МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«ВЕБ додаток - кікстратер для контентмейкерів. »

Завідувач
випускаючої кафедри **А.С. Довбиш**

Керівник роботи **О.Б. Берест**

Студента групи ІН – 71 **М.О. Бірінцев**

СУМИ 2021

ЗМІСТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності "Інформатика" денної форми навчання Бірінцева Михайла Олександровича.

Тема: " ВЕБ додаток - кікстратер для контентмейкерів. "

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) Актуальність проблеми; 2) Огляд аналогів; 3) Вибір технологій; 4) Дизайн додатку; 5) Основна частина; 6) Етап тестування; 7) Кошторис.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ О.Б. Берест

Завдання приняв до виконання _____ М.О. Бірінцев

РЕФЕРАТ

Записка: 91 стор., 21 рис., 2 табл., 1 додаток, 16 джерел.

Об’єкт дослідження (розробки) — система просування творчого контенту.

Мета роботи — аналіз (предметної області, кошторису, аналогів), проектування, прототипування та розробка системи творчого просування контенту.

Методи дослідження (розробки) — розробка WEB-додатку з використанням таких технологій.як Java (Spring, Thymeleaf), Javascript (JQuery) та PostgreSQL.

Результати — на основі попередньо проведених аналізу та проектування, розроблений WEB-додаток із використанням вищезазначених технологій. Проведений пошук аналогів та порівняння з ними, на підставі чого були зроблені висновки щодо подальшої розробки (рефакторинг поточного функціоналу та додавання нових опцій використання). Проведена оцінка бажаного фінансування, необхідного для підтримання життєвого циклу проекту.

ІНФОРМАТИКА, ВСЕСВІТНЯ МЕРЕЖА, ДОДАТОК, ДИЗАЙН,
КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ТВОРЧІСТЬ,
МАРКЕТПЛЕЙС, JAVA, POSTGRES, JAVASCRIPT.

1. АКТУАЛЬНІСТЬ ПРОБЛЕМИ

У сучасному світі індивідуалізму, капіталізму і глобалізму є великий простір для розвитку особистості та творчості. Цьому також сильно посприяв інтернет, його поширеність і доступність. Люди отримали доступ до інформації раніше не доступної. Наприклад: книги, фільми, лекції, філософські роботи та багато іншого. Все це створило людей, які володіють різними вміннями і знаннями.

Але далеко не кожен з них має можливість монетизації своїх умінь. Людині, яка проживає в невеликому містечку чи селі, та котра володіє специфічними знаннями і навичками, наприклад, знання про аналітичну філософію або техніки аерографії, може бути важко знайти своїх глядачів, учнів або патронів.

Допомогти їм у цьому може інтернет. Актуальність, сервісів фінансової підтримки творчих людей або будь-яких інших контент-мейкерів, є безумовною. Досить подивитися статистику подібних сервісів щоб зрозуміти, що на це є великий попит. Люди дійсно готові підтримувати художників, музикантів, ігрових стрімеров, сучасних філософів і багатьох інших творчих діячів своїми грошима, і їм потрібно в цьому допомогти.

Подібні сервіси також можуть бути цікаві для роботодавців, наприклад умовна приватна фірма може виплачувати зарплату своїм працівникам за допомогою цього сервісу.

І на останок, якщо розглядати моральну сторону питання, то я вважаю, що подібні продукти сприяють різнобічному розвитку людей і різних видів мистецтв.

2. ОГЛЯД АНАЛОГІВ

2.1 ПОРІВНЯННЯ СИСТЕМИ З АНАЛОГАМИ

Провівши перегляд аналогів та порівнявши їх функціонал з функціоналом розроблюваної інформаційної системи можемо зробити висновки щодо переваг нашого веб-додатку, які наведені нижче:

Ознайомлюванні безкоштовні пости. Користувачі мають доступ до безкоштовних постів автора, тобто можуть ознайомитись та оцінити контент контент-мейкер, відповідно зможуть прийняти зважане рішення щодо того чи хочуть вони його підтримуватиме чи ні. Після платної підписки їм стануть доступні також і платні пости.

Можливість оплати криптовалютою. Користувач-підписник зможе оплачувати роботу автора не тільки звичайними валютами, такими як дола чи гривня, а і виконувати оплату штучними валютними системами.

Орієнтованість на анонімність. Серед обов'язкових даних для регістрації є ім'я (нікнейм) та електрона пошта, щодо інших даних користувач обирає сам чи хоче він ділитися цією інформацією з іншими користувачами чи ні. Тобто додаток орієнтований на захист персональних даних. Автори і підписники можуть бути повністю анонімними.

Мультимовність. Інтерфейс перекладений на найбільш поширені та найбільш популярні мови, що дає змогу користуватися веб-додатком ще більшій групі людей, ніж наприклад Патреон, інтерфейс якого лише на англійській мові.

Знижена комісія. Кошти йдуть безпосередньо від фаната до контент-мейкера. А додаток заробляє гроші так: коли контент-мейкер отримує в місяць більше певної суми тоді він зобов'язаний платити якийсь податок. Це як перевага в порівнянні з іншими сервісами, так як деякі подібні сервіси приймаю спочатку кошти на свої рахунки а потім їх переводять до контент-мейкерів. а значить вони можуть заморозити гроші мейкерів, якщо вони

раптом вважатимуть що не хочуть з ними співпрацювати. а коли кошти безпосередньо йдуть від фанатів до мейкерам то такого не трапиться.

Зручність інтерфейсу. Інтерфейс сайту побудований з використанням останіх тенденцій розробки графічного дизайну. Дизайн сайту виконаний з приділенням великої уваги на ергономіки. Оформлення контенту, сукупність всіх графічних елементів на веб-сторінці зроблені якнайкраще, щоб познайомити користувача з наповненням та структурою інформації.

Відміна підписки у будь-який час. Користувач-підписник може відмінити платну підписку у будь-який час, адже в людей змінюються смаки, або автор вже не влаштовує підписника.

Корегування щомісячної плати. Користувач-підписник, який хоче отримати доступ до платного контента автора, може сам обрати яку суму перераховувать щомісяця контент-мейкеру. Для користувача-підписника встановлена лише нижня границя не менше 1 долара за місяць, а верхня границя відсутня, адже користувач-підписник може сам визначити настільки дорого він оцінює автора.

2.2 ФОРМУВАННЯ ВИМОГ

Незалежно від того, в якій галузі ви займаєтесь, постійним протягом життєвого циклу проекту є оформлення документів. Технічна документація - це система графічних і текстових документів, необхідних і достатніх для безпосереднього використання її всіх етапах життєвого циклу інформаційної системи або іншого технічного засобу (проектування, виготовлення і експлуатація промислової продукції, при проектуванні, будівництві та експлуатації будівель і споруд, в розробка виробничих процесів, розробка і використання програмного забезпечення).

Завжди є багато документів, які можна створити, затвердити, скласти та заархівувати. Всі ці документи є важливими, але Statement of work (SOW) є

одним із найважливіших, оскільки він складається на початку проекту та описує все, що буде потрібно в ході розробки вашого проекту.

SOW - це документ, який фіксує та визначає всі аспекти вашого проекту. Ви визначаєте в ньому діяльність, результати та графік реалізації проекту. Це надзвичайно детальний документ, оскільки він закладе основу для плану проекту. Це один із перших документів, які створюються, щоб викласти всі деталі проекту перед тим, як планувати та виконувати. [3]

Основним розділом SOW є перелік вимог до програмного продукту. Вимоги (Requirements) – це це перелік параметрів яким має відповідати продукт після завершиння етапу розробки та тестування. Вимоги зазначаються чітко, ясно, в уніфікованій формі, з розкритям всіх особливостей розроблюваної системи.

STATEMENTS OF WORK

Вимоги

ACCOUNT

Кожен користувач повинен мати можливість зареєструватися.

Після подання реєстраційної форми користувач повинен активувати свій акаунт (за допомогою електронного листа з підтвердженням, надісланого на адресу, яку користувач вказав під час реєстрації).

Якщо користувач не вважається активним, він / вона не зможе ввійти.

Нижче наведено перелік параметрів реєстраційної форми (форма в подальшому буде перекладена на інші мови):

- Name (2…128 any characters inclusive, whitespace value is not allowed) *
- Birthday
- E-mail (0…128 characters, unique within the system) *
- Gender
- Password ([8…128] characters)

* – required parameters.

PUBLICS

Кожен зареєстрований користувач повинен мати можливість створити до 128 публік. Коли користувач створює загальнодоступну інформацію, він стає її власником. Кожен паблік може мати лише одного власника. Власник повинен мати можливість розміщувати матеріали на своїй стрічці

Кожен зареєстрований користувач повинен мати можливість підписатися на необмежену кількість пабліків. Одразу після підписки користувач повинен мати можливість побачити видані матеріали від загальнодоступних на веб-сайті каналу. Кожен користувач повинен мати можливість скасувати підписку на загальнодоступну (раніше підписану). Відразу після скасування підписки на паблік користувача не повинен бачити матеріалів цього пабліка в себе в стрічкі.

Назва пабліка має такі обмеження:

- Length [2; 128] characters

Кожен зареєстрований користувач повинен мати можливість перейменувати власні пабліки.

Кожен зареєстрований користувач повинен мати можливість шукати публікації за іменами.

Кожен зареєстрований користувач повинен мати можливість переглянути всі власні пабліки.

Кожен зареєстрований користувач повинен мати можливість переглянути всі свої публікації, на які він підписався.

Лише авторизовані користувачі можуть мати можливість переглядати стрічку сайту.

FREE/PAID PUBLICATIONS

Кожен власник групи повинен мати можливість публікувати платні та безкоштовні матеріали. Якщо користувач безкоштовно підписується на

паблік, його / її стрічка повинна містити лише безкоштовні матеріали. Якщо користувач підписався на підписку з платним тарифом, його / її стрічка повинна містити як безкоштовні, так і платні публікації.

Підписка являється місячною, оплата знімається раз в місяць, щомісяця в той же день коли була здійснена підписка.

Назва публікації повинна відповідати наведеним нижче вимогам:

- Length [2; 128] characters

Текст публікації не повинен перевищувати 4096 символів.

Користувач повинен мати можливість оновити свій тариф до платного або повернутися до безкоштовного в будь який час. При цьому платні пости доступні при платній підписці мають залигитися доступними і користувач має мати до них доступ весь час навіть після відписки.

Після відписки гроші не повертаються, навіть якщо зняття коштів пройшло нещодавно і місяць доступу лише почався

ENTITES SYNONIMS

- public = group = profile
- user = account = person

3. ВИБІР ТЕХНОЛОГІЙ

3.1 МОВА ПРОГРАМУВАННЯ

Для дипломного проекту було необхідно вибрати мову програмування, яка змогла би бути простою використанні, популярною і затребуваною серед сучасних IT компаній, крос-платформною, надійною і безпечною. Всі перераховані вище критерії поєднує в собі, розроблена в 1995 році і популярна досі, мова Java.

Спочатку мова була створена для програмування побутових електронних пристроїв, але дуже скоро стало ясно, що її можливості набагато ширші. Мову взяли на озброєння розробники серверного ПО і клієнтських додатків. Так Java почала підкорення світу. На сьогоднішній день вона стала більш популярною ніж, навіть такі поширені, мови як C ++ і C #. [1]

Сьогодні Java домінує в розробці корпоративних додатків, веб-сайтів для великих проектів e-commerce, мобільних додатків. У світі понад 10 мільйонів Java-розробників і більше 3 мільярдів пристроїв, на яких використовується Java.

Ця мова - беззмінний лідер серед десятків інших. Згідно з рейтингом TIOBE, в якому ЯП розташовані за кількістю пошукових запитах на порталах Wikipedia, Google, YouTube та інших, Java займає перше місце з часткою 16% (результати травня), випереджаючи на кілька відсотків мову C і вдвічі - C ++ і Python.

Головний плюс Java - принцип "написано раз - працює скрізь". Це означає, що ПО, написане на одній платформі, буде запускатися і на інших пристроях. В принципі, Java буквально здатна "співати з кожного праски": вона використовується для створення додатків для мобільних пристроїв, віддалених процесорів, бездротових модулів, датчиків, та й в цілому - практично будь-яких електропристроїв. Адаптивність - одна з причин, по якій Twitter перейшов на JVM.

Java - в числі найпопулярніших мов на GitHub за кількістю коммітов. Мова затребуваний завдяки величезному вибору бібліотек під будь-які завдання. Ну а багатомільйонне співтовариство постійно нарощує їх кількість.

Таким чином, Java постійно розвивається і творцями мови, і його "користувачами".

Завдяки різноманітності бібліотек цей ЯП гнучкий, тому він прекрасно підходить для реалізації нових функцій. І для великих корпорацій, і для компаній меншого масштабу це цінно. Так, гнучкість мови допомогла компанії Spotify побудувати набір модулів для розробки мікросервісів Apollo. [2]

Довгострокова перспектива є одним з неочевидних плюсів даної мови. Легкість у використанні і розумінні дозволяє, при необхідності, розширювати кількість людей, які працюють над продуктом, і спрощує етап підтримки програмного забезпечення.

Можна витратити багато часу на перерахування переваг мови Java, але варто також згадати і недоліки, які безумовно повинні бути. Продуктивність довгий період ставилася під сумнів. Але варто врахувати, що Java постійно оновлюється і її сучасні динамічні компілятори, у швидкодії, практично не поступаються компіляторам з інших платформ. А також не варто забувати, що разом з розвитком програмного забезпечення, розвивається і апаратне забезпечення, отже і обчислювальні потужності. Тому при виборі Java, як основної мови програмування для вашого продукту, не варто сильно хвилюватися про можливі проблеми продуктивності.

Популярність та наявність великої кількості навчальних матеріалів та спільноти програмістів Java та чудова пристосованість цієї мови програмування до розширення та збільшення модульності проекту є гарною підставою для використання її в таких великих та амбіційних проектах, який розробляється в цій роботі.

4. ДИЗАЙН ДОДАТКУ

4.1 ENTITY RELATIONSHIP DIAGRAM

Entity Relationship Diagram (ERD) - діаграма взаємозв’язків, також відома як ERD, ER Diagram або ER модель, - це тип структурної діаграми, що використовується для проектування бази даних. ERD містить різні символи та з'єднувачі, які візуалізують дві важливі відомості: основні об'єкти в межах системи та взаємозв'язки між цими об'єктами. [4]

На основі аналізу аналогів та на основі даних про розробку веб-додатків, спираючися на документ вимоги була розроблена діаграма взаємозв’язків:

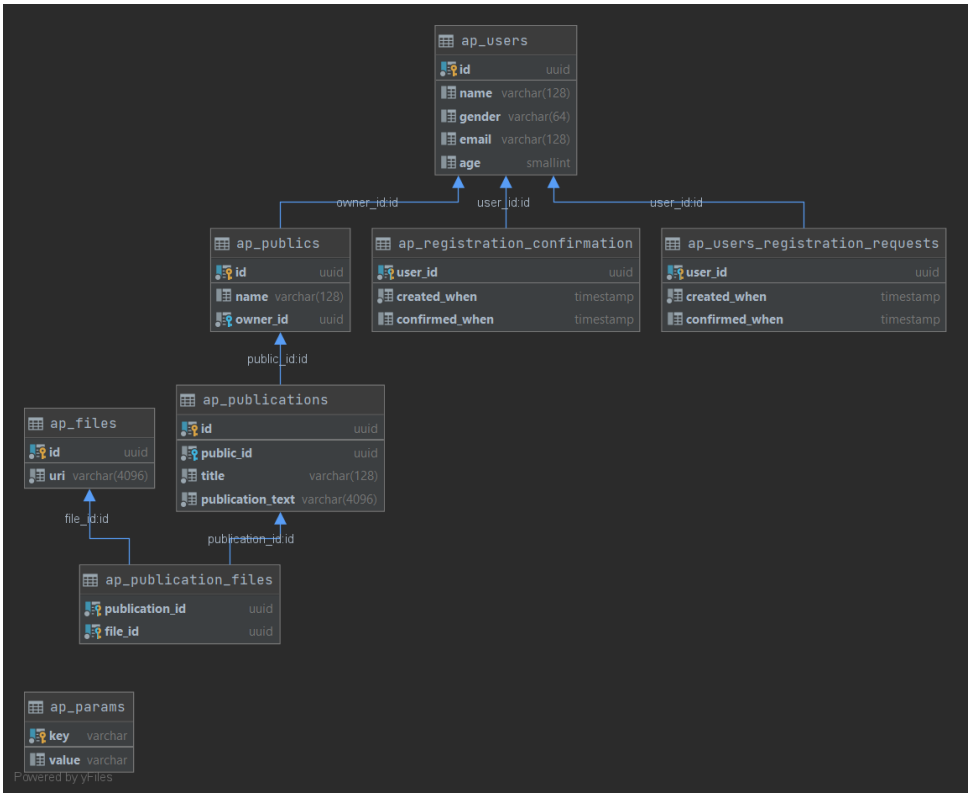

Рисунок 1 – ER діаграма

4.2 USE CASE ДІАГРАМА

UML use case diagram (діаграма використання) - це основна форма системних / програмних вимог до нової слаборозвиненої програми. У випадках використання вказується очікувана поведінка (що), а не точний спосіб її здійснення (як). Описані випадки використання можна позначити як текстове, так і візуальне представлення (тобто діаграму використання). Ключова концепція моделювання випадків використання полягає в тому, що це допомагає нам розробити систему з точки зору кінцевого користувача. Це ефективна техніка передачі поведінки системи з точки зору користувача шляхом визначення всієї видимої зовні системи поведінки. [5]

Основні елементи діаграми - учасник (actor) і прецедент (варіант).

Учасник - це безліч логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистема або клас). Учасником може бути людина або інша система, підсистема або клас, які представляють щось поза суті. Графічно учасник зображується "чоловічком".

Прецедент (use case) - опис безлічі послідовних подій (включаючи варіанти), що виконуються системою, які призводять до спостережуваного учасником результату. Прецедент являє поведінку суті, описуючи взаємодію між учасниками і системою. Прецедент не показує, "як" досягається певний результат, а тільки "що" саме виконується. Прецеденти позначаються дуже простим чином - у вигляді еліпса, всередині якого вказано його назву [6]

Основні елементи діаграми варіантів використання

Існує безліч програм та веб-додатків для полегшення побудови use case діаграм, зокрема draw.io, Flexberry Designer та багато інших. Вони пропонують зручні програмні інструменти для побудови діаграм, готові блоки та інколи підказки, також в багатьох додатках є заготовки. Більшість таких додатків можно використовувати і для побудови інших діаграм користуючись методологією UML

На діаграмі варіантів використання можна відобразити такі елементи нотації UML, доступні в панелі елементів деяких відомих програм та веб додатків:

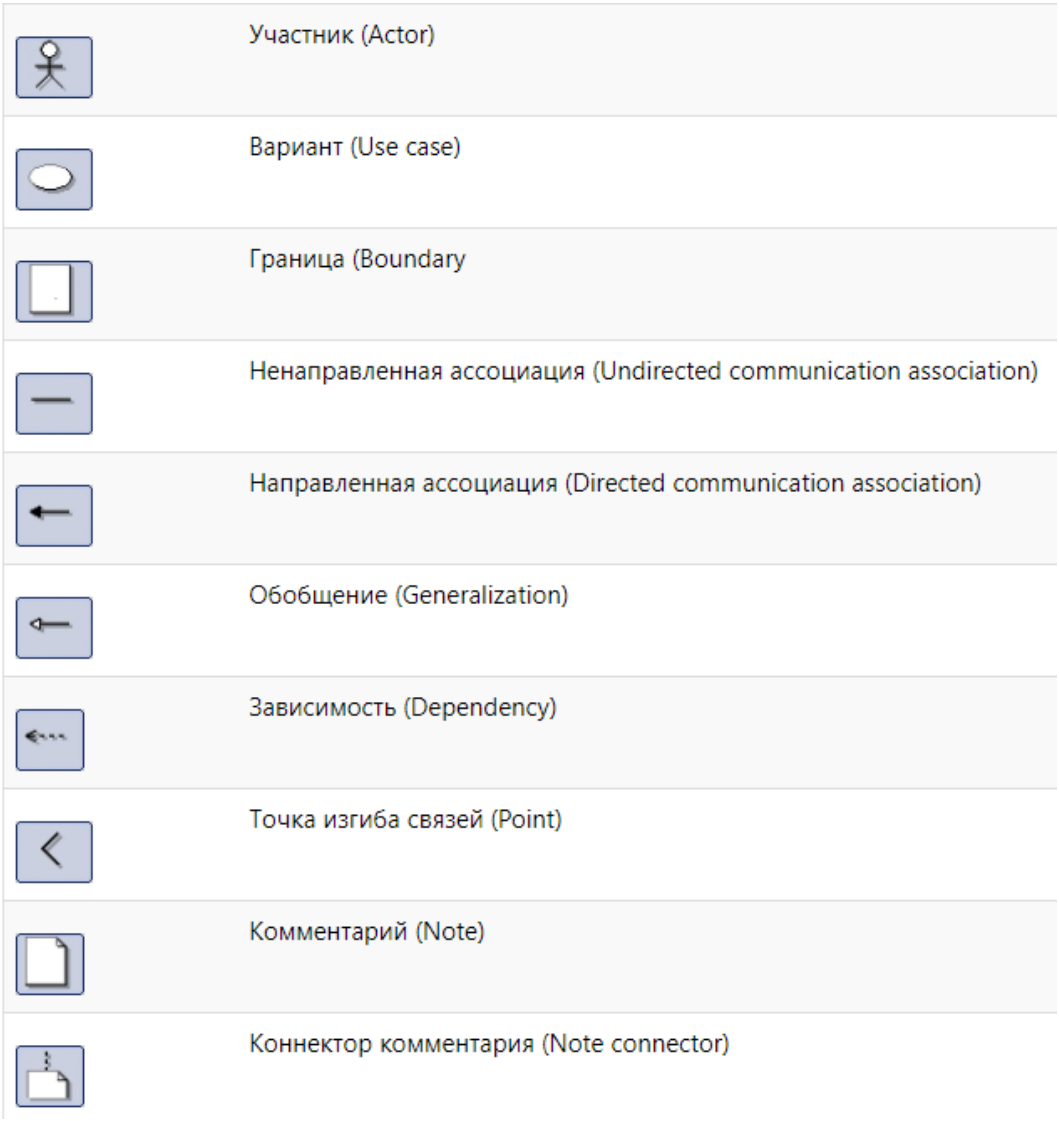

Рисунок 2 – Умовні позначення use case діаграми

Extended використовується, коли варіант використання умовно додає кроки до іншого випадку використання першого класу. Наприклад, уявіть, що "зняття готівки" - це варіант використання банкомата. "Оцінка комісії"розширить Зняття готівки та опише умовну "точку продовження", тобто інстанціюється, коли користувач банкомату не здійснює банку у власнику банкомату. Зверніть увагу, що основний випадок використання "зняти готівку" стоїть сам по собі, без розширення.

Includeed використовується для вилучення фрагментів випадків використання, які дублюються у багатьох випадках використання.включений варіант використання не може стояти окремо, а оригінальний варіант використання не є повним без включений один. Це слід використовувати економно, лише у випадках, коли дублюється значущим і існує за задумом (а не за збігом обставин). Наприклад, потік подій, що відбувається на початку кожного випадку використання банкомату (коли користувач кладе свою карту банкомата, вводить свій PIN-код і відображається головне меню) буде а хороший кандидат для включення.

Ми використовуємо draw.io. Draw.io - інструмент для створення діаграм, блок-схем, інтеллект-карт, бізнес-макетів, відносин сущностей, програмних блоків та іншого. Сервіс розповсюджується на безкоштовній основі з відкритим вихідним кодом. Draw.io має багатий набором функцій для візуалізації більшості завдань користувача.

