МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційно-комунікаційна система для організації соціальних взаємовідносин»

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

Керівник роботи **Шелехов І.В.**

Студента групи ІН – 73 – 9 **Коваль О.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп’ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-73-9 спеціальності “Комп'ютерні науки” денної форми навчання Коваля Олексій Олександровича.

Тема: “**Інформаційно-комунікаційна система для організації соціальних взаємовідносин**”

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) аналітичний огляд методів створення системи; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і критеріїв, що використовуються для розробки системи; 5) розробка інформаційного й програмного забезпечення системи.

Дата видачі завдання “_____”_____2021 г.

Керівник випускної роботи _____ Шелехов І.В.

Завдання приняв до виконання _____ Коваль О.О.

РЕФЕРАТ

Записка: 52 стор., 13 рис., 12 додаток, 15 джерел.

Об'єкт дослідження — процес проектування системи обміну інформацією.

Мета роботи — розробка інформаційної системи обміну інформацією між людьми, знаходження груп з спільними інтересами та розваг.

Методи дослідження — метод проектування інформаційних систем, методи проектування та нормалізації баз даних.

Результати — розроблено інформаційну систему обміну інформацією між людьми, знаходження груп з спільними інтересами та розваг, засвоєно на реалізації програмного продукту безліч бібліотек Node JS, серед яких: React; Redux, – також фреймворк Express JS та написання на ньому rest api сервера з підключенням до бази даних MongoDB.

ІНФОРМАЦІЙНА СИСТЕМА ОБМІНУ ІНФОРМАЦІЄЮ, REACT, REDUX, NODE JS, JAVA SCRIPT, EXPRESS JS, MONGODB.

ЗМІСТ

ВСТУП

Соціальна павутина – це сукупність соціальних відносин, які пов’язують людей через Інтернет. Соціальна мережа охоплює те, як програмне забезпечення веб-сайта розробляються з метою сприяння та підтримки соціальній взаємодії. Ці взаємодії в Інтернеті складають основу великої діяльності, включаючи: веб-магазини, освіту, ігри та соціальні мережі. Соціальний аспект спілкування через Web 2.0 полягав у сприянні взаємодії між людьми зі схожими поглядами. Ці погляди різняться залежно від того, хто цільова аудиторія, і що їм потрібно. Для осіб, які працюють у відділі зв’язків з громадськістю, робота постійно змінюється, і вплив походить від соціальної мережі. Вплив соціальної мережі дуже великий і постійно змінюється.

У міру зростання активності людей в Інтернеті та обміном інформацією про їхні соціальні стосунки стає доступнішою. Сайти соціальних мереж, такі як Telegram та Facebook, а також майбутні Dataweb дозволяють людям та організаціям контактувати один з одним. Сьогодні безліч користувачів Інтернету використовують тисячі соціальних веб-сайтів, щоб залишатися на зв'язку зі своїми близькими та друзями, знаходити нових «друзів» та ділитися створеним користувачами контентом, таким як: фотографії, відео, соціальні закладки та блоги; навіть через мобільну платформу підтримка стільникових телефонів. До другого кварталу 2017 року Facebook повідомив про 1,86 мільярда учасників, а у 2008 році MySpace зареєстрував 100 мільйонів користувачів, а YouTube мав понад 100 мільйонів відео та 2,8 мільйона каналів, і ця сума постійно змінюється у більшу сторону. Соціальна Мережа швидко обновлює себе, виходячи за межі простих веб-додатків, які дають людям доступ до абсолютно нового способу життя.

1 АНАЛІТИЧНИЙ ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Сучасні платформи для соціальних відносин

1 – Facebook

Це легко найбільший сайт соціальних мереж у світі та один із найбільш широко використовуваних. І, мабуть, Facebook був першим, хто перевершив значущість у 1 млрд. Облікових записів користувачів.

Окрім можливості спілкуватися з друзями та родичами, ви також можете отримати доступ до різних додатків Facebook для продажу в Інтернеті, і ви навіть можете продавати або рекламувати свій бізнес, бренд та продукти, використовуючи платні оголошення Facebook

Почати роботу на Facebook легко, оскільки майже весь формат контенту чудово працює на Facebook – текст, зображення, відео, відео в реальному часі та Історії. Але зверніть увагу, що алгоритм Facebook надає пріоритет вмісту, який викликає розмови та змістовні взаємодії між людьми, особливо з родиною та друзями. Якщо ви хочете дізнатись більше про успіх за допомогою оновленого алгоритму Facebook, Браян Пітерс, наш маркетолог стратегічного партнерства, поділився секретами нового алгоритму та тим, що ви можете зробити, щоб процвітати у Facebook.

Також не забудьте оптимізувати свій вміст для мобільних пристроїв, оскільки 94 відсотки користувачів Facebook отримують доступ до Facebook через мобільний додаток.

2 – Instagram

Instagram був запущений як унікальна платформа соціальних мереж, яка повністю базувалася на обміні фотографіями та відео. Таким чином, ця програма для обміну фотографіями в соціальних мережах дозволяє вам зафіксувати найкращі моменти свого життя за допомогою камери телефону чи будь-якої іншої камери та перетворити їх на витвори мистецтва.

Це можливо, оскільки Instagram дозволяє застосовувати до ваших фотографій кілька фільтрів, і ви можете легко розміщувати їх на інших

популярних сайтах соціальних мереж, таких як Facebook і Twitter. Зараз це частина імперії Facebook.

3 - Twitter

Twitter - це сайт у соціальних мережах для новин, розваг, спорту, політики тощо. Що відрізняє Twitter від більшості інших сайтів соціальних медіа, це те, що він робить сильний акцент на інформацію в режимі реального часу - те, що відбувається зараз. Наприклад, одним із визначальних моментів в історії Twitter є те, коли Джаніс Крумс написав у Твіттері зображення літака, який приземлився в річці Гудзон, коли він був на поромі, щоб забрати пасажирів.

Ще однією унікальною характеристикою Twitter є те, що він дозволяє лише 280 символів у твіті (140 для японської, корейської та китайської), на відміну від більшості сайтів у соціальних мережах, які мають набагато вищу межу.

Twitter також часто використовується як канал обслуговування клієнтів. За даними рекламодавців у Twitter, понад 80 відсотків запитів на обслуговування соціальних споживачів відбувається в Twitter. А Salesforce називає Twitter «новим номером 1-800 для обслуговування клієнтів». Зараз існує багато інструментів обслуговування клієнтів у соціальних мережах, таких як Buffer Reply, які допоможуть вам керувати розмовами щодо обслуговування клієнтів у соціальних мережах.

4 - WhatsApp

WhatsApp - це програма обміну повідомленнями, яку використовують люди у понад 180 країнах. Спочатку WhatsApp використовувався лише людьми для спілкування з родиною та друзями. Поступово люди почали спілкуватися з бізнесом через WhatsApp. (Коли я був у Бангкоку, щоб купити новий костюм, я спілкувався з кравцем через WhatsApp.)

WhatsApp розробляє свою бізнес-платформу, щоб дозволити компаніям мати належний профіль бізнесу, надати підтримку клієнтам і ділитися новинами з покупцями про свої покупки. Для малого бізнесу він створив додаток WhatsApp

Business, тоді як для середнього та великого бізнесу існує WhatsApp Business API. Ось кілька історій про те, як підприємства використовують WhatsApp.

5 - Телеграма

Ця мережа обміну миттєвими повідомленнями схожа на WhatsApp і доступна на різних платформах більш ніж на восьми мовах. Однак Telegram завжди більше зосереджувався на конфіденційності та безпеці повідомлень, які ви надсилаєте через Інтернет за допомогою його платформи. Отже, це надає вам змогу надсилати зашифровані та саморуйнуючі повідомлення. Ця функція шифрування доступна лише зараз для WhatsApp, тоді як Telegram завжди надавала її.

6 – YouTube

YouTube - це платформа для обміну відео, де користувачі щодня переглядають мільярд годин відео. Для початку ви можете створити канал YouTube для свого бренду, куди ви можете завантажувати відео для своїх передплатників для перегляду, лайкування, коментування та обміну.

Окрім того, що це другий за величиною сайт у соціальних мережах, YouTube (належить Google) також часто відомий як друга за величиною пошукова система після Google.

1.2 Аналіз програмних засобів і компонентів, необхідних для розробки програмного продукту

Розробка соціальної мережі це довгий і трудомісткий процес. В рамках написання дипломної роботи, проект буде реалізований в демонстраційному вигляді. Створення веб-сайтів охоплює широкий асортимент засобів розробки, мов програмування і т.д. В реалізації цього проекту будуть задіяні такі мови програмування: HTML, JS, CSS, Java, Node, React, Redux. Нижче наведено список, короткий опис, призначення програмний засіб і компонентів, які використовуються в процесі розробки.

HTML

Мова розмітки гіпертексту (HTML) – це стандартна мова розмітки для документів, призначених для відображення у веб-браузері. Йому можуть допомогти такі технології, як каскадні таблиці стилів (CSS) та мови сценаріїв, такі як JavaScript.

Веб-браузери отримують документи HTML з веб-сервера або з локальної пам’яті та надають документи на мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично та включає ознаки зовнішнього вигляду документа.

Елементи HTML – це будівельні блоки HTML-сторінок. За допомогою конструкцій HTML зображення та інші об’єкти, такі як інтерактивні форми, можуть бути вбудовані у відтворену сторінку. HTML забезпечує засіб для створення структурованих документів, позначаючи структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML розмежовані тегами, записаними за допомогою кутових дужок. Такі теги, як <img /> та <input />, безпосередньо вводять вміст на сторінку. Інші теги, такі як <p>, оточують та надають інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не відображають теги HTML, але використовують їх для інтерпретації вмісту сторінки.

HTML може вбудовувати програми, написані мовою сценаріїв, наприклад JavaScript, що впливає на поведінку та вміст веб-сторінок. Включення CSS визначає вигляд та макет вмісту. Консорціум World Wide Web (W3C), колишній супровідник HTML і поточний супровід стандартів CSS, заохочує використання CSS над явним презентаційним HTML з 1997 року.

CSS

Каскадні таблиці стилів (CSS) – це мова таблиць стилів, що використовується для опису презентації документа, написаного мовою розмітки, як HTML. CSS – це наріжна технологія Всесвітньої павутини, поряд з HTML та JavaScript.

