МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система управління навчальною групою»

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

Керівник роботи **Кузіков Б.О.**

Студента групи ІН-63 **Криводуб О.Г.**

СУМИ 2020

РЕФЕРАТ

Записка: 72 стор., 25 рис., 3 табл., 3 додатки, 14 джерел.

Об’єкт дослідження — використання інформаційних систем для управління навчальною групою.

Мета роботи — розробка інформаційної системи управління навчальною групою, що скаладається з декількох незалежно функціонуючих компонентів.

Методи дослідження — об'єктно-орієнтоване проектування.

Результати — розроблено інформаційну систему для управління навчальною групою з використанням клієнт-серверної архітектури, в рамках якої було створено три незалежно функціонуючих компоненти, а саме: серверна частина системи, що працює за принципом API, мобільний додаток для операційної системи IOS та WEB сайт для взаємодії з серевером за допомогою інтернет браузера. Розроблені компоненти реалізовано з використанням мови програмування Swift та програмного середовища XCode.

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ НАВЧАЛЬНОЮ ГРУПОЮ, МИТТЕВИЙ ОБМІН ПОВІДОМЛЕННЯМИ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, APPLICATION PROGRAMMING INTERFACE, MODEL VIEW CONTROLLER.

ЗМІСТ

ВСТУП

На сьогоднішній день кількість інформації про навчальний процес постійно збільшується. Важливим є не тільки факт донесення цієї інформації до студентів, а й форма подачі, її доступність через проміжок часу та своєчасність отримання. Погана інформованість студента може призвести до суттєвого погіршення його академічної успішності та залученості в життя університету в цілому.

Для вирішення цієї проблеми викладачі та старости груп вимушені використовувати різні інформаційні ресурси. Кожен з цих ресурсів може добре вирішувати одну з проблем (своєчасність отримання, зручність форми подання, доступність інформації через проміжок часу), але не може ефективно вирішити їх разом. Рішенням стає комбінація з таких ресурсів, що в свою чергу призводить до роздробленості інформації та створення надлишкових облікових записів.

В зв’язку з актуальністю наведеної проблеми метою роботи є дослідження, аналіз та створення ефективної інформаційної системи управління навчальними групами, яка дозволить студентам своєчасно отримувати актуальну інформацію щодо навчального процесу.

Для досягнення поставленої мети сформульовано задачі роботи:

- провести аналіз та дослідження наявної ситуації, існуючих рішень та актуальних проблем процесу управління навчальної групи,
- виходячи з результатів дослідження сформувати вимоги до нової інформаційної системи,
- розглянути технології створення існуючих рішень та провести огляд літератури,
- визначити перелік технологій для створення інформаційної системи,
- спроектувати інформаційну систему згідно визначених вимог,
- реалізувати систему використовуючи визначені технології,
- провести аналіз роботи створеної інформаційної системи.

1 АНАЛІЗ ПРОБЛЕМИ. ДОСЛІДЖЕННЯ. ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблеми

Для того, щоб зрозуміти актуальність проблем пов’язаних з управлінням навчальною групою, необхідно провести аналіз та дослідження, під час яких:

- встановити сутність процесу управління в контексті роботи з навчальною групою,
- виділити основні суб’єкти що приймають участь в процесі управління,
- визначити ступінь значущості цих суб’єктів та роль, яку вони відіграють,
- визначити перелік методів та механізмів управління,
- провести огляд інформаційних систем, які використовуються в якості механізмів управління,
- виявити основні проблеми існуючих інформаційних систем в процесі управління шляхом проведення опитування серед значущих суб’єктів процесу.

Процес управління - діяльність об'єднаних у визначену систему суб'єктів управління, спрямована на досягнення цілей шляхом реалізації певних функцій з використанням методів управління [14]. В контексті роботи з навчальною групою, визначеною системою взаємодії суб'єктів управління є навчальний процес, методом управління є обмін інформацією, а ціллю управління є створення чітких умов для успішної взаємодії між учасниками навчального процесу.

Серед основних суб’єктів процесу управління навчальною групою можна виділити:

- деканат,
- кафедра,

- викладач,
- староста,
- студент.

Процес обміну інформацією між суб’єктами навчального процесу може значно відрізнятися один від одного в залежності від суб’єктів, що взаємодіють. В навчальному процесі основними споживачами інформації організаційного характеру є суб’єкти типу староста та студент. Ступінь їхньої інформованості є головним фактором впливу на успішність організації всього навчального процесу, тому саме вони мають найвищу значущість. Взаємодія між старостами груп та викладачами зазвичай є організованою та своєчасною, взаємодія між старостами груп та студентами - навпаки, є найменш організованою та потребує детального розгляду. Основні ролі значущих суб’єктів системи вказані в таб. 1.1. **1.11.11.11.11.1**

Таблиця 1.1 Основні ролі в системі

суб’єкт системи	**роль**
староста групи	автор
студент	споживач

Основним методом управління в навчальному процесі є створення та розміщення оголошення. Для цього автори оголошень використовують різні інформаційні системи, саме вони і є основним механізмом управління.

Серед студентів та старост груп найбільшою популярністю користуються інформаційні системи для миттевого обміну повідомленнями (Telegram та Viber), які мають однаковий метод організації повідомлень у вигляді чату. Такий метод організації має свої переваги та недоліки.

Для виявлення основних проблем та недоліків роботи з інформаційними системами Telegram та Viber було проведено опитування, в якому взяли участь старости та студенти з різних груп факультету “Електроніки та інформаційних технологій” Сумського державного університету.

Бланк опитування зображено на рис. 1.1.

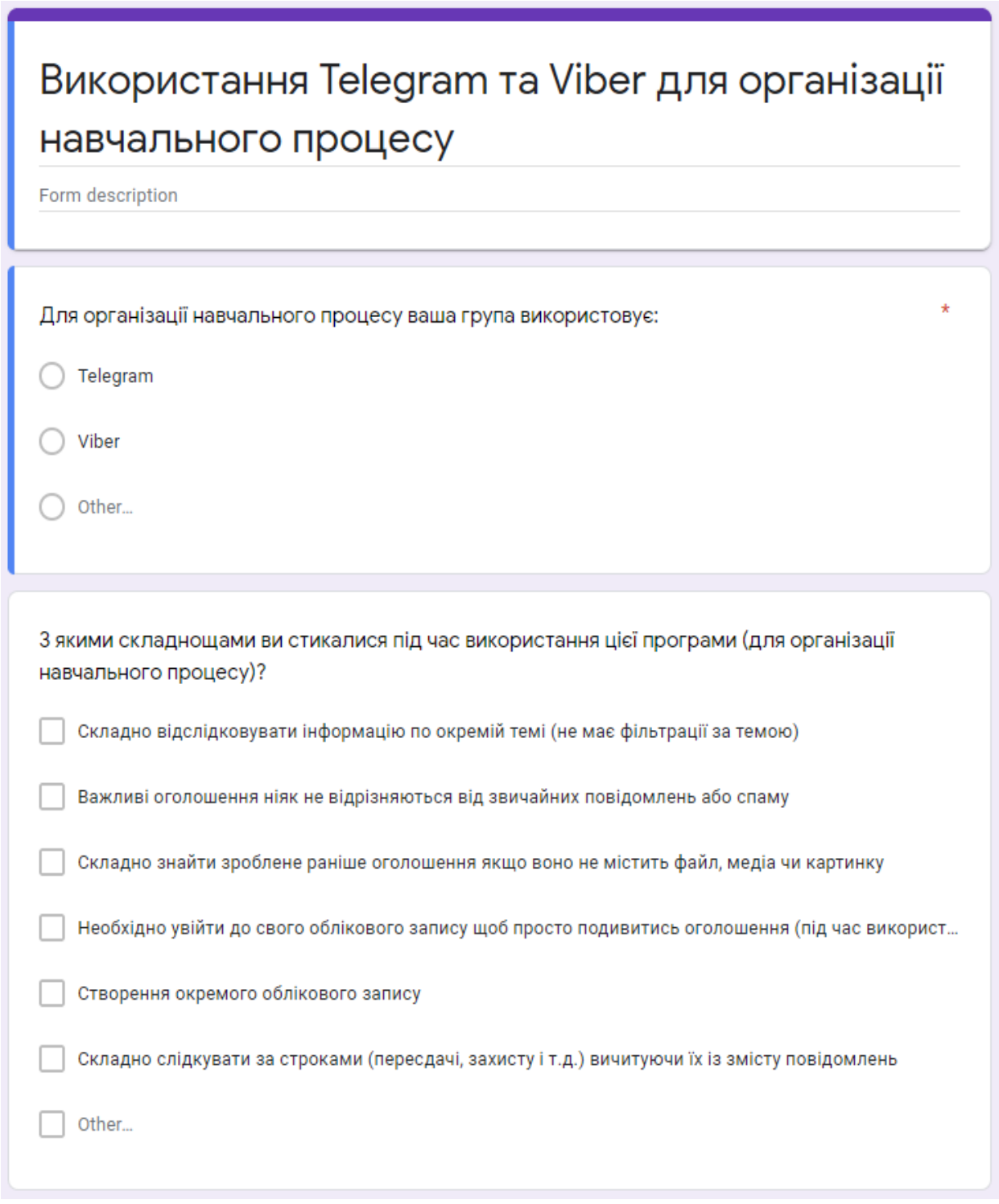


Рисунок 1.1 – Форма опитування

Результати представлені у вигляді діаграм та зображені на рис. 1.2 та 1.3.

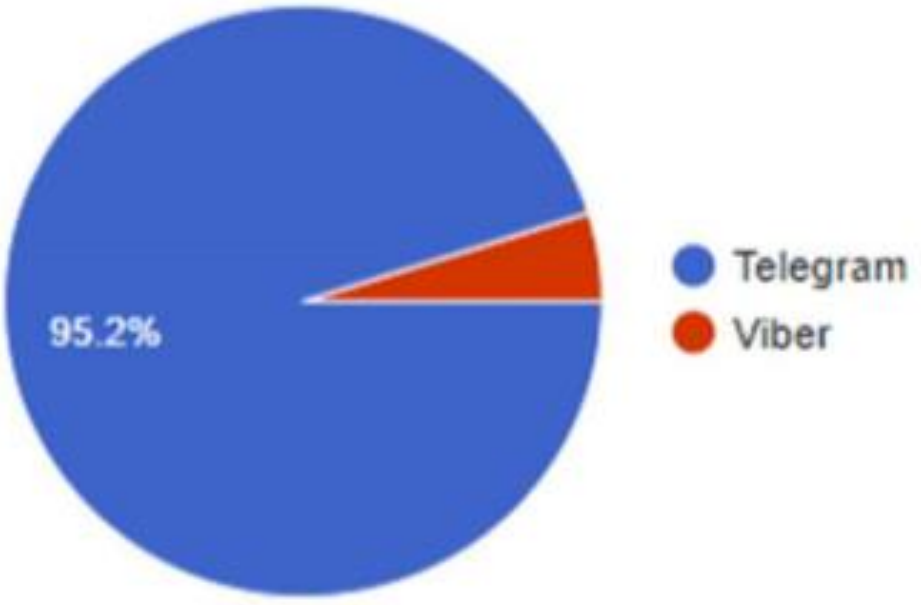


Рисунок 1.2 – Діаграма використання Telegram та Viber

Рисунок 1.3 – Діаграма основних проблем використання

Опитування було створено за допомогою сервісу Google Forms, та розміщено за посиланням https://forms.gle/HResUJmTEAzc5v3J9.

За результатами опитування було сформовано перелік основних проблем, які вказані в таб. 1.2 за ступенем пріорітетності.

Таблиця 1.2 Основні проблеми Telegram та Viber за пріорітетністю

Пріорітетность	Проблема	Відсоток
1	Складно відслідковувати інформацію по окремій темі (відсутня фільтрація за темою)	72.1%
2	Важливі оголошення ніяк не відрізняються від звичайних повідомлень або спаму	60.7%
3	Складно слідкувати за строками (перездачі, захисту і т.д.), вичитуючи їх із змісту повідомлень	52.5%
4	Складно знайти зроблене раніше оголошення, якщо воно не містить файл, медіа чи картинку	45.9%
5	Необхідно увійти до свого облікового запису, щоб просто подивитись оголошення (під час використання веб версії додатку)	18%
6	Створення окремого облікового запису	4.9%

Підсумовуючи результати проведеного опитування можна сказати, що основним недоліком розглянутих інформаційних систем є відсутність категоризації повідомлень, метод їх організації у вигляді чату та відсутність функціоналу для зручної роботи зі строками.

1.2 Формування вимог до інформаційної системи

Для того, щоб ефективно вирішувати актуальні проблеми управління навчальною групою, окрім основного функціоналу який мають розглянуті інформаційні системи, нова повинна:

- підтримувати категоризацію створених оголошень,
- мати розклад та підтримувати створення, редагування та видалення подій в ньому,

- підтримувати посилання на оголошення з будь-якої події,
- дозволяти переглядати оголошення та події без входу або реєстрації в системі,
- підтримувати швидкий вхід та реєстрацію з використанням облікового запису Google.

1.3 Аналіз архітектури та технологій створення існуючих систем

Переважна більшість таких інформаційних систем використовують підхід з клієнт-серверною архітектурою, в якій клієнтами є мобільні або Web додатки, а сервером є система, що працює за принципом API. API (Application Programming Interface) – це набір визначених методів та правил для взаємодії різних компонентів системи. Використання такого підходу дозволяє розробникам цих систем створювати різні їх частини незалежно одна від одної, що в свою чергу спрощує їх реалізацію, масштабування та подальшу підтримку. Всі елементи цих систем проходять регулярні тестування.

Серверні частини більшості систем миттевого обміну повідомленнями побудовані за допомогою фреймворків, написаних на мові програмування Java. Це дозволяє їм працювати з будь-якими операційними системами.

Клієнтські мобільні додатки для операційних систем IOS та Android створені за допомогою бібліотек та фреймворків, що використовують мови програмування Swift та Java відповідно. Такі фреймворки є самостійними або вже вбудованими в потужні редактори коду XCode та Android Studio. Кожен з цих редакторів надає розробникам широкий спектр готових інструментів та вбудованих функцій для ефективного створення мобільних додатків.

В якості систем управління реляційних або не реляційних баз даних найчастіше використовуються PostgreQSL або Firebase. Окрім широкого спектру можливостей вони мають зручний графічний інтерфейс для взаємодії з даними та їх подальшого аналізу.

1.4 Огляд літератури на тему створення подібних систем

Протягом всього свого існування модель клієнт-серверної архітектури не зазнала суттевих змін. Процес її розвитку був і залишається еволюційним. За час такої еволюції з’явилося багато нових рішень, що дозволили використовувати модель клієнт-сервер для рішення нових задач. Після появи поняття API та архітектурного стилю REST стало можливим використовувати цю модель для побудови швидких, надійних та легко масштабованих інформаційних систем. REST (Representational State Transfer) — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів [9].

За останні роки популярність створення API на основі REST зросла неймовірно [11] – каже Leonard Richardson в своїй роботі “RESTful Web APIs: Services for a Changing World”. Що призвело до значного полегшення процесів інтеграції різних систем між собою та підвищення рівню їх надійності.

Ще одним дуже важливим питанням постає вибір правильної моделі тестування. Її вплив на процес створення інформаційних систем важко переоцінити [6,12] – таку думку висловлюють автори літератури з цієї теми. Однак їхні точки зору розділяються стосовно об’єму та пріорітетності тестування в процесі розробки. Вони можуть варіюватися від використання керованої тестами розробки (test driven development) до ручного тестування в залежності від складності інформаційної системи, її архітектури, методології розробки, кількості доступних ресурсів, тощо.

Одним з варіантів є написання автотестів які покривають ключовий з точки зору користувача функціонал [1]. Таку думку висловлює Tim Condon в своїй роботі “Server side Swift with Vapor”, де головну роль в розглянутих ним інформаційних системах відіграють серверні їх частини. Саме для їх тестування автор рекомендує використовувати автотести, які дозволяють

суттєво скоротити час подальшого вдосконалення цих систем на кожному з етапів.

Важливим питанням також є правильний вибір паттерну проектування. В роботах на цю тему розглядаються не тільки переваги чи недоліки тих чи інших паттернів, а ще й підходи до їх реалізації та наслідки, які можуть мати розробники після вибору одного з них.

Зі слів Joshua Green найбільшого поширення здобув паттерн MVC (Model View Controller). Завдяки своїй простоті для розуміння та простоті реалізації паттерн MVC може слугувати відправною точкою для створення багатьох додатків [4]. Таку думку він висловлює в своїй книзі “Design Patterns by Tutorials”.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Визначення архітектури та технологій створення

Для реалізації інформаційної системи на основі клієнт-серверної архітектури в якості технології створення API буде використано фреймворк Vapor. Vapor – це фреймворк з відкритим кодом, написаний на мові програмування Swift. З його допомогою можна створювати API та Web додатки будь-якої складності. Створені за допомогою Vapor додатки можуть працювати на операційних системах MacOS та Linux. Завдяки підтримці великої кількості вбудованих компонентів (інших фреймворків) Vapor має такі можливості:

- асинхронна обробка запитів (swift-nio),
- робота з електронною поштою (SendGrid),
- валідація (Validation),
- авторизація (Auth),
- авторизація за допомогою Google та GitHub (Imperial),
- робота з базами даних за допомогою однієї моделі (Fluent),
- генерація динамічних веб сторінок (Leaf),
- кодування та розкодування JSON (Core),
- хешування паролів та генерація токену (Crypto).

Головною перевагою Vapor є його швидкодія в порівнянні з іншими фреймворками. Результати тестів в Benchmarker на рис. 2.1.