При вході до сервісного користувача підключення до робочого інтерфейсу. У користувача немає можливості авторизації або реєстрації, є лише опція вибору місць для експорту проекту. Процес створення проекту виглядає наступним чином: користувач перетаскує з левої панелі фігури або елементи на робочу поверхню, потім змінює їх - змінює колір, розмір, шрифт тексту, властивості фігур (прозорість, форма та т. Д.) Draw.io дозволяє відслідковувати та відновлювати зміни готових проектів, імпортувати та експортувати в PDF, PNG, XML, VSDX, HTML, а також автоматично публікувати та видаляти роботи.[11]

Спираючись на попередню діаграму, з конкретизацією вимог та включеням до уваги нового функціоналу була розроблена оновлена та покращена use-case діаграма:

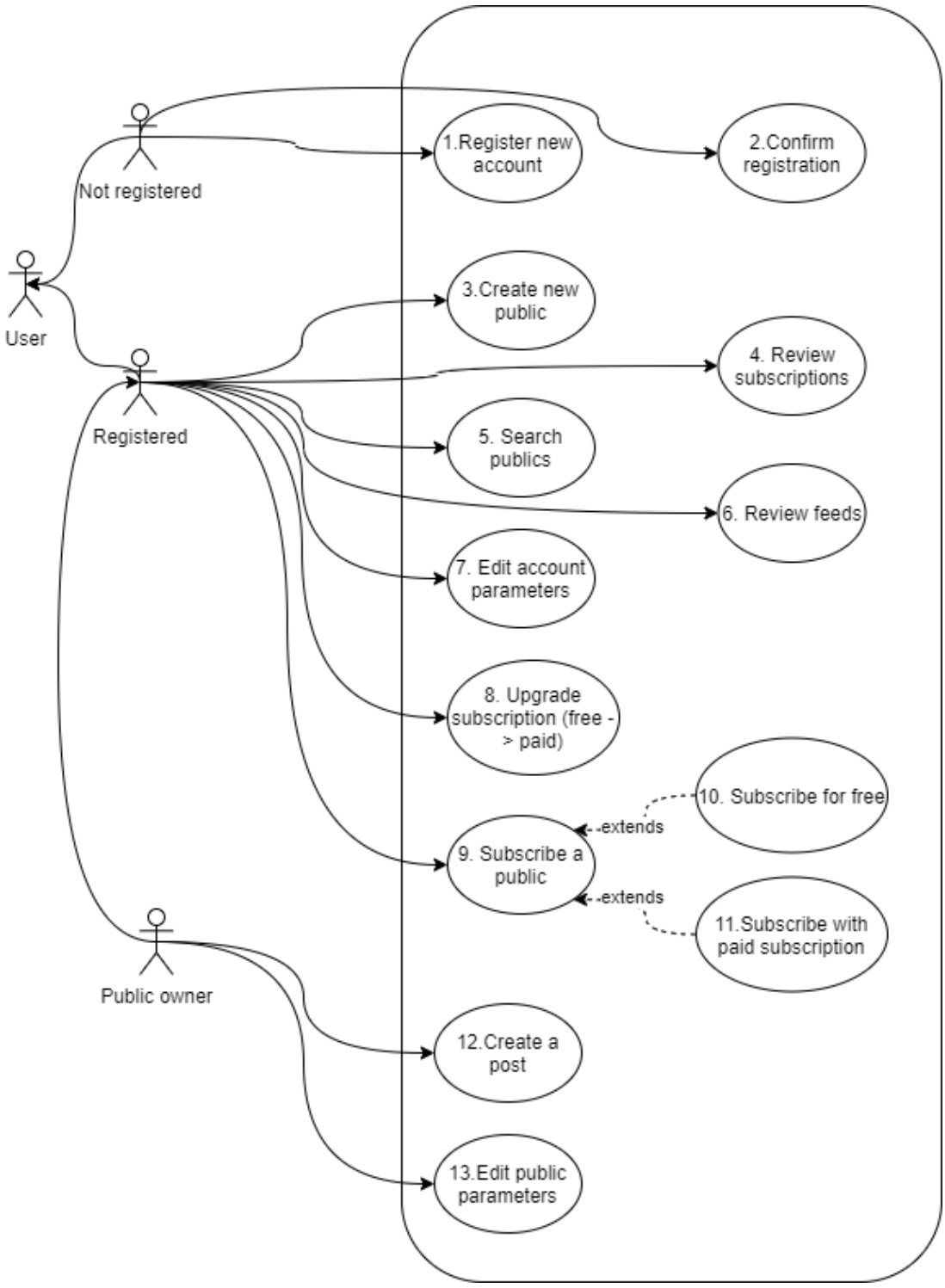

Рисунок 3 – Use case діаграма

Виконавці вони ж учасники - це безліч логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистеми або клас). Учасником може бути людина або інша система, підсистема або клас, які представляють щось поза суті системи. Графічно учасник зображується "чоловічком".[6]

Серед учасників майбутньої системи:

- **User** – потенційний користувач системи
- **Not registered** - користувач системи, котрий ще не має акаунту, та не має доступу до основного функціоналу інформаційної системи
- **Registered** - користувач системи який виконав функцію реєстрації та підтвердження пошти та може використовувати основний функціонал веб-додатку.
- **Public owner** – користувач системи, котрий створив паблік, та має доступ до функціоналу зв’язаного з керуванням цього пабліку.

Наступним і основним елементом use cases діаграми є способи використання, позначаються хмаринками на діаграмі:

1. **Register new account.** Функція реєстрації нового акаунту. Функція доступна для лише not registered користувачів, користувач повинен ввести необхідні згідно документованих вимогу дані, та за бажанням додаткові дані.
2. **Confirm registration .** Not registered користувач може стати registered користувачел лише після того як підтвердить свою пошту та лише після цього йому стане доступний основний функціонал веб додатку.
3. **Create new public.** Функція яка доступна registered користувачу, результатом її виконання є створений новий падбік, а registered користувач стає public owner (власником пабліку) та отримує доступ до функцій зв’язаних з керуванням пабліків 12 і 13.
4. **Review subscriptions.** Registered користувач має можливість переглянути перелік пабліків на які він підписаний, навіть якщо він ще не підписався на жодний – в такому випадку у нього буде відображатись відсутність підписок

5. **Search publics.** Registered користувач може виконати дію пошуку пабліку з контентом за його ім’ям.

6. **Review feeds.** Registered користувач може переглядати свою стрічку, в якій будуть відображатися пости авторів по безкоштовних та платних підписках.

7. **Edit account parameters.** Registered користувач може змінити дані акаунту, які не зазначені як required в документі вимог та також може змінити ім’я, або інші операції по редагуванню профілю.

8. **Upgrade subscription (free > paid).** Registered користувач при умові виконання підписки на паблік 8 може змінити за бажанням безкоштовну підписку на платну та перераховувати кошти автору платних публікацій.

9. **Subscribe a public.** Registered користувач може безкоштовно підписатися (use case 10) на паблік який йому необхідний або сподобався, для підтримки автора registered користувач користувач може скористатися use case 11. При виконанні use case 9 включними use cases є 10 і 11.

10. **Subscribe for free.** Після use case 9, registered користувач може обрати безкоштовну підписку.

11. **Subscribe with paid subscription.** Після use case 9, registered користувач може обрати платну підписку.

12. **Create a post.** Public owner – користувач може викладувати пости власної творчості в себе в паблiку.

13. **Edit public parameters.** Користувач public owner може змінювати та редагувати параметри власного пабліка.

Use case діаграма підкреслює основний функціонал додатку і в подальшій розробці служитиме для підкреслення задач які мають бути в пріоритеті.

4.3 CLASS DIAGRAM

Class diagram UML - це графічне позначення, що використовується для побудови та візуалізації об'єктно-орієнтованих систем. Діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи системні:

- класи,
- їх атрибути,
- операції (або методи),
- та взаємозв'язки між об'єктами.

Класи – це конструкції спеціального виду, які дозволяють об'єднати ряд змінних різних типів в одне ціле. Крім власне даних, класи зазвичай включають підпрограми (в термінології java - методи) і можуть включати блоки (сукупність інструкцій між фігурними дужками {}) та інші класи (внутрішні класи).[9]

Клас представляє концепцію, яка інкапсулює стан (attributes) та поведінку (operations). Кожен атрибут має тип. Кожна операція має підпис. Назва класу є єдиною обов’язковою інформацією (mandatory). Класи позначаються квадратами, а зв’язки між ними стрілочками.

Перспективи класної діаграми

Вибір перспективи залежить від того, наскільки далеко ви знаходитесь у процесі розробки. Наприклад, під час формулювання моделі домену ви рідко проходили повз концептуальну перспективу. Моделі аналізу, як правило, мають поєднання концептуальних та специфікаційних перспектив. Розробка дизайнерської моделі, як правило, починається з великого акценту на перспективі специфікації та переростає у перспективу реалізації. [7]

Діаграму можна інтерпретувати з різних точок зору:

- Концептуальний: представляє поняття в домені

- Специфікація: основна увага приділяється інтерфейсам абстрактних типів даних (ADT) у програмному забезпеченні
- Впровадження: описує, як класи будуть реалізовувати свої інтерфейси

Intellij idea та генерація class diagram.

IntelliJ IDEA - це інтегроване середовище розробки (IDE), написане на Java для розробки комп'ютерного програмного забезпечення. Він розроблений JetBrains (раніше відомий як IntelliJ) і доступний як ліцензований випуск спільноти Apache 2 та у власному комерційному виданні. Обидва вони можуть бути використані для комерційного розвитку.

Перша версія IntelliJ IDEA була випущена в січні 2001 року і була однією з перших доступних IDE Java з розширеними можливостями навігації коду та можливостями рефакторингу коду.

У звіті InfoWorld за 2010 рік IntelliJ отримав найвищий бал центру тестування з чотирьох найкращих інструментів програмування Java: Eclipse, IntelliJ IDEA, NetBeans та JDeveloper.

У грудні 2014 року Google оголосив версію 1.0 Android Studio, IDE з відкритим вихідним кодом для програм для Android, засновану на відкритому виданні спільноти IntelliJ IDEA. Інші середовища розробки, засновані на рамках IntelliJ, включають AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm та MPS. [8]

IntelliJ IDEA дозволяє генерувати діаграму класів на основі пакетів у вашому проекті. Такі діаграми відображають структуру фактичних класів та методів у проекті. Генерація, а не розробка діаграми вручну чи іншими стороніми засобами чи програмами полегшує роботу розробникам на етапі дизайну та створення архітектури інформаційної системи. Також усуває необхідність в зміни діаграми в разі зміни структури класів, адже можно просто згенерувать нову діаграму.

Генерація схеми в середовищі розробки:

- У вікні інструмента «Project» клацніть правою кнопкою миші на пакет, для якого потрібно створити діаграму, та виберіть Diagrams/ Show Diagram (Ctrl + Alt + Shift + U).
- У списку, що відкриється, виберіть «Java Class Diagram». IntelliJ IDEA згенерує діаграму UML для класів та їх залежностей.

Так як для розробки було обране саме це інтегроване середовище розробки, ми виконуємо наведений вище алгоритм та створюємо діаграму класів для розроблюваного проекту.

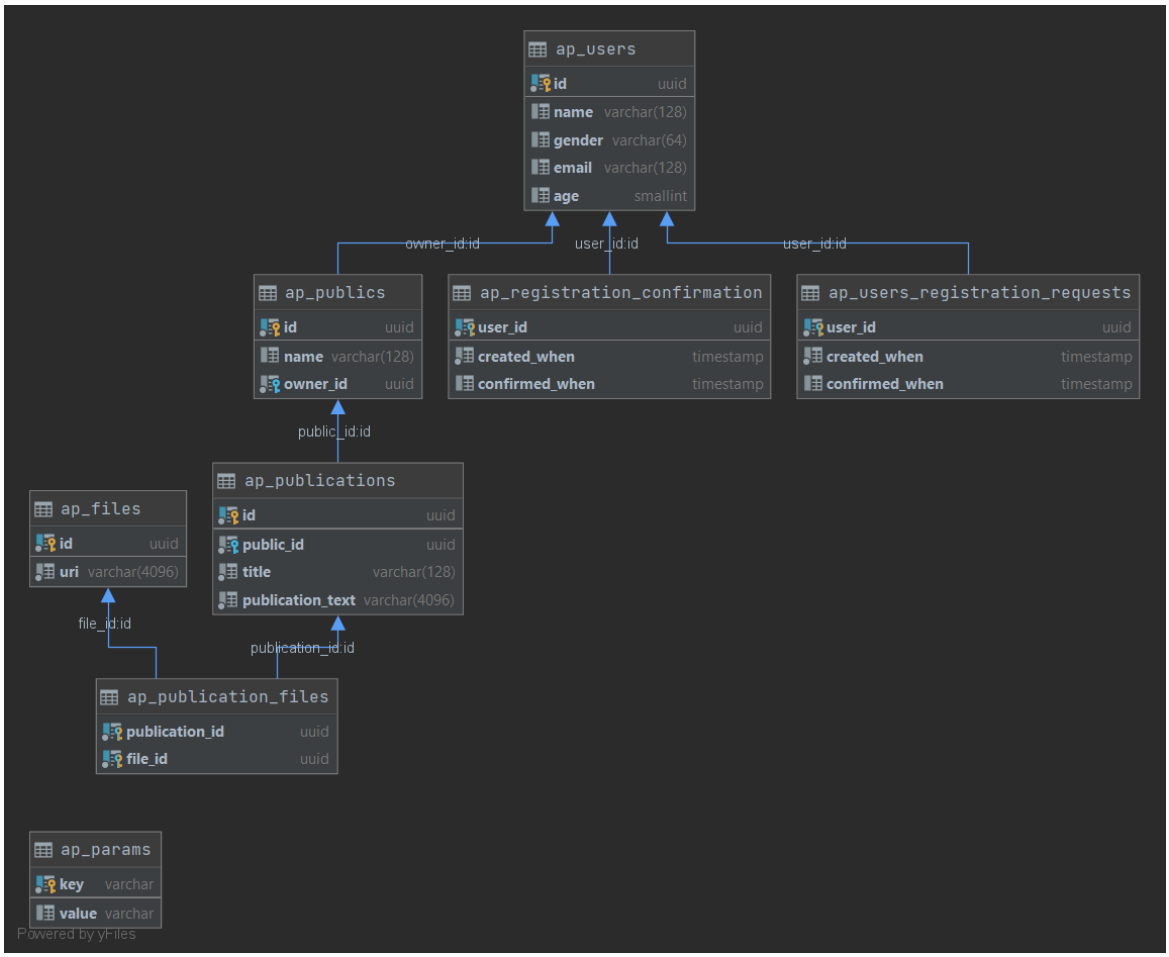

Рисунок 4 – Class diagram

4.4 ЗАГАЛЬНА ДІАГРАМА ФУНКЦІОНУВАННЯ (ПРОЦЕСІВ)

DFD (діаграма потоків даних) - загальноприйняте скорочення від англ. data flow diagrams - діаграми потоків даних. Так називається методологія графічного структурного аналізу, що описує зовнішні по відношенню до системи джерела і адресати даних, логічні функції, потоки даних і сховища даних, до яких здійснюється доступ. Діаграма потоків даних (data flow diagram, DFD) - один з основних інструментів структурного аналізу і проектування інформаційних систем, що існували до широкого поширення UML. [13]

DFD і UML - це різні інструменти, а тому некоректно стверджувати, що DFD - це просто попередник UML. DFD - це нотація, призначена для моделювання інформаційний систем з точки зору зберігання, обробки і передачі даних.[14]

Історично синтаксис цієї нотації застосовується в двох варіантах - Йордана (Yourdon) і Гейне-Сарсона (Gane-Sarson). Відмінності між ними - в таблиці нижче:

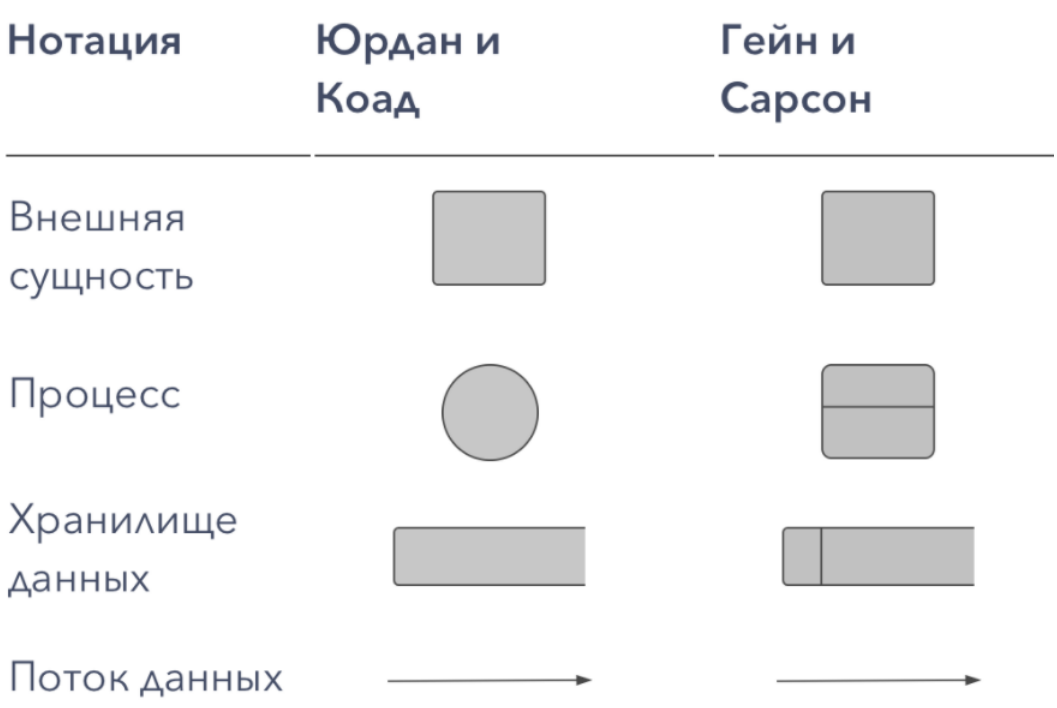

Рисунок 5 – Порівняння нотацій Йордана і Гейне-Сарсона

Безпосередньо DFD нотація складається з наступних елементів:

- **Процес** (англ. Process), тобто функція або послідовність дій, які потрібно зробити, щоб дані були оброблені. Це може бути створення замовлення, реєстрація клієнта і т.д. У назвах процесів прийнято використовувати дієслова, тобто «Створити клієнта» (а не «створення клієнта») або «обробити замовлення» (а не «проведення замовлення»). Тут немає суворої системи вимог, як, наприклад, в IDEF0 або BPMN, де нотації мають жорстко визначений синтаксис, так як вони можуть бути виконуваними. Але все ж певних правил варто дотримуватися, щоб не вносити плутанину при читанні DFD іншими людьми.
- **Зовнішні сутності** (англ. External Entity). Це будь-які об'єкти, які не входять до саму систему, але є для неї джерелом інформації або одержувачами будь-якої інформації з системи після обробки даних. Це може бути людина, зовнішня система, будь-які носії інформації і сховища даних.
- **Сховище даних** (англ. Data store). Внутрішнє сховище даних для процесів в системі. Надійшли дані перед обробкою і результат після обробки, а також проміжні значення повинні десь зберігатися. Це і є бази даних, таблиці або будь-який інший варіант організації та зберігання даних. Тут будуть зберігатися дані про клієнтів, заявки клієнтів, видаткові накладні та будь-які інші дані, які надійшли в систему або є результатом обробки процесів.
- **Потік даних** (англ. Data flow). В нотації відображається у вигляді стрілок, які показують, яка інформація входить, а яка виходить з того чи іншого блоку на діаграмі.

Нотація DFD може описувати будь-які дії, в тому числі, процес продажу або відвантаження товару, роботу з заявками від клієнтів або закупівлі матеріалів, з точки зору опису системи. Ця нотація допомагає зрозуміти, з чого повинна складатися система, що потрібно для автоматизації

бізнес-процесу. Але DFD не є описом безпосередньо бізнес-процесу. Тут, наприклад, немає такого важливого параметра, як час. Також в цій нотації не передбачені умови і «розвилки». У DFD ми розглядаємо звідки з'являються дані, які дані потрібні, їх обробку і куди результати відправити. Тобто в цій нотації описується не стільки безпосередньо процес, скільки рух потоків даних. Для роботи з процесами я рекомендую використовувати BPMN або IDEF3 (про неї я розповім іншим разом).[14]

Які правила необхідно знати, щоб створити DFD діаграму:

- Кожен процес повинен мати хоча б один вхід і один вихід. Сенс процесів тут полягає в обробці даних, а тому процес повинен отримати дані (що входить стрілка) і віддати кудись після обробки (що виходить стрілка);
- Процес обробки даних повинен мати зовнішню входить стрілку (дані від зовнішньої сутності). Задля того, щоб будь-який подібний процес почав працювати, мало використовувати дані зі сховища, має надійти нова інформація для подальшої обробки;
- Стрілки не можуть пов'язувати безпосередньо сховища даних, всі зв'язки йдуть через процеси. Немає сенсу просто переміщати дані з одного місця в інше, а саме так читається прямий зв'язок двох сховищ стрілкою. Дані надходять для того, щоб вироблялися якісь дії, в нашому прикладі - здійснювався процес продажу. А це можливо тільки за допомогою обробки (процесу);
- Всі процеси повинні бути пов'язані або з іншими процесами, або з іншими сховищами даних. Процеси не існують самі по собі, а тому результат повинен кудись передаватися;
- Декомпозиція. У DFD-діаграмах передбачена можливість створювати великі процеси і декомпозировать їх на підпроцеси з докладним описом дій. Наприклад, ми можемо створити процес «створення заявки», який потім декомпозировать на послідовність

дій, наприклад, на отримання заявки, окремо - перевірку і отримання даних клієнта, якщо товар в інтернет-магазині продається під замовлення, то також при формуванні заявки потрібно отримати дані від постачальника про наявність потрібних найменувань і т.д. І тоді на верхній діаграмі у нас буде блок «обробка заявки», а при декомпозіруванії ми отримаємо діаграму з докладною послідовністю дій на цьому етапі. При цьому ні на одному етапі у нас не буде умов і розгалуження. Буде процес і його декомпозиція глибиною до 3-4 рівнів.

Діаграма без декомпозиції, верхній рівень розроблюваного проекту:

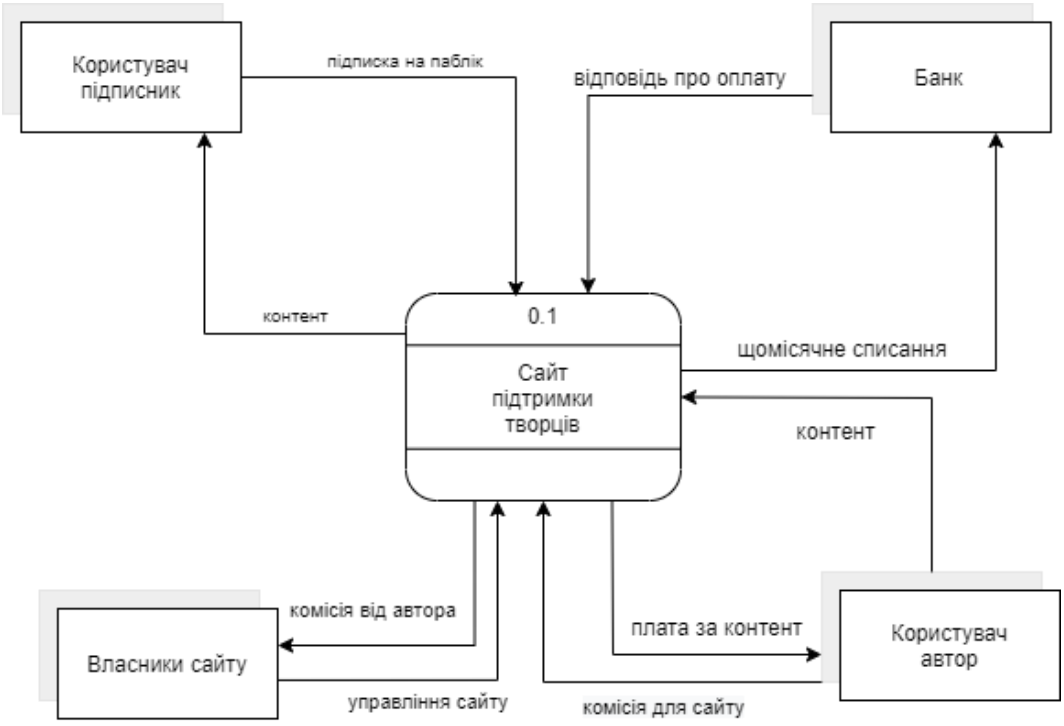

Рисунок 6 – Діаграма DFD0

Провівши детальний перегляд вимог та обдумавши всі майбутні функції ми можемо декомпозувати елемент основного елемента нашої діаграми, тобто перейти на наступний рівень діаграми та розкласти основну функцію «Сайт підтримки творців» на підфункції та розширити потоки даних. Діаграма потоків даних наступного рівня розроблюваного проекта виглядає

таким чином (виконаний з допомогою веб-додатка draw.io – дуже зручний прилад для створення інших схем UML):

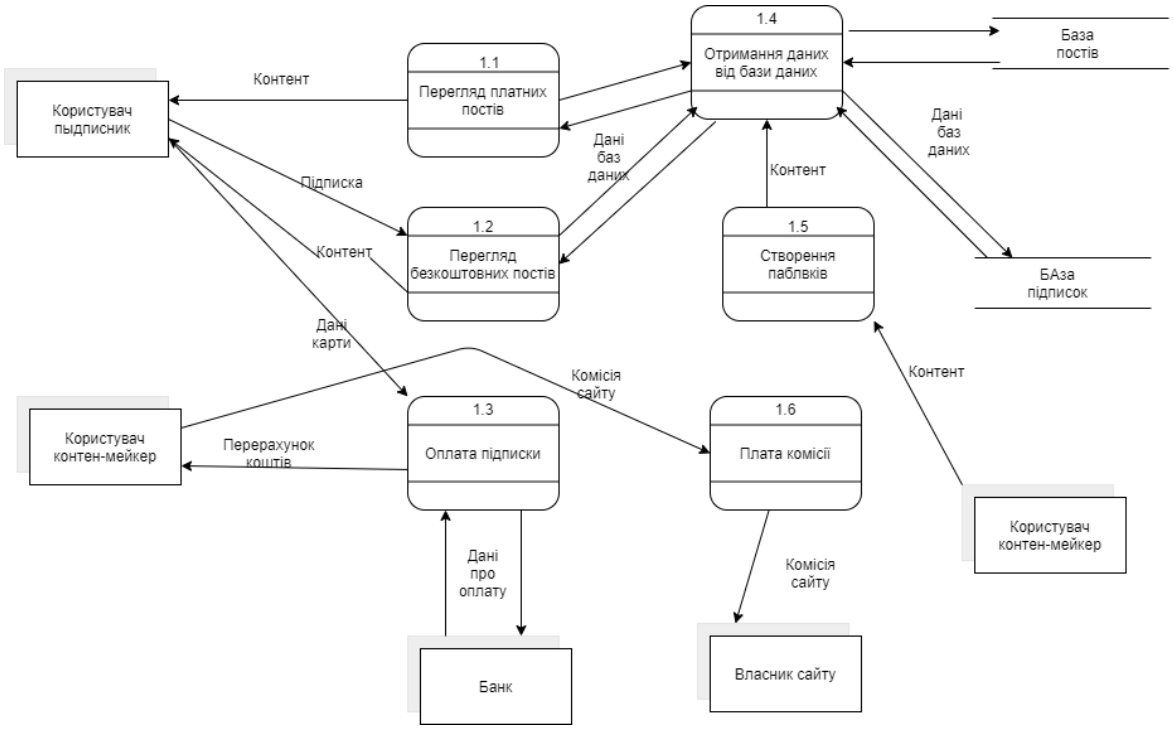

Рисунок 7 – Діаграма DFD1

Елементи (функції) нашого розроблюваного веб-додатку:

1.1 **Перегляд платних постів**. Користувач підписник після використання функції 1.3 отримує доступ до перегляду платних постів та отримує контент. Дані постів зчитуються з бази даних постів через функцію 1.2 Отримання даних від бази даних. Дані про підписку відправляються в базу даних підписок.