CSS призначений для розділення презентації та вмісту, включаючи макет, кольори та шрифти. Це розділення може покращити доступність вмісту, забезпечити більшу гнучкість і контроль у специфікації характеристик презентації, дозволити декільком веб-сторінкам спільно використовувати форматування, вказавши відповідний CSS в окремому файлі .css, що зменшує складність і повторюваність структурного вмісту, а також дозволяє файл .css, який потрібно кешувати, щоб покращити швидкість завантаження сторінки між сторінками, що ділять файл, та його форматування.

Поділ форматування та вмісту також робить можливим подання однієї і тієї ж сторінки розмітки в різних стилях для різних методів візуалізації, таких як екран, друк, голосом (через мовний браузер або зчитувач екрана) та на основі Брайля тактильні пристосування. CSS також має правила альтернативного форматування, якщо доступ до вмісту здійснюється на мобільному пристрої.

Каскадне ім'я походить із зазначеної пріоритетної схеми, щоб визначити, яке правило стилю застосовується, якщо більше одного правила відповідає певному елементу. Ця схема каскадного пріоритету передбачувана.

Специфікації CSS підтримуються Консорціумом World Wide Web (W3C). Текст / css типу Інтернет-носія (тип MIME) зареєстровано для використання з CSS RFC 2318 (березень 1998 р.). W3C забезпечує безкоштовну службу перевірки CSS для документів CSS.

На додаток до HTML, інші мови розмітки підтримують використання CSS, включаючи XHTML, звичайний XML, SVG та XUL.

JS

JavaScript, скорочений як JS, – це мова програмування, яка відповідає специфікації ECMAScript. JavaScript є високорівневим, часто встигуючим до компіляції та мультипарадигмою. Він має синтаксис фігурних дужок, динамічне введення тексту, орієнтацію на об’єкти на основі прототипу та функції першого класу.

Поряд з HTML та CSS, JavaScript є однією з основних технологій Всесвітньої павутини. JavaScript забезпечує інтерактивні веб-сторінки та є важливою частиною веб-додатків. Переважна більшість веб-сайтів використовують його для поведінки на стороні клієнта, і всі основні веб-браузери мають спеціальний механізм JavaScript для його виконання.

Як мова багатопарадигми, JavaScript підтримує керовані подіями, функціональні та імперативні стилі програмування. Він має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та об’єктною моделлю документа (DOM). Однак сама мова не включає жодного введення / виводу (вводу / виводу), наприклад, мережевих, сховищних чи графічних засобів, оскільки хостове середовище (зазвичай веб-браузер) забезпечує ці API.

Спочатку двигуни JavaScript використовувались лише у веб-браузерах, але зараз вони вбудовані в деякі сервери, як правило, через Node.js. Вони також вбудовані в різні програми, створені за допомогою таких фреймворків, як Electron та Cordova.

Незважаючи на те, що між JavaScript та Java є подібність, включаючи назву мови, синтаксис та відповідні стандартні бібліотеки, ці дві мови відрізняються і сильно відрізняються за дизайном.

Node

Node.js – це середовище виконання JavaScript (Framework) із відкритим кодом, міжплатформене, яке виконує код JavaScript поза веб-браузером. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера - запуску сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки до того, як сторінка буде відправлена у веб-браузер користувача. Отже, Node.js представляє парадигму «JavaScript скрізь», що об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для серверних та клієнтських сценаріїв.

Хоча .js є стандартним розширенням імені файлу для коду JavaScript, назва «Node.js» не посилається на певний файл у цьому контексті, а є просто назвою продукту. Node.js має керовану подіями архітектуру, здатну до асинхронного вводу-виводу. Ці варіанти дизайну спрямовані на оптимізацію пропускної здатності та масштабованості у веб-додатках з багатьма операціями введення / виведення, а також для веб-додатків у реальному часі (наприклад, програм спілкування в реальному часі та браузерних ігор).

Раніше розподілений проект розробки Node.js керувався Node.js Foundation, і тепер він об’єднався з JS Foundation, щоб сформувати OpenJS Foundation, що сприяє програмі спільних проектів Linux Foundation.

Java

Java – це об'єктно-орієнтована мова програмування на основі класів, яка розроблена для забезпечення якомога меншої залежності від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники додатків могли писати один раз, запускати їх де завгодно (WORA), тобто скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера. Синтаксис Java схожий на C та C ++, але він має менше засобів низького рівня, ніж будь-який з них. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, що використовується згідно з GitHub, особливо для веб-додатків клієнт-сервер, із 9 мільйонами розробників.

Спочатку Java була розроблена Джеймсом Гослінгом у Sun Microsystems (яка з тих пір була придбана Oracle) і випущена в 1995 році як основний компонент Java-платформи Sun Microsystems. Оригінальні та довідкові реалізатори Java-компілятори, віртуальні машини та бібліотеки класів були спочатку випущені Sun під власні ліцензії. Станом на травень 2007 року, згідно з вимогами Процесу спільноти Java, Sun здійснила переліцензію більшості своїх

технологій Java під загальною публічною ліцензією GNU. Тим часом інші розробили альтернативні реалізації цих технологій Sun, такі як компілятор GNU для Java (компілятор байт-кодів), GNU Classpath (стандартні бібліотеки) та IcedTea-Web (плагін браузера для аплетів).

Останніми версіями є Java 14, випущена в березні 2020 року, та Java 11, яка наразі підтримується довгостроковою підтримкою (LTS), випущена 25 вересня 2018 року; Oracle випустив для застарілої Java 8 LTS останнє безкоштовне загальнодоступне оновлення в січні 2019 року для комерційного використання, хоча в іншому випадку він все ще підтримуватиме Java 8 із загальнодоступними оновленнями для особистого використання принаймні до грудня 2020 року. Oracle (та інші) настійно рекомендують видалити старіші версії Java через серйозні ризики через невирішені проблеми безпеки. Оскільки Java 9, 10, 12 та 13 більше не підтримуються, Oracle радить своїм користувачам негайно перейти до останньої версії (на даний момент Java 14) або випуску LTS.

React

React (також відомий як React.js або ReactJS) – це бібліотека JavaScript з відкритим кодом для побудови користувальницьких інтерфейсів або компонентів інтерфейсу. Він підтримується Facebook та спільнотою окремих розробників та компаній. React можна використовувати як основу при розробці односторінкових або мобільних додатків. Однак React займається лише наданням даних в DOM, тому створення React-програм зазвичай вимагає використання додаткових бібліотек для управління станом та маршрутизації. Redux та React Router є відповідними прикладами таких бібліотек.

Redux

Redux – це бібліотека JavaScript з відкритим кодом для управління станом програми. Найчастіше використовується з такими бібліотеками, як React або Angular, для побудови користувальницьких інтерфейсів. Подібно до (і натхненною) архітектурою Facebook Flux, її створили Дан Абрамов та Ендрю Кларк.

1.3 Постановка задачі

Метою роботи є розробка та програмна реалізація інформаційно-комунікаційної системи для організації соціальних взаємовідносин. Для досягнення поставленої мети необхідно виконати такі завдання:

1. Обрати методологію проектування інформаційної системи
2. Виконати проектування серверної частини інформаційної системи
3. Виконати проектування клієнтської частини інформаційної системи
4. Визначити структуру бази даних інформаційної системи
5. Обрати програмне середовище для реалізації інформаційної системи
6. Програмно реалізувати такі модулі інформаційної системи:
    - модуль аутентифікації користувача

2 Вибір методів рішення

2.1 Вибір методу моделі

Модель водоспаду – це розбивка проектної діяльності на лінійні послідовні фази, де кожна фаза залежить від результатів попередньої та відповідає спеціалізації завдань. У розробці програмного забезпечення він, як правило, є одним з менш ітеративних та гнучких підходів, оскільки прогрес тече в основному в одному напрямку («вниз», як водоспад) через фази задуму, ініціації, аналізу, проектування, побудови, випробування, розгортання та обслуговування .

У оригінальній моделі водоспаду Ройса дотримуються наступних фаз:

1. Вимоги до системи та програмного забезпечення: зафіксовано в документі вимог до продукту.
2. Аналіз: результати моделей, схем та бізнес-правил.
3. Дизайн: результат архітектури програмного забезпечення.
4. Кодування: розробка, перевірка та інтеграція програмного забезпечення.
5. Тестування: систематичне виявлення та налагодження дефектів.
6. Операції: установка, міграція, підтримка та обслуговування повних систем.

Таким чином, модель водоспаду стверджує, що переходити до фази слід лише тоді, коли попередня фаза переглядається та перевіряється.

Однак різні модифіковані моделі водоспаду (включаючи остаточну модель Ройса) можуть включати незначні або великі зміни в цьому процесі. Ці варіації включали повернення до попереднього циклу після виявлення недоліків нижче за течією або повернення до фази проектування, якщо подальші фази вважалися недостатніми.

Час, витрачений на початку циклу виробництва програмного забезпечення, може зменшити витрати на наступних етапах. Наприклад, проблему, виявлену на ранніх стадіях (наприклад, специфікацію вимог), виправити дешевше, ніж ту саму помилку, виявлену пізніше в процесі (у 50–200 разів).

За загальноприйнятою практикою методології водоспадів дають графік проекту, коли 20–40% часу витрачається на перші два етапи, 30–40% часу кодується, а решта присвячується тестуванню та впровадженню. Фактична організація проекту повинна бути високоструктурованою. Більшість середніх та великих проектів включатимуть детальний набір процедур та засобів контролю, які регулюють кожен процес проекту.

Подальшим аргументом для моделі водоспаду є те, що вона робить акцент на документації (такі як документи вимог та проектна документація), а також на вихідний код. У менш ретельно розроблених та задокументованих методологіях знання втрачаються, якщо члени команди виїжджають до завершення проекту, і проекту може бути важко оговтатися від втрат. Якщо присутній повністю робочий проектний документ (як це передбачено Big Design Up Front та модель водоспаду), нові члени команди або навіть зовсім нові команди повинні мати можливість ознайомитись, прочитавши документи.

Модель водоспаду забезпечує структурований підхід; сама модель просувається лінійно через дискретні, легко зрозумілі та пояснювані фази і, отже, її легко зрозуміти; це також забезпечує важко визначені етапи в процесі розробки. Можливо, саме з цієї причини модель водоспаду використовується як початковий приклад моделі розвитку у багатьох текстах та курсах з програмного забезпечення.