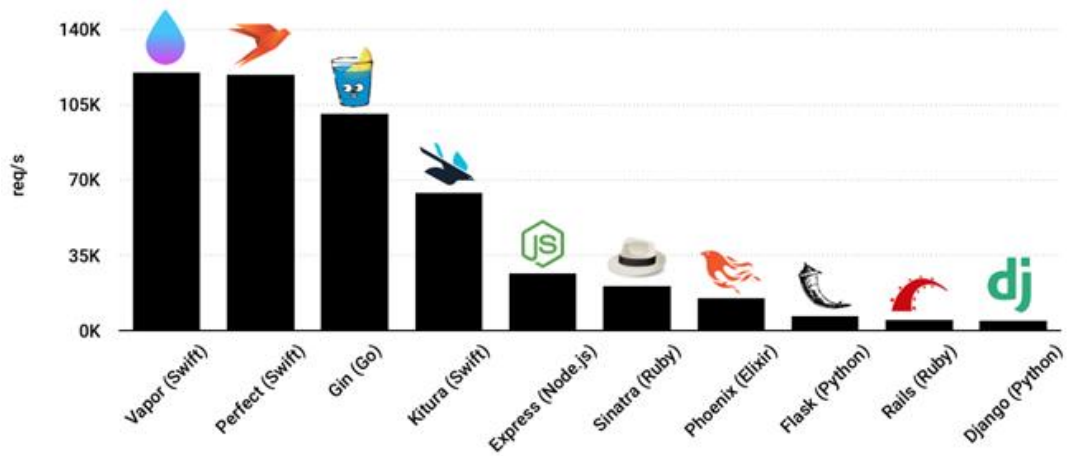


Рисунок 2.1 – Кількість запитів за секунду

В якості системи управління базою даних буде використана PostgreSQL. PostgreSQL – це відкрита, об’єкно-реляційна система управління базою даних (СУБД), що забеспечує високу швидкодію та надійність роботи с даними. Фреймворк Leaf має повну підтримку цієї СУБД, що дозволяє звертатись да бази даних не використовуючи специфічний синтаксис PostgreSQL. База даних інформаційної системи з назвою «postgres» буде працювати в контейнері Docker та буде доступна через порт 5432. Тестова її версія з назвою «postgres-test» буде доступна через порт 5433.

Розробка Web додатку буде здійснюватися за допомогою фреймворку Leaf. Leaf – це потужний вбудований фреймворк, який дозволяє генерувати динамічні HTML сторінки. Головною перевагою Leaf є майже ідентичний з мовою Swift синтаксис, що дозволяє розробникам API швидко створювати Web додатки.

Розробка мобільного додатку для операційної системи IOS буде здійснюватись за допомогою редактору XCode з використанням мови програмування Swift. Swift – це сучасна, жорстко типізована мова програмування з відкритим кодом, яка підтримує функціональну та об’єктно орієнтовану парадигму програмування. Вона дозволяє створювати швидкі та безпечні додатки для операційних систем IOS, IPadOS, WatchOS та TVOS.

2.2 Проектування посилань для роботи з API

Для зручної роботи клієнтських додатків з серверною частиною системи необхідно визначити чіткий набір правил (посилань та методів) для роботи. Перелік всіх посилань знаходиться в таб. 2.1.

Таблиця 2.1 Посилання для роботи з API

Посилання	**Метод**	**Дія**
/api/users	GET	користувачі
/api/users/id	GET	користувач за id
/api/posts	GET	всі оголошення

/api/posts/id	GET	оголошення за id
/api/events	GET	події
/api/events/id	GET	подія за id
/api/categories	GET	категорії
/api/categories/id	GET	категорія за id
/api/users/login	POST	отримати токен користувача
/api/users	POST	додати користувача
/api/posts	POST	додати оголошення
/api/events	POST	додати подію
/api/categories	POST	додати категорію
/api/posts/id/category/id	POST	додати категорію до оголошення
/api/posts/id	PUT	змінити оголошення
/api/posts/id	DELETE	видплити оголошення
/api/posts/sorted	GET	відсортовані за назвою оголошення
/api/posts/first	GET	перше оголошення
/api/posts?term=	GET	пошук по назві та змісту оголошення
/api/events/id	PUT	змінити подію
/api/events/id	DELETE	видплити подію
/api/events/sorted	GET	відсортовані за назвою події
/api/events/first	GET	перша подія
/api/events?term=	GET	пошук по назві та змісту події

Використовуючі різні посилання та методи передачі даних по HTTP протоколу клієнт (мобільний та Web додаток) зможе передавати дані та команди на сервер.

2.3 Проектування інтерфейсу мобільного та Web додатків.

Прототип інтерфейсу Web додатку зображено на рис. 2.2.

Рисунок 2.2 - Прототип сторінок Web додатку

Протитип інтерфейсу мобільного додатку зображено на рис. 2.3.

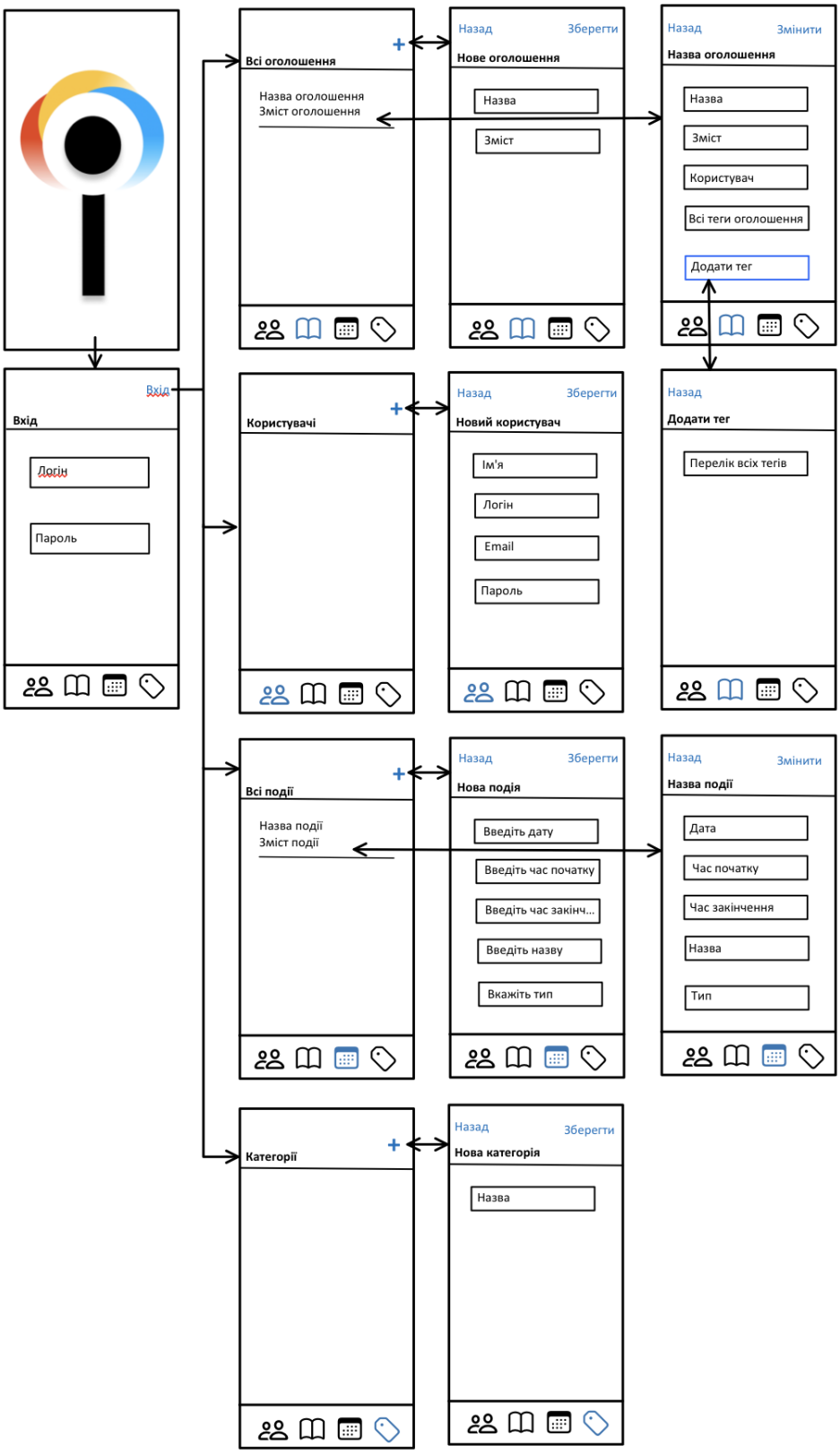


Рисунок 2.3 - Прототип сторінок мобільного додатку

Протитипи Web та мобільного додатку створені за допомогою програми Microsoft OneNote.

2.4 Проектування моделі даних

Для проектування моделі даних була використана утиліта Case Studio. Модель зображено на рис 2.4.

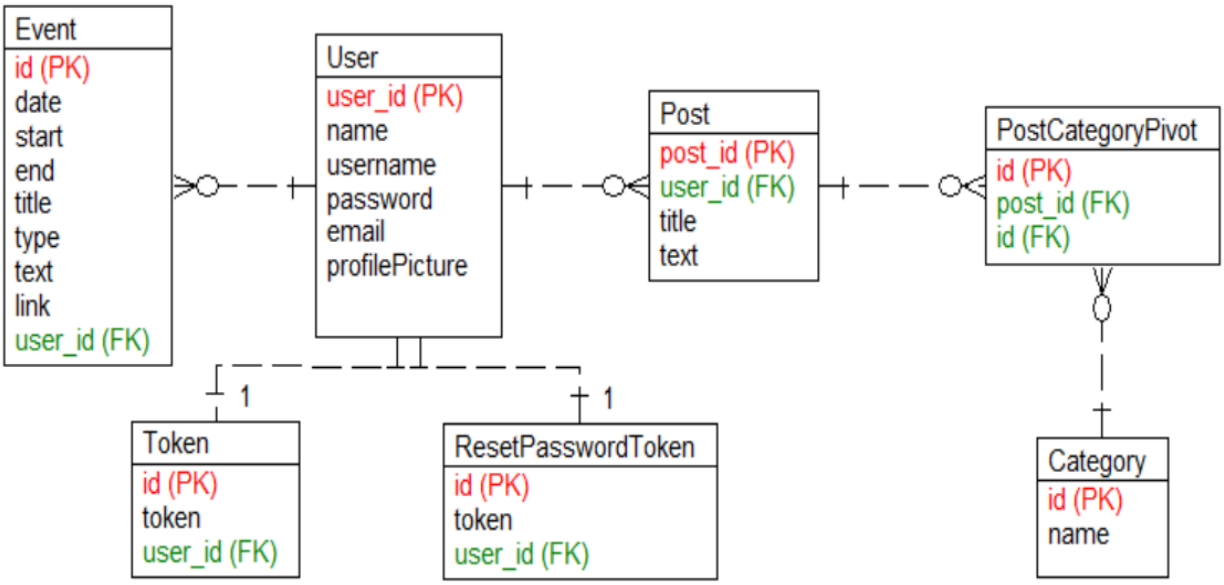


Рисунок 2.4 – ERD діаграма

Для роботи з базою даних необхідно встановити утиліту Docker, за допомогою якої створити два стандартних контейнери з встановленими на них PostgreSQL.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РОБОТИ

3.1 Створення API

Проект API складається з:

- файлів запуску та конфігурації (App),
- файлів моделей даних (Models),
- файлів управління даними (Controllers),
- файлів з автотестами (Tests).

Вся файлова структура проекту API зображена на рис. 3.1.

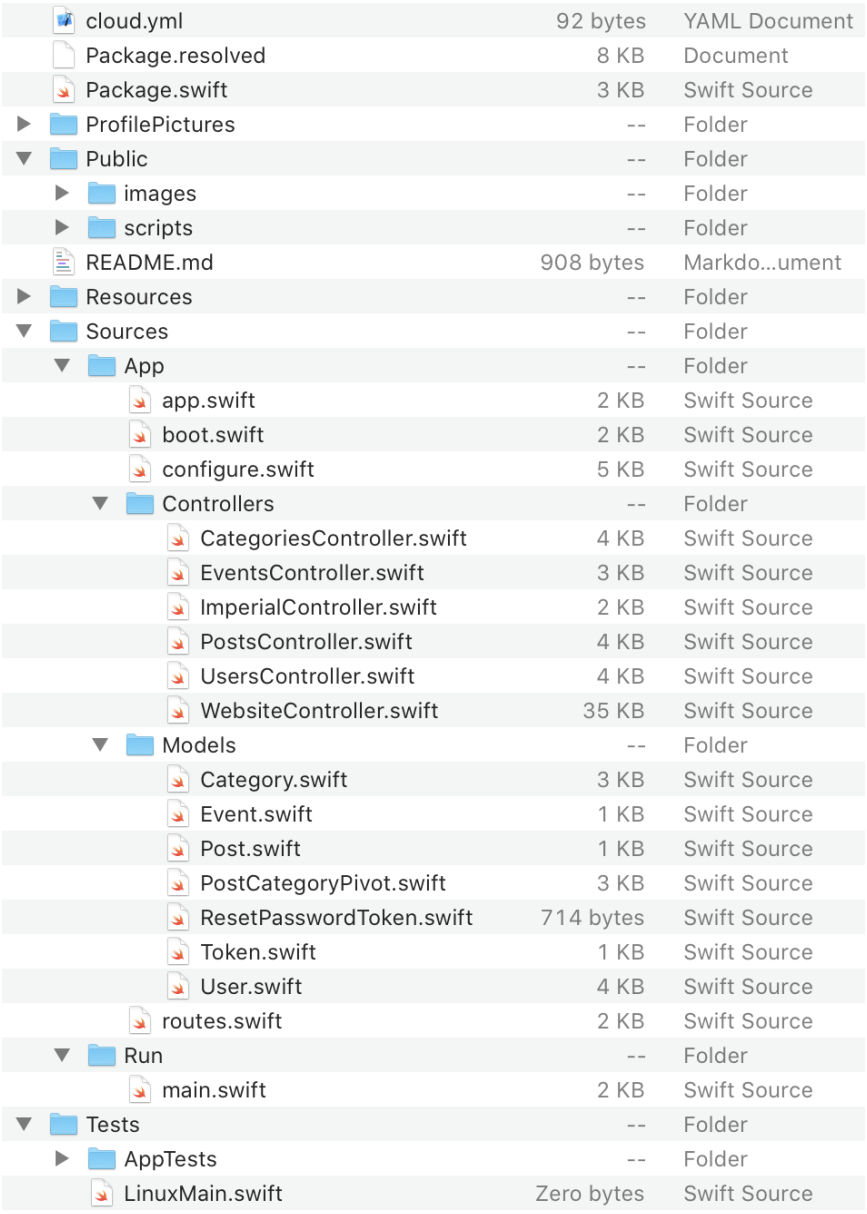


Рисунок 3.1 – Структура файлів API

Код основних файлів проекту створення API розміщено в додатку А.

3.2 Створення мобільного додатку

Проект мобільного додатку складається з:

- файлів інтерфейсу (Base),
- файлів моделей даних (Models),
- файлів управління даними (ViewControllers),
- файлів конфігурації додатку (Utilities),
- допоміжних файлів (Assets).

Вся файлова структура мобільного додатку зображена на рис. 3.2.

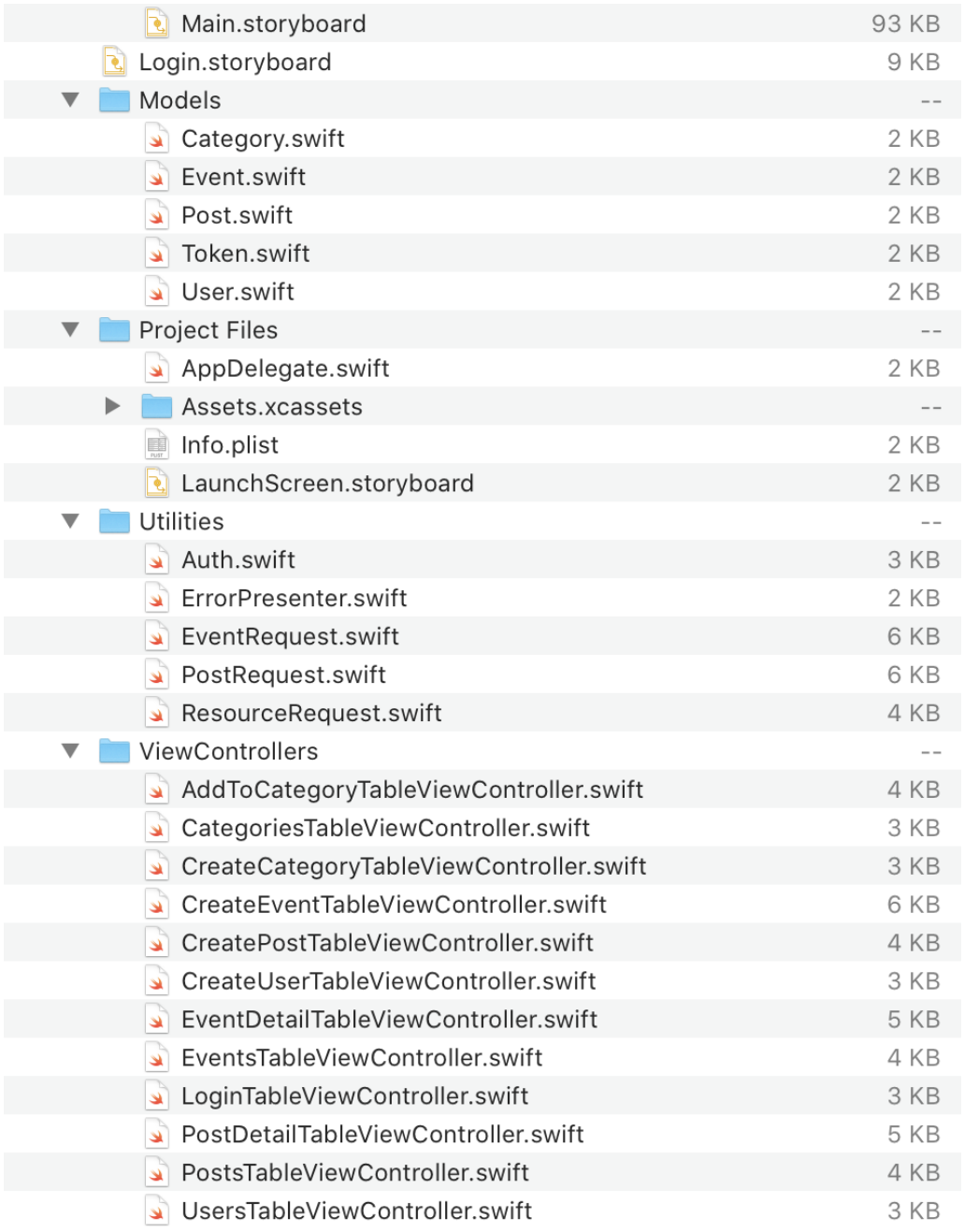


Рисунок 3.2 – Структура файлів мобільного додатку

Конструктор інтерфейсу мобільного додатку зображений на рис. 3.3.