1.2 **Перегляд безкоштовних постів**. Користувач підписник після безкоштовної підписки отримує доступ до перегляду безкоштовних постів та отримує контент. Дані постів зчитуються з бази даних постів через функцію 1.2 Отримання даних від бази даних. Дані про підписку відправляються в базу даних підписок.

1.3 **Оплата підписки**. Користувач підписник надає дані карти та оплачує роботу автора та отримує доступ до функції 1.1 Перегляд платних

постів. Дані надходять до Банку та банк перераховує кошти на рахунок Користувача контент-мейкера.

1.4 **Отримання даних від бази даних**. Функція яка слугує для того щоб обмінюватися даними з базою даних – заносити та селектити їх. Необхідна для функцій 1.1 Перегляд платних постів, 1.2 Перегляд безкоштовних постів та 1.5 Створення пабліків.

1.5 **Створення пабліків**.Користувач контент-мейкер створює паблік та вносить дані в нього дані, а саме контент. Контент відправляється в функцію 1.4 Отримання даних від бази даних та заносяться в баз даних.

1.6 **Плата комісії**. Користувач контент-мейкер виплачує комісію сайту з власних кошт, ці кошти отримує власник сайту.

DFD-діаграми активно застосовуються при розробці програмного забезпечення. При цьому:

- сховища даних - це електронні таблиці і бази даних,
- зовнішні сутності - клієнти або інші бази даних, в тому числі, з інших програм (інтеграція і обмін даними),
- процеси - це виконувані функції і модулі в системі.

Також DFD нотації зручні при аналізі, коли система розглядається з точки зору документообігу. При цьому можна наочно побачити, де зберігаються дані, яким чином проводиться обмін документацією, де в цьому процесі допущені помилки організації бізнес-процесів та ін. Але тут застосування DFD діаграм вимагає особливої обережності. Все ж це не опис бізнес-процесу як такого, а, скоріше, діаграма переміщення даних при реалізації бізнес-процесів. Але як допоміжний варіант, в тому числі, для наочної демонстрації клієнту існуючих проблем і методів оптимізації роботи, цей вид нотацій цілком підійде.[14]

Наприклад, для виявлення проблем документообігу, дублювання документів або, навпаки, відсутньої документації або електронних даних в

системі, дуже зручно створити окремо - опис бізнес-процесу, а потім до нього - DFD-нотацію. Або навпаки, попередньо для розуміння основ роботи бізнесу та особливостей реалізації документообігу створюється DFD-нотація. Вона допомагає виявити, наприклад, відсутність в системі автоматизації важливих документів, які насправді створюються (на папері), але в системі ніяк не відображаються. А потім вже будується оптимізований бізнес-процес з урахуванням виявлених нюансів документообігу.

В DFD-нотаціях особливо зручно, тут не обов'язково дотримуватися строгих правил і синтаксису, як, наприклад, в BPMN. Ці нотації НЕ будуть здійснимими, вони потрібні для розуміння особливостей документообігу, структури та подальшої роботи з даними. А тому, якщо ваша діаграма зрозуміла і вам, і замовнику, якісь відступи від стандартів DFD цілком припустимі.

Малювати діаграми DFD можна, в принципі, де і як вам зручніше. Але якщо ви хочете працювати з декомпозицією, вибудовувати систему на різних рівнях деталізації, то «рисовалки» (Visio, Paint і тому подібні) доведеться забути. Вам будуть потрібні спеціалізовані програми для моделювання. В цьому проекті ми використали draw.io.

4.5 ДІАГРАМА КОМПОНЕНТНИХ БЛОКІВ

У той час як інші UML діаграми, які описують функціональність системи, компонентні діаграми використовуються для моделювання компонентів, які допомагають зробити ці функціональні можливості.

Схема компонентів, також відома як схема компонентів UML, описує організацію та з'єднання фізичних компонентів у системі. Діаграми компонентів часто складаються, щоб допомогти деталям реалізації моделі та перевірте, чи кожен аспект необхідних функцій системи охоплюється запланованою розробкою. У першій версії UML компоненти, включені до цих діаграм, були фізичними: документи, таблиця бази даних, файли та виконувані

файли, усі фізичні елементи з місцем розташування. У світі UML 2 ці компоненти є менш фізичними та більш концептуальними окремими елементами дизайну, такими як бізнес-процес, який забезпечує або вимагає інтерфейсів для взаємодії з іншими конструкціями в системі. Фізичні елементи, описані в UML 1, як файли та документи, тепер називаються артефактами. Компонент UML 2 може містити кілька фізичних артефактів, якщо вони, природно, належать разом [12]

Основні компоненти діаграми, символи та позначення:

- **Компонент** є логічним блоком системи, трохи вищою абстракцією, ніж класи. Він представлений у вигляді прямокутника з меншим прямокутником у верхньому правому куті з вкладками або словом, написаним над назвою компонента, щоб допомогти відрізнити його від класу.

Рисунок 8 – Компонент

- **Інтерфейс** (маленьке коло або напівкруг на паличці) описує групу операцій, які використовуються (потрібно) або створюються (надаються) компонентами. Повне коло представляє інтерфейс, створений або наданий компонентом. Півколо представляє необхідний інтерфейс, як введення людини.

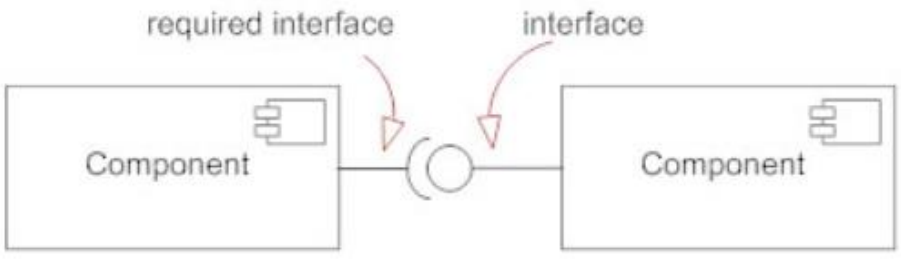

Рисунок 9 – Інтерфейс

- Порти представлені за допомогою квадрата по краю системи або компонента. Порт часто використовується для розкриття необхідних та наданих інтерфейсів компонента.

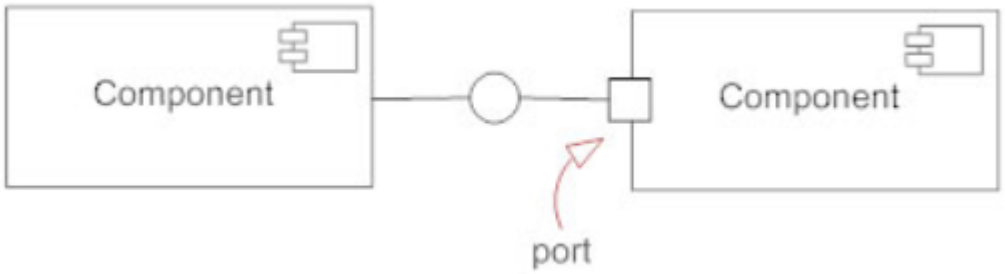

Рисунок 10 – Порт

Ви можете використовувати компонентну діаграму, коли хочете представити свою систему як компоненти і показати їх взаємозв'язок через інтерфейси. Це допомагає отримати уявлення про впровадження системи. Нижче наведені дії, які можна виконати при побудові компонентної діаграми.

Крок 1: З'ясуйте призначення діаграми і ідентифікуйте артефакти, такі як файли, документи і т.д. у вашій системі або додатку, які необхідно подати на діаграмі.

Крок 2: У міру з'ясування взаємозв'язків між елементами, які ви визначили раніше, створіть ментальний макет своєї компонентної діаграми

Крок 3: У міру того, як ви малюєте діаграму, спочатку додайте компоненти, групуючи їх усередині інших компонентів, як вам здається відповідним

Крок 4: Наступним кроком є додавання інших елементів, таких як інтерфейси, класи, об'єкти, залежно тощо в вашу компонентну діаграму і її завершення.

Крок 5: Ви можете докласти примітки до різних частин вашої компонентної діаграми, щоб прояснити деякі деталі іншим.[10]

У цьому навчальному посібнику ми розглянемо, що таке компонентна діаграма, символи компонентної діаграми і як їх намалювати. Ви можете використовувати приклад компонентної діаграми нижче, щоб отримати швидкий старт. [10]

Діаграми компонентів використовуються для візуалізації організації компонентів системи і залежностей між ними. Вони дозволяють отримати високорівневе уявлення про компоненти системи.

Компонентами можуть бути програмні компоненти, такі як база даних або призначений для користувача інтерфейс; або апаратні компоненти, такі як схема, мікросхема або пристрій; або бізнес-підрозділ, таке як постачальник, платіжна відомість або доставка.[10]

Компонентні діаграми

- Використовуються в компонентно-орієнтованих розробках для опису систем з сервіс-орієнтованою архітектурою
- Показати структуру самого коду
- Може використовуватися для фокусування на відносинах між компонентами, приховуючи при цьому деталізацію специфікації
- Допомога в інформуванні та роз'ясненні функцій створюваної системи зацікавленим сторонам

Використовуючи веб додаток draw.io та його панелб інструментів UML на основі даних про наш проект та те що ми від нього очікуємо будуємо діаграму компонентів:

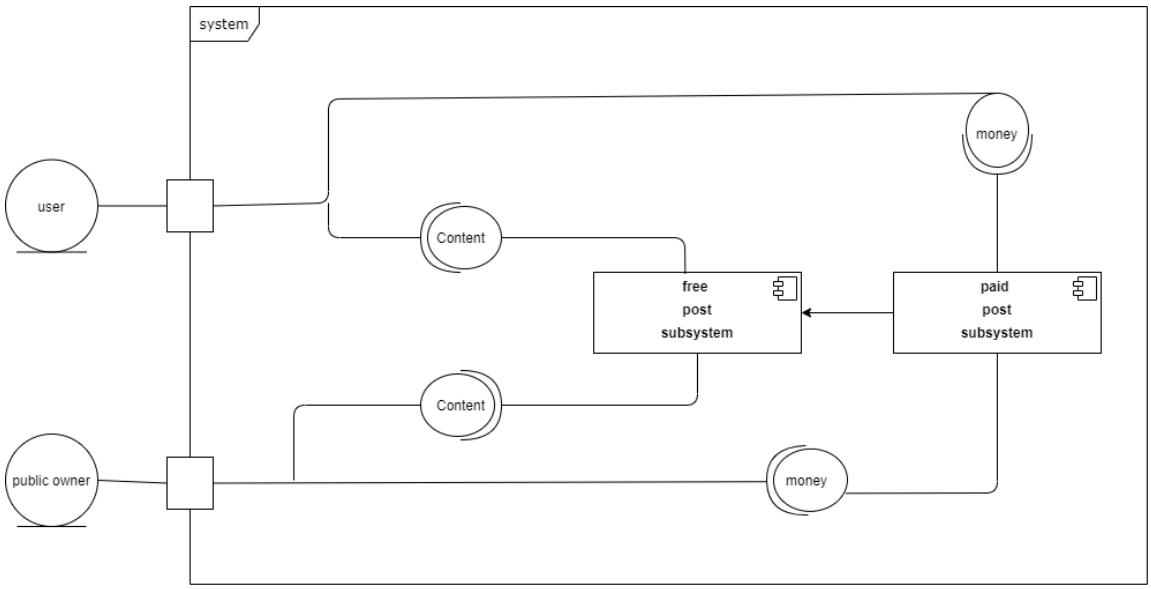

Рисунок 11 – Діаграма компонентів

На основі всіх перелічених діаграм в розділі 4 Дизайн додатку буде виконуватися наступний етап розробки додатку, а саме проектування та програмування, безпосереднє написання коду мовою Java, JavaScript, HTML та інших, з використанням фреймворків згаданих в минулій практиці. Після стадії розробки і під час неї інформаційна система буде піддаватися тестуванню.

5. ОСНОВНА ЧАСТИНА

Основна частина дипломної роботи сфокусована на розробці функціоналу веб додатку спираючись на архітектуру та дизайн розроблених на минулих етапах використовуючи діаграми потоків даних, діаграми класів, діаграми способів використання. В підрозділах буде розглянуто детальний опис розроблених java пакетів.

Пакет, як випливає з назви, являє собою пакет (групу) класів, інтерфейсів та інших пакетів. У Java ми використовуємо пакети для організації наших класів та інтерфейсів. У нас є два типи пакетів на Java: вбудовані пакети та пакети, які ми можемо створити (також відомий як визначений користувачем пакет). Ось підстави, чому ми використовуємо Java пакети:

Багаторазове використання: Розробляючи проект у Java, ми часто відчуваємо, що в нашому коді є кілька речей, які ми пишемо знову і знову. Використовуючи пакети, ви можете створювати такі речі у формі класів всередині пакету, і коли вам потрібно виконати те саме завдання, просто імпортуйте цей пакет і використовуйте клас.

Краща організація: Знову ж таки, у великих Java-проектах, де ми маємо кілька сотень класів, завжди потрібно згрупувати подібні типи класів у значущу назву пакета, щоб ви могли краще організувати свій проект і коли вам щось потрібно, ви могли швидко знайти його і використовувати, що підвищує ефективність.

Конфлікти імен: ми можемо визначити два класи з однаковим іменем у різних пакетах, щоб уникнути зіткнення імен, ми можемо використовувати пакети[1]

Пакети в джава можно поділити на наступні типи:

- Пакети, що визначає користувач джави: пакети які створює програміст, називається user-defined package

- Вбудовані пакети: вже визначений пакет, такий як java.io. *, java.lang. * та ін., їх називають built-in packages.

У розроблюваному проекті використовуються обидва види пакетів.

5.1 РОЗРОБЛЕНІ ПАКЕТИ

Коли ми проглядаємо кориневу систему пакетів розроблюваного java проекту, ми бачимо наступну структуру:

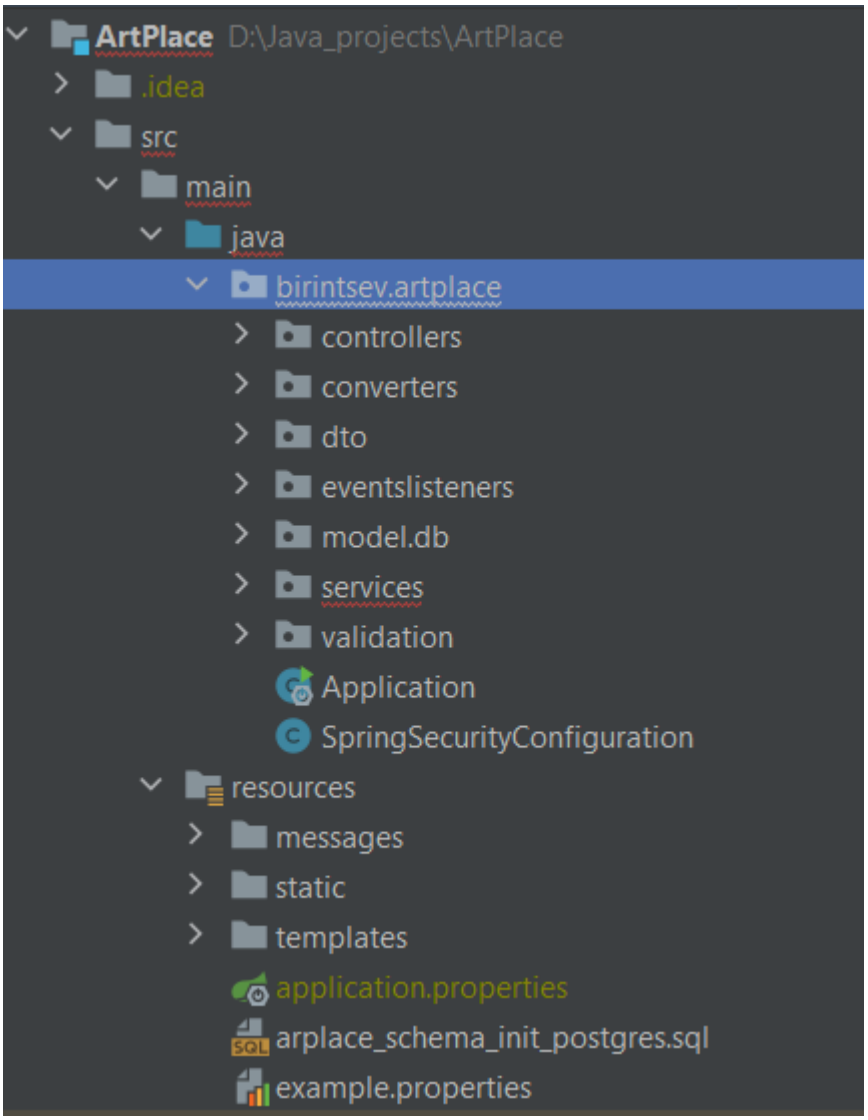

Рисунок 12 – Структура проекту

java – є пакетом вищого рівня. **birintsev.artplace** – є підпакетом пакету java, та кореневий пакет всього додатка, тут знаходяться всі загальні класи, які будуть формувати основний функціонал веб ресурсу.

controllers – пакет містить всі web- контролери для web-view з використанням thymeleaf та rest-контролери. Контролер керує потоками даних в додатку. Thymeleaf - сучасний серверний механізм Java-шаблонів для веб- та автономних засобів, можливість обробки HTML, XML, JavaScript, CSS і навіть простий текст. REST (Representational state transfer) - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який, як правило, використовується для побудови веб-служб. Термін REST був введений у 2000 році Роєм Філдінгом, одним з авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами. У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат. [4] У нашому підході до створення веб-сайту запити HTTP обробляє контролер. Ми сожемо легко оголосити контролер за допомогою анотації @Controller.

У пакеті controllers ми створюємо наступні класи:

- **AdminController.** Клас контролер який відповідає за обробку запросів /admin/ - дії адм іністратора, наприклад перегляд всіх користувачів.
- **FeedController.** Класс контролер який відповідає за обробку запросів /feed/ - дії пов’язані з фідом точніше сторінки перегляду публікацій.
- **RegistrationController.** Клас контролер, що обробляє запроси registration-related.
- **UserAccountController.** Клас контролер займається запросами на перегляд чи редагування акаунта користувача - змінення імені користувача, дати народження чи нікнейму .

converters - root package для всіх конвекторів (перетворювачі) для прикладу для гібернейтовскіх, спрінгових, таймліфових і всіх інших. Гібрейтовський конвектор це клас перетворювач, що слугує для перетворення об’єктів в інші об’єкти, щоб фреймворк Hibernate міг з ними працювати. Hibernate - це фреймворк Java, який спрощує розробку програми Java для взаємодії з базою даних. Це інструмент з відкритим вихідним кодом, легкий, ORM (Object Relational Mapping). Hibernate реалізує специфікації JPA (Java Persistence API) для збереження даних.[5] Спрінговий конвектор виконує туж саму функцію, що і гібрейтовський, але перетворює об’єкти, що використовуються фреймворком Spring. Spring Framework - це платформа Java, яка забезпечує всебічну підтримку інфраструктури для розробки програм Java. Spring обробляє інфраструктуру, щоб ви могли зосередитись на своєму додатку. [6] Таймліф або Thymeleaf - це сучасний механізм шаблонів Java на стороні сервера як для Інтернету, так і для автономних середовищ. Таймліф конвектер це клас перетворювач, що переробляє одні сутності в інші в нашому випадку одні об’єкти в інші, з якими може працювати такий механізм як Таймліф.

У пакеті converters містяться наступні класи:

- **CurrencyConverter.java.** Java класс виконує фунуцію серіалізації валют (представленої класом Currency) в рядок при збереженні її (валюти) в базу даних. Десеріалізація проходить в цьому ж класі. Серіалізація - це механізм перетворення стану об’єкта в байтовий потік. Десеріалізація - це зворотний процес, коли потік байтів використовується для відтворення фактичного об’єкта Java у пам’яті. Цей механізм використовується для збереження об'єкта.
- **PublicationPreparer.java.** Java класс виконує наступну функцію: перетворення запросу на створення публікації в саму публікацію. Тобто юзер передає в запросі всі необхідні дані, та запис про дані

публікації заноситься в базу даних. Javadoc над класом: This class converts, but not persists a request to a Publication. The class was created to unload birintsev.artplace.services.DefaultPublicationService from the conversion logic. In fact, it just prepares a Publication to be stored to the database. However, it might be re-used somewhere else.

Наступний пакет в дереві пакетів **dto** – пакет містить сутності, що транспортують дані зі сторони користувача на сторону сервера, тобто між фронтенд (frontend) та бекенд (backend) сторонами додатку. Frontend та Backend - два найпопулярніші терміни, що використовуються у веб-розробці. Ці терміни дуже важливі для веб-розробки, але досить сильно відрізняються один від одного. Кожна сторона повинна взаємодіяти та ефективно працювати з іншою як єдине ціле, щоб покращити функціональність веб-сайту.

DTO - суффікс, який розширюється як об'єкт передачі даних. Є сутність рівня моделей (пакет birintsev.artplace.model). Ми використовуємо в класах рівня моделей (сервіси, репозиторії). Наприклад - для збереження в БД (база даних), отримання з БД, логіка використання case-ov працює на цих (модельних) класах. Так ось, вони не добре використовують у класах рівня представлення (перегляд). Чому розглянуто в цьому розділі не буде. І якщо ми вирішимо таку ситуацію - просто створимо клони таких класів із підключенням DTO (наприклад, якщо ми захочемо користувачеві показати сторінку групи (пабліка), то будемо використовувати клас PublicDTO).

До пакету dto входять наступні класи:

- **FileDTO.java.** Java класс предоставляє представляння рівня view для сутності birintsev.artplace.model.db.File
- **PublicDTO.java**. Java класс предоставляє представляння рівня view для сутності birintsev.artplace.model.db.Public

- **PublicationDTO.java.** Java класс предоставляє представляння рівня view для сутності birintsev.artplace.model.db.Publication
- **PublishRequest.java.** Java класс потрібен для того щоб екземпляр цього класу, а точніше дані які будуть в цей об’єкт записані, в контролер поступатимуть дані від користувачів на публікацію в групах (пабліках)

Було розділено PublishRequest на інтерфейс і реалізацію для того, що б абстрагуватися від способу отримання аттачменті (прикріплених файлів). Припустимо, програміст захоче отримувати файл за посиланням від користувача і довантажувати його самостійно (замість поточної реалізації, читай нижче) з третього джерела. В цьому випадку я просто створю нову реалізацію інтерфейсу (скажімо, PublishRequestAttachmentsByLink) і всю логіку отримання таких файлів буде написана там.

На даному этапі проекта файли можна прикріпити тільки через спрінговий клас MultipartFile. Spring дозволяє нам увімкнути цю багаточастинну підтримку за допомогою об’єктів MultipartResolver, що підключаються. Структура забезпечує одну реалізацію MultipartResolver для використання з Commons FileUpload та іншу для використання з синтаксичним розбором запитів Servlet 3.0.

Для завантаження нашого файлу ми можемо створити просту форму, в якій ми використовуємо тег вводу HTML із типом = 'файл'. Незалежно від вибраної нами конфігурації обробки завантаження, нам потрібно встановити атрибут кодування форми на multipart / form-data. Це дозволяє браузеру знати, як кодувати форму. Ми додаємо необхідні рядки коду в клас.