Стверджується, що модель водоспаду може підходити для проектів, де вимоги та сфера застосування визначені, сам продукт міцний і стабільний, а технологія чітко зрозуміла.

Клієнти можуть точно не знати, які їх вимоги, перш ніж вони побачать працююче програмне забезпечення, і тому змінять свої вимоги, що призведе до переробки, переробки та повторного тестування та збільшення витрат.

Дизайнери можуть не знати про майбутні труднощі при розробці нового програмного продукту або функції, і в цьому випадку краще переглянути дизайн,

ніж наполягати на дизайні, який не враховує жодних нововиявлених обмежень, вимог чи проблем.

Організації можуть спробувати вирішити відсутність конкретних вимог у клієнтів, використовуючи системних аналітиків для вивчення існуючих ручних систем та аналізу того, що вони роблять і як їх можна замінити. Однак на практиці важко витримати чіткий розділення між системним аналізом та програмуванням. Це пов’язано з тим, що впровадження будь-якої нетривіальної системи майже неминуче відкриє проблеми та крайні випадки, які системний аналітик не розглядав.

У відповідь на сприйняті проблеми з моделлю чистого водоспаду були введені модифіковані моделі водоспаду, такі як «Сашимі (Водоспад із фазами, що перекриваються), Водоспад з підпроектами та Водоспад зі зменшенням ризику».

Деякі організації, такі як Міністерство оборони Сполучених Штатів, тепер заявляють про перевагу щодо методологій типу водоспаду, починаючи з MIL-STD-498, що заохочує еволюційні придбання та ітеративний та додатковий розвиток.

Хоча прихильники гнучкої розробки програмного забезпечення стверджують, що модель водоспаду є неефективним процесом розробки програмного забезпечення, деякі скептики вважають, що модель водоспаду є помилковим аргументом, що використовується суто для виведення на ринок альтернативних методологій розвитку.

Фази Раціонального уніфікованого процесу (RUP) визнають програмну необхідність у визначенні етапів, для підтримки проекту на шляху, але заохочують повторення (особливо в межах дисциплін) протягом етапів. Фази RUP часто називають «подібними до водоспаду».

У відповідь на сприйняті проблеми з «чистою» моделлю водоспаду було представлено багато модифікованих моделей водоспаду. Ці моделі можуть відповісти на деякі чи всі зауваження щодо «чистої» моделі водоспаду.

Сюди входять моделі швидкого розвитку, Стів Макконнелл називає «модифікованими водоспадами»: «модель сашимі» Пітера ДеГрейса (водоспад із фазами, що перекриваються), водоспад із субпроектами та водоспад із зменшенням ризику. Існують також інші комбінації моделей розробки програмного забезпечення, такі як «модель додаткового водоспаду».

Остаточна модель Вінстона В. Ройса, його передбачуване вдосконалення в порівнянні з початковою «моделлю водоспаду», продемонстрував, що зворотний зв'язок може (повинен і часто може) вести від тестування коду до дизайну (як тестування виявлених недоліків коду в дизайні) і від дизайну назад до специфікації вимог (оскільки проблеми проектування можуть вимагати усунення суперечливих або іншим чином незадовільних / невизначених вимог). У тому ж документі Ройс також виступав за велику кількість документації, виконуючи роботу «двічі, якщо це можливо» (почуття, подібне до почуття Фреда Брукса, відомого завдяки написанню місяця міфічної людини, впливової книги з управління програмними проектами, який виступав за планування «викинути одного»), і залучаючи клієнта якомога більше (настрої, подібні до екстремальних програмувань).

Примітки Ройса до остаточної моделі:

1. Повне проектування програми перед початком аналізу та кодування.
2. Документація повинна бути актуальною та повною.
3. Виконайте роботу двічі, якщо це можливо.
4. Випробування повинні плануватися, контролюватися і контролюватися.
5. Залучіть замовника.

2.2 Вибір засобів програмування

2.2.1 Середа виконання

Node.js – це середа виконання, яка має безліч переваг, визначень та особливостей, одними з найвідомих є:

1. Node є асинхронним подієвим JavaScript–оточенням, створеним для побудови масштабованих мережевих додатків. Вона заснована на JS движку V8 з Chrome.
2. Її пакетний менеджер, npm, є однією з найбільших у світі мережею бібліотек з відкритим вихідним кодом.

Середа дозволяє виконувати JavaScript код без участі браузера, що дозволяє використання JS також у серверній частині.

До інших переваг можна додати:

6. Перша перевага у використанні однієї мови у написанні одразу і фронтової частини сайту так і бекендової. Це допомагає зменшити рівень знань, та більшої ефективності взаємодії персоналу у розробці додатку.
7. Швидкість у написанні додатку, полягає у тому що, відкритий код створює багато можливості для розповсюдження своїх методів вирішення проблем, і це потрібно використовувати.
8. Так же вище зазначений відкритий код, дозволяє розвиватися середі не тільки за рахунок її розробників та і за звичайних програмістів які створюють свої бібліотеки, тому з часом актуальність додатків не буде зменшуватися.

2.2.2 Бібліотеки Node.js

2.2.2.1 React

Обрана середа є прогресивною, ефективною, популярною та так далі, але не лише через це вона обрана. Одна з новітніх бібліотек для написання веб-додатків – React, саме вона та її принципи основні у проекті.

Ця бібліотека створення для розробки інтерфейсу користувача.

Її переваги у відносній швидкості та ефективності односторінкових веб-сайтів, за рахунок нового підходу вирішення проблеми. Також використання компонент – функцій або класів, перші з яких приймають у параметрах пропси, дані з вищого рівня після цього в залежності від них повертають JSX. Ця особливість дозволяю дуже зручно писати код для веб-додатків.

Окрім цього також дає можливість використовувати модульні таблиці каскадних стилів або CSS, що зменшує можливі конфлікти стилів між собою. Полегшує їх створення для окремих фрагментів.

Один із принципів, а саме дроблення сайту на менші елементи дозволяє нам їх повторне використання у коді, що зменшує його дублювання, та полегшує створення структури всього додатку.

Ще багато переваг можливо знайти у цій бібліотеці у документацій розробників, що можна знайти нижче.

Разом з React використовують багато інших бібліотек які, надають змогу писати код ще простіше, наприклад, Redux, redux-react, react-router-dom, redux-form, redux-thunk, reselect та багато інших.

Механізм взаємодії React з сайтом.

Бібліотека реактив - це бібліотека яка завантажується в браузер разом з сайтом, для подальших маніпуляцій з ним. Нижче на малюнку 2.1 зображений життєвий цикл відтворення сайту за допомогою React в браузері.

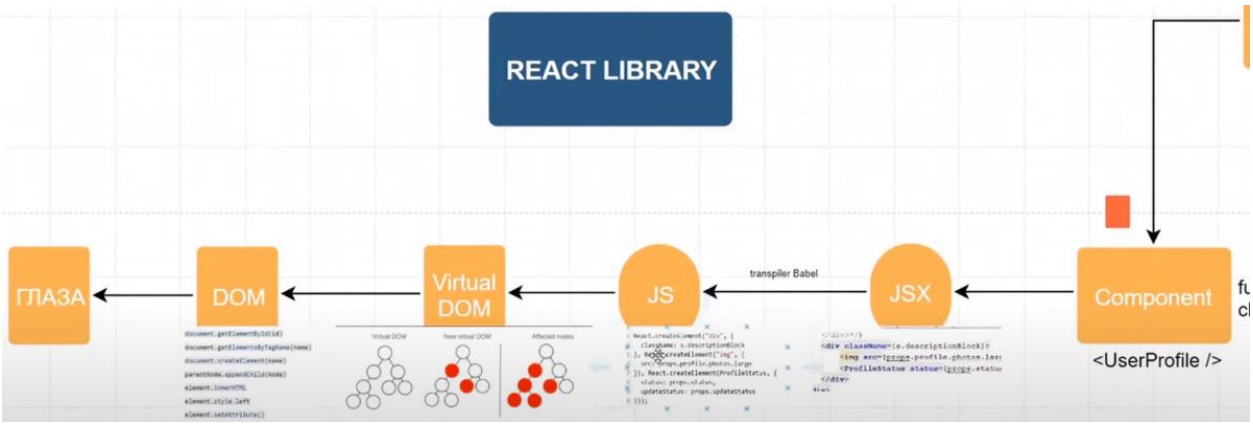


Рисунок 2.1 — Життєвий цикл відтворення сайту

Починаючи життєвий цикл з компоненти (рисунок 2.2) завдання якої компоненти повернути JSX, в залежності від даних в які в неї прийшли. Компоненти створюють дерево (рисунок 2.3), в якому з верхнього рівня передаються дані за деякими пропсам (атрибутам, параметрам).