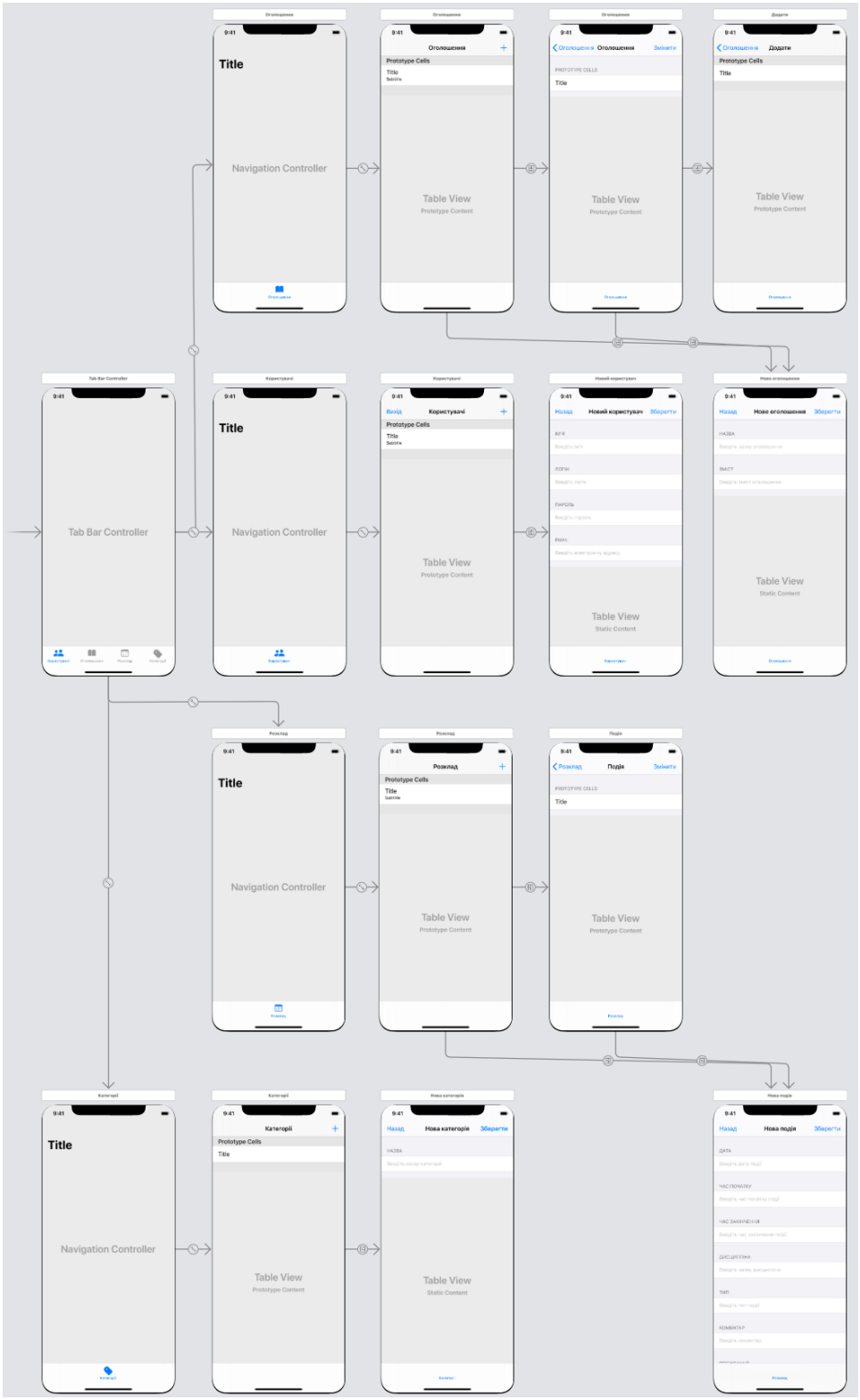


Рисунок 3.3 – Структура інтерфейсу мобільного додатку

Конструктор вбудований в середовище розробки XCode операційної системи MacOS. Код основних файлів проекту створення мобільного додатку розміщено в додатку Б.

3.3 Створення бази даних та Web сторінок

Для роботи з базою даних необхідно встановити утиліту Docker та з її допомогою створити два стандартних контейнери для роботи з PostgreSQL.

Команда для створення контейнеру:

```
docker run --name postgres -e POSTGRES_DB=vapor -e
POSTGRES_USER=vapor -e POSTGRES_PASSWORD=password -d -p
5432:5432 postgres:latest
```

Команда для створення тестового контейнеру:

```
docker run --name postgres-test -e POSTGRES_DB=vapor-test -e
POSTGRES_USER=vapor -e POSTGRES_PASSWORD=password -d -p
5433:5432 postgres:latest
```

Код основних файлів проекту Web додатку розміщено в додатку В. Файли Web сторінок мають роширення .leaf та зображені на рис. 3.4.

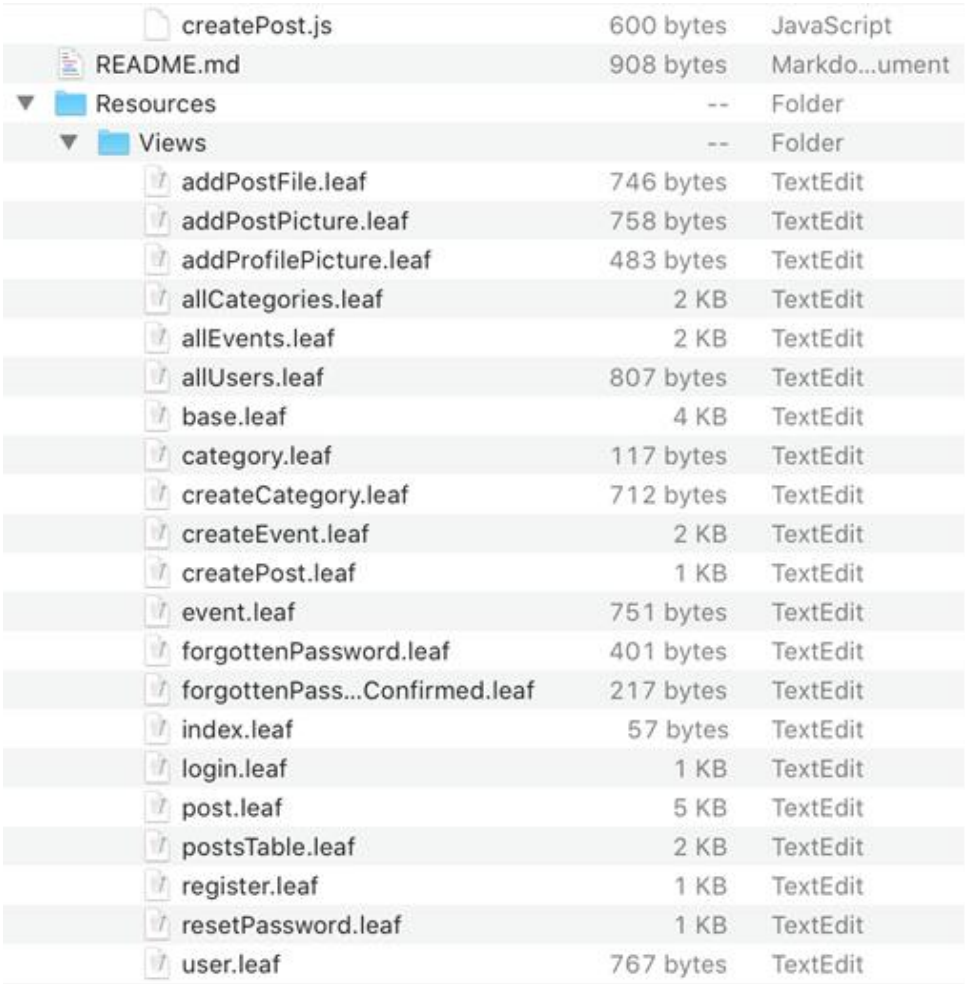


Рисунок 3.4 – Файли для роботи з фреймворком Leaf

За завантаження головної сторінки відповідає файл index.leaf.

3.4 Перевірка роботи API та авто-тестів

Для перевірки роботи API було виконано такі дії:

- показати всіх користувачів,
- показати корстувача за id,
- додати користувача,
- перевірити доданого користувача,
- показати всі оголошення,
- увійти в систему,
- створити оголошення (потрібен токен авторизації),
- перевірити додане оголошення,
- показати всі категорії,
- створити категорію (потрібен токен авторизації),
- перевірити створену категорію,
- показати всі категорії оголошення 2,
- додати категорію до оголошення 2,
- перевірити додану до оголошення категорію,
- змінити зміст оголошення,
- перевірити зміни,
- видалити оголошення,
- перевірити видалення,
- показати події,
- створити подію,
- змінити подію,
- видалити подію,
- вивести відсортовані за назвою оголошення,
- вивести одне (перше) оголошення,
- показати результати пошуку оголошення в полях «назва» та «зміст».

Результати перевірки розміщені в таблиці В.1 додатку В.

Перевірка роботи авто-тестів здійснюється на тестовій базі даних, котра оновлюється перед початком кожного тестування. Всього в проекті 29 тестів які покривають весь основний функціонал API. Вони забражені на рис. 3.5.

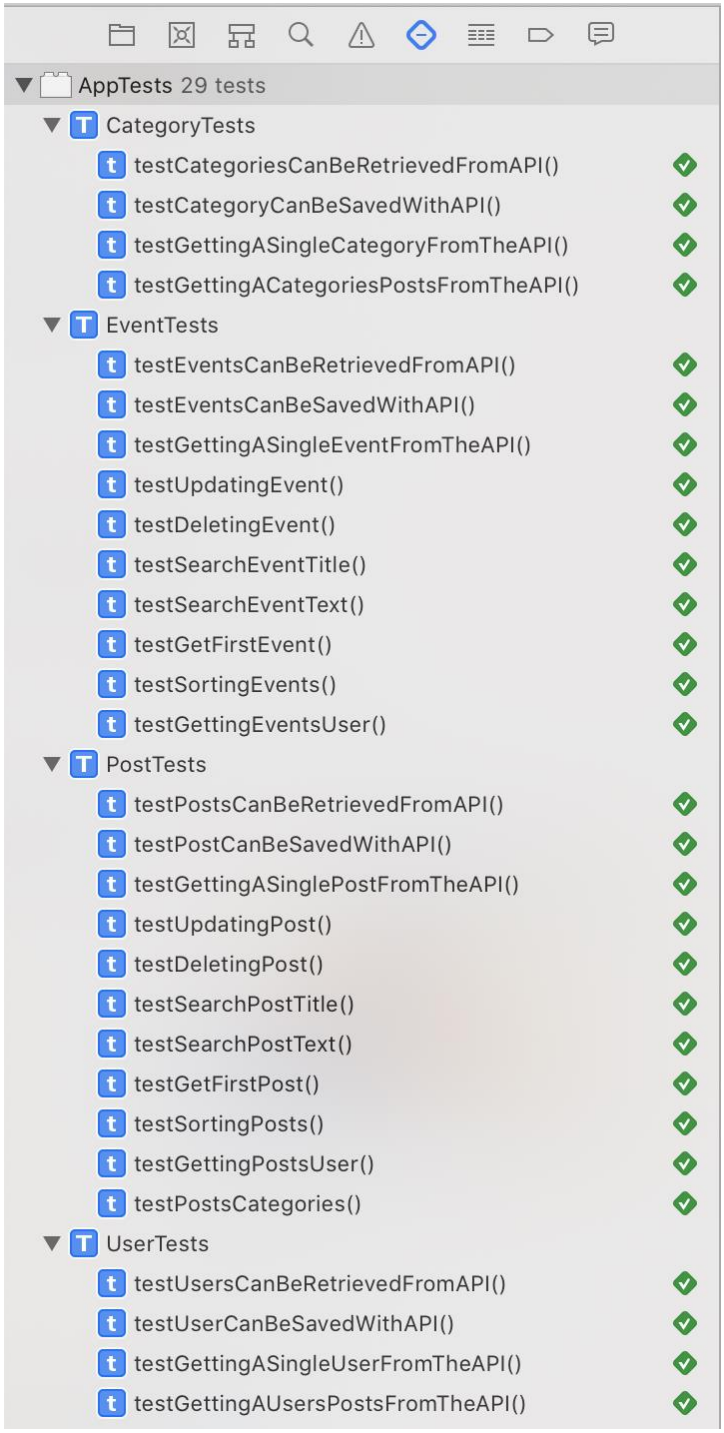


Рисунок 3.5 – Успішно виконані тести

Такі тести проводилися регулярно на кожному з етапів розробки інформаційної системи. Код файлів тестування розміщено в додатку А

3.5 Перевірка роботи Web додатку

Перегляд основних сторінок зображений на рис. 3.6.

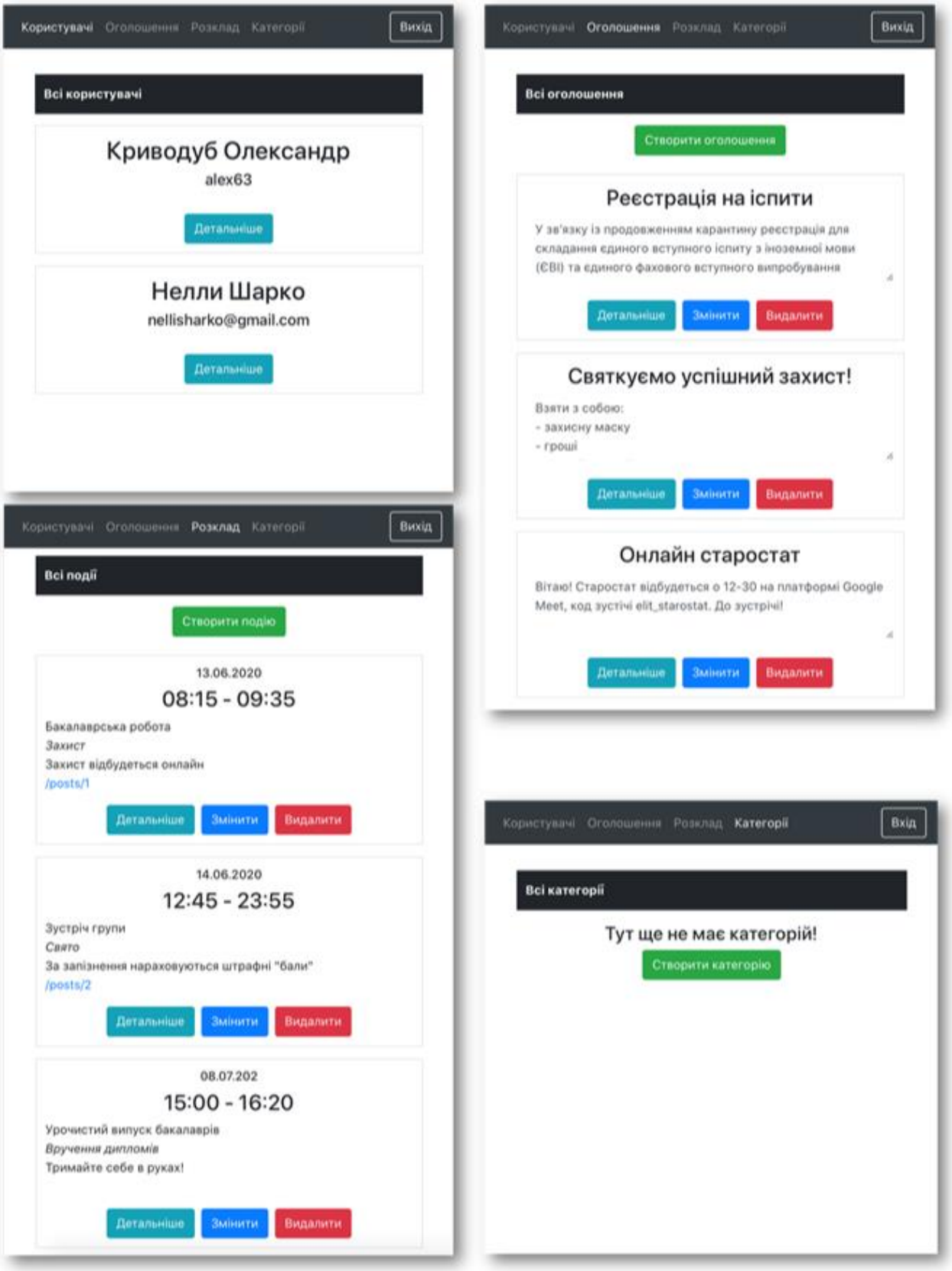


Рисунок 3.6 – Основні сторінки

Для створення, видалення або зміни оголошень, подій чи категорій необхідно увійти використовуючи логін та пароль або зареєструватися. При спробі незареєстрованого користувача додати оголошення, подію чи категорію його автоматично буде переадресовано на сторінку входу та реєстрації, що зображені на рис. 3.7.