Поки атачмент файлу працює тільки при створенні публікації. Надалі ще буде розроблений подгрузка аттачменті при редагуванні публікації. Початкова програма вже містить кілька класів, які стосуються зберігання та завантаження завантажених файлів на диск. Усі вони знаходяться в пакеті

com.example.uploadingfiles.storage. Ви будете використовувати їх у новому FileUploadController

- **PublishRequestImpl.java.** Java класс створений для реалізація PublishRequest
- **RegistrationRequest.java.** Java класс у іигляді об’єктів якого, тобто екземплярів даного класу, дані надходять в контролер. Під даними в даному контексті і з даним класом розуміються дані від користувача про регістрацію.
- **UserDTO.java.** Java класс предоставляє представляння рівня view для сутності birintsev.artplace.model.db.User

Eventlisteners - пакет всіх слухачів (listener) і подій (events). Подія в Java - це об’єкт, який створюється, коли щось змінюється в графічному інтерфейсі користувача. Якщо користувач натискає кнопку, клацає комбіноване поле або вводить символи в текстове поле тощо, тоді подія запускається, створюючи відповідний об’єкт події. Така поведінка є частиною механізму обробки подій Java та включена в бібліотеку графічного інтерфейсу Swing. Наприклад, скажімо, у нас є JButton. Якщо користувач натискає кнопку JButton, запускається подія натискання кнопки, подія буде створена, і вона буде надіслана відповідному прослуховувачу подій (у цьому випадку ActionListener). Відповідний слухач матиме впроваджений код, який визначає дію, яку потрібно вжити, коли подія відбувається. Зверніть увагу, що джерело події має бути у парі зі слухачем подій, інакше його активація не призведе до жодних дій. [7]

Як працюють події в нашому проекті. Обробка подій у Java складається з двох ключових елементів:

- Джерело події, яке є об’єктом, який створюється при настанні події. Java пропонує кілька типів цих джерел подій, про які йдеться у розділі Типи подій нижче.

- Прослуховувач подій, об'єкт, який "прослуховує" події та обробляє їх, коли вони відбуваються.

Цей пакет містить івенти і обробники для них в рамках використання Spring, а саме - його модуля з лістенерамі. В пакеті Eventlisteners створюємо наступні класи:

- **SendRegistrationConfirmationEvent.java.** Java класс який представляє собою подію, яка публікується в систему, коли додаток хоче отправити користувачу електроного листа, для підтвердження реєстрації.
- **SendRegistrationConfirmationListener.java.** Java класс який представляє собою обробником попереднього класу, котрому фреймворк Spring надає івенти (події), які є опублікованими, потім обробник відправляє електроного листа користувачу з посиланням про підтверження реєстрвції.

Наступний пакет **model.db** – це пакет відповідає за роботу з базою даних в якій ми зберігаємо дані користувачів. Пакет містить сутності бази даних та класів репозиторіїв для роботи з сутностями. Бази даних є невід’ємною частиною будь якого веб додатку, база даних (БД) - це організована колекція структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді в комп’ютерній системі. База даних зазвичай контролюється системою управління базами даних (СУБД). Дані та СУБД разом із пов'язаними з ними програмами називають системою баз даних, часто скороченою до просто бази даних.

В даному пакеті (model.db) знаходяться класи які напряму працюють та взаємодіють з БД (базою даних). Витягують із БД () сутності, тобто дані сформовані у певні об’єкти, зберігають ці дані в БД , тобто роблять запис даних у відповідні таблиці даних. Ці класи проводять підрахунок даних і виконують ряд інших елементарних операцій DML. Мова маніпулювання

даними (DML) - це мова комп'ютерного програмування, яка використовується для додавання (вставки), видалення та модифікації (оновлення) даних у базі даних. DML часто є підмовою ширшої мови баз даних, такої як SQL, де DML включає деякі оператори цієї мови. Вибір даних лише для читання іноді виділяють як частину окремої мови запитів даних (DQL), але він тісно пов’язаний і іноді також вважається компонентом DML; деякі оператори можуть виконувати як вибір (читання), так і запис. В даному пакеті (model.db) створюємо перелічені класи та репозиторії:

- **Embeddable.** Даний репозиторій містить допоміжні класи для роботи з базою даних. JPA забезпечує анотацію @Embeddable, щоб оголосити, що клас буде вбудований іншими сутностями. Анотація JPA @Embedded використовується для вбудування типу в іншу сутність.

Річ у тім, що наші поля називалися такими як contactFirstName у нашому початковому класі компанії, а тепер firstName у нашому класі ContactPerson. Отже, JPA захоче зіставити їх із контактом_перше_ім'я та ім'ям відповідно. Окрім того, що він є менш ніж ідеальним, насправді він розіб’є нас з нашою дубльованою телефонною колонкою. Отже, ми можемо використовувати @AttributeOverrides та @AttibuteOverride, щоб замінити властивості стовпців нашого вбудованого типу.

Репозиторій Embeddable містить клас **UserPublicationAssociation.java**, який відповідає за зіставлені ключі - як primary key, так і foreign key. Вони створюються для того, що їх можна було використовувати багатократно в різних класах-сущностях для БД. Наприклад, клас-сутність A має набір атрибутів {a, b, c, d}, з яких a і b - складений первинний ключ. Поруч живе класу B, у якого атрибути - {a, b, e, f}, з яких первинний ключ - теж a і b. Так ось ми можемо створити такий вбудований клас (скажем, ABPrimaryKey), у якому винесем атрибути a і b. У такому випадку цей клас ми будемо використовувати для класів-сущностей A і B в Java.

- **repo.** Репозиторій, що містить ряд класів: **PublicRepo.java** – репозиторій для Public, **PublicationRepo.java –** репозиторій для Publication**, RegistrationRepo.java –** репозиторій для RegistrationConfirmation**, TariffRepo.java –** репозиторій для Tariff**, UserRepo.java** – репозиторій для User**.** Дані класи відповідальні за коректне збереження даних.
- **Authority.java** Java клас, що відповідає за реалізацію інтерфейса org.springframework.security.core.GrantedAuthority в проекті ArtPlace**.** Коротко - це права (гранти, гранти, повноваження,, ролі - все це по суті одне і теж). Необхідно обмежити доступ людини до якрогось ресурсу / ендпоінту.

Приклад: скажімо, ми хочемо, що за всіма запрошеннями / admin / ** імена доступні лише для адміністраторів. Просто говорим спрингом (spring security), що пускає на ендпоінти за шаблоном / admin / ** лише юзери, у яких є role / grant / authorty (у розроблюваному проекті саме авторитет) із іменем ADMIN. Усі інші - не пускайте (відповідь, наприклад, з HTTP 403)

У Spring Security ми можемо сприймати кожен GrantedAuthority як індивідуальну привілей. Прикладом може бути READ_AUTHORITY, WRITE_PRIVILEGE або навіть CAN_EXECUTE_AS_ROOT. Важливо зрозуміти, що назва довільна. При безпосередньому використанні GrantedAuthority, наприклад, за допомогою виразу типу hasAuthority („READ_AUTHORITY“), ми обмежуємо доступ детально. Як ви, мабуть, можете зрозуміти, ми можемо посилатися на поняття авторитету, використовуючи також привілей. Подібним чином, у Spring Security ми можемо розглядати кожну роль як грубовизначений GrantedAuthority, який представляється як String і має префікс "ROLE". При безпосередньому використанні Ролі, наприклад, через вираз типу hasRole ("АДМІНІСТРАТОР"), ми обмежуємо доступ грубої форми. Варто зазначити, що префікс “ROLE” за замовчуванням можна налаштувати, але пояснити, як

це зробити, виходить за рамки цієї статті. Основна різниця між цими двома полягає в семантиці, яку ми додаємо до того, як ми використовуємо функцію. Щодо фреймворку, різниця мінімальна - і в основному вона має справу з ними однаково. Тепер, коли ми побачили, як фреймворк використовує концепцію ролі, давайте також швидко обговоримо альтернативу - а це використання ролей як контейнерів повноважень / привілеїв. Це підхід вищого рівня до ролей - що робить їх більш бізнес-концепцією, а не орієнтованою на реалізацію. Структура Spring Security не дає жодних вказівок щодо того, як нам слід використовувати цю концепцію, тому вибір повністю залежить від реалізації.

- **File.java** Java клас, екземпляри якого є сутностями файлів, які додає користувач. Ці файли працюють як прикріплені файли до публікацій, тобто атачменти (attachment).
- **Public.java** Java клас, екземпляри якого є сутностями пабліка (групи), які створює користувач паблік овнер. Працює на рівні моделі.
- **PublicSubscription.java** Java клас, екземпляри якого є сутностями пілписки користувача на групу (паблік). Забезпечує зв’язок many-to-many, працює на рівні моделі.
- **Publication.java** Java клас, екземпляри якого є сутностями публікації користувача паблік овнера в групу (паблік), котра йому належить. Працює на рівні моделі.
- **RegistrationConfirmation.java** Java клас, екземпляри якого є сутностями, що є підтвердженням, того що користувач зареєстрований. Робиться відповідна відмітка в таблиці користувачів. Сутність, що відображена на рівні моделі.

- **SubscriptionTariff.java** Java клас, екземпляри якого є сутностями, що відповідають за тарифи (безкоштовні та платні) Сутність, що відображена на рівні моделі.
- **User.java** Java клас, екземпляри якого є сутностями, що відповідають за користувача. Сутність, що відображена на рівні моделі.
- **UserPermanentPublication.java** Java клас, екземпляри якого є сутностями, що відповідають за те що користувачі мали постійний доступ до контенту за який вони вже заплатили в групі (пабліку). Сутність, що відображена на рівні моделі.

services – пакет в якому знаходяться класи, що визначають методи, що пов’язані з наданням бізнес логіки. Бізнес-логіка - це спеціальні правила або алгоритми, які обробляють обмін інформацією між базою даних та користувальницьким інтерфейсом. Бізнес-логіка - це, по суті, частина комп’ютерної програми, яка містить інформацію (у формі ділових правил), яка визначає або обмежує функціонування бізнесу. Такі бізнес-правила є операційною політикою, яка зазвичай виражається в істинних або помилкових двійкових даних. Бізнес-логіку можна побачити в робочих процесах, які вони підтримують, наприклад, у послідовностях або етапах, що детально визначають належний потік інформації або даних, а отже, і прийняття рішень. Бізнес-логіка також відома як "логіка домену". Сервіс - це контейнер для бізнес-логіки. код для 90% випадків використання прописаний у них. Ім'я для інтерфейсу - це $ {ім'я_Набагато більше інформації про функції окремих блоків можно знайти в javadoc.

Перелік створених класів та репозиторіїв в пакеті services:

- **exceptions**. Репозиторій, що містить класи виключень, а саме: **RegistrationException.java –** згідно джава доку: This exception is raised when user registration process went wrong**,** тобто ексепшинси (виключення) викидаються тоді коли не проходить реєстрація,

тобто фейлиться. **UnauthorizedOperationException.java** згідно джава доку: Is thrown when a User lacks privileges on performing an operation (e.g. creating a Publication in not owned Public), тобто виключення викидається, коли користувачеві не вистачає привілегій для виконання певних функцій і дій, наприклад – створення публікації в пабліку, який він не має як власник. **UserExistException.java.** згідно джава доку цей клас: Is commonly thrown when it is necessary to markup, that a user being registered already exist. Also, may be used in registration-related cases (that are semantically related), тобто цей клас викидає виключення коли треба проінформувати систему, що користувач уже існує в ній.

- **DefaultPublicService.java** Java клас, щоб реалізувати інтерфейс PublicService Цей PublicService (послуга, функціонал) вфдповідає за роботу з пабліками (групами), створенням, видалення і редагування полів (назви чи інших атрибутів) пабліку. DefaultPublicService – це реалізація інтервіфейсу PublicService , яка на даний момент задовольняє вимогам проекту і в дальнійшому може наслідуватись іншими класами за необхідністю.
- **DefaultPublicationService.java** Java клас, щоб реалізувати інтерфейс реалізувати PublicationService. Цей PublicationService (послуга, функціонал) вфдповідає за роботу з публікаціями, додавання, видалення і редагування. DefaultPublicationService – це реалізація інтервіфейсу PublicationService, яка на даний момент задовольняє вимогам проекту і в дальнійшому може наслідуватись іншими класами за необхідністю.
- **DefaultUserService.java** Java клас, щоб реалізувати інтерфейс UserService. Цей UserService (послуга, функціонал) вфдповідає за

роботу з користувачами, створенням, авторизацію і логінацію, підтвердження і редагування даних. DefaultUserService – це реалізація інтерфейсу UserService, яка на даний момент задовольняє вимогам проекту і в дальнійшому може наслідуватись іншими класами за необхідністю.

- **FileService.java** Java клас, що описує сервіс, що містить методи, які забезпечують бізнес логіку для роботи з файлами, які користувачі завантажують у інформаційну систему, наприклад фото, відео, зображення та аудіофайли та інще.
- **NetworkToFileSystemFileService.java** Java клас, що реалізує FileService. FileService необхідний для того, щоб преобразовувати дані з інтернет мережі в файли з якими може працювати інформаційна система.
- **PublicService.java** Java клас, що описує сервіс, що містить методи які забезпечують бізнес логіку для роботи з групами (пабліками), а саме такі функціїяк створити паблік, редагувати паблік, знайти паблік, перевірити паблік та інше.
- **PublicationService.java** Java клас, що описує сервіс, що містить методи відплвідальні за бізнес логіку для роботи з публікаціями, та мають наступні функції: опублікувати публікацію, видалити публікацію, знайти публікацію та інше.
- **UserService.java** Java клас, що описує сервіс, що містить наступні методи, що забезпечують бізнес логіку для роботи з користувачем.

Пакет **validation** є останнім пакетом, що надає зміст функціоналу додатку. Цей пакет містить класи відповідальні за валідацію даних. Пакет розширює можливості валідації в джаві та містить кастомні анотації валідацій (javaee/hibernate) і іх же валідатори і інші побічні функції цієї невід'ємної частини перевірки даних на коректність і валідність. Перевірка - це

автоматизована перевірка, що виконується для гарантування того, що введення даних є раціональним та прийнятним. Він не перевіряє правильність даних. Як приклад, припустимо, кав'ярня наймає барістів у віці від 18 до 25 років. Система може бути запрограмована на прийняття лише цифр від 18 до 25 для вікового поля. Це називається перевіркою дальності. Однак це не гарантує правильності введеної цифри. Вік заявника може бути 20, але якщо введено 18, він все одно буде дійсним, просто неправильним. Перевірка - це спосіб спроби зменшити кількість помилок при введенні даних. Перевірка виконується комп'ютером під час введення даних. Це спосіб перевірки вхідних даних із заданим набором правил перевірки. Мета перевірки полягає в тому, щоб переконатися, що будь-який набір даних є логічним, раціональним, повним та у допустимих межах.[8] У пакетах перевірки лежать сутності, які відповідають за проведення перевірки в інших даних. Під вхідними данними ми розумію дані від користувача. У майбутньому сюди будуть додані класи перевірки даних, що йдуть [back-end] -> [database] , тобто від бек енда-сервісної сторони в базу даних, для подальшого збереження.

В пакеті validation містяться наступні класи:

- **UserNotExists.java** Java клас – анотація, котрий вішається над класом RegistrationRequest, Ця анотація підрзуміває, що користувач, який сюди відправляє запрос не існує.
- **UserNotExistsValidator.java** Java клас валідатор анотації, що описана в попередньому класі, виконує перевірку існування користувача за допомогою його електроної поштової скриньки (електроної пошти)

5.2 ІМПОРТОВАНІ ПАКЕТИ

У проекті також використовується широкий спект імпортованих пакетів, тобто бібліотеки, модулі та фреймворки для полегшення етапу розробки та зменшення часу розроблення.

Для автоматизації розробки використовується Maven. Apache Maven є наріжним каменем розробки Java та найпоширенішим інструментом управління збіркою для Java. Впорядкована модель конфігурації на основі XML від Maven дозволяє розробникам швидко описувати або розуміти контури будь-якого проекту на основі Java, що робить запуск та спільний доступ до нових проектів швидким. Maven також підтримує тестову розробку, довгострокове обслуговування проектів, а його декларативна конфігурація та широкий спектр плагінів роблять його популярним варіантом для CI / CD. Як і багато чудових інструментів, Maven бере те, що колись було занадто складно (пекло конфігурації), і спрощує його до засвоюваних частин. Maven складається з трьох компонентів:[2]

Maven POM: файл, що описує проект Maven та його залежності. Кожен проект Java, який використовує Maven, має файл POM (об’єктна модель проекту) у своєму кореневому каталозі. Pom.xml описує залежності проекту та розповідає, як його будувати. (Залежності - програмне забезпечення сторонніх розробників, яке вимагає проект. Деякі загальні приклади - JUnit та JDBC. Перегляньте перелік усіх доступних інструментів та популярних залежностей у Центральному сховищі Maven.)

Каталог: стандартизований формат опису проекту Maven у POM. Каталог Maven реалізує те, що називається домовленістю щодо конфігурації, елегантне рішення пекла конфігурації. Замість того, щоб вимагати від розробників визначати компонування та вручну налаштовувати компоненти для кожного нового проекту (як це було у випадку з makefile та Ant), Maven запроваджує загальну структуру проекту та пропонує стандартний формат

файлу для опису того, як це працює. Ви просто підключаєте свої вимоги, і Maven викликає залежності та налаштовує проект для вас.

Сховища: де зберігається та виявляється стороннє програмне забезпечення. нарешті, Maven використовує централізовані сховища як для виявлення, так і для публікації пакетів проектів як залежностей. Коли ви посилаєтесь на залежність у своєму проекті, Maven виявить її в централізованому сховищі, завантажить до локального сховища та встановить у ваш проект. Здебільшого все це є невидимим для вас як розробника.[2]

В файлі залежностей Maven можно переглянути додаткові пакети, сторонні бібліотеки та інщі модулі, що використовує веб додаток. Перелік maven залежностей:

- **spring-boot-starter-data-jpa.**
- **spring-boot-starter-thymeleaf**
- **spring-boot-starter-validation**
- **spring-boot-starter-web**
- **spring-boot-devtools**
- **postgresql**
- **lombok**
- **spring-boot-starter-test**
- **spring-boot-starter-security**
- **modelmapper**
- **spring-boot-starter-mail**
- **spring-boot-maven-plugin**

spring-boot-starter-data-jpa – забезпечує роботу з базрою даних. Spring Boot забезпечує залежність spring-boot-starter-data-jpa для ефективного підключення програми Spring до реляційної бази даних. Spring-boot-starter-data-jpa внутрішньо використовує залежність spring-boot-jpa (починаючи з Spring Boot версії 1.5.3).

Бази даних мають таблиці / відношення. Раніше підходи (JDBC) передбачали написання SQL-запитів. У JPA ми будемо зберігати дані від об’єктів у таблиці та навпаки. Однак JPA еволюціонувала в результаті іншого процесу мислення.

До JPA, ORM був терміном, який частіше використовувався для позначення цих систем. Саме тому Hibernate називають структурою ORM.

JPA дозволяє нам зіставити класи додатків із таблицями в базі даних.

- Entity Manager: Щойно ми визначимо відображення, воно обробляє всі взаємодії з базою даних.
- JPQL (Java Persistence Query Language): Він забезпечує спосіб написання запитів для виконання пошукових запитів щодо сутностей. Це відрізняється від запитів SQL. Запити JPQL вже розуміють відображення, яке визначено між сутностями. За потреби ми можемо додати додаткові умови.
- Criteria API: Він визначає API на основі Java для виконання пошукових запитів у базі даних.[3]

spring-boot-starter-thymeleaf - спрінговий модуль інтеграції з thymeleaf. Spring Boot – Thymeleaf - це бібліотека на основі Java, яка використовується для створення веб-програми. Він забезпечує хорошу підтримку обслуговування XHTML / HTML5 у веб-додатках. У цьому розділі ви детально дізнаєтеся про Чебрець.

Шаблони Thymeleaf. Thymeleaf перетворює ваші файли у добре сформовані файли XML. Він містить 6 типів шаблонів, як зазначено нижче -

- XML
- Дійсний XML
- XHTML
- Дійсний XHTML
- HTML5

- Спадковий HTML5

Усі шаблони, крім Legacy HTML5, посилаються на добре сформовані дійсні файли XML. Спадковий HTML5 дозволяє нам відображати теги HTML5 на веб-сторінці, включаючи незакриті теги.[9]

spring-boot-starter-validation - спрінговий модуль дозволяє валідувати сутності, задіяний у таких пакетах як: dto та model.db. Що стосується перевірки вводу користувача, Spring Boot надає потужну підтримку цій загальній, але критичній задачі прямо з коробки. Хоча Spring Boot підтримує безперебійну інтеграцію з користувацькими валідаторами, фактичним стандартом для перевірки є Hibernate Validator, довідкова реалізація Bean Validation.

spring-boot-starter-web Є дві важливі особливості spring-boot-starter-web:

- Він сумісний для веб-розробки
- Автоконфігурація

Кожна програма Spring Boot включає вбудований сервер. Вбудований сервер вбудований як частина розгортається програми. Перевага вбудованого сервера полягає в тому, що ми не вимагаємо попередньо встановленого сервера в середовищі. У Spring Boot вбудованим сервером за замовчуванням є Tomcat. Spring Boot також підтримує ще два вбудовані сервери: [5]

- Jetty Server
- Undertow Server

spring-boot-devtools Spring Boot включає додатковий набір інструментів, які можуть зробити досвід розробки додатків трохи приємнішим. Модуль spring-boot-devtools може бути включений в будь-який проект, щоб забезпечити додаткові функції часу розробки. Щоб включити підтримку devtools, додайте залежність модуля до вашої збірки використовуючи мавен.

postgresql - це потужна об'єктно-реляційна система баз даних з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші робочі навантаження даних. Походження PostgreSQL бере свій початок у 1986 році в рамках проекту POSTGRES в Університеті Каліфорнії в Берклі і має більше 30 років активного розвитку на базовій платформі.

PostgreSQL заслужив сильну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти з відкритим кодом, що стоїть за програмним забезпеченням, для постійного забезпечення продуктивних та інноваційних рішень. PostgreSQL працює на всіх основних операційних системах, сумісний з ACID з 2001 року та має потужні доповнення, такі як популярний розширювач геопросторових баз даних PostGIS. Не дивно, що PostgreSQL став реляційною базою даних з відкритим кодом для багатьох людей та організацій. [10] Нижче наведено невичерпний перелік різних функцій, знайдених у PostgreSQL, які ми використовували:

Типи даних:

- Primitives: Integer, Numeric, String, Boolean
- Structured: Date/Time, Array, Range, UUID
- Document: JSON/JSONB, XML, Key-value (Hstore)
- Geometry: Point, Line, Circle, Polygon
- Customizations: Composite, Custom Types

Цілісність даних:

- UNIQUE, NOT NULL
- Primary Keys
- Foreign Keys
- Exclusion Constraints
- Explicit Locks, Advisory Locks

Надійність, відновлення після катастрофи:

- Write-ahead Logging (WAL)
- Replication: Asynchronous, Synchronous, Logical
- Point-in-time-recovery (PITR), active standbys
- Tablespaces

Безпека:

- Аутентифікація: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, сертифікат тощо
- Надійна система контролю доступу
- Безпека на рівні стовпців і рядків
- Багатофакторна автентифікація за допомогою сертифікатів та додаткового методу

lombok Project Lombok - це бібліотека Java, яка автоматично підключається до вашого редактора та створює інструменти, приваблюючи вашу Java. Щоб ніколи більше не писати інший метод getter або equals, за допомогою однієї анотації ваш клас має повнофункціональний конструктор, автоматизує ваші змінні реєстрації та багато іншого.

spring-boot-starter-test Тест spring-boot-starter-test є основною залежністю для тесту. Він містить більшість елементів, необхідних для наших тестів. Існує декілька різних типів тестів, які ми можемо написати для тестування та автоматизації роботи програми. Перш ніж починати будь-яке тестування, нам потрібно інтегрувати рамки тестування. З Spring Boot нам потрібно додати до нашого проекту стартер, для тестування нам потрібно лише додати залежність spring-boot-starter-test за допомогою Мавен [5]

spring-boot-starter-security Spring MVC Security створив простий приклад Spring MVC Security за допомогою базової автентифікації. Але потрібно налаштувати багато конфігурації. У цьому розділі ми бачимо, наскільки просто налаштувати безпеку за допомогою Spring Boot. Якщо до

шляху класу додається залежність Spring Boot Security, програма Spring Boot автоматично вимагає базової автентифікації для всіх кінцевих точок HTTP. Кінцева точка “/” та “/ home” не вимагає ніякої автентифікації. Усі інші Кінцеві точки вимагають автентифікації.

modelmapper Додатки часто складаються з подібних, але різних об'єктних моделей, де дані в двох моделях можуть бути подібними, але структура та проблеми моделей різні. Зіставлення об’єктів полегшує перетворення однієї моделі в іншу, дозволяючи окремим моделям залишатися відокремленими. Мета ModelMapper - спростити відображення об'єктів, автоматично визначаючи, як одна об'єктна модель відображається в іншу, на основі домовленостей, так само, як це робила б людина, - забезпечуючи при цьому простий, безпечний для рефакторингу API для обробки конкретних випадків використання.