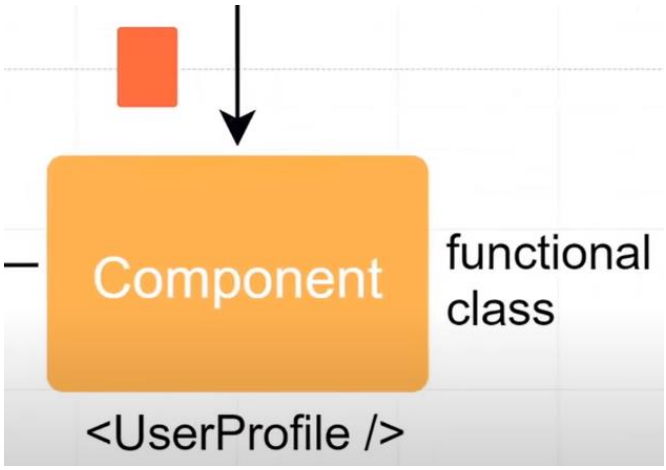


Рисунок 2.2 — Компонента

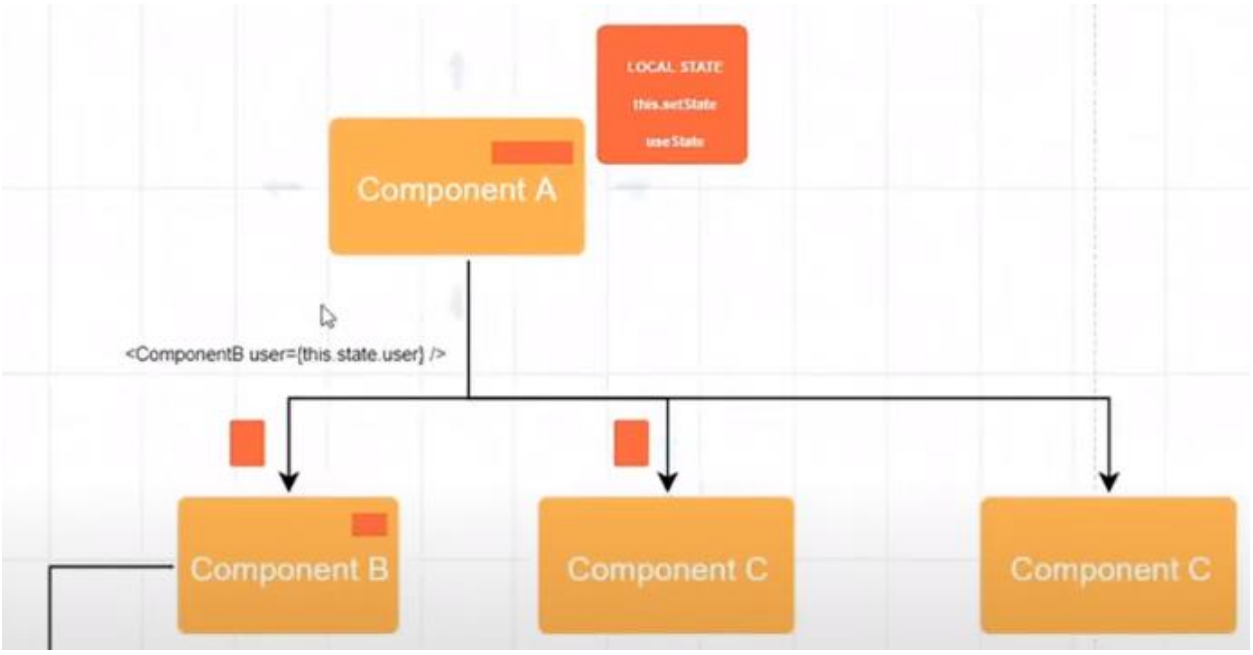


Рисунок 2.3 — Дерево компонент по якому передаються дані

JSX це розширення JS яке за допомогою транспілятора Babel перетворюватися в JS (рисунок 2.4).

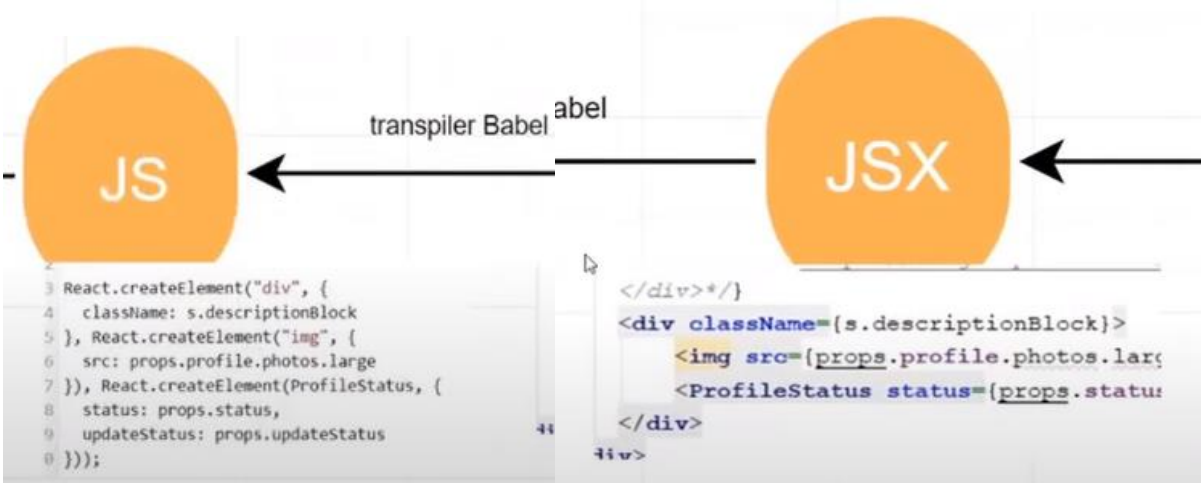


Рисунок 2.4 — Транспіляція JSX

Після чого реактив створює віртуальний будинок (рисунок 2.5), за рахунок якого відбувається оптимізація процесу відтворення сайту в браузері. Після створення віртуального будинку, реактив порівнює його з поточним будинком (рисунок 2.6) і перемальовує не весь сайт цілком, а тільки змінився компонент або компоненти.

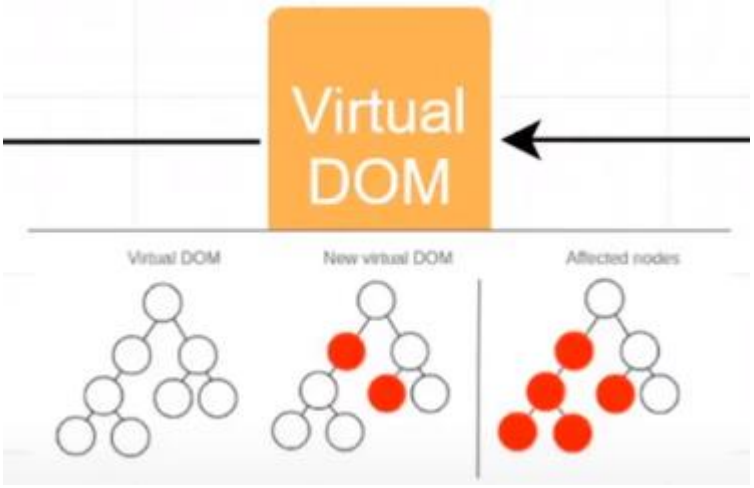


Рисунок 2.5 — Віртуальний будинок

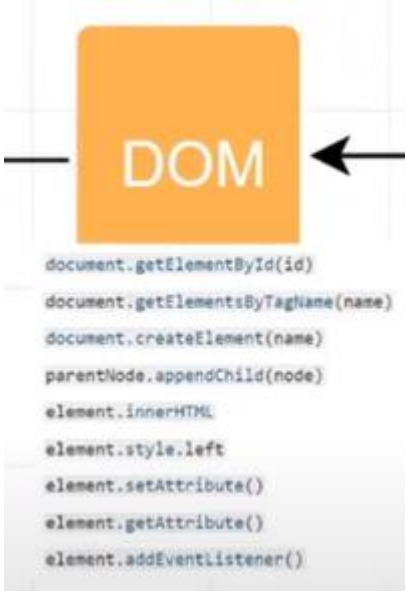


Рисунок 2.6 — Будинок

Решту інформацію по бібліотеці React можливо переглянути по посиланню [13] із списку літератури.

2.2.2.2 Redux

Іншою але не менш потрібною є бібліотека Redux. Її переваги використання можна знайти на головній сторінці їх сайту розробника вони такі:

Передбачуваність. Redux допомагає писати програми, які поводяться послідовно, працюють в різних середовищах (клієнтських, серверних та власних) і легкі для тестування.

Централізованість. Централізація стану та логіки вашого додатка забезпечує потужні можливості, такі як скасування/повтор, стійкість стану та багато іншого.

Налагодження. Redux DevTools дозволяє легко простежити, коли, де, чому та як змінився стан вашого додатка. Архітектура Redux дозволяє реєструвати зміни, використовувати «налагодження подорожей у часі» і навіть надсилати на сервер повні звіти про помилки.

Гнучкість. Redux працює з будь-яким шаром інтерфейсу та має велику екосистему аддонів, яка відповідає вашим потребам.

В цілому ця бібліотека полегшує взаємодію бізнес логіки та інтерфейсу користувача, зберігаючи дані користувача окрему від інтерфейсу.

Redux так само як і реактив це бібліотека яка виконує свою функцію. Функція Redux полягає в тому що він повинен зберігати локальні дані всього сайту, також він входить в BLL. Він складається з одного головного об'єкта store (сховище), в свою чергу store складається з:

- state - це об'єкт, який складається з двох частин;
- reducers - функції-перетворювачі, завдання якого отримати action (подія) - це об'єкт у якого як мінімум є тип, - а також reducers мають частина state і імьютабельно перетворюють state.

Також у store є 3 основних способи:

1. getState () - дозволяє отримати з store state.
2. subscribe (subscriber) - дозволяє subscriber (це функція яку ми передаємо всередину і коли state всередині store зміниться перемальовує все додаток з актуальним state) store підписатися на зміни state.
3. dispatch (action) - дозволяє змінити state з зовнішнього світу за допомогою action (інструкція по якій буде виконуватися дія з state).

Більше про цю бібліотеку можна прочитати по посиланню [12] із списка літератури.

2.2.2.3 React-Redux

Бібліотека React-Redux допомагає нам спростити взаємодію React з Redux. У неї є функція connect яка є hoc (higher order component або функція вищого порядку), вона приймає в себе одну компоненту і навколо неї створює контейнерну компоненту, для того щоб це компонента забезпечила нашу компоненту даними і функціями.

Також є функції mapStateToProps (mstp) і mapDispatchToProps (mdtp).

Завдання mstp взяти з усього state якусь частину призначити її ім'я і передати в пропси. Також завдання mstp повернути якийсь об'єкт з callbacks який може задіспатчіть подія.

Це все відбувається локально і перемальовуватись дерево не буде, у кожної контейнерної компонентний є своя функція subscribe яка засовує свою компоненту.

Функція коннект і контейнерна компонента бере дані з контексту (рисунок 2.7), а дані в контекст додаються за допомогою провайдера.