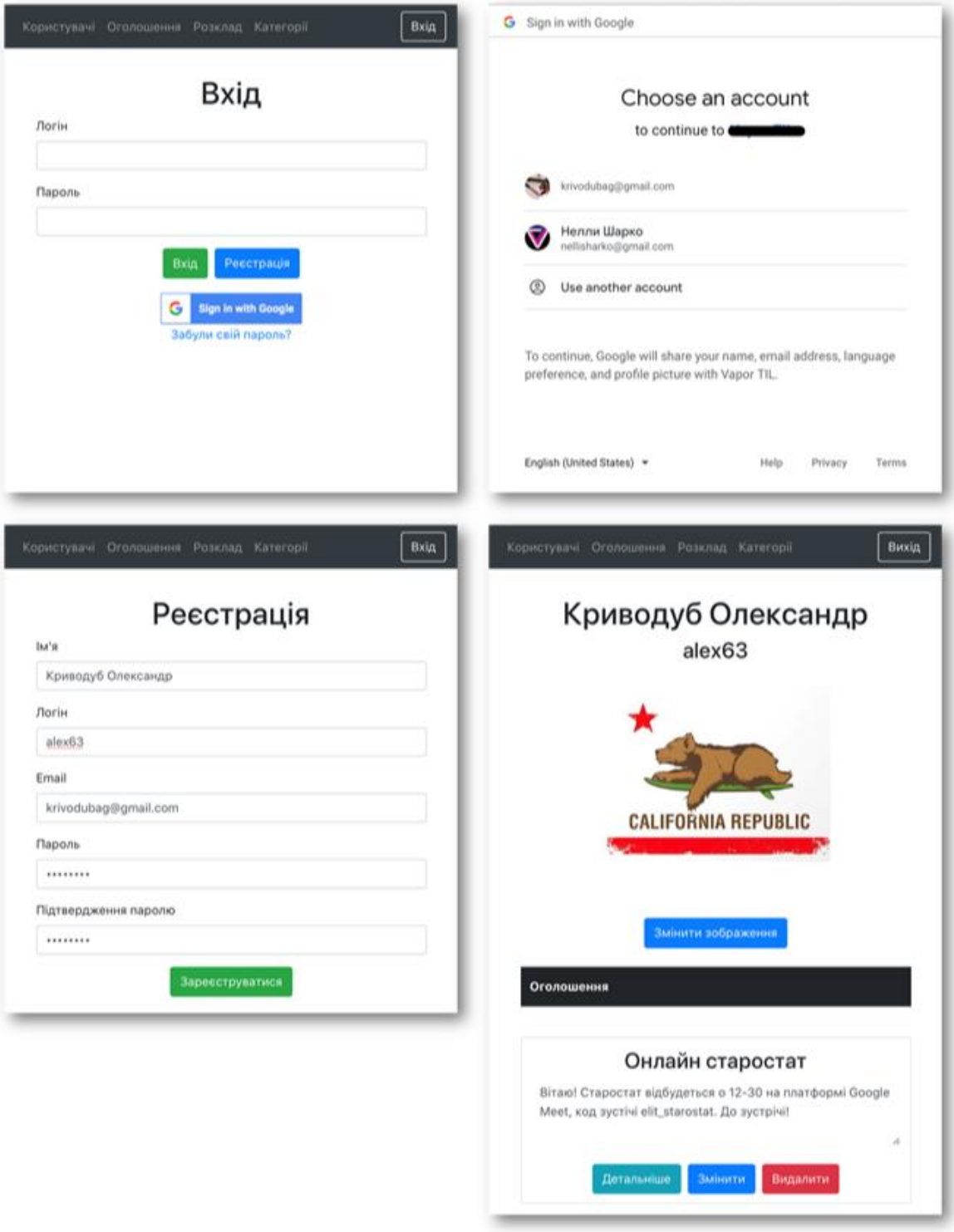


Рисунок 3.7 – Сторінки входу та реєстрації

Система дає змогу прискорити процес входу та реєстрації за допомогою використання облікового запису Google.

Для покращення користувацького досвіду під час створення або редагування оголошення існує додаткова можливість створення та додавання нових категорій, цей процес зображено на рис. 3.8.

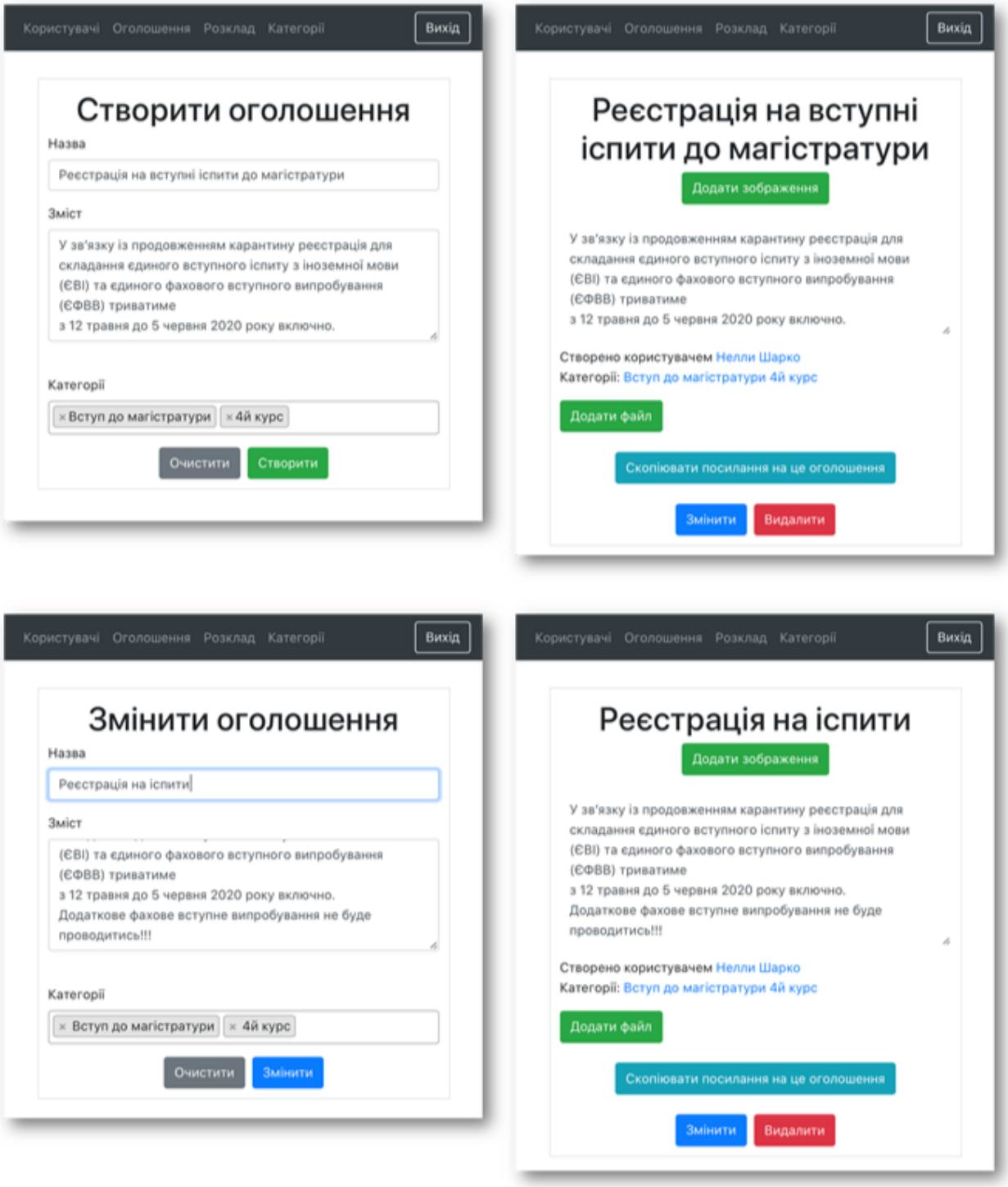


Рисунок 3.8 – Створення та редагування нового оголошення

Під час детального перегляду оголошення можна отримати повну інформацію про його автора перейшовши за посиланням у вигляді імені користувача. Також кожне з оголошень може мати своє зображення та посилання на файли. Приклад їх додавання зображено на рис. 3.9.

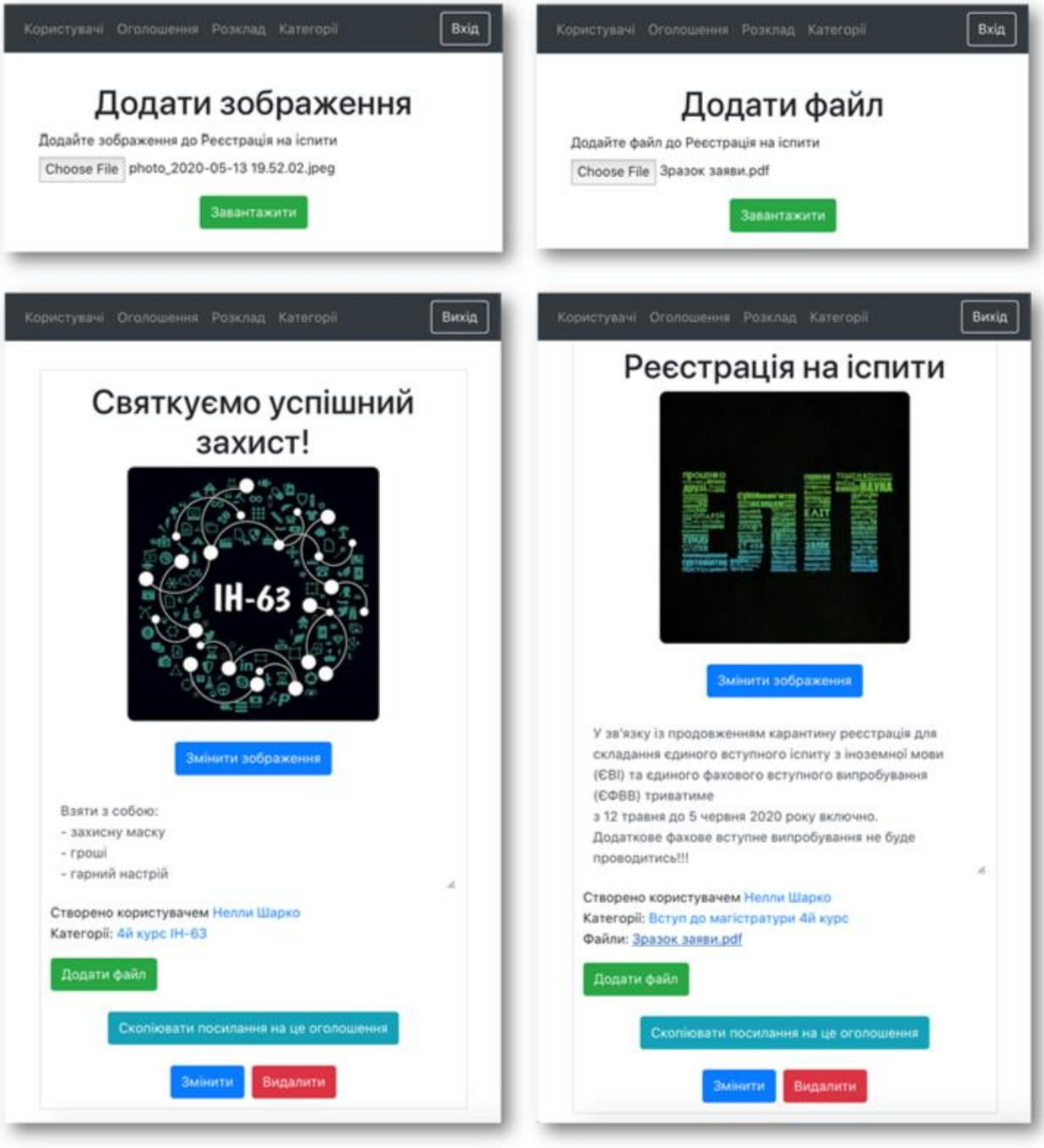


Рисунок 3.9 – Перегляд, редагування та видалення оголошення

Для перегляду змісту файлів необхідно перейти за посиланням у вигляді їх імені. Якщо інтернет браузер підтримує формат файлу то його зміст з’явиться у додатковому вікні. Якщо ні, то файл буде завантажено.

Процес створення, редагування та детальний перегляд події зображено на рис. 3.10.

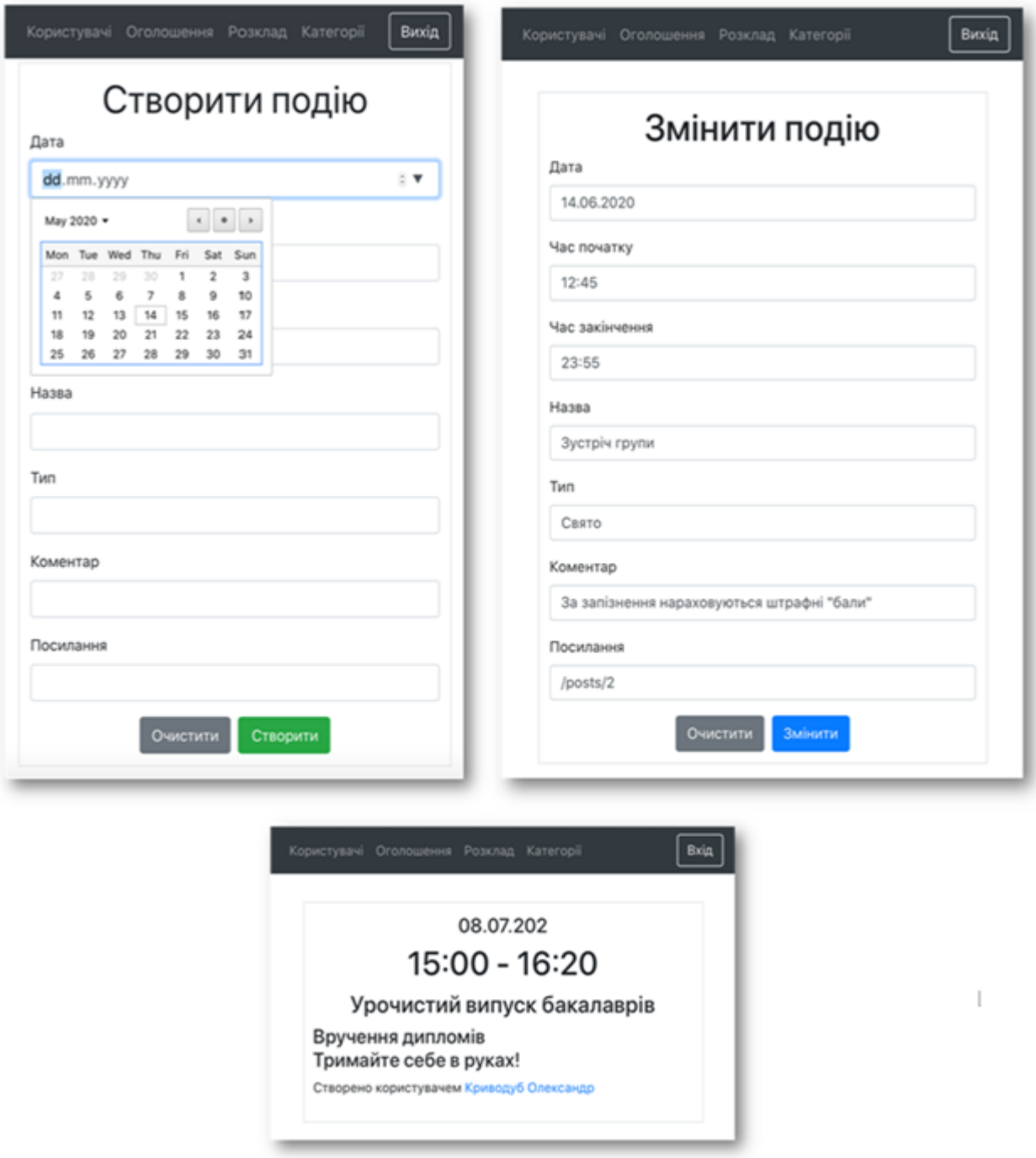


Рисунок 3.10 – Створення, редагування та детальний перегляд події

Подія може містити посилання на будь-яке оголошення в системі а також на сторінку детального огляду користувача що її створив.

Якщо користувач загубив свій пароль, то він має змогу пройти процес зміни паролю, який зображено на рис. 3.11. Для цього необхідно ввести адресу поштової скриньки, яка була вказана користувачем під час реєстрації.