Додаємо modelmapper за допомогою залежностей з використанням Мавену. Коли викликається метод map, аналізуються типи джерела та призначення, щоб визначити, які властивості неявно збігаються відповідно до стратегії відповідності та іншої конфігурації. Далі дані відображаються відповідно до цих збігів. Навіть коли джерела та об'єкти призначення та їх властивості відрізняються, як у прикладі вище, ModelMapper зробить все можливе, щоб визначити розумні збіги між властивостями відповідно до налаштованої стратегії відповідності.

spring-boot-starter-mail Інтерфейси та класи для підтримки пошти Java у структурі Spring організовані таким чином:

- Інтерфейс MailSender: інтерфейс верхнього рівня, який забезпечує базову функціональність для надсилання простих електронних листів
- Інтерфейс JavaMailSender: підінтерфейс вищезазначеного MailSender. Він підтримує повідомлення MIME і здебільшого

використовується разом із класом MimeMessageHelper для створення MimeMessage. З цим інтерфейсом рекомендується використовувати механізм MimeMessagePreparator.

- Клас JavaMailSenderImpl забезпечує реалізацію інтерфейсу JavaMailSender. Він підтримує MimeMessage та SimpleMailMessage.
- Клас SimpleMailMessage: використовується для створення простого поштового повідомлення, що включає поля від, до, копію, тему та текстове поле
- Інтерфейс MimeMessagePreparator забезпечує інтерфейс зворотного виклику для підготовки повідомлень MIME.
- Клас MimeMessageHelper: допоміжний клас для створення повідомлень MIME. Він пропонує підтримку зображень, типових вкладень до пошти та текстового вмісту в HTML-макеті.[11]

У нашому додатку можно знайти приклади реалізації та використання цих інтерфейсів.

spring-boot-maven-plugin Плагін Spring Boot Maven забезпечує підтримку Spring Boot в Apache Maven. Це дозволяє упаковувати виконувані архіви jar або war, запускати програми Spring Boot, генерувати інформацію про збірку та запускати програму Spring Boot перед запуском тестів інтеграції.

Як і попередні плагіни підключаємо його за допомогою Мавен залежностей. Користувачі Maven можуть успадковувати проект spring-boot-starter-parent для отримання розумних значень за замовчуванням. Батьківський проект надає такі функції:

- Java 1.8 як рівень компілятора за замовчуванням.
- Вихідне кодування UTF-8.
- Розділ управління залежностями, успадкований від POM Spring-boot-зависимостей, який керує версіями загальних залежностей.

Це управління залежностями дозволяє вам опустити теги <version> для цих залежностей, коли вони використовуються у вашому власному POM.

- Виконання цілі переупаковки з ідентифікатором виконання переупаковки.
- Помітна фільтрація ресурсів.
- Помітна конфігурація плагіна (ідентифікатор коміту Git і відтінок).
- Помітна фільтрація ресурсів для application.properties та application.yml, включаючи файли для конкретних профілів (наприклад, application-dev.properties та application-dev.yml)

Попереднє налаштування зразка не дозволяє вам перевизначити окремі залежності за допомогою властивостей, як пояснено вище. Щоб досягти того самого результату, вам потрібно додати записи в розділі dependencyManagement вашого проекту перед входом spring-boot-dependencies.

5.3 ПРОТОТИП

В даному розділі ми розглянемо графічний прототип веб додатку підтримки авторів творчого контенту. Ми опишемо графічний дизайн та його можливості. Розглянемо UX (user experience design) та UI (user interface design) додатку. Продемонструємо основні функції сайту.

UX та UI: два терміни, які часто використовуються як взаємозамінні, але насправді є дуже різні речі. UX-дизайн означає термін "дизайн інтерфейсу користувача", тоді як UI означає "дизайн інтерфейсу користувача". Обидва елементи мають вирішальне значення для товару і тісно співпрацюють. Але, незважаючи на їхні професійні стосунки, самі ролі досить різні, маючи на увазі дуже різні аспекти процесу розробки товару та дисципліни дизайну. Перш ніж

ми розглянемо ключові відмінності між UX та UI, спершу визначимось, що кожен термін означає окремо. [12]

По суті, UX застосовується до будь-чого, що може бути досвідченим - будь то веб-сайт, кавоварка або відвідування супермаркету. Частина “взаємодія з користувачем” стосується взаємодії між користувачем та продуктом чи послугою. Тоді дизайн досвіду користувача враховує всі різні елементи, що формують цей досвід. Дизайнер UX думає про те, як досвід відчуває користувача, і про те, як легко користувачеві виконувати бажані завдання. Наприклад: Наскільки легким є процес оформлення замовлення під час покупок в Інтернеті? Наскільки вам легко вхопити овочечистку? Чи спрощує ваш додаток для онлайн-банкінгу управління своїми грошима? Кінцева мета дизайну UX - створити легкі, ефективні, відповідні та всебічні приємні враження для користувача. Ми відповімо на запитання "Що робить дизайнер UX?" у розділі четвертому. Наразі ось що вам потрібно знати про UX-дизайн у двох словах: [12]

- Дизайн інтерфейсу користувача - це процес розробки та поліпшення якості взаємодії між користувачем та усіма аспектами компанії.
- Теоретично, дизайн користувацького досвіду є нецифровою (когнітивно-науковою) практикою, але використовується та визначається переважно цифровими галузями.
- UX-дизайн НЕ стосується візуалів; він фокусується на загальному відчутті переживання.

Незважаючи на те, що ця сфера є давнішою та практичнішою, питання "Що таке дизайн інтерфейсу користувача?" важко відповісти через широкий спектр помилкових тлумачень. Незважаючи на те, що користувальницький досвід - це сукупність завдань, спрямованих на оптимізацію продукту для ефективного та приємного використання, дизайн інтерфейсу користувача є його доповненням; зовнішній вигляд, презентація та інтерактивність товару.

Але, як і UX, його легко і часто плутають галузі, в яких працюють дизайнери інтерфейсу - до тієї міри, що різні посади часто позначають професію як абсолютно різні речі.

Як і дизайн користувацького досвіду, дизайн інтерфейсу користувача - це багатогранна і складна роль. Він відповідає за перетворення розробки, дослідження, вмісту та верстки продукту в привабливий, спрямовуючий та чуйний досвід для користувачів.

Ми розглянемо процес проектування інтерфейсу користувача та конкретні завдання, на які може розраховувати дизайнер інтерфейсу, у четвертому розділі. Перш ніж ми розглянемо основні відмінності між UX та UI, давайте швидко підведемо підсумки дизайну користувацького інтерфейсу (UI):

- Дизайн інтерфейсу користувача - це суто цифрова практика. Він враховує всі візуальні, інтерактивні елементи інтерфейсу продукту, включаючи кнопки, піктограми, інтервали, типографіку, кольорові схеми та адаптивний дизайн.
- Мета дизайну інтерфейсу користувача - візуально вести користувача через інтерфейс продукту. Вся справа в створенні інтуїтивного досвіду, який не вимагає від користувача занадто багато думати!
- Дизайн інтерфейсу передає сильні сторони та візуальні цінності бренду в інтерфейс продукту, забезпечуючи його узгодженість, узгодженість та естетичність.

Дизайнер UX розглядає всю подорож користувача для вирішення певної проблеми; які кроки вони роблять? Які завдання їм потрібно виконати? Наскільки прямий досвід? Значна частина їхньої роботи зосереджена на тому, щоб з’ясувати, з якими проблемами та проблемними точками стикаються користувачі та як певний продукт може їх вирішити. Вони проведуть обширне

дослідження користувачів, щоб з’ясувати, хто цільові користувачі та які їхні потреби щодо певного товару. Потім вони складуть карту подорожі користувача через товар, враховуючи такі речі, як інформаційна архітектура - тобто. Як впорядковано та марковано вміст у продукті та які функції можуть знадобитися користувачеві. Врешті-решт вони створять каркасні каркаси, в яких викладаються голі кістки продукту. Як бачите, UX та UI міцно поєднуються, і хоча є мільйони прикладів чудових продуктів з одним, а не з іншим, уявіть, наскільки успішнішими вони могли б бути, коли були сильними в обох сферах. Щодо відмінностей UX та UI можемо дізнатися з даного малюнка, де основна різниця зображена з використанням грфічних форм:

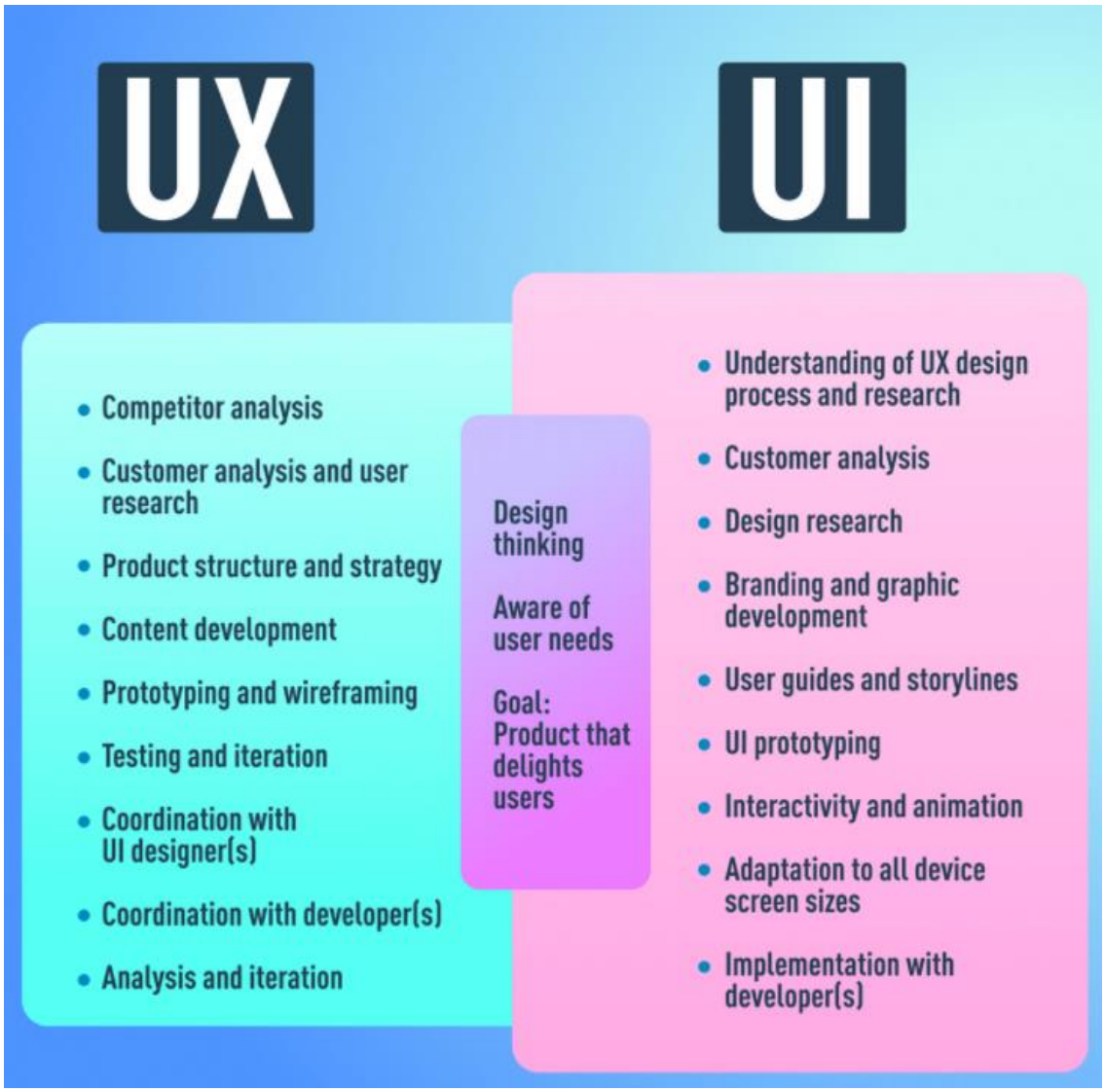

Рисунок 13 – Відміності UX та UI

Приступимо до розробки і опису прототипу веб-додатку. А саме сторони користувача – фронтенду. Для допомоги у розробкі була використані сучасні інструменти, які полегшують розробку дизайну програми схожі на фотошоп та інші графічні редактори, а саме Figmaю Figma - це програма для інтерфейсу, яка працює в браузері, але насправді це набагато більше, ніж це. Я пішов би так далеко, сказавши, що це, мабуть, найкращий додаток для спільних дизайнерських проектів на основі команд.

На рисунку 3 продемонстрована сторінка входу (Login page). Де користувач повинен вести дані у форму входу, а саме: логін користувача (user name) та пароль (password) у відповідні поля форми. Після вводу даних користувач натискає на кнопку “Login”, введені дані відправляються на серверну сторону додатку, на бекенд, де валідуються та обробляються, якщо користувач існує в системі, то вхід виконується, якщо користувача немає в системі, тобто дані які він ввів не знаходяться в жодній таблиці користувачів, вхід на сайт підтримки творчого контенту не виконується.

Також під формою входу (логін формою) знаходиться повідомлення, точніше питання до користувача, чи має він акаунт (“Not register?”), яке служить для того, щоб не зареєстрований користувач міг створити свій акаунт у інформаційній системі підтримки авторів, тобто нашому сайті. Коли користувач натискає “Create an account”, то він (не зареєтрований користувач) аотрапляє на форму реєстацій нового акаунта, яка приведена на рисунку 4 і буде розглянута трохт далі.

На формі входу знаходиться також логотип сайту з його назвою “ArtPlace”, що перекладається як творче місце, або арт плейс, або мистецьке розташування, що підкреслює націленість на творчу аудиторію та контент криейторів. Логотипом нашої інформаційної системи підтримки творчих людей є силует профіля людини (homo sapiens), з силуетом мозку, а саме верхньої кори головного мозку з помітною великою кількістю пагорбів та впадин, що символізують про високий рівень розвитку даного мозку, що

підкреслює, що користувачі сайту є інтелігентні та розумово розвинуті представники суспільства. Хоч спираючись на несвідомий люський досвід ми можемо відмітити, що профіль належить чоловікові (не жінкі), це ми можемо зрозуміти з виступаючої нижньої челюсті, це не означає, що даний веб додаток спрямований на чоловічу аудиторію. Вибір чоловічого профілю був зроблений, як результат натхнення яке найшло на дизайнера цього додатку (також програміста), джерелом натхнення став замовник.

Головну увагу в логотипі бере на себе пензлик і палітра с фарбами в середині селуета мозку. Хоч і інформаційна веб система розрахована на митців будь якого цифрового виду мистецтва чи музики, чи поезії, чи дизайну. Але так як при згадку про мистецтво і творців цього мистецтва приходить на думку його широкий розділ малювання (образотворче мистецтво), то саме ці атрибути були вибрані, хоча і художниками в наші дні вони вже використовуються рідко. Ідея вибору палітри також пересікається з оформленням правої сторони форми входу – акварельних розводів – до них пізніше.

Вибір шрифту аргументується його відаленою схожістю на написання від руки, що є референсом на поезію і прозу. Заокруглені форми створюють відчуття привітності, ніж різкі квадратні чи закруглені квадрати.

Основним кольором оформлення сайту був обраний синій, та його похідні. Цей вибір був зроблений не просто так адже синій кольор є фаворитом серед кольорів серед людей, особливо серед чоловіків. Він є досить стримани, консервативним, тому традиційним. Синій колір викликає почуття спокою та безтурботності. Його часто описують як мирний, спокійний, безпечний і впорядкований. Синій часто розглядається як знак стабільності та надійності. Підприємства, які хочуть спроектувати образ безпеки, часто використовують синій колір у своїх рекламних та маркетингових зусиллях.

Хоча синій колір може викликати відчуття смутку і відаленості чи тоски. В даному випадку, при створенні інформаційної системи підтримки творчого контенту, це є плюс, перевага синього, адже багато артистів, митців, творчих людей є меланхолійні і апатичні, і часто вибудовують свої твори на цих відчуттях, тому їх буде надихати цей колір. Також синій колір не вважається апетитним, але цей аспект ніяк не впливає на нашу бізнес логіку. Із перелічених вище причин синіій і палітра близбких до нього кольорів були обрані як основні відтінки веб додатку.

Саме тому акварельні розводи, точніше зображення з ними обрані саме в цьому кольорі. Акварельні розводи – кольорові візерунки, що створюються вкликою кількістю води і водною акварельною фарбою з використанням круглого пензля передають легкість та романтизм сторінки входу до акаунту. Їх зображення було взято з безкоштовного сайту стічних фотографій і зображень, тому його використання є законим.

Також на сторінкі входу є іконка, де буде знаходитись зображення користувача, тобто фото його профала. Ці дані при першому вході, а також без реєстрацій не відображаються, також якщо користувач заборонив використання кукі файлів. Файловий файл cookie - це невеликий фрагмент тексту, переданий у браузер із веб-сайту, який ви відвідуєте. З його допомогою веб-сайт запам'ятовує інформацію про ваші відвідування та з кожним разом, що знаходиться зручніше та полезніше для вас. Якщо користувач дозволив кукі, то з наступним входом зображення його профайлу і інші дані будуть автоматично підставленні.

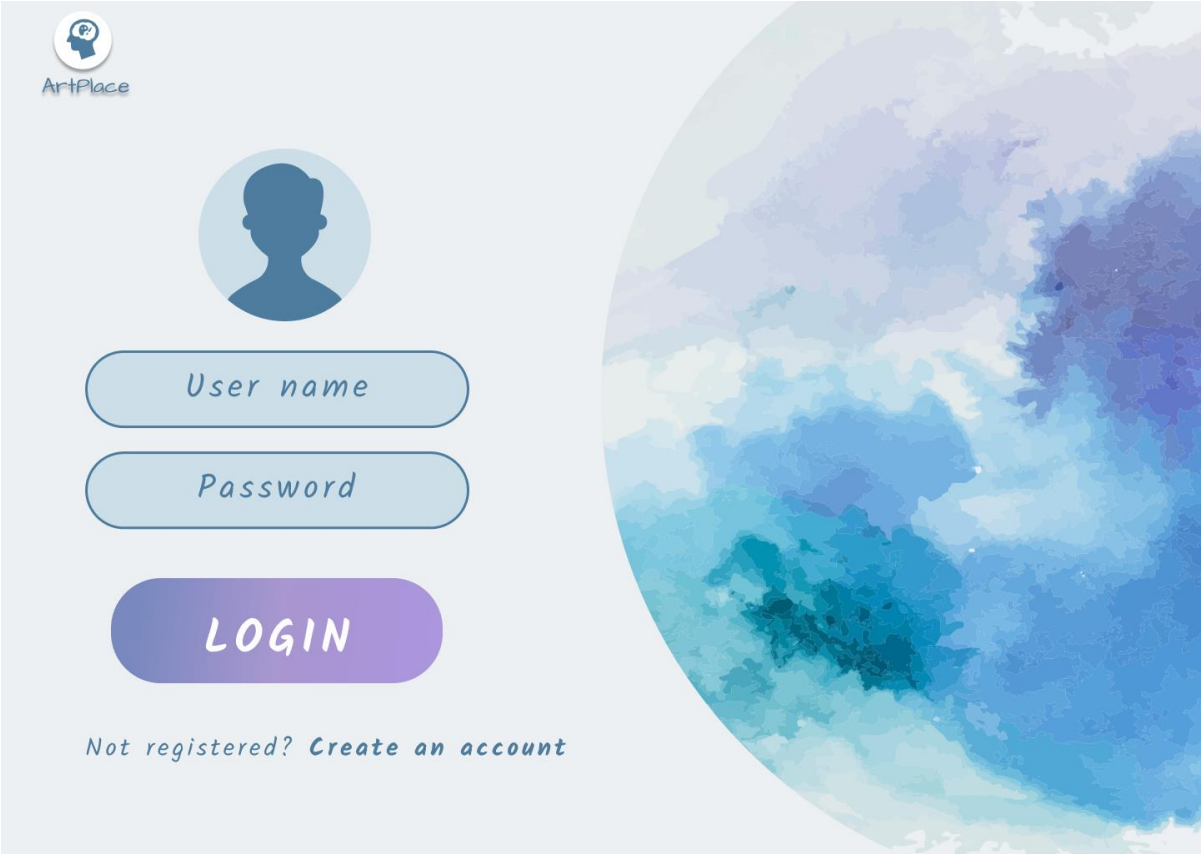

Рисунок 14 – Login page

Наступною детально розглянемо сторінку реєстрацію, точніше реєстраційну форму. На реєстраційній формі присутні наступні поля:

- Name - ім’я користувача чи його нікнейм, текстове поле, обов’язкове поле
- Email – електрона пошта кристувача, текстове поле, згідно з валідацією обов’язковим є наявність одного равлика – @ , поле є обов’язковим
- Birthday – дата народження користувача, не є обов’язковим полем, складається з групи інпутів – трьох випадаючих списків: дня, місяця та року. За замовчуванням стоїть 1 january 2000,
- Password – пароль до акаунту користувача, є обов’язквим текстовим полем, поле можно скривати
- Confirm password – повтор паролю до акаунту користувача, є обов’язквим текстовим полем, поле можно скривати

- Gender – стать користувача, поле свічер (перемикач, radio button), не є обов’язковим, за замовчуванням не стоїть. Щодо того чому статі (гендера) всього дві, розробники і власники даного веб додатку не відкидають гендерну теорію і наявність людей інтерсексів, дана фукція є опціональною і не обов’язковою і зроблена для економії ресурсів серверної частини, а саме для економії пам’яті у сховищі даних. Поле Gender помічаються поряд картинками схематичної жінки і схематичного чоловіка в офісному вбрані.

Також на формі присутня кнопка “Create an account” – створити акаунт, після натискання на неї форма перевіряється з фронт енд сторони, проходить перевірка наявності всіх mandatory (мандоторі – обов’язкових) даних, їх валідація і далі форма запросом POST відправляється на бекенд сторону, де також дані валідуються, перевіряється їх достатність. Далі за допомогою класів в пакеті model.db, зокрема класу **UserRepo.java** дані додаються в базу даних і користувач офіційно стає зареєстрованим, але не є ще підтвердженим, далі за допомогою класу **RegistrationRepo.java,** який відповідає за функціонал з’язаний з підтвердженням електроної адреси. Тобто користувач має зайти на свою поштову скриньку і підтвердити через посилання свій імейл і тоді користувач буде зареєстронаним і підтвердженим і матиме повний доступ до функціоналу сайту. Графічне зображення форми та приклад введених даних є на малюнку нижче.

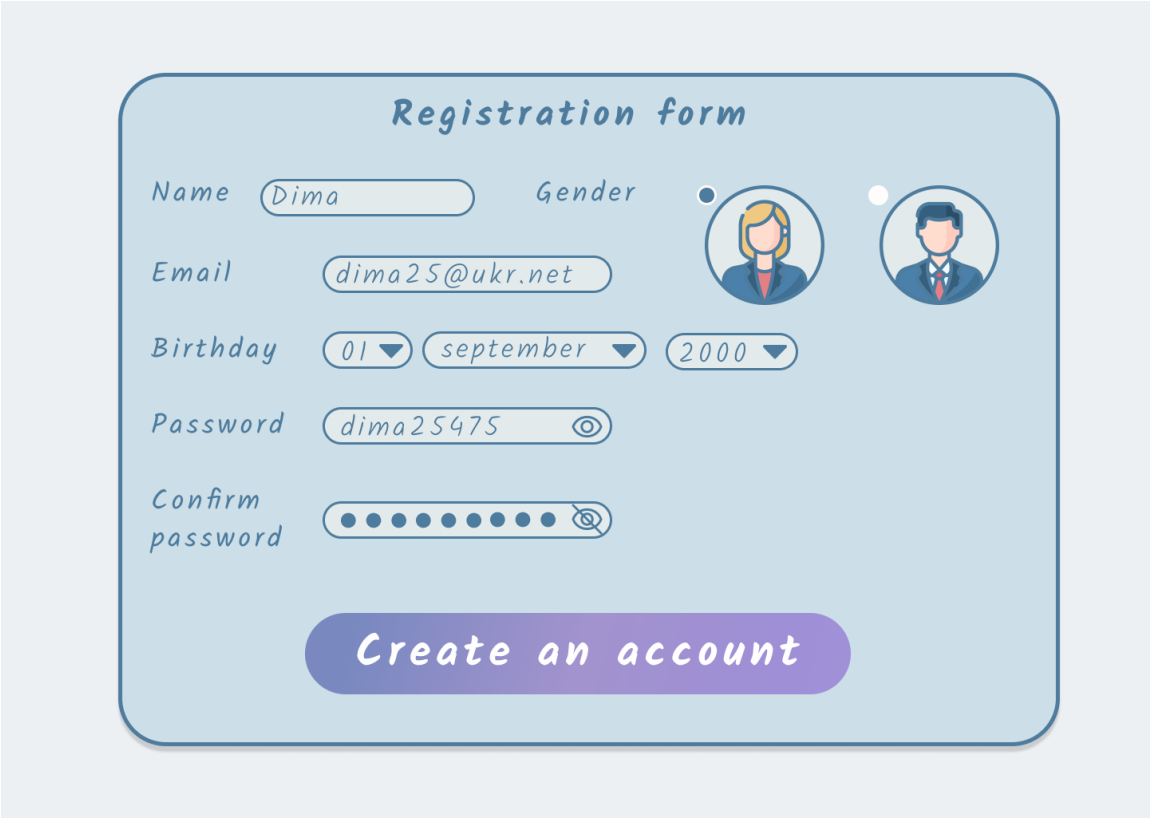

Рисунок 15 – Registration form

Наступним буде розглянуто прототип дизайну фіду (feed) чи стрічки публікацій. Feed виконаний по типу як в Instagram, тобто публікаціє це є безкінечна стрічка публікацій (чому безкінечна буде поянено в розділі 7), яку можно скролити вниз і переглядати фото, читати підписи чи статті, чи вірші. Переглядати відео, чи слухати музику, а також можлива наявність інших файлів. Розглянути як будуть розташовані публікації можно на малюнку 5. Фід буде головною сторінкою сайта і першою на яку потрапляє користувач після входу в акаунт.