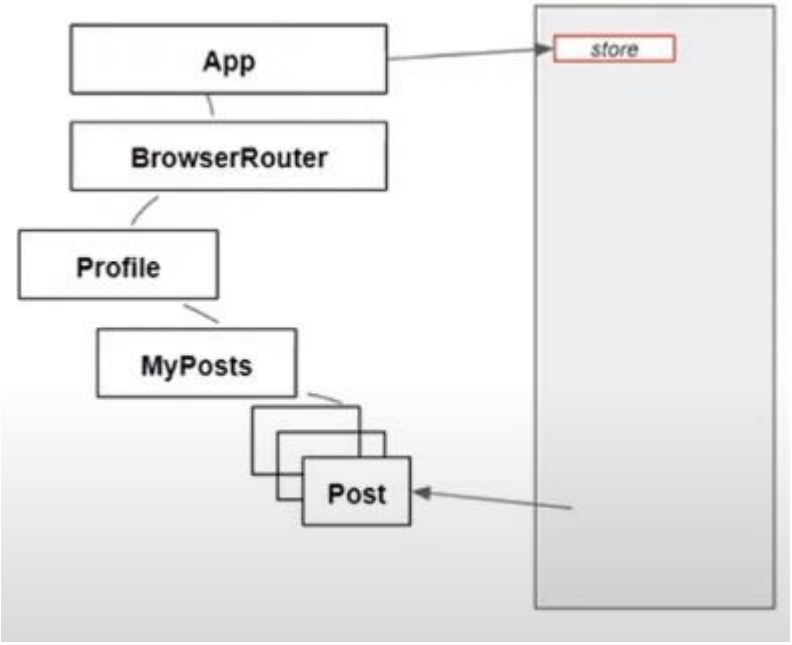


Рисунок 2.7 — Контекст

2.2.2.4 Selectors

Селектори це функції дозволяють зручно брати дані з бізнес логіки і передавати зручно їх для призначеного для користувача інтерфейсу не знаючи деталі структури state. І в цьому нам допомагає бібліотека Reselect.

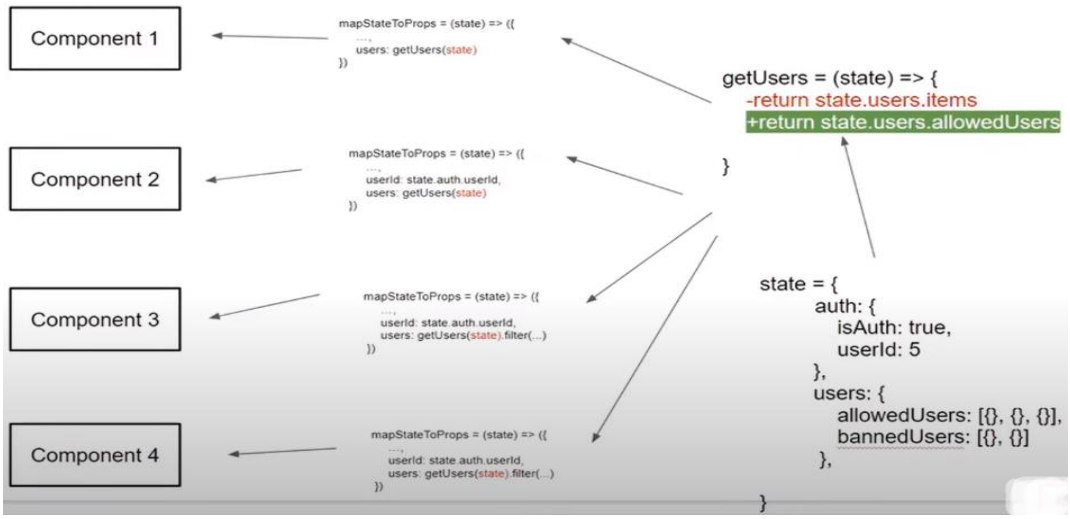


Рисунок 2.8 — Селектори

2.2.3 Бекенд

Для зберігання даних користувача та їх використання були використанні популярні застосунки такі як: express для написання rest api серверу, та mongodb для зберігання даних користувача.

2.2.3.1 Express

Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків.

Має в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів для швидкого і легкого створення надійний API.

Express надає тонкий шар фундаментальних функцій веб-додатків, які не заважають вам працювати з давно знайомими і улюбленими вами функціями Node.js.

У той час як сам express досить мінімалістичний, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення практично будь-якої проблеми з веб-розробкою. Існують бібліотеки для роботи з куки-файлами, сеансами, входами користувачів, параметрами URL, даними POST, заголовками безпеки і багатьма іншими.

Офіційний сайт де можна переглянути іншу документацію можливо по посиланню [14] із списка літератури.

2.2.3.1 Mongodb

MongoDB - це загальнодоступна розподілена база даних, що базується на документах, створена для сучасних розробників додатків та для епохи хмар. MongoDB - це база даних документів, що означає, що вона зберігає дані у документах, подібних до JSON.

У нашому проекті ми використовуємо формат JSON для зберігання даних і MongoDB має такі переваги роботи з ними:

1. Найбільш природний і продуктивний спосіб роботи з даними.
2. Підтримує масиви та вкладені об'єкти як значення.
3. Дозволяє створювати гнучкі та динамічні схеми.
4. Розширена та виразна мова запитів, що дозволяє фільтрувати та сортувати за будь-яким полем, незалежно від того, наскільки воно вкладено в документ.
5. Підтримка агрегувань та інших сучасних випадків використання, таких як географічний пошук, пошук у графіках та пошук тексту.
6. Запити самі по собі JSON і, таким чином, легко складаються. Більше немає об’єднання рядків для динамічного генерування запитів SQL.
7. Розподілені мультидокументальні транзакції ACID із ізоляцією знімків.
8. Підтримка об’єднань у запитах.
9. Два типи відносин замість одного: посилання та вбудовані.

Більше інформації можна переглянути на сторінці офіційного розробника по посиланню [15] із списка літератури.

3 Моделювання веб-сервісу

3.1 Інформаційна модель додатку

Наш додаток поділений на декілька частин. Перша частина це – UI або інтерфейс користувача, до нього надходять дані із бізнес логіки додатку і він відображає гіперрозмітку на сайті. Друга частина – це вище зазначена бізнес логіка у якій ми зберігаємо локальні дані користувача які надходять з серверу. Наступна частина – DAL або шар доступу до даних, він відповідає за зв'язок з сервером даних. Остання частина – це сам сервер який приймає запити, та дає відповіді на них з даними нашої бази. Зв'язок зображений на рисунку 3.1.

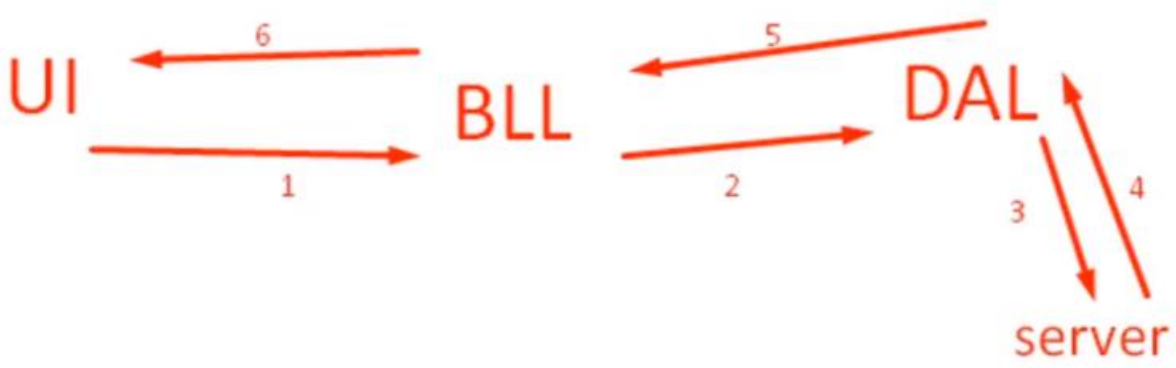


Рисунок 3.1 — Бізнес модель додатку

Схематичне зображення взаємодії об'єктів додатку можна побачити в додатку А. На ньому зображені компоненти та локальні дані (state.js) які далі розподіляються.

3.2 Опис коду

Насамперед у проекті використовується контроль версій і весь код можливо побачити по посиланню: https://github.com/ToshnoWor/learn-react та https://github.com/ToshnoWor/RestAPIServer. Написання коду програми було по етапне вивчення уроків [16] та більш розширене поглиблення у тему [1-15]. Основні фрагменти коду будуть представлені нижче у додатках.

Для початку інтерфейс користувача. Вже відомо, що сайт розбивається на дрібні частини, які мають назву компоненти приклад такої компоненти у додатку Б, ця компонента відповідає за зображення основний частин сайту:

- Голови сайту;
- Навігаційної панелі;
- Основної частини.

В основній частині сайту знаходиться маршрути за вдяки яким відображається потрібний контейнер який зазначений у адресній строчці. У React застосовується принцип чистих функцій, тому для отримання даних з бізнес логіки використовують контейнерні компоненти, приклад у додатку В.

Перейдемо до бізнес логіки тут головним файлом є redux-store.js у якому знаходиться комбінація reducers, та створення самого store з підключенням до нього middleware, код знаходиться у додатку Г. Далі BLL розбивається на reducers приклад якого у додатку Ґ.

Далі NAL на даний момент в ньому один файл у додатку Д.

В остання частина складається з основного файлу старту сервера(додаток Е), файлу аутентифікації(додаток Є), файлом зі схемою JSON файлу який буде зберігатися у базі даних(додаток Ж), файл з основними запитами(додаток З), файл з функціями-валідаторами даних(додаток И), файл з веріфікацією токена користувача(додаток І).

3.3 Інструкція користувача

Щоб потрапити на сайт потрібно перейти по посиланню: https://toshnowor.github.io/learn-react/#/.

Головна сторінка поки що порожня із-за того, що сайт ще в розробці зображена на рисунку 3.2.

Рисунок 3.2 — Головна сторінка сайту

Після завантаження сторінки потрібно авторизуватися або зареєструватись на сайті, це можливо зробити на сторінці після натиснення кнопки Login.