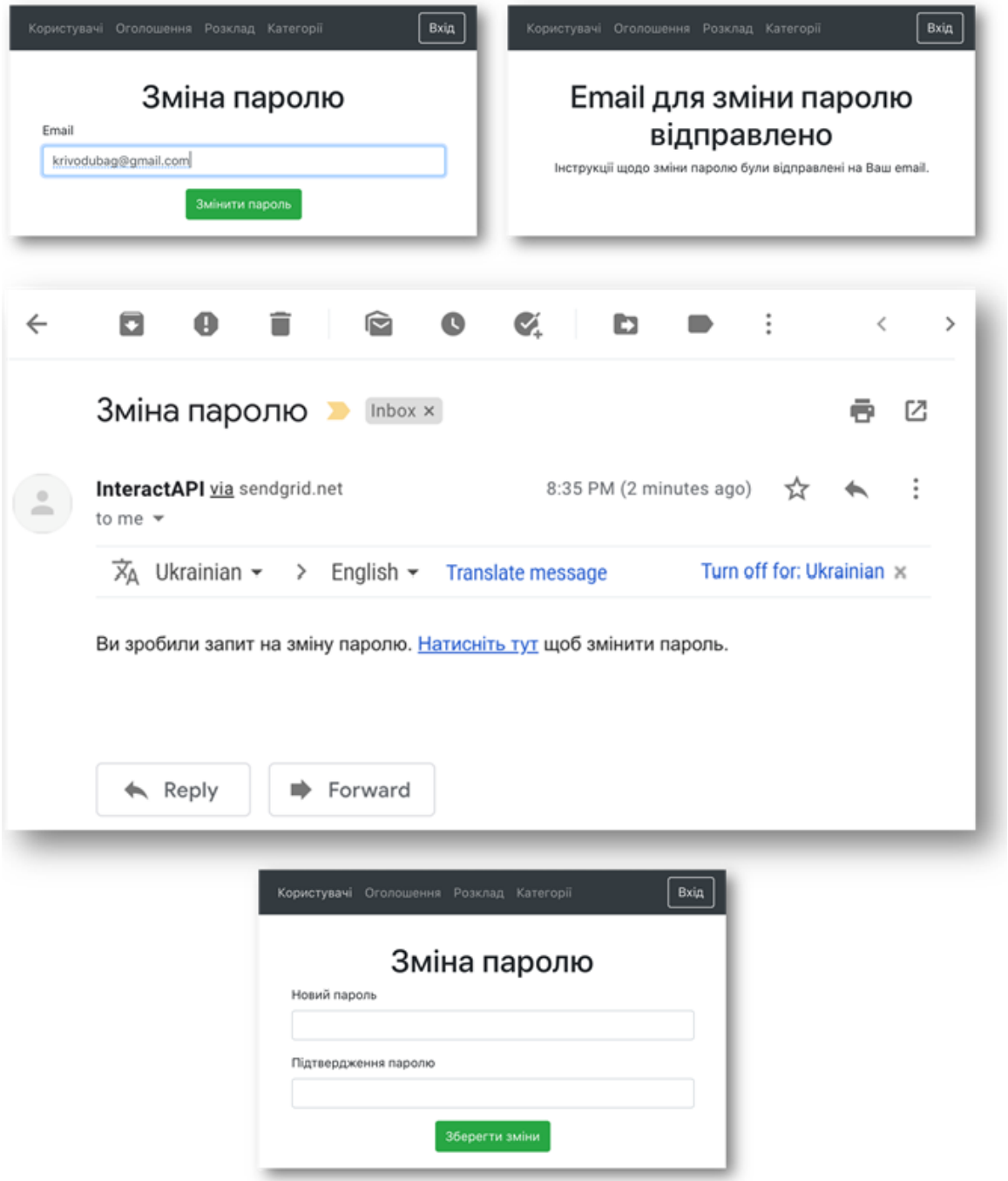


Рисунок 3.11 – Зміна паролю облікового запису

Пароль буде змінено тільки в тому разі, якщо введені користувачем паролі будуть співпадати, та якщо під час переходу на сторінку зміни паролю буде переданий спеціально згенерований токен. Скріншоти роботи Web додатку були зроблені в браузері Safari за адресою http://localhost:8080/.

3.6 Перевірка роботи мобільного додатку

Сторінки завантаження та входу зображені на рис. 3.12.

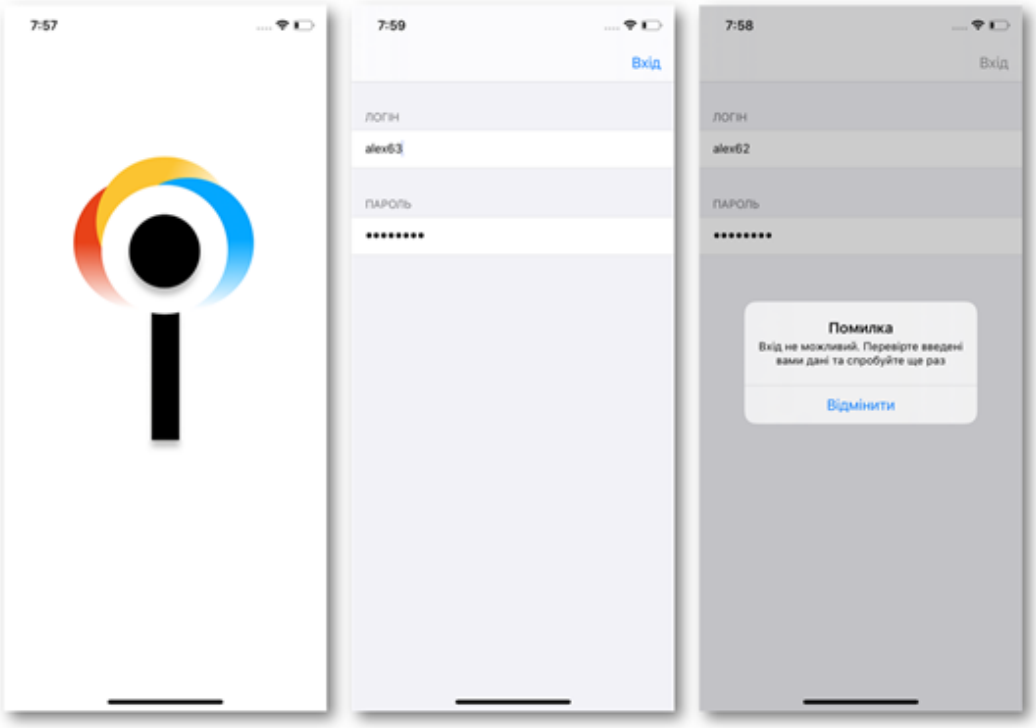


Рисунок 3.12 – Сторінки завантаження та входу

Огляд всіх користувачів та створення нового зображено на рис. 3.13.

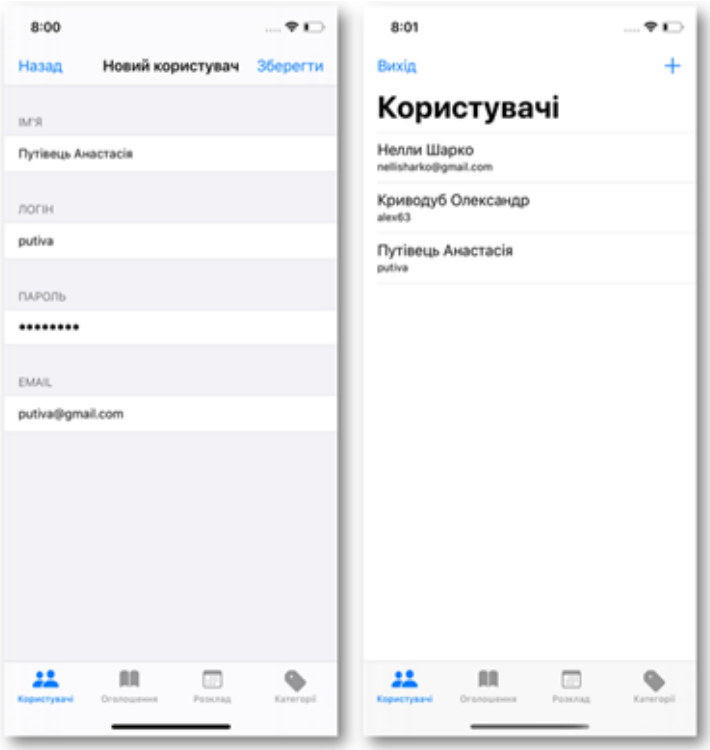


Рисунок 3.13 – Огляд та створення нового користувача

Огляд оголошення та додавання категорії зображено на рис. 3.14.

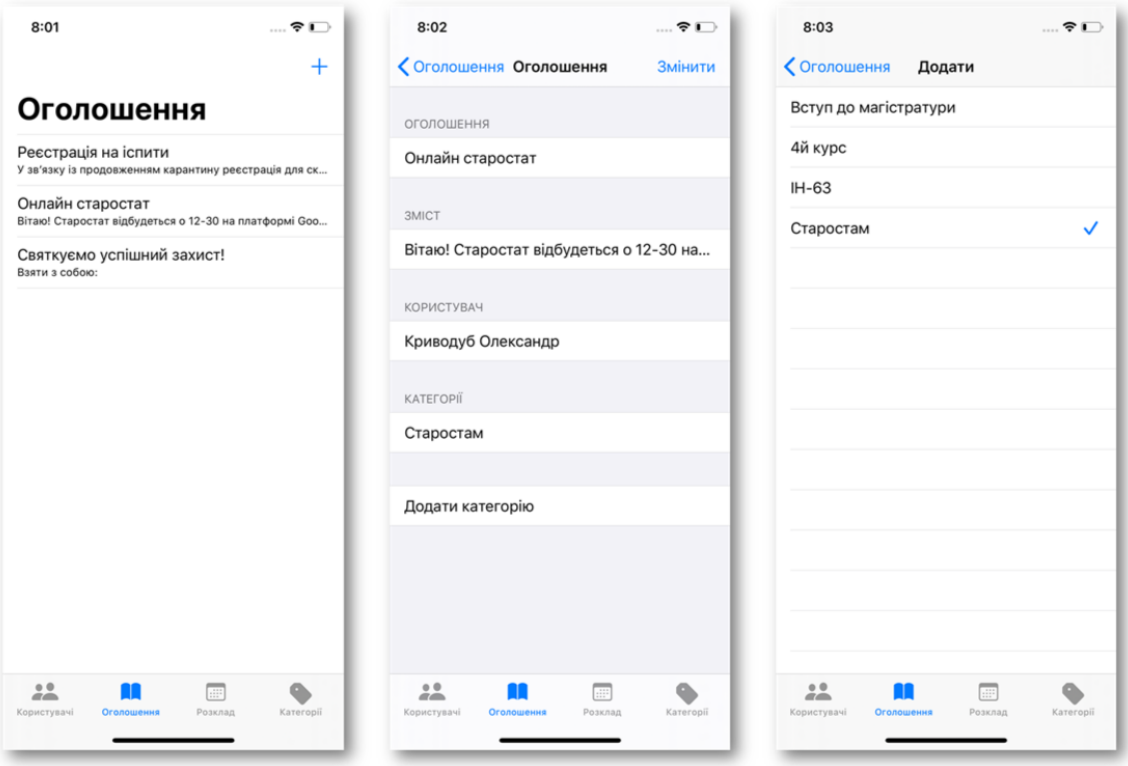


Рисунок 3.14 – Огляд оголошення та додавання категорії

Процес редагування оголошень та їх видалення жестом змахування з права на ліво зображено на рис. 3.15.

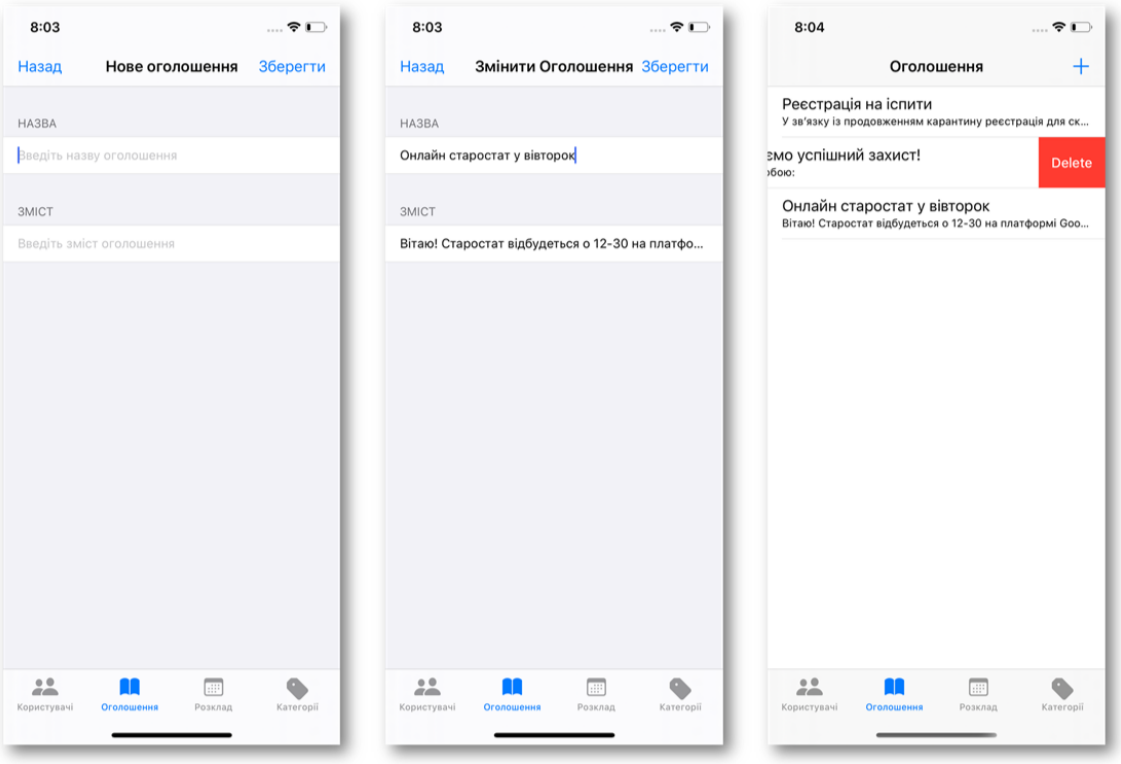


Рисунок 3.15 – Редагування та видалення оголошень

Повний та детальний перегляд подій зображено на рис. 3.16.

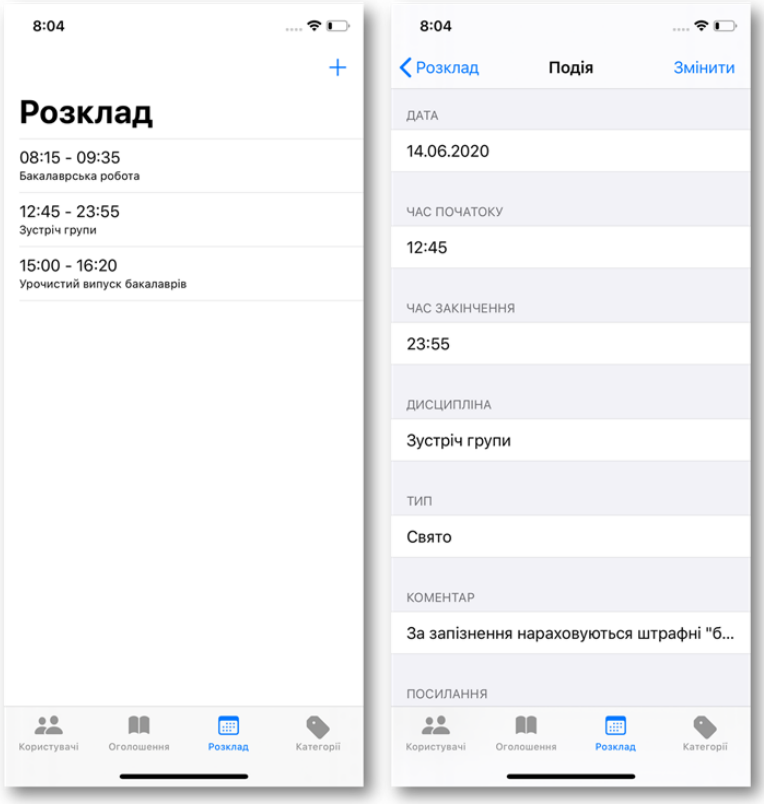


Рисунок 3.16 – Повний та детальний перегляд подій

Процес створення та видалення подій зображено на рис. 3.17.

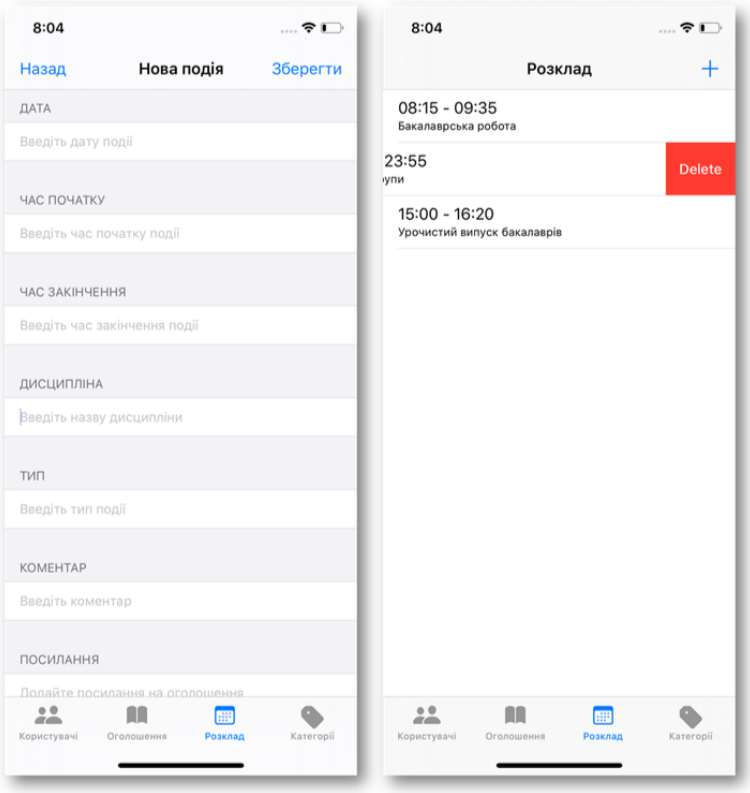


Рисунок 3.17 – Створення та видалення подій

Перегляд всіх категорій та створення нових зображено на рис. 3.18.