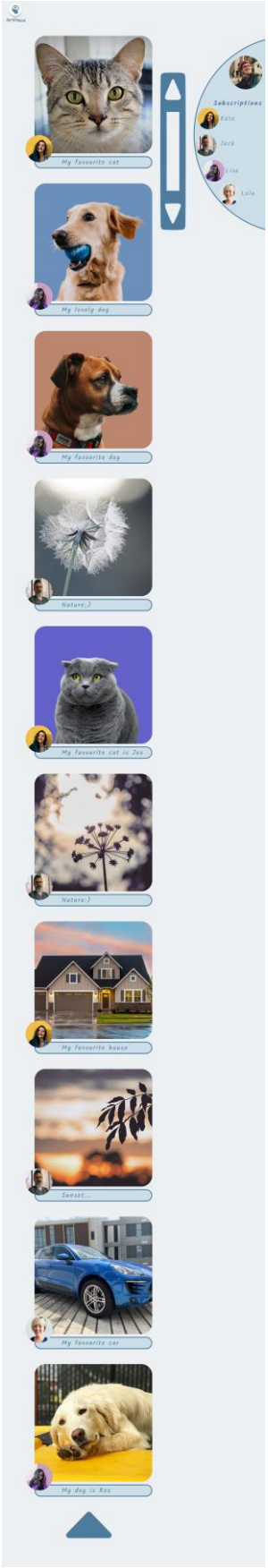

Рисунок 16 – Feeds

Давайте розглянемо публікацію ближче і сам користуватьський інтерфейс головної сторінки. Зліва у верхньому куті знаходиться логотип сайту. Справа знаходиться колесо контенту, лівіше від нього скролер, а центральну частину займає пост. Пост – це одиниця інформації в пабліку, тобто одиниця передачі контенту користувачу.

Пост (публікація) складається з тьрох чи більше елементів елеметів. Цими елементами є автатарка автора публікації, тобто фото (чи інще зображення) митця, якому належить публікація. Фото-іконка круглої форми і знаходиться знизу зліва блоку публікації. Обов'язково публікація повина мати підпис, який розташований у блакитному закругленому блокі внизу блоку публікації. Підпис може бути єдиним елементом публікації, так як акористувачами сайту є не тільки художники та діджитал артисти чи музиканти, а також поети, які пишуть вірші чи письменики, які пишуть романи, повісті чи новели, а також драматурги, які відповідно займаються п'єсами і мюзиклами. Рамки підпису скруглені по максимуму, це зроблено для того, щоб дизайн перекликувався з формою відображення автарки автора, яка лівіше від підпису. Для того, щоб підпис,я кщо він містить велику кількість тексту не займав все місце його можно буде скривати частично та відображати. На початку вміст буде прихований для зручності користувача.

Так як все ж дизайн додатку виконаний з елементами атрибутів образотворчого мистецтва, то основне місце в публікації займає блок зображення. Зображення займає верхню частину блоку публікації та має скруглені края. Користувачі-художники, користувачі-фотографи матимуть змогу викладувати свої роботи і їх їх підписники зможуть побачити завдяки зображенню.

Справа на колесі контенту (колесі підписок) користувач може бачитти авторів на яких він оформив безкоштовну чи платну підписку, а також ті пабліки на які він підписався (безкоштовно чи платно неважливо). При наведені на картинку автора (користувача контент-мейкера) головна сторінка

сайту буде затимнятися та вспливуть зображення, що відповідають паблікам даного автора. Фото знаходяться в круглих рамках. Користувач може натиснути на ту іконку на який паблік він хоче перейти. При натисканні на фото можно відкрити його повноформатний режим для більш детального огляду зображення, адже багато робіт мистецтва чи це фото чи картина мають дрібні деталі, які приковують погляд і вимагають пильного перегляду. Можно побачити це на малюнку нижче.

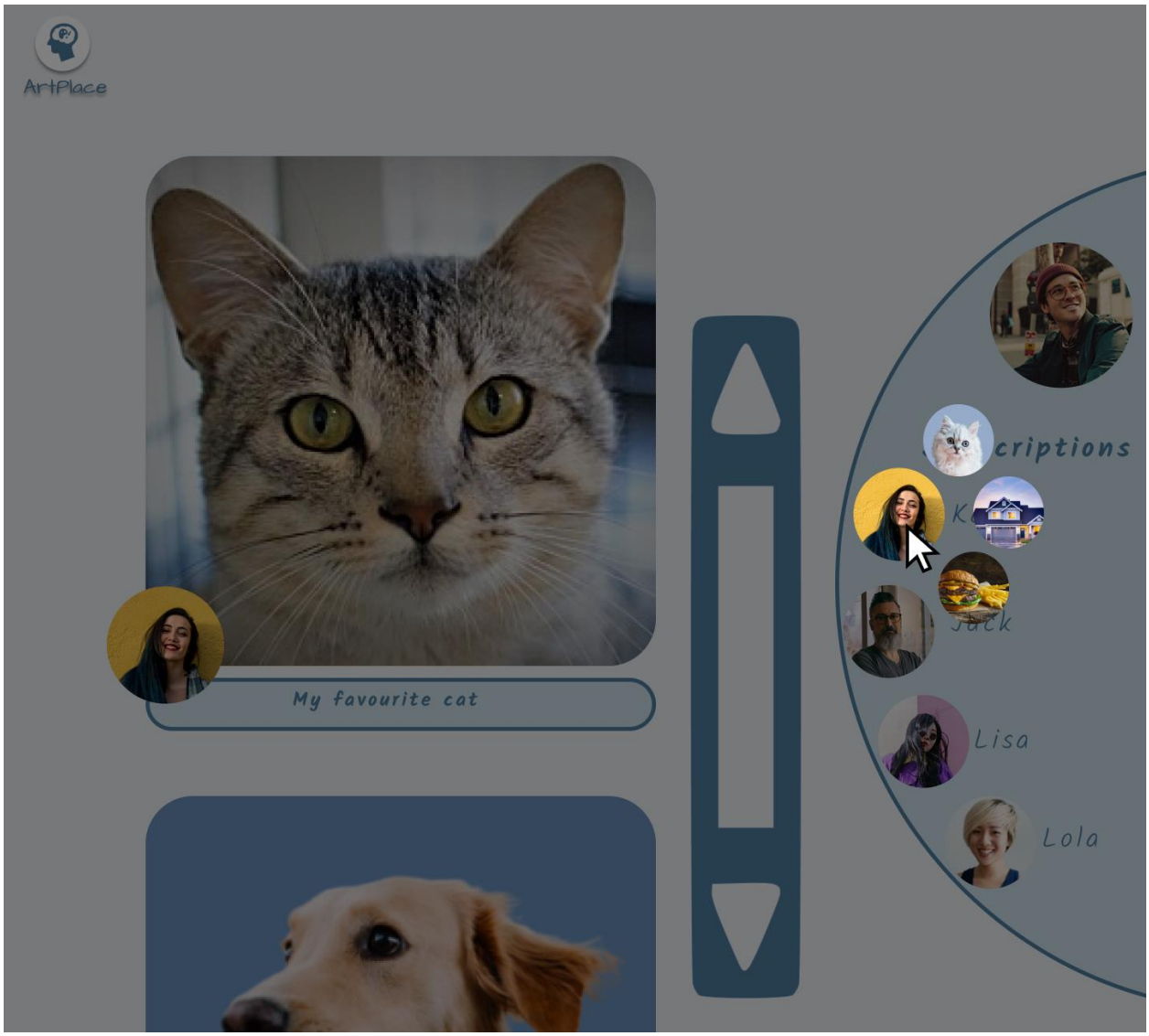

Рисунок 17 – Subscription publics

При наведені на зображення групи (пабліку) буде вспливати завдяки хуверу (hover) назва цього пабліку. Селектор: hover використовується для виділення елементів під час наведення на них курсора. Тобто завдяки цьому

селектору ми можемо визначити, який івент трапиться при наведені мишкою на певний об’єкт на веб сторінці. А відбудеться в нашому випадку всплив нотатки з назвою групи. Це потрібно для того, щоб користувач орієнтувався в назвах пабліків, якщо він не запам’ятав їх візуальне зображення (фото профайлу пабліка) чи іконку змінили.

Також на головній сторінці є скролер. Він знаходиться між блоком публікації та колесом підписок. Наявність його на сторінці має бути аргументоване, адже і так всі браузери мають скролери в правій частині вікна програми (веб-браузера). Причини його наявності в нашому веб додатку наступні: по-перше майбутня інтеграція і адаптація додатка до мобільних пристроїв (про неї в розділі 5.4), по-друге більш зручніше користування додатком для власників ноутбуків. Не всі ноутбуки мають функцію скролінгу двома пальцями і не всі користувачі ноутбуків користуються проводними чи безпроводними мишами. А наш скролер більш легкий у використанні: по-перше він більший і по-друге він знаходиться поряд з колесом контенту (воно ж колесо підписок), що означає, що рухи користувача по сторінкі не будуть виходити за межі центральної частини, що хоч і мізерно але економить час користувачу.

Також такий скролер вписується в чанкі дизайн з такою заокругленою формою, тобто він довершує абстракту композицію головної сторінки.

В верхній частині колеса підписок в більшом за розміром колі ніж підписки, знаходиться автарка користувача – зображення його профайлу. При натискані на яке ми можнмо перейти на сторінку де можна редагувати дані акаунту. А також переглянути свої пабліки і публікації. На рисунку нижче можно розглянути функціонал описаний вище.

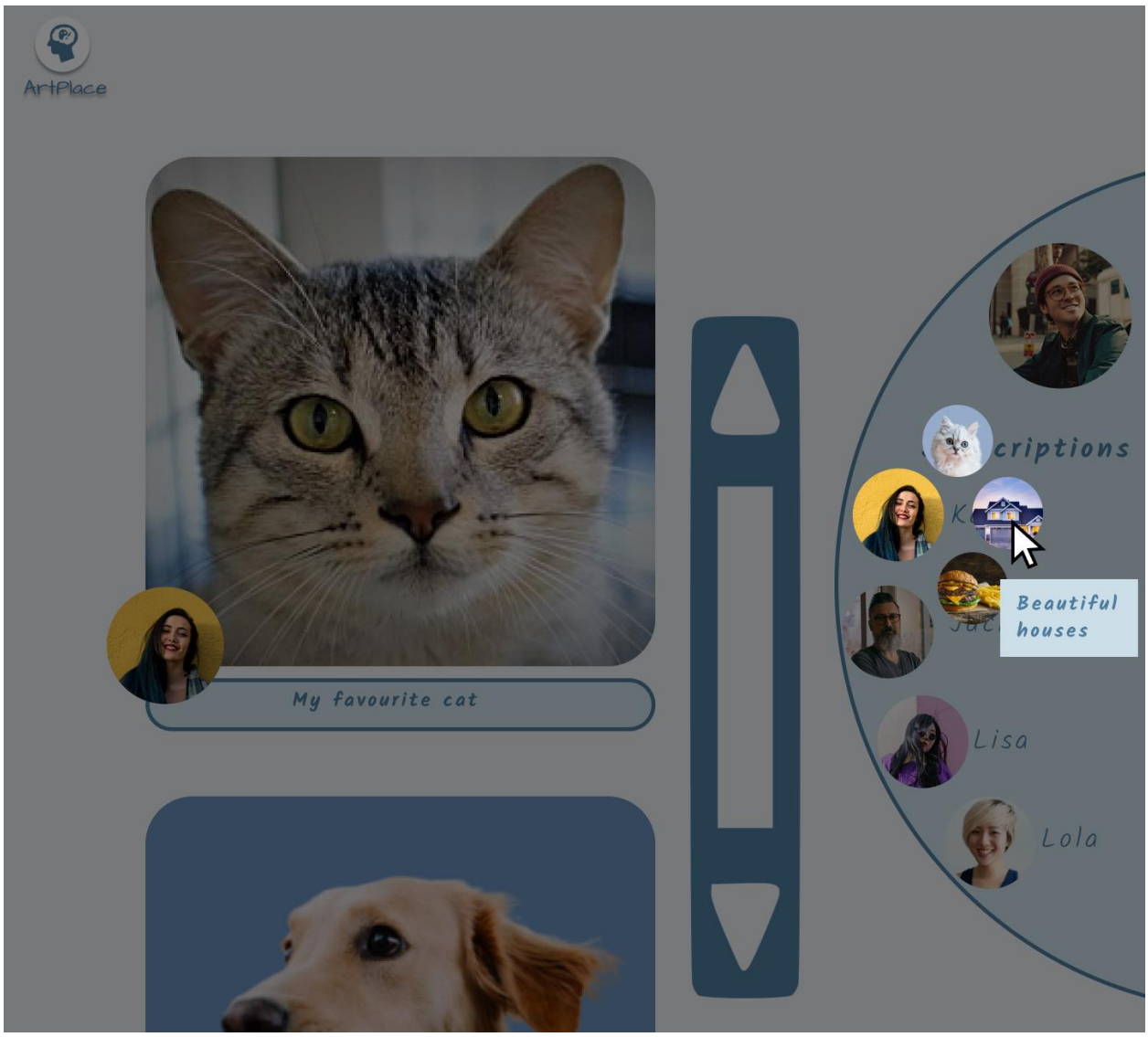

Рисунок 18 – Subscription publics (names)

На рисунку нижче розглянуто ще одну функцію анімації інтерфейсу зв’язану з хувером (селектор hover). При наведені на автора (користувача криейтора контенту) в блокі публікації, над його іконкою зображення вспливають зображення (міні іконки) його пабліків, груп якими він володіє. Та користувач може натиснути на них та перейти в ці групи. При цьому так само затемняється вся головна сторінка для хайлайтингу того на що навів мишку, точніше курсор користувач.

Для затемнення фона використовуються можливості мови html, css та JavaScript. Нам необхідно створити перекраваючий слой overlay. Overlay (оверлей), або перекриває шар - це шар, який зазвичай накладають поверх

фонового зображення для того, щоб затемнити або освітлити його. Ми робимо joverlay темного кольору. Існує декілька способів зробити такий ефект:

- Використання окремого шару з негативними margin
- Використання окремого шару з position: absolute
- Використання псевдоелементів :: before або :: after з position: absolute
- Використання множинного фону
- Використання position: fixed для overlay на всю висоту екрану

Ми скористалися третім варіантом. Та написали необхідні функції.

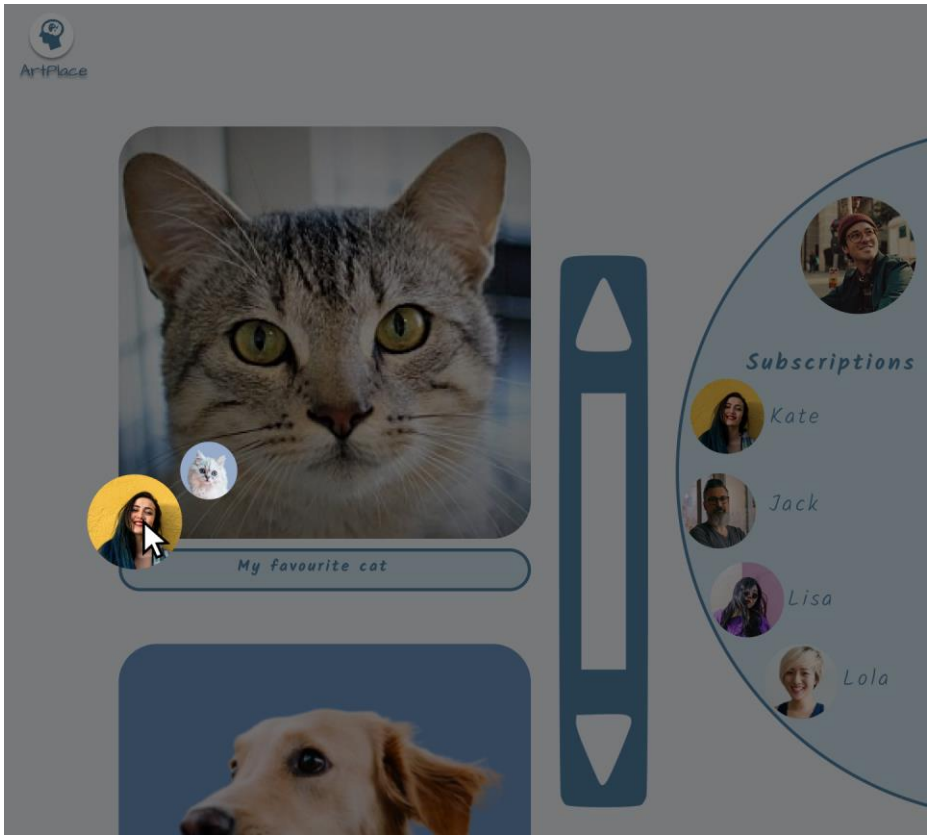

Рисунок 19 – Publics

Так само як із колесом підписок при наведені на автора публікації у блоці поста, та наведені на паблік, що належить автору вспливає назва цього пабліку, дивись рисунок 20.

Ця спливаюча назва розроблена наступним способом: Першим кроком створення спливаючого вікна є створення елемента (або будь-якого іншого елемента) і його оформлення. Цей і буде використовуватися в якості спливаючого вікна. Тепер ми його приховуємо за допомогою значення none властивості display і додаємо посилання, при наведення на яку буде з'являтися спливаюче вікно. Використовуючи псевдо-клас: target ми вибираємо і застосовуємо стилі до елементу, до якого був здійснений перехід. Таким чином після переходу по посилання значення властивості display елемента зміниться з none на block.

Пару слів про те як щробити такі круглф зображення. Для заокруглення кутів у елементів в CSS3 призначена властивість border-radius, значенням якого виступає радіус заокруглення. Якщо взяти квадратне зображення і додати до нього це властивість, то ми отримаємо вже не квадратне, а кругле зображення. Як значення слід задати половину ширини малюнка. Правда, можна вчинити і простіше і значенням вказати свідомо велике число, що перевищує розміри зображення. Так ми в будь-якому випадку отримаємо круглу картинку і зможемо застосовувати стиль до зображень різного розміру.

Плюсом використання border-radius є те, що ми можемо додавати в стилях до елементу рамку, тінь і вона буде повторювати контур. У прикладі 1 показано створення круглих зображень, для чого вводиться клас round, з тінню і рамкою.

Далі роглянемо сторінку акаунта (рисунок 21), для того щоб на нього перейти як було сказано раніше необхідно натиснути на кругле зображення вашого профілю (профілю користувача). Ви перейдете за посиланням на ваш акаунт. Тут ви побачите інформацію свого профілю, ваші підписки, і ваші публікації, якщо ви маєте власні пабліки.

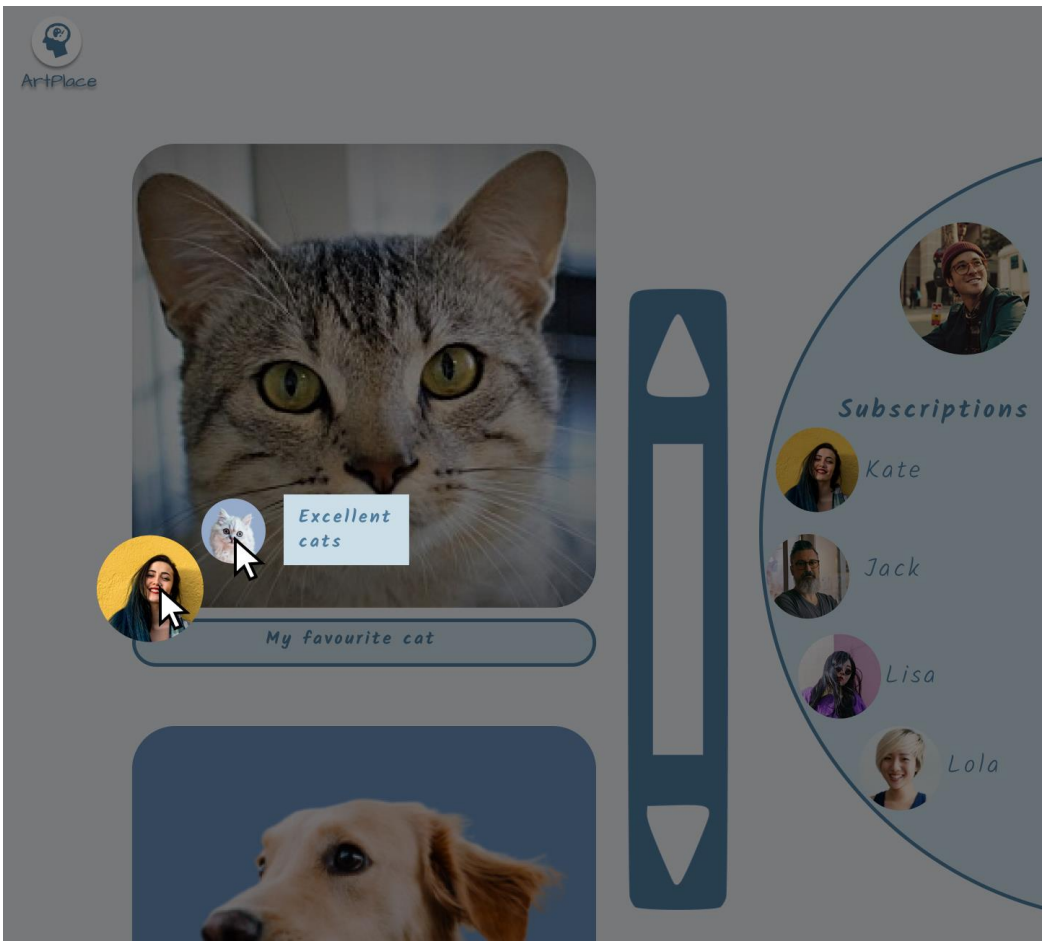

Рисунок 20 – Publics name

У лівій верхній частині знаходиться фото профайлу користувача, його ім’я чи нікнейм трохи правіше від фото та під ним дата народження, яка є опціональною і не буде відображатись, якщо вона була не введена перед цим.

В правому верхньому куті знаходиться іконка налаштувань (setting), де можна змінити чи додати будь яку наявну інформацію до вашого профілю, тобто змінити ім’я, дату народження чи прибрати її, вибрати стать чи заінити пароль, якщо користувач цього потребує.

Рисунок 21 – Your account (user account)

Наступним елементом сторінки користуватьського акаунту є меню бар для підписок (як безкоштовних, так і тих які оплачуються щомісяця). Він складається з рядка підписок, які представлені зображенням профілю пабліка. Заключені з двох сторін своєрідними стрілочками, які виконані у формі

рівнобічних трикутників зі скругленими кутами, трикутники є симетричними і напрямлені виступаючим кутом (вершиною) в різні сторони.

Правіше від меню підписок знаходиться значок плюсу в колі, при натискані на який можно додати нову підписку на групу чи паблік, тобто при натискані на плюз ми можемо виконати пошук нового пабліку і підписатися на нього.

В самому низу сторінки, майже в футері знаходиться стрілочка, такйже трикутник як і в меню барі підписок, яка дозволяє перегорнути сторінку публікацій. Виконаний він у формі трикутника, але являється просто блоком html.

Саме публікації займають основну і найбільшу частину вікна браузера. Ця основна зона розбита на три колонки та на два рядка. Якщо паблік має більше публікацій ніж 6 (3x2), то користувач натискає на трикутник, який додає іще 6 нових публікацій до цієї сітки. Мінімальна кількість публікацій 1 при прогортувані, максимальна як зрозуміло 6. Але якщо група тільки створена, то публікацій може бути нуль.

Для створення колонок є чотири методи. Перелічемо їх: фіктивні колонки, псевдоколонки, границі і негативні (від’ємні) відступи, зміщення колонок і контейнера.

Фіктивні колонки існують вже довгий час. За цей час вони були і, можливо, все ще залишаються методом де-факто для створення колонок однакової висоти. Цей простий трюк використовує фонове зображення яке повторюється по вертикалі. Зображення зазвичай робиться висотою в один піксель, а його ширина збігається з шириною макету. У тому місці, де одна колонка переходить в іншу, зображення для фіктивних колонок також має змінюватися. Серед плюсів методу: легко налаштувати, працює незалежно від того яка колонка (стовбець) розташовані нижче чи вище. Серед мінусів: необхвдне зображення, яка створює додатковий HTTP-запит, будь-яка зміна

макета вимагає переробки зображення, потрібно заздалегідь знати, як буде виглядати макет для створення зображення.