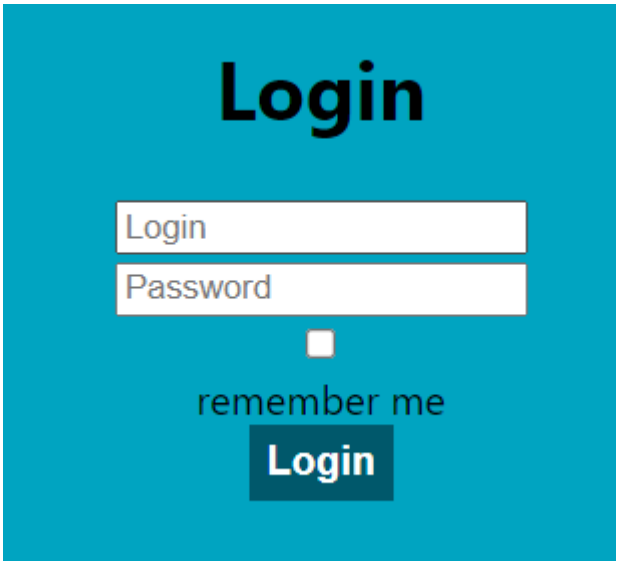


Рисунок 3.3 — Форма для авторизації

Після авторизації потрапляємо на нашу сторінку зображену на рисунку 3.4. На ній є наша фотографія, ім’я та прізвище, вік, пол, статус та пости. На ній можливо змінити всі дані об власнику та додати або видали пости.

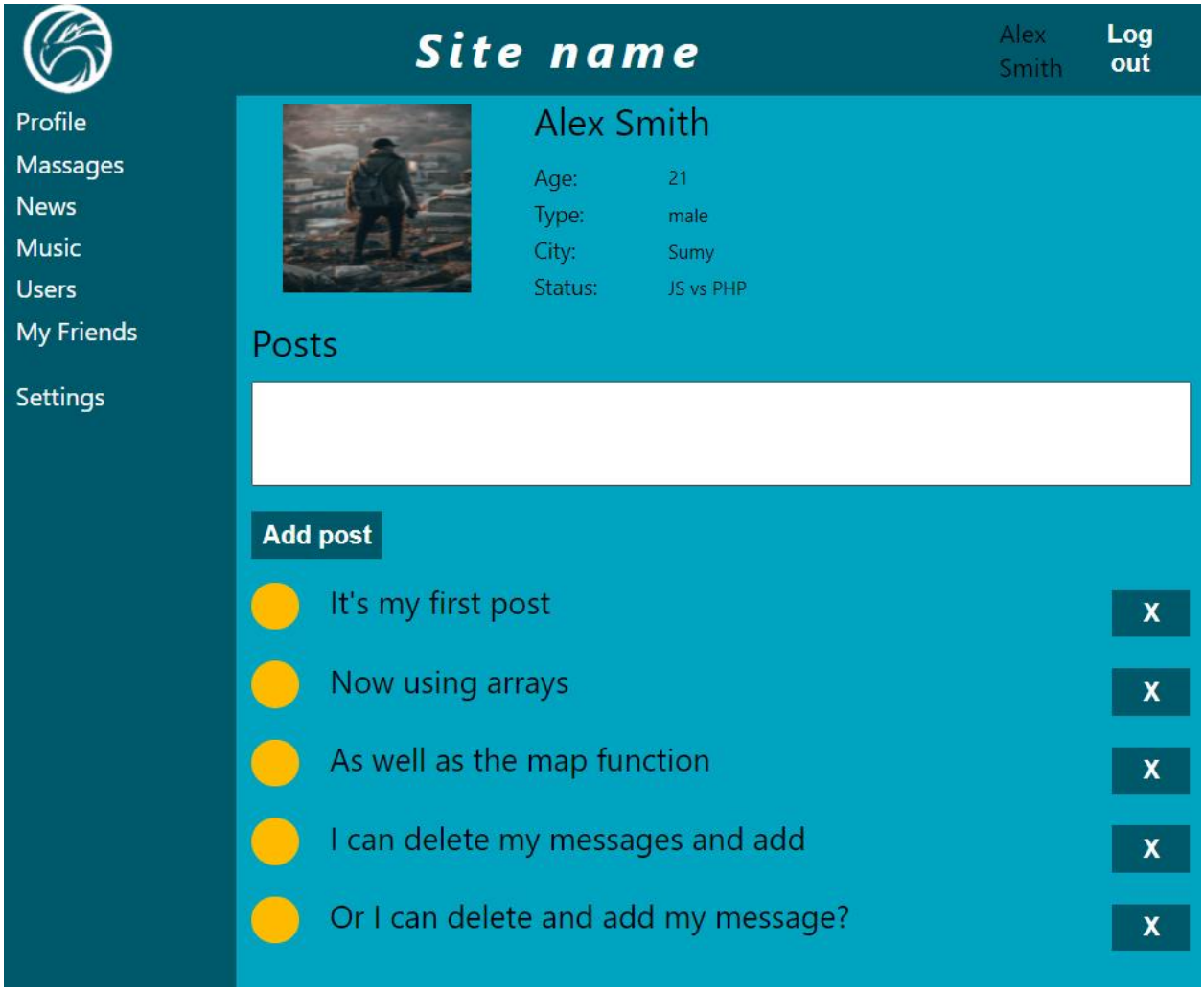


Рисунок 3.4 — Форма для авторизації

На сторінці користувачі можливо побачити всіх зареєстрованих користувачів та додати їх у список друзів також можливо і видалити.

Рисунок 3.5 — Сторінка користувачів

На сторінці мої друзі можливо переглянути ваших друзів, перейти на їх профіль та видалити за необхідністю.

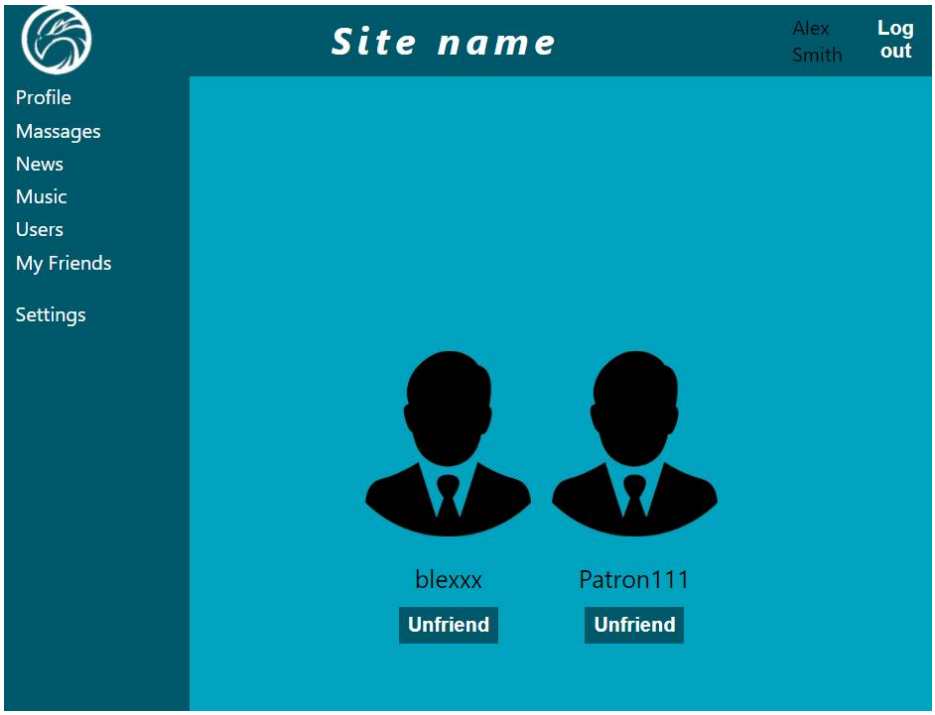


Рисунок 3.5 — Сторінка моїх друзів

ВИСНОВКИ

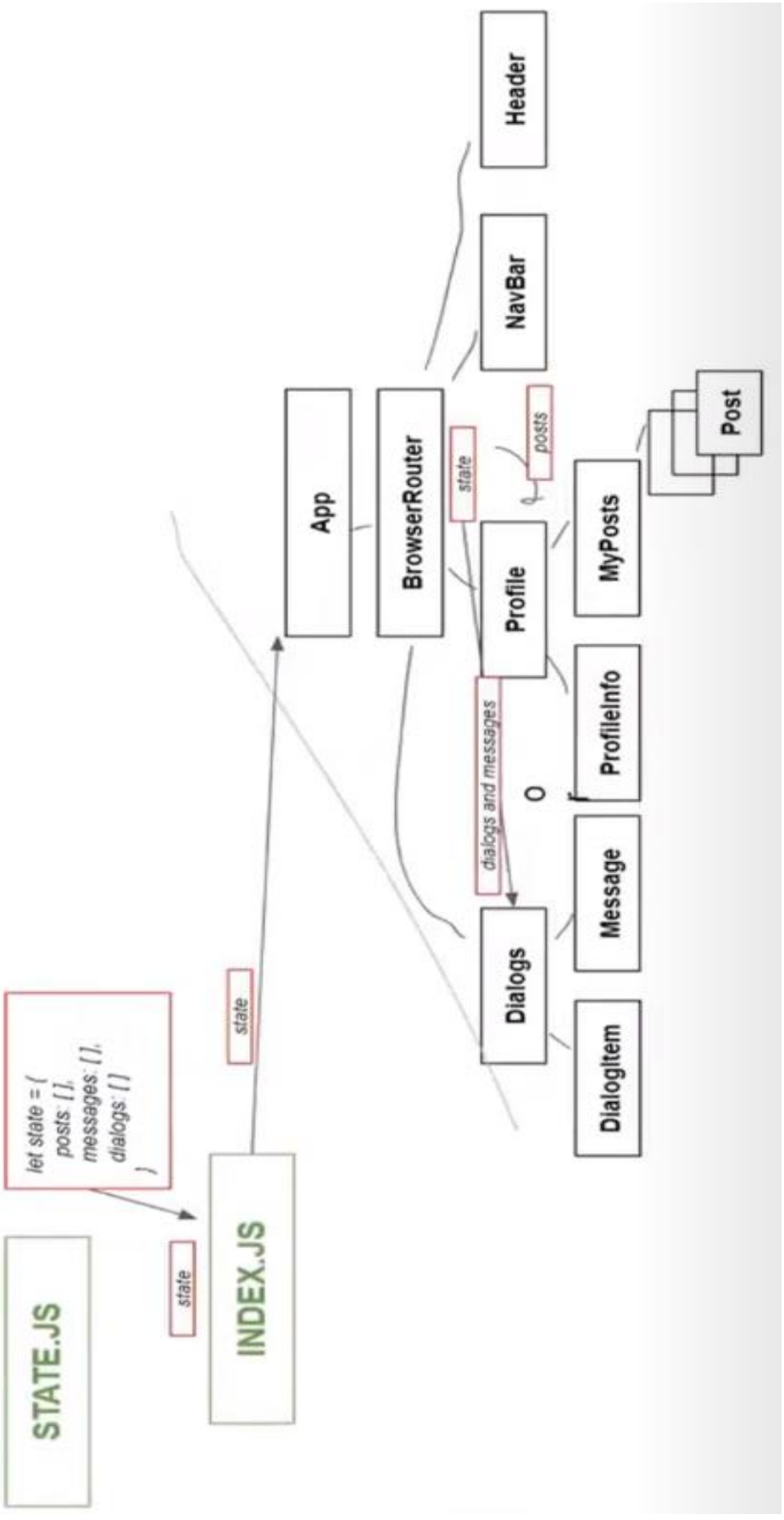
В ході виконання кваліфікаційної роботи бакалавра було проведено розробку та програмну реалізацію інформаційно-комунікаційної системи для організації соціальних взаємовідносин. При цьому було виконано такі завдання:

1. Обрати методологію проектування інформаційної системи
2. Виконати проектування серверної частини інформаційної системи
3. Виконати проектування клієнтської частини інформаційної системи
4. Визначити структуру бази даних інформаційної системи
5. Обрати програмне середовище для реалізації інформаційної системи
6. Програмно реалізувати такі модулі інформаційної системи:
    - модуль аутентифікації користувача

СПИСОК ЛІТЕРАТУРИ

1. React and React Native Paperback. – Adam Boduch, 2017 . – 540p.
2. Learn React with TypeScript 3: Beginner's guide to modern React web development with TypeScript . – Carl Rippon, 2018 . – 492p.
3. React Design Patterns and Best Practices: Build easy to scale modular applications using the most powerful components and design patterns. – Michele Bertoli, 2017. – 326p.
4. Learn React Hooks: Build and refactor modern React.js applications using Hooks. – Daniel Bugl, 2019 . – 426p.
5. MERN Quick Start Guide: Build web applications with MongoDB, Express.js, React, and Node. – Eddy Wilson Iriarte Koroliova, 2018 . – 536p.
6. Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js, 2nd Edition. – Shama Hoque, 2020. – 440p.
7. Redux in Action. – Marc Garreau, Will Faurot, 2018. – 329p.
8. Analyzing the Social Web. – Jennifer Golbeck, 2013. – 291p.
9. Building Social Web Applications. – Gavin Bell, 2009. – 448p.
10. Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub and More. – Matthew A. Russell, 2-е видання., 2013. – 448p.
11. Project Management: Concepts, Methodologies, Tools, and Applications. – Information Resources Management Association, 2016. – 2410p.
12. Документація офіційного розробника Redux – https://redux.js.org/
13. Документація офіційного розробника React – https://reactjs.org/
14. Документація офіційного розробника ExpressJS – https://expressjs.com/
15. Документація офіційного розробника MongoDB – https://www.mongodb.com/
16. Сайт з відео-уроками – https://www.youtube.com/playlist?list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8

ДОДАТОК А


STATE.JS
let state = {
posts: [],
messages: [],
dialogs: []
}
state
INDEX.JS
state
App
BrowserRouter
Header
NavBar
state
posts
Profile
MyPosts
Post
dialogs and messages
Dialogs
Message
ProfileInfo
DialogItem

ДОДАТОК Б

```
class App extends React.Component {
      componentDidMount() {
            this.props.initializeApp();
      }

      render() {
            if (!this.props.initialized)
                  return <Preloader />

            return (
                  <div className='app_wrapper'>
                        <HeaderContainer/>
                        <NavContainer/>
                        <div className='app_wrapper-content'>
                              <Route path='/profile/:userId?'
                                      render={() => <ProfileContainer/>}/>
                              <Route path='/dialogs'
                                      render={() => <DialogsContainer/>}/>
                              <Route path='/users'
                                      render={() => <UsersContainer/>}/>
                              <Route path='/friends'
                                      render={() => <FriendsContainer/>}/>
                              <Route path='/login'
                                      render={() => <LoginContainer/>}/>
                        </div>
                  </div>
            );
      }
}

const mapStateToProps = (state) => ({
      initialized: state.app.initialized
})

export default compose(
      withRouter,
      connect(mapStateToProps,{initializeApp}))(App);
```

ДОДАТОК В

```
class ProfileContainer extends React.Component{

    componentDidMount() {
        let userId = this.props.match.params.userId;
        if (!userId){
            userId = this.props.authId;
            if (!userId)
                this.props.history.push('/login');
        }
        if (userId !== null || userId !== 'undefined')
            this.props.getProfile(userId);
    }

    render(){
        return (
            <Profile
                profile={this.props.profile}
            />
        )
    }
}

let mapStateToProps = (state) => ({
    profile: state.profilePage.profile,
    authId: state.auth.userId,
    isAuth: state.auth.isAuth
});

export default compose(
    connect(mapStateToProps, {
        getProfile
    }),
    withRouter,
    //withAuthRedirect
)(ProfileContainer);
```

ДОДАТОК Г

```
let reducers = combineReducers({
    profilePage: profileReducer,
    messagesPage: messageReducer,
    sidebar: sidebarReducer,
    usersPage: usersReducer,
    auth: authReducer,
    friendsPage: friendsReducer,
    form: formReducer,
    app: appReducer
});

let store = createStore(reducers, applyMiddleware(thunk));

export default store;
```

ДОДАТОК Ґ

```
import {profileAPI} from "../api/api";
import {reset} from "redux-form";

const ADD_POST = 'ADD-POST';
const SET_USER_PROFILE = 'SET_USER_PROFILE';
const SET_USER_POST = 'SET_USER_POST';
const SAVE_STATUS = 'SAVE_STATUS';
const DELETE_POST = 'DELETE_POST';

let initialize = {
    profile: null,
    posts: null
};

const profileReducer = (state = initialize, action) => {
    switch (action.type) {
        case ADD_POST:
            return {
                ...state,
                posts: [...state.posts, action.newPost]
            };
        case SET_USER_PROFILE:
            return {
                ...state,
                profile: action.profile
            };
        case SET_USER_POST:
            return {
                ...state,
                posts: action.posts,
                nextId: action.posts ? action.posts.length : 0
            }
        case SAVE_STATUS:
        {
            return {
                ...state,
                profile: {...state.profile, status: action.status},
                newStatus: ''
            };
        }
        case DELETE_POST:
        {
            state.posts.splice(action.postId, 1);
            return {
                ...state,
                posts: [...state.posts]
            };
        }
        default :
            return state;
    }
}
```

```
export const createPost = (newPost) => ({type: ADD_POST, newPost})
export const setUserProfile = (profile) =>
    ({type: SET_USER_PROFILE, profile})
export const setUserPosts = (posts) =>({ type: SET_USER_POST, posts:
posts})
export const saveStatusSuccess = (status) => ({type: SAVE_STATUS, status})
export const deletePost = (postId) => ({type: DELETE_POST, postId})

export const getProfile = (userId) => {
    return (dispatch) => {
        if (userId)
            profileAPI.getProfile(userId).then(r => {
                dispatch(setUserProfile(r.data[0]));
                dispatch(setUserPosts(r.data[0].posts));
            });
        if (userId!==null) {
            dispatch(setUserProfile(null));
            dispatch(setUserPosts(null));
        }
    }
}

export const saveStatus = (auth, newStatus) => {
    return (dispatch) => {
        let data = null;
        if (auth.isAuth){
            data = {
                auth: auth,
                content: {
                    status: newStatus
                },
                config:{
                    headers: {
                        'auth-token': auth.token
                    }
                }
            }
            profileAPI.changeStatus(data)
                .then(status => {
                    if(status === 200)
                        dispatch(saveStatusSuccess(newStatus));
                });
        }
    }
}

export const addPost = (auth, post) => {
    return (dispatch) => {
        if (auth.isAuth){
            profileAPI.addPost({
                config: {
                    headers: {
                        'auth-token': auth.token
                    }
                },
                content: {
```

```
                    post
                }
            })
                .then(resultCode => {
                    if (resultCode === 0){
                        dispatch(createPost(post));
                        dispatch(reset('post'));
                    }
                });
        }
    }
}
export const removePost = (auth, postId) =>{
    return (dispatch) => {
        if (auth.isAuth){
            profileAPI.removePost({
                config: {
                    headers: {
                        'auth-token': auth.token
                    }
                },
                id: postId
            })
                .then(resultCode => {
                    if (resultCode === 0){
                        dispatch(deletePost(postId));
                    }
                })
        }
    }
}

export default profileReducer;
```

ДОДАТОК Д

```
import * as axios from "axios";

const instanceLogin = axios.create({
    withCredentials: true,
    baseURL: 'http://localhost:3033/api/'
});
const instanceNotLogin = axios.create({
    baseURL: 'http://localhost:3033/api/'
});

export const userAPI = {
    getUsers(currentPage = 1 , pageSize = 2) {
        return instanceLogin.get("profile/get?" +
            "_page=" + currentPage +
            "&_limit=" + pageSize).then(r => r.data);
    },
    follow (auth, id) {
        return instanceLogin.post("profile/follow/"+ id, {},
            {
                headers: {
                    'auth-token': auth ? auth.token : ''
                }
            }
        );
    },
    unfollow(auth, id) {
        return instanceLogin.delete("profile/unfollow/"+id,
            {
                headers: {
                    'auth-token': auth ? auth.token : ''
                }
            }
        )
    },
}

export const profileAPI = {
    changeStatus(data){
        return instanceLogin.put(
            "profile/status",
            data.content,
            data.config
        ).then(r => {
            return r.status;
        });
    },
    getProfile(userId) {
        return instanceNotLogin.get("profile/" + userId).then(r => {
            return  {
                data: r.data,
                status: r.status
            }});
    },
```

```
    addPost(data){
        return instanceLogin.post("profile/post",data.content,data.config)
            .then(r => {
                return r.data.resultCode;
            })
    },
    removePost(data){
        return instanceLogin.delete("profile/post/" + data.id, data.config)
            .then(r => {
                return r.data.resultCode;
            })
    }
}

export const authAPI = {
    auth(data) {
        return  instanceNotLogin.post("user/login", data);
    }
}
```