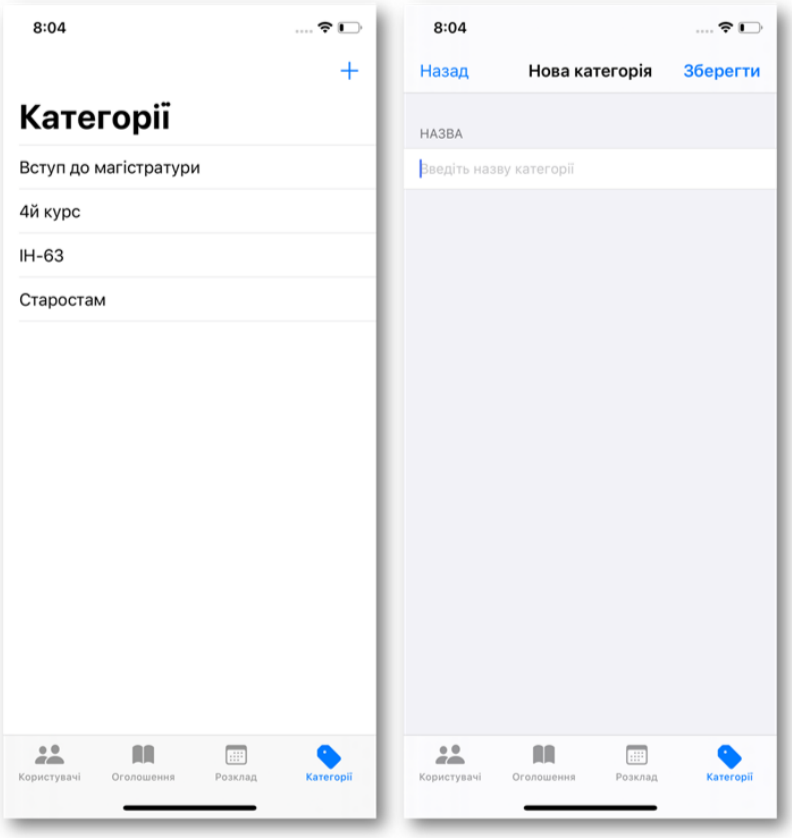


Рисунок 3.18 – Всі категорії та процес їх створення

Скріншоти роботи мобільного додатку були зроблені на вбудованому в середовище розробки симуляторі.

ВИСНОВКИ

Під час дипломного проектування було проведено аналіз та дослідження наявної ситуації, існуючих рішень та актуальних проблем процесу управління навчальної групи, під час яких:

- встановлено сутність процесу управління в контексті роботи з навчальною групою,
- виділено основні суб’єкти що приймають участь в процесі управління,
- визначено ступінь значущості цих суб’єктів та роль яку вони відіграють,
- визначено перелік методів та механізмів управління,
- проведено огляд інформаційних систем які використовуються в якості механізмів управління,
- виявлено основні проблеми існуючих інформаційних систем в процесі управління шляхом проведення опитування серед значущих суб’єктів процесу.

Проведено аналіз літератури по темі проектування подібних систем, завдяки чому було встановлено переваги використання деяких технологій створення, паттернів проектування, архітектурних стилів та типів тестування.

Визначено перелік методів проектування та створення, що відповідають умовам обмеження часу та ресурсам на розробку інформаційної системи, та перелік технологій, що мають високий рівень безпеки та швидкодії, підтримку декількох операційних систем та середовищ розробки з широкими функціональними можливостями.

Дотримуючись визначених умов та обмежень виконано проектування системи, яке дозволило:

- отримати чітке бачення кінцевого результату,
- визначити шляхи взаємодії між компонентами системи,
- знайти та заздалегідь виправити дефекти,

- визначити слабкі з точки зору безпеки місця,
- ефективно спланувати час на розробку.

Реалізовано інформаційну систему використовуючи визначені технології, в результаті чого було отримано:

- готову до роботи та тестування серверну частину системи,
- мобільний додаток для операційної системи IOS,
- Web додаток.

Проведено перевірку роботи різних компонентів інформаційної системи, виявлено та виправлено існуючі помилки.

В результаті виконання перелічених робіт отримано надійну та швидку інформаційну систему, яка відповідає визначеним вимогам та стане основою для проведення робіт з подальшого вдосконалення. В перспективах розвитку планується провести додаткові дослідження, збільшити функціонал системи, покращити рівень безпеки та швидкодію, вдосконалити дизайн та розробити мобільний додаток для операційної системи Android.

Тези на тему «Створення інформаційної системи управління навчальною групою» увійшли в матеріали міжнародної науково-технічної конференції студентів та молодих вчених “Інформатика, математика, автоматика :: 2020”, секція «Комп’ютерні науки та Кібербезпека».

СПИСОК ЛІТЕРАТУРИ

1. Jonas Schwartz, Tanner Nelson and Tim Condon. Server Side Swift with Vapor, 2018. – 562 p.

2. Eli Ganim, Joey deVilla and Matthijs Hollemans. IOS Apprentice (Eighth Edition), 2019. – 1031 p.

3. Alexis Gallagher, Janie Clayton and Matt Galloway. Swift Apprentice: Beginning Programming, 2015. – 680 p.

4. Joshua Greene & Jay Strawn. Design Patterns, 2018. – 359 p.

5. Oliver Rösner. OAuth2 with ORY Hydra, Vapor 3 and iOS 12 – https://medium.com

6. Лайза Кріспін. Джанет Грегорі. Agile-тестирование. Обучающий курс для всей команды. 2015. – 528 с.

7. Tim Condon. Using Fluent and Persisting Models in Vapor – https://www.raywenderlich.com

8. Моргунов Е. PostgreSQL. Основы языка SQL. Учебное пособие. 2016. – 336 с.

9. Jim Webber, Savas Parastatidis, Ian Robinson. REST in Practice: Hypermedia and Systems Architecture, 2010. – 415 p.

10. Дэвид Марк, Джек Наттинг, Ким Топли, Фредрик Олссон, Джефф Ламарш. Swift: разработка приложений в среде Xcode для iPhone и iPad с использованием iOS SDK. 2017. – 816 с.

11. Leonard Richardson. RESTful Web APIs: Services for a Changing World. 2013. – 406 p

12. В. Романько. Автотесты – барское дело – https://habr.com

13. Tanmay Deshpande. RESTful API Design — Step By Step Guide – https://hackernoon.com/

14. Акбердин Р. З., Кибанов А. Я. "Вдосконалення структури, функцій і економічних взаємовідносин управлінських підрозділів підприємств при формах господарювання", 1993.

ДОДАТКИ

Додаток А

Програмний код проекту створення API.

Папка Controllers

Файл PostsController.swift

```
import Vapor
import Fluent
import Authentication

struct PostsController: RouteCollection {
    func boot(router: Router) throws {
        let postsRoutes = router.grouped("api", "posts")
        postsRoutes.get(use: getAllHandler)
        postsRoutes.get(Post.parameter, use: getHandler)
        postsRoutes.get("search", use: searchHandler)
        postsRoutes.get("first", use: getFirstHandler)
        postsRoutes.get("sorted", use: sortedHandler)
        postsRoutes.get(Post.parameter, "user", use: getUserHandler)
        postsRoutes.get(Post.parameter, "categories", use:
getCategoriesHandler)
        let tokenAuthMiddleware = User.tokenAuthMiddleware()
        let guardAuthMiddleware = User.guardAuthMiddleware()
        let tokenAuthGroup = postsRoutes.grouped(tokenAuthMiddleware,
guardAuthMiddleware)
        tokenAuthGroup.post(PostCreateData.self, use: createHandler)
        tokenAuthGroup.delete(Post.parameter, use: deleteHandler)
        tokenAuthGroup.put(Post.parameter, use: updateHandler)
        tokenAuthGroup.post(Post.parameter, "categories", Category.parameter,
use: addCategoriesHandler)
        tokenAuthGroup.delete(Post.parameter, "categories",
Category.parameter, use: removeCategoriesHandler)
    }

    func getAllHandler(_ req: Request) throws -> Future<[Post]> {
        return Post.query(on: req).all()
    }

    func createHandler(_ req: Request, data: PostCreateData) throws ->
Future<Post> {
        let user = try req.requireAuthenticated(User.self) // 3
        let post = try Post(
            title: data.title,
            text: data.text,
            userID: user.requireID())
        return post.save(on: req)
    }

    func getHandler(_ req: Request) throws -> Future<Post> {
        return try req.parameters.next(Post.self)
    }

    func updateHandler(_ req: Request) throws -> Future<Post> {
        return try flatMap(to: Post.self, req.parameters.next(Post.self),
req.content.decode(PostCreateData.self)
        ) { post, updateData in
            post.title = updateData.title
```

```
            post.text = updateData.text
            let user = try req.requireAuthenticated(User.self)
            post.userID = try user.requireID()
            return post.save(on: req)
        } }

    func deleteHandler(_ req: Request) throws -> Future<HTTPStatus> {
        return try req.parameters.next(Post.self).delete(on:
req).transform(to: .noContent)
    }

    func searchHandler(_ req: Request) throws -> Future<[Post]> {
        guard let searchTerm = req.query[String.self, at: "term"] else {
            throw Abort(.badRequest)
        }
        return Post.query(on: req).group(.or) { or in
            or.filter(\.title == searchTerm)
            or.filter(\.text == searchTerm)
        }.all()
    }

    func getFirstHandler(_ req: Request) throws -> Future<Post> {
        return Post.query(on: req).first().unwrap(or: Abort(.notFound))
    }

    func sortedHandler(_ req: Request) throws -> Future<[Post]> {
        return Post.query(on: req).sort(\.title, .ascending).all()
    }

    func getUserHandler(_ req: Request) throws -> Future<User.Public> {
        return try req.parameters.next(Post.self)
            .flatMap(to: User.Public.self) { post in
                post.user.get(on: req).convertToPublic() }
    }

    func addCategoriesHandler(_ req: Request) throws -> Future<HTTPStatus> {
        return try flatMap(to: HTTPStatus.self,
req.parameters.next(Post.self),
                           req.parameters.next(Category.self)) { post,
category in
                            return post.categories.attach(category, on:
req).transform(to: .created)
        }
    }

    func getCategoriesHandler(_ req: Request) throws -> Future<[Category]> {
        return try req.parameters.next(Post.self).flatMap(to:
[Category].self) { post in
            try post.categories.query(on: req).all()
        }
    }

    func removeCategoriesHandler(_ req: Request) throws -> Future<HTTPStatus>
{
        return try flatMap(to: HTTPStatus.self,
req.parameters.next(Post.self),
                           req.parameters.next(Category.self)) { post,
category in
                            return post.categories.detach(category, on:
req).transform(to: .noContent)
        }
    }
}
```

```
struct PostCreateData: Content {
    let title: String
    let text: String
}
```

Файл CategoriesController.swift

```
import Vapor

struct CategoriesController: RouteCollection {
    func boot(router: Router) throws {
        let categoriesRoute = router.grouped("api", "categories")
        categoriesRoute.get(use: getAllHandler)
        categoriesRoute.get(Category.parameter, use: getHandler)
        categoriesRoute.get(Category.parameter, "posts", use:
getPostsHandler)
        let tokenAuthMiddleware = User.tokenAuthMiddleware()
        let guardAuthMiddleware = User.guardAuthMiddleware()
        let tokenAuthGroup = categoriesRoute.grouped(tokenAuthMiddleware,
guardAuthMiddleware)
        tokenAuthGroup.post(Category.self, use: createHandler)
        tokenAuthGroup.delete(Category.parameter, use: deleteHandler)
        tokenAuthGroup.put(Category.parameter, use: updateHandler)

    }

    func createHandler(_ req: Request, category: Category) throws ->
Future<Category> {
        return category.save(on: req)
    }

    func deleteHandler(_ req: Request) throws -> Future<HTTPStatus> {
        return try req.parameters.next(Category.self).delete(on:
req).transform(to: .noContent)
    }

    func updateHandler(_ req: Request) throws -> Future<Category> {
        return try flatMap(to: Category.self,
req.parameters.next(Category.self),
req.content.decode(CategoryCreateData.self)
        ) { category, updateData in
            category.name = updateData.name
            return category.save(on: req)
        } }

    func getAllHandler(_ req: Request) throws -> Future<[Category]> {
        return Category.query(on: req).all()
    }

    func getHandler(_ req: Request) throws -> Future<Category> {
        return try req.parameters.next(Category.self)
    }

    func getPostsHandler(_ req: Request) throws -> Future<[Post]> {
        return try req.parameters.next(Category.self).flatMap(to:
[Post].self) { category in
            try category.posts.query(on: req).all()
        }
    }
}
```

```
struct CategoryCreateData: Content {
    let name: String
}
```

Файл configure.swift

```
import FluentPostgreSQL
import Vapor
import Leaf
import Authentication
import SendGrid

// Called before your application initializes.
public func configure(_ config: inout Config, _ env: inout Environment, _
services: inout Services) throws {
    // Register providers first
    try services.register(FluentPostgreSQLProvider())
    try services.register(LeafProvider())
    try services.register(AuthenticationProvider())
    try services.register(SendGridProvider())

    // Register routes to the router
    let router = EngineRouter.default()
    try routes(router)
    services.register(router, as: Router.self)

    // Register middleware
    var middlewares = MiddlewareConfig() // Create _empty_ middleware config
    middlewares.use(FileMiddleware.self) // Serves files from `Public/`
directory
    middlewares.use(ErrorMiddleware.self) // Catches errors and converts to
HTTP response
    middlewares.use(SessionsMiddleware.self) // Enables sessions for all
requests
    services.register(middlewares)

    // Configure a database
    var databases = DatabasesConfig()
    let databaseConfig = PostgreSQLDatabaseConfig(hostname: "localhost",
username: "vapor",
                                                  database: "vapor",
password: "password")
    let database = PostgreSQLDatabase(config: databaseConfig)
    databases.add(database: database, as: .psql)
    services.register(databases)

    // Configure migrations
    var migrations = MigrationConfig()
    migrations.add(model: User.self, database:
DatabaseIdentifier<User.Database>.psql)
    migrations.add(migration: User.self, database: .psql)

    migrations.add(model: Post.self, database:
DatabaseIdentifier<Post.Database>.psql)
    migrations.add(migration: Post.self, database: .psql)

    migrations.add(model: Event.self, database:
DatabaseIdentifier<Event.Database>.psql)
    migrations.add(migration: Event.self, database: .psql)

    migrations.add(model: Category.self, database:
DatabaseIdentifier<Category.Database>.psql)
```

```
    migrations.add(migration: Category.self, database: .psql)

    migrations.add(model: PostCategoryPivot.self, database:
DatabaseIdentifier<PostCategoryPivot.Database>.psql)
    migrations.add(migration: PostCategoryPivot.self, database: .psql)

    migrations.add(model: Token.self, database:
DatabaseIdentifier<Token.Database>.psql)
    migrations.add(migration: Token.self, database: .psql)

    migrations.add(model: ResetPasswordToken.self, database:
DatabaseIdentifier<ResetPasswordToken.Database>.psql)
    services.register(migrations)
    migrations.add(migration: ResetPasswordToken.self, database: .psql)

    config.prefer(LeafRenderer.self, for: ViewRenderer.self)
    config.prefer(MemoryKeyedCache.self, for: KeyedCache.self) // Service for
using cache

    guard let sendGridAPIKey = Environment.get("SENDGRID_API_KEY") else {
      fatalError("No Send Grid API Key specified")
    }
    let sendGridConfig = SendGridConfig(apiKey: sendGridAPIKey)
    services.register(sendGridConfig)
}
```

Файл EventsController.swift

```
import Vapor
import Fluent

struct EventsController: RouteCollection {
    func boot(router: Router) throws {
        let eventsRoutes = router.grouped("api", "events")
        eventsRoutes.get(use: getAllHandler)
        eventsRoutes.get(Event.parameter, use: getHandler)
        eventsRoutes.get("search", use: searchHandler)
        eventsRoutes.get("first", use: getFirstHandler)
        eventsRoutes.get(Event.parameter, "user", use: getUserHandler)
        let tokenAuthMiddleware = User.tokenAuthMiddleware()
        let guardAuthMiddleware = User.guardAuthMiddleware()
        let tokenAuthGroup = eventsRoutes.grouped(tokenAuthMiddleware,
guardAuthMiddleware)
        tokenAuthGroup.post(EventCreateData.self, use: createHandler)
        tokenAuthGroup.delete(Event.parameter, use: deleteHandler)
        tokenAuthGroup.put(Event.parameter, use: updateHandler)
    }

    func getAllHandler(_ req: Request) throws -> Future<[Event]> {
        return Event.query(on: req).sort(\.start, .ascending).all()
    }

    func createHandler(_ req: Request, data: EventCreateData) throws ->
Future<Event> {
        let user = try req.requireAuthenticated(User.self)
        let event = try Event(
            date: data.date,
            start: data.start,
            end: data.end,
            title: data.title,
            type: data.type,
            text: data.text,
```

```
            link: data.link,
            userID: user.requireID())
        return event.save(on: req)
    }

    func getHandler(_ req: Request) throws -> Future<Event> {
        return try req.parameters.next(Event.self)
    }

    func updateHandler(_ req: Request) throws -> Future<Event> {
        return try flatMap(to: Event.self, req.parameters.next(Event.self),
req.content.decode(EventCreateData.self)
        ) { event, updateData in
            event.date = updateData.date
            event.start = updateData.start
            event.end = updateData.end
            event.title = updateData.title
            event.type = updateData.type
            event.text = updateData.text
            event.link = updateData.link
            let user = try req.requireAuthenticated(User.self)
            event.userID = try user.requireID()
            return event.save(on: req)
        } }

    func deleteHandler(_ req: Request) throws -> Future<HTTPStatus> {
        return try req.parameters.next(Event.self).delete(on:
req).transform(to: .noContent)
    }

    func searchHandler(_ req: Request) throws -> Future<[Event]> {
        guard let searchTerm = req.query[String.self, at: "term"] else {
            throw Abort(.badRequest)
        }
        return Event.query(on: req).group(.or) { or in
            or.filter(\.title == searchTerm)
            or.filter(\.text == searchTerm)
        }.all()
    }

    func getFirstHandler(_ req: Request) throws -> Future<Event> {
        return Event.query(on: req).first().unwrap(or: Abort(.notFound))
    }

    func getUserHandler(_ req: Request) throws -> Future<User> {
        return try req.parameters.next(Event.self).flatMap(to: User.self) {
event in
            event.user.get(on: req)
        }
    }
}

struct EventCreateData: Content {
    let date: String
    let start: String
    let end: String
    let title: String
    let type: String
    let text: String
    let link: String
}
```