Псевдоколонки. Цей метод не важають самим вдалим. Він працює аналогічно методу фіктивних колонок, який передбачає додавання фону до контейнера. Це не загальний підхід до колонок однакової висоти, тому він має обмежену сферу застосування, але якщо ви стикаєтеся з таким випадком, все це працює легко і просто. Обмеження наступне - ми заздалегідь повинні знати, яка колонка буде коротше і в ідеалі ця колонка повинна бути такою на всіх сторінках сайту. Звучить, як досить серйозне обмеження, але на практиці, я вважаю, одна з двох колонок зазвичай коротше на більшості, якщо не на всіх, сторінках. На тих сторінках, де це не так, для виправлення досить просто додати більше або менше контенту. Серед плюсів даного методу: легко налаштувати, легко обслуговувати. Серед мінусів: важко трьох і більше колонок (як в нашому випадку), потрібна заздалегіть відома висота колонок, цей метод не буде працювати, коли колонки вище чи нижче на різних сторінках.

Межі і негативні відступи. На цей метод з’явився не так давно на Smashing Magazine в статті Тьєррі Кобленца, хоча пізніше був у статті Алана Пірса на A List Apart, написану кілька років тому про це ж методі. Використовуються кордону і негативні відступи, щоб створити видимість колонок рівної висоти. Спочатку дамо бічній панелі негативний правий margin рівний її ширині (або ширині лівої межі контенту, вони однакові). Це поверне бічну панель туди, куди ми і хочемо, але вона як і раніше не видно. Проблема в порядку накладення двох Дивов. #content розташовується поверх #sidebar, так що ми повинні перемістити #sidebar на передній план. Зробимо це шляхом додавання position: relative для бічній панелі і тепер його зміст стає видно. Серед переваг: працює незалежно від того, яка колонка вище чи нижче (корисно якщо розміри зображень користувачів будуть різного розміру), просте налаштування, але за умови гарної практичною бази, легко

обслуговувать.Серед недоліків: ширина бокової панелі повина бутти фіксована, оскільки border-width розуміє тільки абсолютні значення, егативні margin потенційно можуть привести до помилки в деяких старих версіях IE. Але як стало відомо IE. Про відмову від бренду Internet Explorer компанія оголосила ще в 2015 році і з тих пір робила це поступово. У минулому році IE 11 перестав підтримувати сервіс Microsoft Teams, а з 17 серпня 2021 року його більше не будуть підтримувати додатки і сервіси Microsoft 365.

Зсув колонок і контейнера. Останній метод створив Метью Джеймс Тейлор. З усіх методів представлених тут, на цей раз він буде працювати в самих різних випадках використання. В CSS відбувається дещо більше, ніж ми бачили до цього моменту. Для #sidebar і #content задається float із значенням left і встановлюється ширина, все інше крім цього є новим. Серед переваг: працює незалежно від того яка колонка вище чи коротше, підтримує будь-які макети (фіксованим, гумовим, еластичним і ін.), можна зробити скільки завгодно колонок. Серед мінусів: важка реалізація, потрібні додаткові несемантичні блоки div.

Передивившись всі переваги і недоліки методів, формування колонок виконана третім методом – зміщення колонок і контейнера.

В даному розділі були описані основні елементи інтерфейсу і методи їх виконання мовами HTML, CSS і javascript.

5.4 ПОДАЛЬША РОЗРОБКА

Так як період для розробки веб-додатку був досить короткий і зжатий, то деякі важливі функції не були реалізовані програмно із-за недостатку часу. В цьому розділі опишеться принцип дії та реалізація майбутнього функціоналу.

Серед особливостей додатку є також відсутність лайкі (вподобайок) чи іншої системи оцінки контенту користувачами. Це є необхідне для того, щоб єдиною формою оцінки робіт автора було оплата його роботи. Тобто

користувачі підписники оцінюють автора якра тим, що платятять йому грошові чи криптові кошти в тій сумі яку вважають запотрібну для даного автора. Спираючись на те як часто вони його переглядають, читають чи слухають і від того настільки їм подобається його контент.

Як було сказано в минулих роботах при перегляді аналогів та аналізів конкурентів основною перевагою є спосіб оплати. Гроші йдуть безпосередньо на рахунок контент криєртору. А він (власник пабліку) вже відає певну комісію власникам сайту. Тобто фактично сайт отримує доход, тоді коли безкоштовні пости залучать людей які захочуть платить за платні. Тобто оплата власником пабліку сайту відбувається тоді, коли хтось розблокує платний тариф. Фактично автор творчого контенту купляє можливість створення своєї платної підписки, але вносить оплату власникам тільки у випадку, коли його творчиство починає приносити кошти, що є дуже зручно, адже контент криєртор відчуває безпеку своїх коштів, адже він не буде їх втрачати, якщо його твори не будуть цікавими іншим юзерам.

У подальшому будуть додані **алгоритми підбору контенту за вподобаннями користуавачів**. Але як такої системи з лайками як сказано вище у доддатку немає. А оцінюватися контент буде таким чином: по часу затраченому на перешляд користувачаем одного поста, чи те чи дослухав користувач трек чи пісню, чи очитав до кінця пост чи вірш. Тобто те на що користувачі затрачують більше часу шоб переглянути чи ознайомитися буде вважатися популярним контентом.

Відповідно на перших порах використання ддодатку, коли користувачів буде небагато і бажано пропонувати людям нові підписки, буде видаватися в рекомендацію популярний контент. В дальнійшому будуть вираховуватися кластери вподобань.

Вираховуючи людей по сходим пабліка, тобто один кластер людей які підписані на одні й тіж пабліки по максимуму наближені будуть відправлятись

в одну категорію, далі нові користувачі, після невеликого періоду використання, коли алгорит зрозуміє, що цьому користувачеві довподоби буде пропонувать йому іще авторів і пабліки ті, які він вирахував зі схожих вподобань інших людей.

Розробка такого алгоритма це задача сфери машиного навчання. А саме нейроних мереж чи трансформерів, тобто так званий штучний інтелект. Машине навчання – це метод дата аналізу, що автоматизує аналітичну модель і імітує аналітичну діяльність людини. Це підрозділ штучного інтелекту, яке загалом визначається як здатність машини імітувати розумну поведінку людини. Системи штучного інтелекту використовуються для виконання складних завдань, подібних до того, як люди вирішують проблеми.

В нашому випадку машине навчання, алгорит буде підбирати людям авторі і ті пабліки, які б їх заохотили, тобто рекомендувати їм шось. Такі системи називаються рекомендаційними, наприклад як рекомендації фільмів чи кинижок, чи одна із найвідоміших сфер – рекомендація товару.

Можно выдылити два типи рекомендаційних систем, їх набагато бвльше, але наступні є основними:

Content-based.

- Користувачу рекомендуються об'єкти, подібні до тих, що цей користувач вже використовував.
- Схожості оцінюються за визнаками змістовного об'єкта.
- Сильна залежність від предметної області, корисність рекомендацій обмежена.

Collaborative Filtering.

- Для рекомендацій використовується історія оцінки як самого користувача, так і інших користувачів.
- Більш універсальний підхід, часто дає найкращий результат.

- Є свої проблеми (наприклад, холодний старт). [16]

Формалізовано задача звучить так: ми маємо множину користувачів $u \in U$, множину об’єктів $i \in I$ (атори чи пости нашого веб-додатку) і множину подій (R$ui, u, i, ..$) $\in \mathcal{D}$ (дії які користувачі виконують над об’єктнами чи з ними). Кожна подія задається користувачем u і об’єктом i, і його результатом rui. І задача здовиться до наступного:

- передбачити перевагу:

$r'ui = \text{Predict}\ (u, i, ...) \approx rui.$

- персональні рекомендації:

$u \longmapsto (i1, ..., iK) = \text{Recommend}K\ (u, ...).$

- схожі об'єкти:

$u \longmapsto (i1, ..., iM) = \text{Similar}M\ (i).$

Саме ці задачі повина бути вирішити машина можель, для того щоб додати систему рекомендацій до розробленого веб-додатку.

Функціонал другої основної переваги буде також в майбутньому. Можливість оплати криптовалютою. Користувач-підписник зможе оплатити роботу автора не лише звичайними валютами, такими як наприклад гривні чи долари, а також виплатою коштів штучними валютними системами.

Криптовалюти дозволяють купувати товари та послуги або торгувати ними з метою отримання прибутку. Криптовалюта (cryptocurrency or crypto) – це цифрова валюта, що використовує сильні алгоритми шифрування для онлайн-транзакцій.

Криптовалюти звертаються до своїх прихильників з різних причин. Ось декілька найпопулярніших:

- Прихильники розглядають такі криптовалюти, як біткойн, як валюту майбутнього і мчать купувати їх зараз, мабуть, ще до того, як вони стануть більш цінними
- Деяким прихильникам подобається той факт, що криптовалюта усуває центральні банки від управління грошовою масою, оскільки з часом ці банки, як правило, знижують вартість грошей через інфляцію
- Іншим прихильникам подобається технологія криптовалют, блокчейн, оскільки це децентралізована система обробки та запису і може бути більш безпечною, ніж традиційні платіжні системи
- Деякі дільці люблять криптовалюти, тому що вони зростають у ціні і не зацікавлені в довгостроковому прийнятті валют як способу переміщення грошей

Найпопулярнішмими криптовалютами (на сьогоднійшній день червень 2021) є Bitcoin, Ethereum, Tether, Binance Coin, Cardano та деякі інші.

У подальшому веб-додаток можливо адаптувати до Android-додатку. Адже структура прототипу дозволяє це зробити дуже легко. А також це зробить додаток більш зручним для користувачів, які споживають контент, адже не завжди сайт може відобразитись коректно на браузерах мобільних пристроїв.

Також в дальнійшому хочеться так би мовити “оживити” сайт і додати більше руху і анімацій. Анімація як правило асоціюється з мультипликацією або створенням мультфільмів. У веб-середовищі теж є поняття анімації, але воно дещо відміне. Елементи на сторінці браузеру можуть змінювати положення, прозорість, форму, розміри і так далі.

Але аикористання анімації повино бути в міру. Треба слідувати хочаб загальновідомим правилам: слідкувати за тим, щоб елемнти не були надто

нав’язливі; анімувати функціональні елементи – пункти меню, кнопки, плити чи активні посилання. Обов’язково слідкувати за якістю відображення анімації, адже анімація на веб ресурсі не має тормозити, це дуже відторгує користувачів.

Тому в дальнійшому колесо контенту (колесо підписок) буде анімовано так щоб при вході із зображення акварельних розводів воно перетворювалось в коесо з авторами та їх підписками. Також буде додані ефекти то вспливаючих пабліків та їх назв.

Зі збільшенням попиту на додаток будуть водитися нові зміни і додаватися новий функціонал.

6. ЕТАП ТЕСТУВАННЯ

Даний розділ продемонструє розгляд та опис етапу тестування. Основних виконаних пунктів, що забезпечують стабільність роботи системи та відповідають за якість розроблюваного продукту.

Етап тестування один з основних і найдовших етапів розробки будь якого програмного продукту. Життєвий цикл тестування програмного забезпечення (STLC) - це низка чітко визначених заходів, які тестери програм повинні виконати для забезпечення якості програмного забезпечення. Кожен з етапів процесу STLC повинен виконуватися систематично та послідовно.

Крім того, кожен з цих кроків має різні цілі та результати в кінці своєї фази. Хоча різні компанії можуть встановлювати власні життєві цикли тестування програмного забезпечення, основна структура цієї процедури тестування залишається незмінною. Простіше кажучи, це метод, який формалізує, як буде проводитися процес тестування. [14]

Тестування - один із найскладніших етапів процесу розробки програмного забезпечення. Він вимагає пильної уваги до деталей і не може бути завершений, якщо ви не застосуєте методичний підхід. Ось чому тестування програмного забезпечення розбивається на різні етапи.

Загалом існує чотири етапи тестування програмного забезпечення, які включають модульне тестування, інтеграційне тестування, системне тестування та перевірку прийнятності. З огляду на це, ці чотири етапи можна розділити на два типи, причому перші два - етапи перевірки, тоді як останні два є частиною етапу перевірки.

Ключова відмінність між ними полягає в тому, що тестування, проведене на етапах верифікації, базується на процесах, що використовуються під час розробки. На відміну від цього, етап перевірки перевіряє функціональність готового продукту і використовує зворотний зв’язок користувачів у підсумку.

Крім того, попередній етап процедури тестування використовує тестування White Box у своїх процесах. Це означає, що внутрішня структура програмного забезпечення не прихована, і фахівці, які проводять тестування, повинні знати про впровадження програмного забезпечення.

З іншого боку, при тестуванні чорних ящиків внутрішня структура програмного забезпечення прихована, і тестувальники використовують цей прийом на завершальному етапі перевірки. Тестери повинні перевірити функціональність програми відповідно до специфікацій або вимог, заявлених користувачами.

Давайте детально обговоримо кожен окремий етап, щоб чіткіше зрозуміти рівні тестування при тестуванні програмного забезпечення.

Unit Testing

Модульне тестування - це перший етап рівня тестування програмного забезпечення. На цьому етапі ми оцінюєм окремі компоненти системи, щоб перевірити, чи ці компоненти функціонують належним чином самостійно. В нашому проекті юніт тестування проходить мануально, та за допомогою білого тестування.

Integration Testing

Ми виконуємо інтеграційне тестування на наступному етапі тестування. Тут ми тестуємо окремі компоненти системи, а потім перевіряють їх як колективну групу. Це дозволяє нам при тестуванні програм визначати продуктивність окремих компонентів як групу та виявляти будь-які проблеми в інтерфейсі між модулями та функціями. Тестування було виконано манульно, та за допомогою білого тестування.

System Testing

Тестування системи є завершальним етапом процесу верифікації. На цьому етапі ми визначаємо, чи працює колективна група інтегрованих

компонентів оптимально. Процес є вирішальним для життєвого циклу якості, і ми прагнемо оцінити, чи може система відповідати стандартам якості та відповідає усім основним вимогам документації. Тестування було виконано мануально, та за допомогою білого тестування.

Acceptance Testing

Приймальне тестування є завершальним етапом тестового циклу контролю якості. Воно допомагає оцінити, чи готовий додаток до випуску для споживання користувачами. Ми проводимо цю фазу за допомогою представників замовника, тобто при його присутності, і він самостійно тестує додаток, використовуючи його. Тому вони він перевірить, чи може програма виконувати всі зазначені функції. Буде виконуватися за допомогою чорного ящика та мануально, але може бути виконано в будь якому іншому стилі за бажанням замовника.

Робота буде вестися посилено. Будуть аналізуватися помилки і на результатах будуть робитися висновки та імплементуватиметься поліпшення. Тобто в дальнійшому будуть розроблені класи автотестів (автоматизовані функціональні тести, що виконуються на повністю розгорнутому продукті в оточенні, максимально наближеному до реального в якому додаток буде функціонувать) та залучені інші інструменти для тестування інформаційних систем.

7. КОШТОРИС

Даний розділ присвячений грошовій стороні питання розробки даної інформаційної системи. Як ми бачимо із назви розділу – кошторис, тобто розділ про кошти. Знайдемо визначення даного терміна згідно з толкового словника української мови. Кошторис витрат — повне зведення витрат підприємства на виробництво та реалізацію продукції. [13] Тому в цьому розділі будуть описані необхідні витрати на повну розробку та підтримку даного веб додатку.

Для того, щоб отримати вартість розроблюваного продукту, нам необхідно підрахувати години витрачені на розробку веб-додатка, написання всього коду, аналітичну роботу, написання документації та проведеня тестування. Розділити ці години за типом роботи, тобто скільки часу пішло на кожну окрему вид діяльності. Знайти актуальну оплату за клжен з даних видів роботи за годину та підрахувати повну суму затрачену на розробку нашого веб-дадатку.

№	Назва діяльності	Затрачено годин, год	Вартість за годину, грн	Сума, грн
1	Написання кода мовою Java	160	343.75	55 000
2	Розробка прототипу сайту	30	125	3 750
3	Проведення тестування	60	134.4	8 064
4	Розробка бізнес логіки	15	156.25	2 345
5	Написання документації	30	187.5	4 125
6	Написання коду html та front-end	60	187.5	11 250
7	Розробка бази даних	45	165.6	7 452
	Всього	**400**		**91 986**

Таблиця 1 – Вартість продукту

Дані про оплату фахіфвців з різних сфер були взяті за актуальними цінами на сьогоднішній день на ринку праці в Україні. Наприклад середня заробітна праця Java розробника в Україні за версією сайта work.ua складає 55000 грн за п’яти денний робочий тиждень з обов’язковими 8 годинами роботи. Тобто шляхом неважких матиматичних підрахунків 55000/(4*5*8)=343.75 , знаходимо вартість години розробки мовою джава. Важаємо, що в місяці приблизно 4 тижні. За таким же принципом розаховуємо і вартість іншої діяльності, за необхідністю.

Перерахуємо спеціалістів і їх місячну зарплату по Україні на сьогоднішній день (червень 2021)

№	Посада	Місячна зарплата, грн
1	SQL - програміст	26 500
2	Java-програміст	55 000
3	Front-end-розробник	30 000
4	Web-developer	20 000
5	Тестувальник	21 500
6	Бізнес аналітик	25 000
7	Технічний письменик	30 000

Таблиця 2 – Зарплата в сфері IT

З таблиці бачимо, що вартість розробки даного програмного продукту становить 91 тисячу 986 гривень. Як бачимо даний проект це зовсі не дешеве задоволення.

ВИСНОВКИ

У даній роботі було розроблено веб-додаток для просування творчого контенту та підтримки його авторів.

У ході виконання цієї роьоти було розглянуто велику кількість джерл інформації про розробку веб-додатків, методологій, патернів та технологій. Виконаний аналіз цих даних та зроблено рішення про вибір мов програмування, методологій, допоміжних засобів.

Проаналізовані найбільш популярні аналоги, як мобільні, настільні так і веб-додатки, щоб врахувати всі переваги і недоліки конкурентів і визначити сильні сторони на той момент розроблюваного продукту.

Було обрано найбільш зручні інструменти для розробки за допомогою яких писався, компілювався код, а також інструменти для розробки фронтенду.

Були розроблені декілька діаграм (потоків даних, використання, функціоналу, класів та відносин між сутностями бази даних). З використанням мови UML, що дало змогу фокусуватись під час розробки на основний функціонал.

Було розроблено головні сервіси відповідно до бізнес логіки додатку. Створений прототип головних сторінок сайту.

Були виконанні всі необхідні етапи тестування для забезпечення якості програмного продукту.

Створений веб-додаток є прекрасним джерелом заробітку для працівників творчої сфери та для власників сайту. Також додаток слугує пізнавально розважальним місцем для його користувачів. Додаток є інтуітивно зрозумілим у використанні та частично безкоштовним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Packages in java and how to use them [Електронний ресурс] – 2018. – Режим доступу: https://beginnersbook.com/2013/03/packages-in-java/
2. What is Maven? Build and dependency management for Java] – 2020. – Режим доступу: https://www.infoworld.com/article/3516426/what-is-maven-build-and-dependency-management-with-apache-maven.html
3. Spring Data JPA [Електронний ресурс] – 2018. – Режим доступу: https://www.javatpoint.com/spring-boot-starter-data-jpa#:~:text=Spring%20Boot%20provides%20spring%2Dboot,since%20Spring%20Boot%20version%201.5.
4. Архитектура REST[Електронний ресурс] – 2008. – Режим доступу: https://habr.com/ru/post/38730/
5. Hibernate Tutorial [Електронний ресурс] – 2018. - Режим доступу: https://www.javatpoint.com/hibernate-tutorial
6. Introduction to Spring Framework [Електронний ресурс] – 2020. - Режим доступу: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html
7. A Java Event Represents a GUI Action in Java's Swing GUI API Framework [Електронний ресурс] – 2019. - Режим доступу: https://www.thoughtco.com/event-2034091#:~:text=An%20event%20in%20Java%20is,within%20a%20graphical%20user%20interface.&text=The%20relevant%20listener%20will%20have,will%20result%20in%20no%20action.
8. Data Validation [Електронний ресурс] – 2021. - Режим доступу: https://teachcomputerscience.com/validation/

9. Spring Boot – Thymeleaf [Електронний ресурс] – 2019. - Режим доступу: https://www.tutorialspoint.com/spring_boot/spring_boot_thymeleaf.htm#:~:text=Thymeleaf%20is%20a%20Java%2Dbased,XHTML%2FHTML5%20in%20web%20applications.
10. PostgreSQL [Електронний ресурс] – 2021. - Режим доступу: https://www.postgresql.org/about/
11. Guide to Spring Email [Електронний ресурс] – 2021. - Режим доступу: https://www.baeldung.com/spring-email
12. The Difference Between UX And UI Design [Електронний ресурс] – 2021. - Режим доступу: https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/
13. Кошторис [Електронний ресурс] – 2021. - Режим доступу: https://classes.ru/all-ukrainian/dictionary-ukrainian-explanatory-term-74212.htm
14. Software Testing Life Cycle Definition [Електронний ресурс] – 2021. - Режим доступу: https://performancelabus.com/software-test-life-cycle-stlc-importance/
15. What Is Cryptocurrency? Here's What You Should Know [Електронний ресурс] – 2021. - Режим доступу: https://www.nerdwallet.com/article/investing/cryptocurrency-7-things-to-know
16. Як працюють рекомендаційні системи. Лекция в Яндексе [Електронний ресурс] – 2021. - Режим доступу: https://habr.com/ru/company/yandex/blog/241455/