ДОДАТОК Е

```
const express = require('express');
const dotenv = require('dotenv');
const mongoose = require('mongoose');
const cors = require('cors');
const authRoute = require('./routes/auth');
const postsRoute = require('./routes/posts');
const parseurl = require('parseurl');
const session = require('express-session');

const app = express();
dotenv.config();
mongoose.connect(
    process.env.DB_CONNECT,
    { useNewUrlParser: true},
    () => console.log('connect to db!')
);
mongoose.set('useFindAndModify', true);
app.use(express.json());
app.use(cors({
    origin: 'http://localhost:3000',
    credentials: true
}));
app.set('trust proxy', 1);
app.use(
    session({
        name: 'localhostSiteCookieName',
        saveUninitialized: true,
        secret: process.env.SESSION_SECRET,
        resave: false,
        cookie:{
            maxAge: parseInt(process.env.SESSION_MAX_AGE),
            secure: true,
            sameSite: 'none'
        }
    })
);
app.use((req,res,next) => {
    if (!req.session.item) {
        req.session.item = {}
    }
    if (!req.session.views) {
        req.session.views = {}
    }
    var pathname = parseurl(req).pathname;
    req.session.views[pathname] = (req.session.views[pathname] || 0) + 1;
    next();
})
app.use('/api/user', authRoute);
app.use('/api/profile', postsRoute);

app.listen(3033, ()=> console.log('Server Up and running'));
```

ДОДАТОК Є

```
const router = require('express').Router();
const Item = require('../model/Item');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');
const { registerValidation, loginValidation } = require('../validation');

router.post('/register', async (req, res) => {

    const {error} = registerValidation(req.body);
    if (error)
        return res.status(400).send(error.details[0].message);

    const emailExist = await Item.findOne({ email: req.body.email});
    if(emailExist) return res.status(400).send("Email already exists.");

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(req.body.password, salt);

    const item = new Item({
        name: req.body.name,
        email: req.body.email,
        password: hashedPassword,
        photo: null,
        age: null,
        type: null,
        city: null,
        status: null,
        posts: [],
        followers: [],
    });
    try{
        const savedItem = await item.save();
        req.session.item = item;
        res.json({item: item._id});
    }catch(err){
        res.status(400).json(err);
    }
});

router.post('/login', async (req,res) => {
    const {email, password} = req.body;
    const {error} = loginValidation({email, password});
    if (error)
        return res.status(200).json({
            ressoltCode: 1,
            message: error.details[0].message,
            field: error.details[0].context.key
        });
    const item = await Item.findOne({ email });
    if(!item)
        return res.status(200).json({
            ressoltCode: 2,
            message:'Email is not found.',
```

```
            field: 'email'
        });
    const validPass = await bcrypt.compare(password, item.password);
    if (!validPass)
        return res.status(200).json({
            ressoltCode: 3,
            message:'Invalid password',
            field: 'password'
        });

    const token = jwt.sign({_id: item._id}, process.env.TOKEN_SECRET);

    var data = {
        id: item._id,
        login: item.name,
        email: item.email,
        token: token,
        ressoltCode: 0
    };

    req.session.item = data;
    res.status(200).json(data);
});

requireAuth = (req, res, next) =>{
    const { user } = req.session;
    console.log(user);
    if(!user){
        return res
        .status(401)
        .json({message: 'Unauthorized'});
    }
    next();
};

module.exports = router;
module.exports.requireAuth = requireAuth;
```

ДОДАТОК Ж

```
const mongoose = require('mongoose');
const mongoosePaginate = require('mongoose-paginate-v2');
const itemSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        max: 255,
        min: 3
    },
    email: {
        type: String,
        required: true,
        max: 255,
        min: 6
    },
    password: {
        type: String,
        required: true,
        max: 1024,
        min: 6
    },
    photo: {
        type: String
    },
    age: {
        type: Number
    },
    type: {
        type: String,
        max: 6,
        min: 4
    },
    city: {
        type: String,
        max: 255,
        min: 2
    },
    status: {
        type: String,
        max: 255,
        min: 0
    },
    posts: {
        type: [String]
    },
    followers: {
        type: [String]
    },
    dateCreate: {
        type: Date,
        default: Date.now
itemSchema.plugin(mongoosePaginate);
module.exports = mongoose.model('Item', itemSchema);
```

ДОДАТОК 3

```
const router = require('express').Router();
const Item = require('../model/Item');
const verify = require('./verifyToken');
const {profileAddValidation} = require('../validation');


router.put('/age', verify, async (req,res) => {
    if (!req.body.age)
    return res.status(400).send("Dont enter age!");
    filter = {_id: req.item._id};
    update = {age: req.body.age};
    await Item.updateOne(filter, update);
    res.status(200).send({
        message: "Age update",
        ressoultCode: 0
    });
});
router.put('/city', verify, async (req,res) => {
    if (!req.body.city)
    return res.status(400).send("Dont enter city!");
    filter = {_id: req.item._id};
    update = {city: req.body.city};
    await Item.updateOne(filter, update);
    res.status(200).send({
        message: "City update",
        ressoultCode: 0
    });
});
router.put('/photo', verify, async (req,res) => {
    if (!req.body.photo)
    return res.status(400).send("Dont enter photo!");
    filter = {_id: req.item._id};
    update = {photo: req.body.photo};
    await Item.updateOne(filter, update);
    res.status(200).send({
        message: "Photh update",
        ressoultCode: 0
    });
});
router.put('/type', verify, async (req,res) => {
    if (!req.body.type)
    return res.status(400).send("Dont enter type!");
    filter = {_id: req.item._id};
    update = {type: req.body.type};
    await Item.updateOne(filter, update);
    res.status(200).send({
        message: "Type update",
        ressoultCode: 0
    });
});
router.put('/status', verify, async (req,res) => {
    /* if (!req.body.status)
    return res.status(400).send("Dont enter status!"); */
```

```
    filter = {_id: req.item._id};
    update = {status: req.body.status};
    await Item.updateOne(filter, update);
    res.status(200).json({
        message: "Status update",
        ressoultCode: 0
    });
});
router.post('/update', verify, async (req,res) => {
    const {error} = profileAddValidation(req.body);
    if (error)
    return res.status(400).send(error.details[0].message);
    filter = {email: req.body.email};
    update = {
        name: req.body.name,
        photo: req.body.photo,
        age: req.body.age,
        type: req.body.type,
        city: req.body.city,
        status: req.body.status,
        posts: req.body.posts
    };

    res.status(200).send(await Item.updateOne(filter, update));
});
router.post('/post', verify, async (req,res) => {
    const filter = {_id: req.item._id};
    const user = await Item.findOne(filter);
    //res.send({user: user, filter: filter});
    const update = { posts: [...user.posts, req.body.post]};
    await Item.findOneAndUpdate(filter, update);
    res.status(200).json({
        message: "Post add",
        resultCode: 0
    });
});
router.delete('/post/:id', verify, async (req,res) => {
    var id = req.params.id;
    const filter = {_id: req.item._id};
    const user = await Item.findOne(filter);
    const remove = user.posts.splice(id, 1);
    await Item.findOneAndUpdate(filter, {posts: user.posts});
    res.status(200).json({
        message: `Remove "${remove}" post`,
        resultCode: 0
    });
});
router.get('/get', async (req,res) => {

    if ( req.query._page && req.query._limit) {

        Item.paginate({},{page: req.query._page, limit: req.query._limit},
            (err, rezz) => {
                res.send(rezz);
            });
    }
```

```
    else
        await Item.find({}, function(err, item){
            res.send({items: item});
        });
});
router.get('/:id', async(req, res)=>{
    var id = req.params.id;
    if (id) {
        Item.find({_id: id}, (err, rezz)=>{
            res.send(rezz);
        });
    }
});
router.delete('/unfollow/:id', verify, async (req, res) => {
    var id = req.params.id;
    var filter = {_id: req.item._id};
    //var filter = {_id: req.session.user.id};
    var item = await Item.findOne(filter);
    item.followers = item.followers.filter((value, index, arr)=>{
        return value != id;
    });
    await Item.findOneAndUpdate(filter, item);
    res.status(200).json({
        message: "Unfollowed",
        ressoltCode: 0
    });
});
router.post('/follow/:id', verify, async (req, res) => {
    var id = req.params.id;
    //console.log(req.session.user);
    var filter = {_id: req.item._id};
    var item = await Item.findOne(filter);
    item.followers.push(id);
    await Item.findOneAndUpdate(filter, item);
    res.status(200).json({
        message: "Followed",
        ressoltCode: 0
    });
});

requireAuth = (req, res, next) =>{
    const { user } = req.session;
    req.session.item = {};
    console.log(req.session);
    if(!user){
        return res
        .status(401)
        .json({message: 'Unauthorized'});
    }
    next();
}

module.exports = router;
```

ДОДАТОК И

```
const Joi = require('@hapi/joi');

const profileAddValidation = data => {
    const schema = {
        name: Joi.string().max(255).min(3),
        email: Joi.string().max(255).min(6).required().email(),
        photo: Joi.string(),
        age: Joi.number(),
        type: Joi.string().max(6).min(4),
        city: Joi.string().max(255).min(2),
        status: Joi.string().max(255).min(0),
        posts: Joi.array().items(Joi.string())
    };
    return Joi.validate(data, schema);
};

const registerValidation = data => {
    const schema = {
        name: Joi.string().min(6).required(),
        email: Joi.string().min(6).required().email(),
        password: Joi.string().min(6).required()
    };
    return Joi.validate(data, schema);
};

const loginValidation = data => {
    const schema = {
        email: Joi.string().min(6).required().email(),
        password: Joi.string().min(6).required()
    };
    return Joi.validate(data, schema);
};

module.exports.profileAddValidation = profileAddValidation;
module.exports.registerValidation = registerValidation;
module.exports.loginValidation = loginValidation;
```

ДОДАТОК І

```
const jwt = require('jsonwebtoken');

module.exports = function auth (req, res, next) {
    const token = req.header('auth-token');
    if(!token) return res.status(401).send('Access Denied!');

    try {
        const verified = jwt.verify(token, process.env.TOKEN_SECRET);
        req.item = verified;
        next();
    } catch (err) {
        res.status(400).send('Invalid Token!');
    }
}
```