Файл UsersController.swift

```
import Vapor
import Crypto

struct UsersController: RouteCollection {
    func boot(router: Router) throws {
        let usersRoute = router.grouped("api", "users")
        usersRoute.post(User.self, use: createHandler)
        usersRoute.get(use: getAllHandler)
        usersRoute.get(User.parameter, use: getHandler)
        usersRoute.get(User.parameter, "posts", use: getPostsHandler)
        usersRoute.get(User.parameter, "events", use: getEventsHandler)
        let basicAuthMiddleware = User.basicAuthMiddleware(using:
BCryptDigest())
        let basicAuthGroup = usersRoute.grouped(basicAuthMiddleware)
        basicAuthGroup.post("login", use: loginHandler)
    }

    func createHandler(_ req: Request, user: User) throws ->
Future<User.Public> {
        user.password = try BCrypt.hash(user.password)
        return user.save(on: req).convertToPublic()
    }

    func getAllHandler(_ req: Request) throws -> Future<[User.Public]> {
        return User.query(on: req).decode(data: User.Public.self).all()
    }

    func getHandler(_ req: Request) throws -> Future<User.Public> {
        return try req.parameters.next(User.self).convertToPublic()
    }

    func getPostsHandler(_ req: Request) throws -> Future<[Post]> {
        return try req.parameters.next(User.self).flatMap(to: [Post].self) {
user in
            try user.posts.query(on: req).all()
        }
    }

    func getEventsHandler(_ req: Request) throws -> Future<[Event]> {
        return try req.parameters.next(User.self).flatMap(to: [Event].self) {
user in
            try user.events.query(on: req).all()
        }
    }

    func loginHandler(_ req: Request) throws -> Future<Token> {
        let user = try req.requireAuthenticated(User.self)
        let token = try Token.generate(for: user)
        return token.save(on: req)
    }
}
```

Файл ImperialController.swift

```
import Vapor
import Imperial
import Authentication

struct ImperialController: RouteCollection {
  func boot(router: Router) throws {
```

```
    guard let googleCallbackURL = Environment.get("GOOGLE_CALLBACK_URL") else
{
      fatalError("Google callback URL not set")
    }
    try router.oAuth(from: Google.self, authenticate: "login-google",
callback: googleCallbackURL,
                     scope: ["profile", "email"], completion:
processGoogleLogin)
  }

  func processGoogleLogin(request: Request, token: String) throws ->
Future<ResponseEncodable> {
    return try Google.getUser(on: request).flatMap(to:
ResponseEncodable.self) { userInfo in
      return User.query(on: request).filter(\.username == userInfo.email)
                 .first().flatMap(to: ResponseEncodable.self) { foundUser in
        guard let existingUser = foundUser else {
          let user = User(name: userInfo.name, username: userInfo.email,
password: UUID().uuidString,
                          email: userInfo.email)
          return user.save(on: request).map(to: ResponseEncodable.self) {
user in
            try request.authenticateSession(user)
            return request.redirect(to: "/")
          }
        }
        try request.authenticateSession(existingUser)
        return request.future(request.redirect(to: "/"))
      }
    }
  }
}

struct GoogleUserInfo: Content {
  let email: String
  let name: String
}

extension Google {
  static func getUser(on request: Request) throws -> Future<GoogleUserInfo> {
    var headers = HTTPHeaders()
    headers.bearerAuthorization = try BearerAuthorization(token:
request.accessToken())

    let googleAPIURL =
"https://www.googleapis.com/oauth2/v1/userinfo?alt=json"
    return try request.client().get(googleAPIURL, headers: headers).map(to:
GoogleUserInfo.self) { response in
      guard response.http.status == .ok else {
        if response.http.status == .unauthorized {
          throw Abort.redirect(to: "/login-google")
        } else {
          throw Abort(.internalServerError)
        }
      }
      return try response.content.syncDecode(GoogleUserInfo.self)
    }
  }
}
```

Файл WebsiteController.swift

```
import Vapor
import Leaf
import Authentication
import SendGrid

struct WebsiteController: RouteCollection {
    let imageFolder = "ProfilePictures/"
    func boot(router: Router) throws {

        let authSessionRoutes = router.grouped(User.authSessionsMiddleware())
        authSessionRoutes.get(use: allPostsHandler)
        authSessionRoutes.get("posts", Post.parameter, use: postHandler)
        authSessionRoutes.get("users", User.parameter, use: userHandler)
        authSessionRoutes.get("users", use: allUsersHandler)
        authSessionRoutes.get("events", use: allEventsHandler)
        authSessionRoutes.get("events", Event.parameter, use: eventHandler)
        authSessionRoutes.get("categories", use: allCategoriesHandler)
        authSessionRoutes.get("categories", Category.parameter, use:
categoryHandler)
        authSessionRoutes.get("login", use: loginHandler)
        authSessionRoutes.post(LoginPostData.self, at: "login", use:
loginPostHandler)
        authSessionRoutes.post("logout", use: logoutHandler)
        authSessionRoutes.get("register", use: registerHandler)
        authSessionRoutes.post(RegisterData.self, at: "register", use:
registerPostHandler)
        authSessionRoutes.get("forgottenPassword", use:
forgottenPasswordHandler)
        authSessionRoutes.post("forgottenPassword", use:
forgottenPasswordPostHandler)
        authSessionRoutes.get("resetPassword", use: resetPasswordHandler)
        authSessionRoutes.post(ResetPasswordData.self, at: "resetPassword",
use: resetPasswordPostHandler)
        authSessionRoutes.get( "users", User.parameter, "profilePicture",
use: getUsersProfilePictureHandler)
        authSessionRoutes.get( "posts", Post.parameter, "postFile", use:
getPostsFileHandler)
        authSessionRoutes.get( "posts", Post.parameter, "postPicture", use:
getPostsPictureHandler)

        let protectedRoutes = authSessionRoutes
.grouped(RedirectMiddleware<User>(path: "/login"))

        protectedRoutes.get("posts", "create", use: createPostHandler)
        protectedRoutes.post(CreatePostData.self, at: "posts", "create", use:
createPostPOSTHandler)
        protectedRoutes.get("posts", Post.parameter, "edit", use:
editPostHandler)
        protectedRoutes.post("posts", Post.parameter, "edit", use:
editPostPOSTHandler)
        protectedRoutes.post("posts", Post.parameter, "delete", use:
deletePostHandler)

        protectedRoutes.get("events", "create", use: createEventHandler)
        protectedRoutes.post(CreateEventData.self, at: "events", "create",
use: createEventPOSTHandler)
        protectedRoutes.get("events", Event.parameter, "edit", use:
editEventHandler)
        protectedRoutes.post("events", Event.parameter, "edit", use:
editEventPOSTHandler)
```

```
        protectedRoutes.post("events", Event.parameter, "delete", use:
deleteEventHandler)

        protectedRoutes.get("categories", "create", use:
createCategoryHandler)
        protectedRoutes.post(Category.self, at: "categories", "create", use:
createCategoryPOSTHandler)
        protectedRoutes.get("categories", Category.parameter, "edit", use:
editCategoryHandler)
        protectedRoutes.post("categories", Category.parameter, "edit", use:
editCategoryPOSTHandler)
        protectedRoutes.post("categories", Category.parameter, "delete", use:
deleteCategoryHandler)

        protectedRoutes.get( "users", User.parameter, "addProfilePicture",
use: addProfilePictureHandler)
        protectedRoutes.post( "users", User.parameter, "addProfilePicture",
use: addProfilePicturePostHandler)

        protectedRoutes.get( "posts", Post.parameter, "addPostFile", use:
addPostFileHandler)
        protectedRoutes.post( "posts", Post.parameter, "addPostFile", use:
addPostFilePOSTHandler)

        protectedRoutes.get( "posts", Post.parameter, "addPostPicture", use:
addPostPictureHandler)
        protectedRoutes.post( "posts", Post.parameter, "addPostPicture", use:
addPostPicturePOSTHandler)

    func allUsersHandler(_ req: Request) throws -> Future<View> {
        return User.query(on: req) .all()
            .flatMap(to: View.self) { users in

                let userLoggedIn = try req.isAuthenticated(User.self)
                let context = AllUsersContext(title: "Користувачі", users:
users, userLoggedIn: userLoggedIn)
                return try req.view().render("allUsers", context)
        } }

    func allPostsHandler(_ req: Request) throws -> Future<View> {
        return Post.query(on: req).all().flatMap(to: View.self) { posts in
            let userLoggedIn = try req.isAuthenticated(User.self)
            let context = AllPostsContext(title: "Оголошення", posts: posts,
userLoggedIn: userLoggedIn)
            return try req.view().render("index", context)
        } }

    func allEventsHandler(_ req: Request) throws -> Future<View> {
        return Event.query(on: req).sort(\.start,
.ascending).all().flatMap(to: View.self) { events in
            let userLoggedIn = try req.isAuthenticated(User.self)
            let context = AllEventsContext(title: "Розклад", events: events,
userLoggedIn: userLoggedIn)
            return try req.view().render("allEvents", context)
        } }

    func allCategoriesHandler(_ req: Request) throws -> Future<View> {
        let categories = Category.query(on: req).all()
        let userLoggedIn = try req.isAuthenticated(User.self)
        let context = AllCategoriesContext(categories: categories,
userLoggedIn: userLoggedIn)
        return try req.view().render("allCategories", context)
    }
```

```
    func userHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(User.self) .flatMap(to: View.self) {
user in
            return try user.posts.query(on: req).all().flatMap(to: View.self)
{ posts in
                let userLoggedIn = try req.isAuthenticated(User.self)
                let loggedInUser = try req.authenticated(User.self) // 2
                let context = UserContext(title: user.name, user: user,
posts: posts, userLoggedIn: userLoggedIn, authenticatedUser: loggedInUser)
                return try req.view().render("user", context) }
        } }

    func postHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Post.self) .flatMap(to: View.self) {
post in
            return post.user.get(on: req).flatMap(to: View.self) { user in
                let categories = try post.categories.query(on: req).all()
                let userLoggedIn = try req.isAuthenticated(User.self)
                let context = PostContext(
                    title: post.title, post: post,
                    user: user,
                    categories: categories, userLoggedIn: userLoggedIn)
                return try req.view().render("post", context) }
        } }

    func eventHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Event.self) .flatMap(to: View.self) {
event in
            return event.user.get(on: req).flatMap(to: View.self) { user in
                let userLoggedIn = try req.isAuthenticated(User.self)
                let context = EventContext( title: event.title, event: event,
user: user, userLoggedIn: userLoggedIn)
                return try req.view().render("event", context) }
        } }

    func categoryHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Category.self) .flatMap(to: View.self)
{ category in
            let posts = try category.posts.query(on: req).all()
            let userLoggedIn = try req.isAuthenticated(User.self)
            let context = CategoryContext(
                title: category.name, category: category, posts: posts,
userLoggedIn: userLoggedIn)
            return try req.view().render("category", context)
        } }

    func createPostHandler(_ req: Request) throws -> Future<View> {
        let userLoggedIn = try req.isAuthenticated(User.self)
        let context = CreatePostContext(userLoggedIn: userLoggedIn)
        return try req.view().render("createPost", context)
    }

    func createEventHandler(_ req: Request) throws -> Future<View> {
        let userLoggedIn = try req.isAuthenticated(User.self)
       let context = CreateEventContext(userLoggedIn: userLoggedIn)
        return try req.view().render("createEvent", context)
    }

    func createCategoryHandler(_ req: Request) throws -> Future<View> {
        let userLoggedIn = try req.isAuthenticated(User.self)
       let context = CreateCategoryContext(userLoggedIn: userLoggedIn)
        return try req.view().render("createCategory", context)
```

```
    }

    func createPostPOSTHandler(_ req: Request, data: CreatePostData) throws -
> Future<Response> {
        let user = try req.requireAuthenticated(User.self)
        let post = try Post(title: data.title, text: data.text, userID:
user.requireID())
        return post.save(on: req).flatMap(to: Response.self) { post in
            guard let id = post.id else {
                throw Abort(.internalServerError)
            }
            var categorySaves: [Future<Void>] = []
            for category in data.categories ?? [] {
                try categorySaves.append(
                    Category.addCategory(category, to: post, on: req))
            }
            let redirect = req.redirect(to: "/posts/\(id)")
            return categorySaves.flatten(on: req).transform(to: redirect)
        }
    }

    func createEventPOSTHandler(_ req: Request, data: CreateEventData) throws
-> Future<Response> {
        let user = try req.requireAuthenticated(User.self)
        let event = try Event(date: data.date, start: data.start, end:
data.end, title: data.title, type: data.type, text: data.text, link:
data.link, userID: user.requireID())
        return event.save(on: req).map(to: Response.self) { event in
            return req.redirect(to: "/events/")
        } }

    func createCategoryPOSTHandler(_ req: Request, category: Category) throws
-> Future<Response> {
        return category.save(on: req).map(to: Response.self) { category in
            return req.redirect(to: "/categories/")
        } }

    func editPostHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Post.self) .flatMap(to: View.self) {
post in
            let categories = try post.categories.query(on: req).all()
            let userLoggedIn = try req.isAuthenticated(User.self)
            let context = EditPostContext(post: post, categories: categories,
userLoggedIn: userLoggedIn)
            return try req.view().render("createPost", context)
        } }

    func editEventHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Event.self) .flatMap(to: View.self) {
event in
            let userLoggedIn = try req.isAuthenticated(User.self)
           let context = EditEventContext(event: event, userLoggedIn:
userLoggedIn)
            return try req.view().render("createEvent", context)
        } }

    func editCategoryHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Category.self) .flatMap(to: View.self)
{ category in
            let userLoggedIn = try req.isAuthenticated(User.self)
           let context = EditCategoryContext( category: category,
userLoggedIn: userLoggedIn)
            return try req.view().render("createCategory", context)
```

```
        } }

    func editPostPOSTHandler(_ req: Request) throws -> Future<Response> {
        return try flatMap(to: Response.self, req.parameters.next(Post.self),
req.content.decode(CreatePostData.self)) { post, data in
            let user = try req.requireAuthenticated(User.self)
            post.title = data.title
            post.text = data.text
            post.userID = try user.requireID()
            guard let id = post.id else {
                throw Abort(.internalServerError) }
            return post.save(on: req) .flatMap(to: [Category].self) { _ in
                try post.categories.query(on: req).all() }.flatMap(to:
Response.self) { existingCategories in
                    let existingStringArray = existingCategories.map {
$0.name }
                    let existingSet = Set<String>(existingStringArray)
                    let newSet = Set<String>(data.categories ?? [])
                    let categoriesToAdd = newSet.subtracting(existingSet)
                    let categoriesToRemove = existingSet.subtracting(newSet)
                    var categoryResults: [Future<Void>] = []
                    for newCategory in categoriesToAdd {
                        categoryResults.append( try
Category.addCategory(newCategory,to: post,on: req))
                    }
                    for categoryNameToRemove in categoriesToRemove {
                        let categoryToRemove = existingCategories.first {
                            $0.name == categoryNameToRemove }
                        if let category = categoryToRemove {
categoryResults.append(
                            post.categories.detach(category, on: req)) }
                    }
                    let redirect = req.redirect(to: "/posts/\(id)")
                    return categoryResults.flatten(on: req)
                        .transform(to: redirect) }
        } }

    func editEventPOSTHandler(_ req: Request) throws -> Future<Response> {
        return try flatMap(to: Response.self,
req.parameters.next(Event.self), req.content.decode(CreateEventData.self)
        ) { event, data in
            let user = try req.requireAuthenticated(User.self)
            event.date = data.date
            event.start = data.start
            event.end = data.end
            event.title = data.title
            event.type = data.type
            event.text = data.text
            event.link = data.link
            event.userID = try user.requireID()
            let redirect = req.redirect(to: "/events/")
            return event.save(on: req).transform(to: redirect)
        } }

    func editCategoryPOSTHandler(_ req: Request) throws -> Future<Response> {
        return try flatMap(to: Response.self,
req.parameters.next(Category.self), req.content.decode(Category.self)
        ) { category, data in
            category.name = data.name
            let redirect = req.redirect(to: "/categories/")
            return category.save(on: req).transform(to: redirect)
        } }
```

```
    func deletePostHandler(_ req: Request) throws -> Future<Response> {
        return try req.parameters.next(Post.self).delete(on: req)
            .transform(to: req.redirect(to: "/"))
    }

    func deleteEventHandler(_ req: Request) throws -> Future<Response> {
        return try req.parameters.next(Event.self).delete(on: req)
            .transform(to: req.redirect(to: "/events/"))
    }

    func deleteCategoryHandler(_ req: Request) throws -> Future<Response> {
        return try req.parameters.next(Category.self).delete(on: req)
            .transform(to: req.redirect(to: "/categories/"))
    }

    func loginHandler(_ req: Request) throws -> Future<View> { let context:
LoginContext
       if req.query[Bool.self, at: "error"] != nil {
            context = LoginContext(loginError: true)
        } else {
            context = LoginContext()
        }
       return try req.view().render("login", context)
    }

    func loginPostHandler(_ req: Request,userData: LoginPostData) throws ->
Future<Response> {
        return User.authenticate( username: userData.username, password:
userData.password, using: BCryptDigest(),
                                  on: req).map(to: Response.self) { user in
                                    guard let user = user else {
                                        return req.redirect(to:
"/login?error")
                                    }
                                    try req.authenticateSession(user)
                                    return req.redirect(to: "/")
        } }

    func logoutHandler(_ req: Request) throws -> Response {
        try req.unauthenticateSession(User.self)
        return req.redirect(to: "/")
    }

    func registerHandler(_ req: Request) throws -> Future<View> {
        let context: RegisterContext
        if let message = req.query[String.self, at: "message"] {
            context = RegisterContext(message: message)
        } else {
            context = RegisterContext()
        }
        return try req.view().render("register", context)
    }

    func registerPostHandler(_ req: Request, data: RegisterData) throws ->
Future<Response> {
        // Validation start
        do {
            try data.validate()
        } catch (let error) { // Adding error messege
            let redirect: String
            if let error = error as? ValidationError,
                let message = error.reason.addingPercentEncoding(
withAllowedCharacters: .urlQueryAllowed) {
```

```
                redirect = "/register?message=\(message)"
            } else {
                redirect = "/register?message=Unknown+error"
            }
            // End
            return req.future(req.redirect(to: redirect)) }
        let password = try BCrypt.hash(data.password)
        let user = User(name: data.name, username: data.username, password:
password, email: data.emailAddress)
        return user.save(on: req).map(to: Response.self) { user in
            try req.authenticateSession(user)
            return req.redirect(to: "/")
        } }

    func forgottenPasswordHandler(_ req: Request) throws -> Future<View> {
        return try req.view().render(
            "forgottenPassword",
            ["title": "Зміна паролю"])
    }

    func forgottenPasswordPostHandler(_ req: Request) throws -> Future<View>
{
        let email = try req.content.syncGet(String.self, at: "email")
        return User.query(on: req) .filter(\.email == email) .first()
            .flatMap(to: View.self) { user in
                guard let user = user else { return try req.view().render(
                    "forgottenPasswordConfirmed",
                    ["title": "Email для зміни паролю відправлено"])
                }
                let resetTokenString = try CryptoRandom()
.generateData(count: 32) .base32EncodedString()
                let resetToken = try ResetPasswordToken( token:
resetTokenString,
                                                         userID:
user.requireID())
                return resetToken.save(on: req).flatMap(to: View.self) { _ in
                    let emailContent = """
                    <p>Ви зробили запит на зміну паролю. <a
href="http://localhost:8080/resetPassword?token=\(resetTokenString)">
                    Натисніть тут</a> щоб змінити пароль.</p>
                    """
                    let emailAddress = EmailAddress(
                        email: user.email,
                        name: user.name)
                    let fromEmail = EmailAddress(
                        email: "krivodubag@gmail.com",name: "InteractAPI")
                    let emailConfig = Personalization(
                        to: [emailAddress],
                        subject: "Зміна паролю")
                    let email = SendGridEmail(
                        personalizations: [emailConfig],
                        from: fromEmail,
                        content: [
                            ["type": "text/html",
                             "value": emailContent]
                    ])
                    let sendGridClient = try req.make(SendGridClient.self)
                    return try sendGridClient.send([email], on:
req.eventLoop)
                        .flatMap(to: View.self) { _ in
                            return try req.view().render(
"forgottenPasswordConfirmed",
```

```
                                                        ["title": "Email
для зміни паролю відправлено"]
                                )
                    } }
        } }

    func resetPasswordHandler(_ req: Request) throws -> Future<View> {
        guard let token = req.query[String.self, at: "token"] else {
            return try
req.view().render("resetPassword",ResetPasswordContext(error: true))
        }
        return ResetPasswordToken.query(on: req) .filter(\.token == token)
            .first().map(to: ResetPasswordToken.self) { token in
                guard let token = token else {
                    throw Abort.redirect(to: "/") }
                return token }.flatMap { token in
                    return token.user.get(on: req).flatMap { user in
                        try req.session().set("ResetPasswordUser", to: user)
                        return token.delete(on: req)
                    } }.flatMap {
                        try req.view().render( "resetPassword",
                                               ResetPasswordContext()
                        )
        } }

    func resetPasswordPostHandler(
        _ req: Request,
        data: ResetPasswordData) throws -> Future<Response> {
        // 2
        guard data.password == data.confirmPassword else { return try
req.view().render("resetPassword", ResetPasswordContext(error: true))
.encode(for: req)
        }
        let resetPasswordUser = try req.session() .get("ResetPasswordUser",
as: User.self)
        try req.session()["ResetPasswordUser"] = nil
        let newPassword = try BCrypt.hash(data.password)
        resetPasswordUser.password = newPassword
        return resetPasswordUser
            .save(on: req)
            .transform(to: req.redirect(to: "/login"))
    }

    func addProfilePictureHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(User.self) .flatMap { user in
            try req.view().render( "addProfilePicture", ["title": "Додати
зображення", "username": user.name])
        } }

    func addProfilePicturePostHandler(_ req: Request) throws ->
Future<Response> {
        return try flatMap(
        to: Response.self, req.parameters.next(User.self),
req.content.decode(ImageUploadData.self)) {
            user, imageData in
            let workPath = try req.make(DirectoryConfig.self).workDir
            let name = try "\(user.requireID())-\(UUID().uuidString).jpg"
            let path = workPath + self.imageFolder + name
            FileManager().createFile(atPath: path, contents:
imageData.picture, attributes: nil)
            user.profilePicture = name
            let redirect =
                try req.redirect(to: "/users/\(user.requireID())")
```

```
            return user.save(on: req).transform(to: redirect)
        } }

    func getUsersProfilePictureHandler(_ req: Request) throws ->
Future<Response> {
        return try req.parameters.next(User.self) .flatMap(to: Response.self)
{ user in
            guard let filename = user.profilePicture else { throw
Abort(.notFound)
            }
            let path = try req.make(DirectoryConfig.self)
                .workDir + self.imageFolder + filename
            return try req.streamFile(at: path) }
    }

    func addPostPictureHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Post.self) .flatMap { post in
            try req.view().render( "addPostPicture", ["title": "Додати
зображення", "postTitle": post.title])
        } }

    func addPostPicturePOSTHandler(_ req: Request) throws -> Future<Response>
{
        return try flatMap(
        to: Response.self, req.parameters.next(Post.self),
req.content.decode(FileUploadData.self)) {
            post, imageData in
            let workPath = try req.make(DirectoryConfig.self).workDir
            let name = "\(imageData.name)"
            let path = workPath + self.imageFolder + name
            FileManager().createFile(atPath: path, contents: imageData.file,
attributes: nil)
            post.postPicture = name
            let redirect = req.redirect(to: "/posts/\(post.id!)")
            return post.save(on: req).transform(to: redirect)
        } }

    func getPostsPictureHandler(_ req: Request) throws -> Future<Response> {
        return try req.parameters.next(Post.self) .flatMap(to: Response.self)
{ post in
            guard let filename = post.postPicture else { throw
Abort(.notFound)
            }
            let path = try req.make(DirectoryConfig.self)
                .workDir + self.imageFolder + filename
            return try req.streamFile(at: path)
        } }



    func addPostFileHandler(_ req: Request) throws -> Future<View> {
        return try req.parameters.next(Post.self) .flatMap { post in
            try req.view().render( "addPostFile", ["title": "Додати файл",
"postTitle": post.title])
        } }

    func addPostFilePOSTHandler(_ req: Request) throws -> Future<Response> {
        return try flatMap(
        to: Response.self, req.parameters.next(Post.self),
req.content.decode(FileUploadData.self)) {
            post, fileData in
            let workPath = try req.make(DirectoryConfig.self).workDir
            let name = "\(fileData.name)"
```

```
            let path = workPath + self.imageFolder + name
            FileManager().createFile(atPath: path, contents: fileData.file,
attributes: nil)
            post.postFile = name
            let redirect = req.redirect(to: "/posts/\(post.id!)")
            return post.save(on: req).transform(to: redirect)
        } }

    func getPostsFileHandler(_ req: Request) throws -> Future<Response> {
        return try req.parameters.next(Post.self) .flatMap(to: Response.self)
{ post in
            guard let filename = post.postFile else { throw Abort(.notFound)
            }
            let path = try req.make(DirectoryConfig.self)
                .workDir + self.imageFolder + filename
            return try req.streamFile(at: path)
        } }

}

struct AllUsersContext: Encodable {
    let title: String
    let users: [User]
    let userLoggedIn: Bool
}

struct AllPostsContext: Encodable {
    let title: String
    let posts: [Post]
    let userLoggedIn: Bool
}

struct AllEventsContext: Encodable {
    let title: String
    let events: [Event]
    let userLoggedIn: Bool
}

struct AllCategoriesContext: Encodable {
    let title = "Категорії"
    let categories: Future<[Category]>
    let userLoggedIn: Bool
}

struct UserContext: Encodable {
    let title: String
    let user: User
    let posts: [Post]
    let userLoggedIn: Bool
    let authenticatedUser: User?
}

struct PostContext: Encodable {
    let title: String
    let post: Post
    let user: User
    let categories: Future<[Category]>
    let userLoggedIn: Bool
}

struct EventContext: Encodable {
    let title: String
    let event: Event
```

```
    let user: User
    let userLoggedIn: Bool
}

struct CategoryContext: Encodable {
    let title: String
    let category: Category
    let posts: Future<[Post]>
    let userLoggedIn: Bool
}

struct CreatePostContext: Encodable {
    let title = "Створити оголошення"
    let userLoggedIn: Bool
}

struct CreateEventContext: Encodable {
    let title = "Створити подію"
   let userLoggedIn: Bool
}
struct CreateCategoryContext: Encodable {
    let title = "Створити категорію"
   let userLoggedIn: Bool
}

struct EditPostContext: Encodable {
    let title = "Змінити оголошення"
    let post: Post
   let editing = true
    let categories: Future<[Category]>
    let userLoggedIn: Bool
}
struct EditEventContext: Encodable {
    let title = "Змінити подію"
    let event: Event
   let editing = true
    let userLoggedIn: Bool
}
struct EditCategoryContext: Encodable {
    let title = "Змінити категорію"
    let category: Category
   let editing = true
    let userLoggedIn: Bool
}
struct CreatePostData: Content {
   let title: String
    let text: String
    let categories: [String]?
}
struct CreateEventData: Content {
   let date: String
    let start: String
    let end: String
    let title: String
    let type: String
    let text: String
    let link: String
}
struct LoginContext: Encodable {
    let title = "Вхід"
    let loginError: Bool
    init(loginError: Bool = false) {
        self.loginError = loginError
```

```
    }
}
struct LoginPostData: Content {
    let username: String
    let password: String
}
struct RegisterContext: Encodable {
    let title = "Реєстрація"
    // Error messege start
    let message: String?
    init(message: String? = nil) {
        self.message = message
    }
    // End
}
struct RegisterData: Content {
    let name: String
    let username: String
    let password: String
    let confirmPassword: String
    let emailAddress: String
}
extension RegisterData: Validatable, Reflectable {
    static func validations() throws -> Validations<RegisterData> {
        var validations = Validations(RegisterData.self)
        try validations.add(\.name, .ascii)
        try validations.add(\.username, .alphanumeric && .count(3...))
        try validations.add(\.password, .count(8...))
        try validations.add(\.emailAddress, .email)
        // Custom validation start
        validations.add("passwords match") { model in
            guard model.password == model.confirmPassword else {
                throw BasicValidationError("passwords don’t match")
            }
        }
        // End
        return validations
    } }

struct ResetPasswordContext: Encodable {
    let title = "Зміна паролю"
    let error: Bool?
    init(error: Bool? = false) {
        self.error = error }
}
struct ResetPasswordData: Content {
    let password: String
    let confirmPassword: String
}

struct FileUploadData: Content {
    var name: String
    var file: Data
}

struct ImageUploadData: Content {
    var picture: Data
}
```