ДОДАТОК А

Основні елементи архітектури додатку (Java-код).

```
package birintsev.artplace.controllers;


import birintsev.artplace.services.UserService;
import lombok.AllArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;


@Controller
@RequestMapping(AdminController.RESOURCE_ADMIN)
@AllArgsConstructor
public class AdminController {

    public static final String RESOURCE_ADMIN = "/admin";

    public static final String RESOURCE_USERS = "/users";

    private static final String VIEW_NAME_USERS = "users";

    private static final Logger LOGGER = LoggerFactory.getLogger(
        AdminController.class
    );

    private final UserService userService;

    @RequestMapping(RESOURCE_USERS)
```

```
            protected ModelAndView users() {
                String usersModelAttrName = "users";
                return new ModelAndView(
                    VIEW_NAME_USERS,
                    usersModelAttrName,
                    userService.allUsers()
                );
            }
        }


        package birintsev.artplace.controllers;


        import birintsev.artplace.dto.PublicationDTO;
        import birintsev.artplace.dto.UserDTO;
        import birintsev.artplace.model.db.Publication;
        import birintsev.artplace.model.db.User;
        import birintsev.artplace.services.PublicService;
        import birintsev.artplace.services.PublicationService;
        import lombok.RequiredArgsConstructor;
        import org.modelmapper.ModelMapper;
        import org.modelmapper.TypeToken;
        import org.springframework.data.domain.Pageable;
        import org.springframework.http.HttpStatus;
        import org.springframework.http.ResponseEntity;
        import org.springframework.security.core.annotation.AuthenticationPrincipal;
        import org.springframework.stereotype.Controller;
        import org.springframework.web.bind.annotation.RequestMapping;
        import org.springframework.web.bind.annotation.RequestMethod;
        import org.springframework.web.servlet.ModelAndView;
        import java.util.ArrayList;
```

```
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;


@Controller
@RequiredArgsConstructor
@RequestMapping(path = "/feed")
public class FeedController {


    private static final String FEED_VIEW_NAME = "feed";


    private final ModelMapper modelMapper;


    private final PublicService publicService;


    private final PublicationService publicationService;


    @RequestMapping(path = "", method = RequestMethod.GET)
    protected ModelAndView feed(
        @AuthenticationPrincipal User user
    ) {
        ModelAndView mav = new ModelAndView(FEED_VIEW_NAME);
        mav.addObject(
            "publics",
            publicService.userSubscriptions(user)
        );
        mav.addObject(
            "user",
            modelMapper.map(user, UserDTO.class)
        );
```

```
                mav.addObject(
                    "publications",
                    mapPublications(
                        publicationService.findForUserFirstPage(user)
                            .stream()
                            .collect(Collectors.toList())
                    )
                );
                return mav;
            }

            @RequestMapping("/page")
            protected ResponseEntity<List<PublicationDTO>> feedPage(
                @AuthenticationPrincipal User user,
                Pageable pageable
            ) {
                return new ResponseEntity<>(
                    new ArrayList<>(
                        mapPublications(
                            publicationService.findForUser(user, pageable)
                                .stream()
                                .collect(Collectors.toList())
                        )
                    ),
                    HttpStatus.OK
                );
            }

            private Collection<PublicationDTO> mapPublications(
                Collection<Publication> publications
```

```
            ) {
                return modelMapper.map(
                    publications,
                    new TypeToken<List<PublicationDTO>>() {}.getType()
                );
            }
        }
        package birintsev.artplace.controllers;


        import birintsev.artplace.dto.PublicDTO;
        import birintsev.artplace.dto.PublicationDTO;
        import birintsev.artplace.dto.PublishRequestImpl;
        import birintsev.artplace.dto.UserDTO;
        import birintsev.artplace.model.db.Public;
        import birintsev.artplace.model.db.Publication;
        import birintsev.artplace.model.db.User;
        import birintsev.artplace.services.PublicService;
        import birintsev.artplace.services.PublicationService;
        import birintsev.artplace.services.exceptions.UnauthorizedOperationException;
        import lombok.RequiredArgsConstructor;
        import org.modelmapper.ModelMapper;
        import org.modelmapper.TypeToken;
        import org.slf4j.Logger;
        import org.slf4j.LoggerFactory;
        import org.springframework.data.domain.Pageable;
        import org.springframework.http.HttpStatus;
        import org.springframework.http.ResponseEntity;
        import org.springframework.security.core.annotation.AuthenticationPrincipal;
        import org.springframework.stereotype.Controller;
        import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.UUID;
import java.util.stream.Collectors;

@Controller
@RequiredArgsConstructor
@RequestMapping(path = "/publication")
public class PublicationController {

    private static final Logger LOGGER = LoggerFactory.getLogger(
        PublicationController.class
    );

    private static final String BOUGHT_PUBLICATIONS_VIEW_NAME =
        "boughtPublications";

    private final PublicationService publicationService;

    private final ModelMapper modelMapper;

    private final PublicService publicService;
```

```
    @RequestMapping("/bought")
    protected ModelAndView bought(
        @AuthenticationPrincipal User user
    ) {
        ModelAndView mav = new ModelAndView(BOUGHT_PUBLICATIONS_VIEW_NAME);
        mav.addObject(
            "user",
            modelMapper.map(user, UserDTO.class)
        );
        mav.addObject(
            "publications",
            mapPublications(
                publicationService.findPermanentPublicationsByUserFirstPage(
                    user
                )
                    .stream()
                    .collect(Collectors.toList())
            )
        );
        return mav;
    }

    @RequestMapping("/bought/page")
    protected ResponseEntity<List<PublicationDTO>> boughtPage(
        @AuthenticationPrincipal User user,
        Pageable pageable
    ) {
        return new ResponseEntity<>(
            new ArrayList<>(
                mapPublications(
```

```
                            publicationService
                                .findPermanentPublicationsByUser(user, pageable)
                                .stream()
                                .collect(Collectors.toList())
                        )
                    ),
                    HttpStatus.OK
                );
            }

            @RequestMapping(
                path = "/publish/{publicId}",
                method = RequestMethod.GET
            )
            protected ModelAndView publicationForm(
                @AuthenticationPrincipal User user,
                @PathVariable(name = "publicId") UUID publicId
            ) {
                Public parentPublic = publicService
                    .findById(publicId)
                    .orElseThrow(
                        () -> new NoSuchElementException(
                            String.format("Public (id = %s) not found", publicId)
                        )
                    );
                return new ModelAndView(
                    "createPublication",
                    Map.of(
                        "parentPublic",                 modelMapper.map(parentPublic,
PublicDTO.class)
                    )
```

```
        );
    }

    @ExceptionHandler(value = {
        NoSuchElementException.class
    })
    protected ModelAndView onPublicationFormRequest(
        NoSuchElementException exception
    ) {
        LOGGER.error(exception.getMessage(), exception);
        return new ModelAndView(
            "redirect:/error",
            Map.of("message", "Public with provided id not found")
        );
    }

    @RequestMapping(
        path = "/publish",
        method = RequestMethod.POST
    )
    protected ResponseEntity<?> publish(
        @AuthenticationPrincipal User user,
        PublishRequestImpl publishRequest
    ) {
        Publication p = publicationService.publish(user, publishRequest);
        LOGGER.debug("New publication has been created: " + p);
        return ResponseEntity
            .ok()
            .body(
                modelMapper.map(
```

```
                        p,
                        PublicationDTO.class
                    )
                );
        }

        @ExceptionHandler(value = {
            UnauthorizedOperationException.class
        })
        protected ResponseEntity<?> onPublicationCreation(
            UnauthorizedOperationException exception
        ) {
            LOGGER.error(exception.getMessage(), exception);
            return ResponseEntity
                .status(HttpStatus.FORBIDDEN)
                .body(new Object() {
                    final String message = "The user is not the public owner";
                });
        }

        private Collection<PublicationDTO> mapPublications(
            Collection<Publication> publications
        ) {
            return modelMapper.map(
                publications,
                new TypeToken<List<PublicationDTO>>() {}.getType()
            );
        }
    }
```

```
package birintsev.artplace.controllers;


import birintsev.artplace.dto.PublicDTO;
import birintsev.artplace.dto.PublicationDTO;
import birintsev.artplace.model.db.Public;
import birintsev.artplace.model.db.Publication;
import birintsev.artplace.model.db.User;
import birintsev.artplace.services.PublicService;
import birintsev.artplace.services.PublicationService;
import lombok.AllArgsConstructor;
import org.modelmapper.ModelMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.http.HttpSession;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.UUID;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;
```

```
/**
 * Controller for handling
 * {@link birintsev.artplace.model.db.Public Public}-related requests
 * */
@AllArgsConstructor
@RequestMapping("/public")
@Controller
public class PublicController {

    private static final Logger LOGGER = LoggerFactory.getLogger(
        PublicController.class
    );

    private final PublicService publicService;

    private final PublicationService publicationService;

    private final ModelMapper modelMapper;

    @RequestMapping(value = "/{publicId}", method = RequestMethod.GET)
    protected ModelAndView publicPage(
        @PathVariable("publicId") UUID publicId,
        HttpSession userSession
    ) {
        Public aPublic = publicService
            .findById(publicId)
            .orElseThrow(
                () -> new NoSuchElementException(
                    String.format("Public (id = %s) not found.", publicId)
                )
```

```
                );


            /*
            * TODO: add publications from UserPermanentPublication
            *       for the case if user visits not subscribed public
            * */
            Iterable<PublicationDTO> publications = mapPublications(
                publicationService.findByPublicFirstPage(aPublic)
            );
            return new ModelAndView(
                "publicPage",
                Map.of(
                    "public",
                        modelMapper.map(aPublic, PublicDTO.class),
                    "publications",
                        publications,
                    "subscribersAmount",
                        publicService.getTotalSubscribersAmount(aPublic),
                    "publicationsAmount",
                        publicationService.getTotalPublicationsCount(aPublic),
                    "pageSize",
                        10
                ),
                HttpStatus.OK
            );
        }


        @RequestMapping(
            value = "/unsubscribe/{publicId}",
            method = RequestMethod.GET,
```

```
            produces = MediaType.APPLICATION_JSON_VALUE
        )
        protected ResponseEntity<?> unsubscribe(
            @AuthenticationPrincipal User user,
            @PathVariable("publicId") Public publicToUnsubscribe
        ) {
            if (publicToUnsubscribe == null) {
                return ResponseEntity
                    .status(HttpStatus.BAD_REQUEST)
                    .contentType(MediaType.APPLICATION_JSON)
                    .body(new Object() {
                        public final String message = "Public not found";
                    });
            }

            if (!publicService.isSubscriber(user, publicToUnsubscribe)) {
                return ResponseEntity
                    .status(HttpStatus.BAD_REQUEST)
                    .contentType(MediaType.APPLICATION_JSON)
                    .body(new Object() {
                        public final String message = String.format(
                            "The user (id = %s) is not a subscriber"
                                + " of the public (id = %s).",
                            user.getId(),
                            publicToUnsubscribe.getId()
                        );
                    });
            }

            publicService.unsubscribe(user, publicToUnsubscribe);
```

```
                return ResponseEntity
                    .status(HttpStatus.OK)
                    .contentType(MediaType.APPLICATION_JSON)
                    .body(new Object() {
                        public final String message = String.format(
                            "The user (id = %s) has been unsubscribed"
                                + " from the public (id = %s).",
                            user.getId(),
                            publicToUnsubscribe.getId()
                        );
                    });
            }

            @ExceptionHandler(NoSuchElementException.class)
            protected ModelAndView onGetPublicPage(NoSuchElementException exception) {
                LOGGER.error(exception.getMessage(), exception);
                return new ModelAndView(
                    "redirect:/error",
                    Map.of("message", "Public with such id does not exist.")
                );
            }

            private Iterable<PublicationDTO> mapPublications(
                Iterable<Publication> publications
            ) {

                return StreamSupport.stream(publications.spliterator(), false)
                    .map(p -> modelMapper.map(p, PublicationDTO.class))
                    .collect(Collectors.toList());
            }
```

```
    }


    package birintsev.artplace.controllers;


    import birintsev.artplace.dto.RegistrationRequest;
    import birintsev.artplace.model.db.User;
    import birintsev.artplace.services.UserService;
    import birintsev.artplace.services.exceptions.UserExistException;
    import lombok.AllArgsConstructor;
    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.context.MessageSource;
    import org.springframework.http.HttpStatus;
    import org.springframework.stereotype.Controller;
    import org.springframework.validation.BindingResult;
    import org.springframework.web.bind.annotation.ModelAttribute;
    import org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RequestMethod;
    import org.springframework.web.bind.annotation.RequestParam;
    import org.springframework.web.servlet.ModelAndView;
    import javax.validation.Valid;
    import java.util.Locale;
    import java.util.NoSuchElementException;


    @AllArgsConstructor
    @Controller
    @RequestMapping(RegistrationController.REG_PAGE_PATH)
    public class RegistrationController {


        private static final Logger LOGGER = LoggerFactory.getLogger(
```

```
            RegistrationController.class
        );

        private static final String REG_REQUEST_MODEL_ATTR_NAME =
            "registrationRequest";

        private static final String ERROR_MODEL_ATTR_NAME =
            "error";

        private static final String ERRORS_MODEL_ATTR_NAME =
            "errors";

        private static final String MESSAGE_MODEL_ATTR_NAME =
            "message";

        public static final String REG_PAGE_PATH =
            "/registration";

        public static final String REG_FORM_HANDLER_PATH =
            "/form-submit";

        public static final String REG_CONFIRM_HANDLER_PATH =
            "/confirm";

        private static final String REG_VIEW_NAME =
            "registration";

        private static final String ACCOUNT_VIEW_NAME =
            "account";
```

```
            private static final String ERROR_VIEW_NAME =
                "error";


            private final UserService userService;


            private final MessageSource messageSource;


            private final Locale localeDefault;


            @RequestMapping(
                method = RequestMethod.GET
            )
            protected ModelAndView registrationPage() {
                ModelAndView modelAndView = new ModelAndView(REG_VIEW_NAME);
                modelAndView.getModelMap().addAttribute(
                    REG_REQUEST_MODEL_ATTR_NAME,
                    new RegistrationRequest()
                );
                return modelAndView;
            }


            @RequestMapping(
                method = RequestMethod.POST,
                path = REG_FORM_HANDLER_PATH
            )
            protected ModelAndView registrationFormSubmit(
                @ModelAttribute(REG_REQUEST_MODEL_ATTR_NAME)
                @Valid
                RegistrationRequest request,
                BindingResult errors
```

```
            ) {
                ModelAndView modelAndView;


                if (errors.hasErrors()) {
                    modelAndView = new ModelAndView(
                        REG_VIEW_NAME,
                        REG_REQUEST_MODEL_ATTR_NAME,
                        request
                    );
                    modelAndView.addObject(ERRORS_MODEL_ATTR_NAME, errors);
                    modelAndView.setStatus(HttpStatus.BAD_REQUEST);


                    return modelAndView;
                }
                try {
                    User user = userService.register(request);
                    modelAndView = new ModelAndView(ACCOUNT_VIEW_NAME);


                    modelAndView.addObject(
                        MESSAGE_MODEL_ATTR_NAME,
                        messageSource.getMessage(
                            "registration.success.message",
                            new Object[] {user.getName()},
                            localeDefault
                        )
                    );
                    modelAndView.setStatus(HttpStatus.OK);
                } catch (Exception e) {
                    LOGGER.error(e.getMessage(), e);
                    modelAndView = new ModelAndView(REG_VIEW_NAME);
```

```
                modelAndView.addObject(ERROR_MODEL_ATTR_NAME, e);
                modelAndView.addObject(REG_REQUEST_MODEL_ATTR_NAME, request);
                modelAndView.setStatus(HttpStatus.INTERNAL_SERVER_ERROR);
            }
            return modelAndView;
        }

        @RequestMapping(
            path = REG_CONFIRM_HANDLER_PATH,
            method = RequestMethod.GET
        )
        protected ModelAndView confirm(@RequestParam(value = "token") String
token) {
            ModelAndView mav;
            try {
                userService.confirm(token);
                mav = new ModelAndView("redirect:/login");
            } catch (UserExistException e) {
                LOGGER.error(e.getMessage(), e);
                mav = new ModelAndView("redirect:" + ACCOUNT_VIEW_NAME);
            } catch (NoSuchElementException e) {
                mav = new ModelAndView(
                    ERROR_VIEW_NAME,
                    ERROR_MODEL_ATTR_NAME,
                    messageSource.getMessage(
                        "registration.token.expired.message",
                        new Object[]{},
                        localeDefault
                    )
                );
            }
```

```
            return mav;
        }
    }

    package birintsev.artplace.controllers;

    import birintsev.artplace.dto.UserDTO;
    import birintsev.artplace.model.db.User;
    import lombok.AllArgsConstructor;
    import org.modelmapper.ModelMapper;
    import org.springframework.security.core.annotation.AuthenticationPrincipal;
    import org.springframework.stereotype.Controller;
    import org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RequestMethod;
    import org.springframework.web.servlet.ModelAndView;

    @Controller
    @AllArgsConstructor
    public class UserAccountController {

        private final ModelMapper modelMapper;

        @RequestMapping(
            method = RequestMethod.GET,
            path = "/account"
        )
        protected ModelAndView account(@AuthenticationPrincipal User user) {
            ModelAndView mav = new ModelAndView("account");
            mav.addObject(
                "userDto",
```

```
                    modelMapper.map(user, UserDTO.class)
            );
            mav.addObject("readOnly", Boolean.TRUE);
            return mav;
        }
    }
    package birintsev.artplace.services;


    import birintsev.artplace.converters.PublicationPreparer;
    import birintsev.artplace.dto.PublishRequest;
    import birintsev.artplace.model.db.Public;
    import birintsev.artplace.model.db.Publication;
    import birintsev.artplace.model.db.User;
    import birintsev.artplace.model.db.UserPermanentPublication;
    import birintsev.artplace.model.db.embeddable.UserPublicationAssociation;
    import birintsev.artplace.model.db.repo.PublicRepo;
    import birintsev.artplace.model.db.repo.PublicationRepo;
    import birintsev.artplace.model.db.repo.UserPermanentPublicationRepo;
    import birintsev.artplace.services.exceptions.UnauthorizedOperationException;
    import lombok.AllArgsConstructor;
    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.context.ApplicationEventPublisher;
    import org.springframework.core.convert.ConversionService;
    import org.springframework.data.domain.Pageable;
    import org.springframework.data.domain.Slice;
    import org.springframework.stereotype.Service;
    import java.math.BigInteger;
    import java.util.stream.Collectors;
```

```
        @Service
        @AllArgsConstructor
        public class DefaultPublicationService implements PublicationService {


            private static final Logger LOGGER = LoggerFactory.getLogger(
                DefaultPublicationService.class
            );


            private final PublicationRepo publicationRepo;


            private final PublicRepo publicRepo;


            private final ApplicationEventPublisher eventPublisher;


            private final ConversionService conversionService;


            private final UserPermanentPublicationRepo permanentPublicationRepo;


            @Override
            public Slice<Publication> findForUser(User subscriber, Pageable pageable)
{
                return publicationRepo.findAllBySubscriber(
                    subscriber,
                    pageable
                );
            }


            /**
             * @see PublicationPreparer See how the publishRequest
             *                          is converted to a Publication
             * */
```

```
            @Override
            public Publication publish(
                User publisher,
                PublishRequest publishRequest
            ) {
                if (
                    !publicRepo.isOwner(publisher,
publishRequest.getParentPublicId())
                ) {
                    throw new UnauthorizedOperationException(
                        String.format(
                            "The user (id = %s) is not the public (id = %s) owner.",
                            publisher.getId(),
                            publishRequest.getParentPublicId()
                        )
                    );
                }
                return publicationRepo.save(
                    conversionService.convert(
                        publishRequest,
                        Publication.class
                    )
                );
            }


            @Override
            public Slice<Publication> findByPublic(
                Public parentPublic,
                Pageable pageable
            ) {
                return publicationRepo.findAllByParentPublic(parentPublic, pageable);
```

```
            }

            @Override
            public int getTotalPublicationsCount(Public aPublic) {
                return publicationRepo.countAllByParentPublic(aPublic);
            }

            @Override
            public void bindForPaidSubscribers(Publication publication) {
                if (!isPaid(publication)) {
                    LOGGER.info(
                        String.format(
                            "The publication (id = %s) is free. Skipping binding.",
                            publication.getId()
                        )
                    );
                    return;
                }
                permanentPublicationRepo.saveAll(
                    publicRepo.findSubscribersByPublicAndTariff(
                        publication.getParentPublic(),
                        publication.getTariff()
                    ).stream()
                        .map(user -> new UserPermanentPublication(
                            new UserPublicationAssociation(user, publication)
                        )).collect(Collectors.toList())
                );
            }

            @Override
```

```
    public Slice<Publication> findPermanentPublicationsByUser(
        User subscriber,
        Pageable pageable
    ) {
        return permanentPublicationRepo.findAllByUser(subscriber, pageable);
    }

    private boolean isPaid(Publication publication) {
        return BigInteger.ZERO.compareTo(
            publication.getTariff().getPrice()
        ) < 0;
    }
}

package birintsev.artplace.services;

import birintsev.artplace.model.db.Public;
import birintsev.artplace.model.db.PublicSubscription;
import birintsev.artplace.model.db.User;
import birintsev.artplace.model.db.repo.PublicRepo;
import birintsev.artplace.model.db.repo.PublicSubscriptionRepo;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import java.util.Optional;
import java.util.UUID;
```

```
@Service("DefaultPublicService")
@AllArgsConstructor
public class DefaultPublicService implements PublicService {

    private final PublicRepo publicRepo;

    private final PublicSubscriptionRepo publicSubscriptionRepo;

    @Override
    public Page<Public> userSubscriptions(User subscriber) {
        return publicRepo.findAllBySubscriber(
            subscriber,
            defaultSubscriptionsFirstPage()
        );
    }

    @Override
    public Optional<Public> findById(UUID publicId) {
        return publicRepo.findById(publicId);
    }

    @Override
    public int getTotalSubscribersAmount(Public aPublic) {
        return publicRepo.getTotalSubscribersCount(aPublic);
    }

    @Override
    public void unsubscribe(User user, Public subscribedPublic) {
        Optional<PublicSubscription> optionalPublicSubscription =
            publicSubscriptionRepo.findById(
```

```
                        new PublicSubscription.PublicSubscriptionId(
                            user.getId(),
                            subscribedPublic.getId()
                        )
                    );
                optionalPublicSubscription.ifPresent(publicSubscriptionRepo::delete);
            }

            @Override
            public boolean isSubscriber(User user, Public aPublic) {
                return publicRepo.isSubscriber(user, aPublic);
            }

            /**
             * A default {@link Pageable} for querying the first page
             * of a user subscriptions.
             * */
            private Pageable defaultSubscriptionsFirstPage() {
                final int defaultPageSize = 5;
                return PageRequest.of(0, defaultPageSize, Sort.by("name"));
            }
        }

        package birintsev.artplace.services;

        import birintsev.artplace.dto.RegistrationRequest;
        import birintsev.artplace.eventslisteners.SendRegistrationConfirmationEvent;
        import birintsev.artplace.model.db.Authority;
        import birintsev.artplace.model.db.RegistrationConfirmation;
        import birintsev.artplace.model.db.User;
```

```
import birintsev.artplace.model.db.repo.RegistrationRepo;
import birintsev.artplace.model.db.repo.UserRepo;
import birintsev.artplace.services.exceptions.UserExistException;
import lombok.AllArgsConstructor;
import org.modelmapper.ModelMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.NoSuchElementException;
import java.util.Optional;
import java.util.Set;
import java.util.UUID;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;

@Service
@AllArgsConstructor
public class DefaultUserService implements UserService {

    private static final Logger LOGGER = LoggerFactory.getLogger(
        DefaultUserService.class
    );

    private final ApplicationEventPublisher eventPublisher;

    private final UserRepo userRepo;
```

```
            private final RegistrationRepo regRepo;


            private final ModelMapper modelMapper;


            private final PasswordEncoder passwordEncoder;


            @Override
            // todo @transactional
            public User register(RegistrationRequest registrationRequest) {
                User newUser = modelMapper.map(registrationRequest, User.class);
                newUser.setPassword(passwordEncoder.encode(newUser.getPassword()));


                newUser.addAuthority(Authority.REG_CONF_PENDING);


                RegistrationConfirmation registrationConfirmation =
                    new RegistrationConfirmation(
                        newUser.getId(),
                        newUser,
                        Timestamp.valueOf(LocalDateTime.now()),
                        null,
                        Timestamp.valueOf(
                            LocalDateTime.now()

.plus(RegistrationConfirmation.DEFAULT_TOKEN_EXPIRATION)
                        ),
                        UUID.randomUUID().toString()
                    );


                // ! order is important
                /* [1] */ userRepo.save(newUser);
```

```
        /* [2] */ regRepo.save(registrationConfirmation);
        /* [3] */ eventPublisher.publishEvent(
            new SendRegistrationConfirmationEvent(registrationRequest)
        );

        return newUser;
    }

    @Override
    public Set<User> allUsers() {
        return StreamSupport
            .stream(userRepo.findAll().spliterator(),false)
            .collect(Collectors.toSet());
    }

    @Override
    // todo @transactional
    public void confirm(String token) throws UserExistException {
        User user;
        RegistrationConfirmation registrationConfirmation;
        Optional<RegistrationConfirmation> _regConf =
            regRepo.findByToken(token);
        if (_regConf.isEmpty() || isExpired(_regConf.get())) {
            throw new NoSuchElementException(
                String.format(
                    "The registration (token=%s) can not be confirmed."
                        + " Reasons: it does not exist or has got expired",
                    token
                )
            );
```

```
            } else {
                registrationConfirmation = _regConf.get();
                user = registrationConfirmation.getUser();
            }
            if (isConfirmed(registrationConfirmation)) {
                throw new UserExistException(
                    "This registration has already been confirmed"
                );
            }
            registrationConfirmation.setConfirmedWhen(
                Timestamp.valueOf(LocalDateTime.now())
            );
            user.addAuthority(Authority.REG_CONFIRMED);
            regRepo.save(registrationConfirmation);
            userRepo.save(user);
        }

        /**
         * <strong>Note!</strong>
         * <p>
         * This method does not take into account
         * if the checked RegistrationConfirmation object
         * has already been confirmed.
         * It only informs if the token
         * <strong>expiration date is in the future or not</strong>.
         * <p>
         * {@code expirationDate = null} means that this token can not be outdated.
         * */
        private boolean isExpired(RegistrationConfirmation regConf) {
            Timestamp expiresWhen = regConf.getExpiresWhen();
```

```
            return expiresWhen != null
                && Timestamp.valueOf(LocalDateTime.now()).after(expiresWhen);
        }

        private boolean isConfirmed(RegistrationConfirmation regConf) {
            return regConf.getConfirmedWhen() != null;
        }
    }

    package birintsev.artplace.services;

    import lombok.AllArgsConstructor;
    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.stereotype.Service;
    import java.io.File;
    import java.io.FileInputStream;
    import java.io.FileNotFoundException;
    import java.io.IOException;
    import java.io.InputStream;
    import java.net.URI;
    import java.net.URISyntaxException;
    import java.net.URL;
    import java.nio.file.Files;
    import java.nio.file.Paths;
    import java.nio.file.StandardOpenOption;

    /**
     * Saves incoming files from the network to files system.
     * */
```

```
@Service
@AllArgsConstructor
public class NetworkToFileSystemFileService implements FileService {

    private static final Logger LOGGER = LoggerFactory.getLogger(
        NetworkToFileSystemFileService.class
    );

    /**
     * {@inheritDoc}
     *
     * This method does not close the passed input stream.
     * */
    @Override
    public URI saveFile(
        InputStream fileStream,
        String fileName,
        URL location
    ) {
        validProtocol(location);
        final File targetFile;
        try {
            targetFile = new File(new File(location.toURI()), fileName);
        } catch (URISyntaxException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(
                "An error happened during converting location URL -> URI",
                e
            );
        }
```

```
                try {
                    return Files.write(
                        targetFile.toPath(),
                        fileStream.readAllBytes(),
                        // TODO: configure StandardOpenOptions to not to overwrite
files
                        StandardOpenOption.WRITE,
                        StandardOpenOption.TRUNCATE_EXISTING,
                        StandardOpenOption.CREATE
                    ).toUri();
                } catch (IOException e) {
                    LOGGER.error(e.getMessage(), e);
                    throw new RuntimeException(
                        "An error happened during file writing",
                        e
                    );
                }
            }

            @Override
            public InputStream getFile(URI fileId) throws FileNotFoundException {
                validProtocol(fileId);
                return new FileInputStream(Paths.get(fileId).toFile());
            }

            private void validProtocol(URI uri) {
                if (!isOfSupportedProtocol(uri)) {
                    throw new UnsupportedOperationException(
                        String.format("The protocol of %s is not supported", uri)
                    );
                }
```

```
        }

        private void validProtocol(URL url) {
            if (!isOfSupportedProtocol(url)) {
                throw new UnsupportedOperationException(
                    String.format("The protocol of %s is not supported", url)
                );
            }
        }

        private boolean isOfSupportedProtocol(URL url) {
            return url != null && isProtocolSupported(url.getProtocol());
        }

        private boolean isOfSupportedProtocol(URI uri) {
            return uri != null && isProtocolSupported(uri.getScheme());
        }

        private boolean isProtocolSupported(String protocol) {
            return "file".equalsIgnoreCase(protocol);
        }
    }
```