Файл routes.swift

```
import Vapor
import Fluent

public func routes(_ router: Router) throws {
    let postsController = PostsController()
    try router.register(collection: postsController)

    let eventsController = EventsController()
    try router.register(collection: eventsController)

    let usersController = UsersController()
    try router.register(collection: usersController)

    let categoriesController = CategoriesController()
    try router.register(collection: categoriesController)

    let websiteController = WebsiteController()
    try router.register(collection: websiteController)

    let imperialController = ImperialController()
    try router.register(collection: imperialController)
}
```

Додаток Б

Програмний код проекту створення Web додатку.

Папка Views

Файл post.leaf

```
#set("content") {
<table class="table table-bordered">
    <tbody>
        <tr>
            <td>
                <h1 style="text-align:center">#(post.title)</h1>

                #if(post.postPicture) {
                <div style="text-align:center">
                    <img style="border-radius:8px; width:300px"
src="/posts/#(post.id)/postPicture" alt="#(post.title)">
                        </div><br>
                }

                <div style="text-align:center">
                    #if(post.postPicture){
                    <a class="btn btn-primary"
href="/posts/#(post.id)/addPostPicture" role="button">Змінити зображення</a>
                    } else {
                    <a class="btn btn-success"
href="/posts/#(post.id)/addPostPicture" role="button">Додати зображення</a>
                    }
                </div><br>

                <div class="form-group">
                    <textarea rows="1" style="height:1em;border:none"
class="form-control" id="text">#(post.text)</textarea>
                </div>
```

```
                <p>Створено користувачем <a
href="/users/#(user.id)/">#(user.name)</a><br>
                #if(count(categories) > 0) {
                Категорії:
                #for(category in categories) {
                <a href="/categories/#(category.id)">#(category.name)</a>
                }
                }<br>

                #if(post.postFile) {
                Файли: <a
href="/posts/#(post.id)/postFile/">#(post.postFile)</a><br>
                }
                #//#if(post.postFile){
                <div>
                    #//<a class="btn btn-primary"
href="/posts/#(post.id)/addPostFile" role="button">Змінити файл</a>
                    #//} else {
                    <a class="btn btn-success"
href="/posts/#(post.id)/addPostFile" role="button">Додати файл</a>
                    #//}
                </div><br>

                <div style="text-align:center">
                    <button id="write-btn" class="btn btn-info">Скопіювати
посилання на це оголошення</button>
                </div><br>

                <form method="post" action="/posts/#(post.id)/delete"
style="text-align:center">
                    #if(userLoggedIn){
                    <a class="btn btn-primary" href="/posts/#(post.id)/edit"
role="button">Змінити</a> 
                    <input class="btn btn-danger" type="submit"
value="Видалити" />
                    }
                </form>
            </td>
        </tr>
    </tbody>
</table>
}
#embed("base")
```

Файл allUsers.leaf

```
#set("content") {
<table class="table">
    <thead class="thead-dark">
        <tr>
            <th>Всі користувачі</th>
        </tr>
    </thead>
    <tbody>
    </tbody>
</table>
#if(count(users) > 0) {
#for(user in users) {
<table class="table table-bordered">
    <tbody>
        <tr >
            <td>
```

```
                <h2 style="text-align:center">#(user.name)<h2>
                    <h5 style="text-align:center">#(user.username)</h5><br>
                    <div style="text-align:center">
                        <a class="btn btn-info" href="/users/#(user.id)"
role="button">Детальніше</a>
                    </div>
            </td>
        </tr>
    </tbody>
</table>
}
} else {
<h4 style="text-align:center">Тут ще не має користувачів!</h4>
}
}
#embed("base")
```

Файл base.leaf

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
            <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
                <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.cs
s" integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">

                    #if(title == "Створити оголошення" || title == "Змінити
оголошення") {
                    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/css/select2.min.css" integrity="sha384-
RdQbeSCGSeSdSlTMGnUr2oDJZzOuGjJAkQy1MbKMu8fZT5G0qlBajY0n0sY/hKMK"
crossorigin="anonymous">
                        }
                        <title>#(title)</title>
                        </head>
    <body onload="init();">
        <nav class="navbar navbar-expand navbar-dark bg-dark fixed-top">
            #//<a class="navbar-brand" href="/">Interact</a>
            <button class="navbar-toggler" type="button"
                data-toggle="collapse" data-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
                <span class="navbar-toggler-icon"></span> </button>
            <div class="collapse navbar-collapse"
                id="navbarSupportedContent">
                <ul class="navbar-nav mr-auto">
                    <li class="nav-item #if(title == "Користувачі"){active}">
<a href="/users" class="nav-link">Користувачі</a>
                    </li>
                    <li class="nav-item #if(title == "Оголошення"){active}">
<a href="/" class="nav-link">Оголошення</a>
                    </li>
                    <li class="nav-item #if(title == "Розклад"){active}"> <a
href="/events" class="nav-link">Розклад</a>
                    </li>
```

```
                    <li class="nav-item #if(title == "Категорії"){active}">
<a href="/categories" class="nav-link">Категорії</a>
                    </li>
                </ul>
                #if(userLoggedIn) {
                <form class="form-inline" action="/logout" method="POST">
                    <input class="btn btn-outline-light" type="submit"
value="Вихід">
                        </form> } else {
                <form class="form-inline" action="/login" method="GET">
                    <input class="btn btn-outline-light" type="submit"
value="Вхід">
                        </form>
                }
            </div>
        </nav>
        <br><br><br>

        <div class="container mt-3">
            #get(content)
        </div>
        <script src="https://code.jquery.com/jquery-3.3.1.min.js"
integrity="sha384-
tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT"
crossorigin="anonymous"></script>
        #if(title == "Створити оголошення" || title == "Змінити оголошення")
{
        <script src="https://cdnjs.cloudflare.com/ajax/libs/select2/4.0.6-
rc.0/js/select2.min.js" integrity="sha384-
uQwKPrmNkEOvI7rrNdCSs6oS1F3GvnZkmPtkntOSIiPQN4CCbFSxv+Bj6qe0mWDb"
crossorigin="anonymous"></script>
        <script src="/scripts/createPost.js"></script>
        }
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.j
s" integrity="sha384-
ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
crossorigin="anonymous"></script>
        <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-
ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>
    </body>
</html>
```

Файл category.leaf

```
#set("content") {
<h1 style="text-align:center">#(category.name)</h1><br>
#embed("postsTable")
}
#embed("base")
```

Файл createPost.leaf

```
#set("content") {
<table class="table table-bordered">
    <tbody>
        <tr>
            <td>
                <h1 style="text-align:center">#(title)</h1>
```

```
                <form method="post">
                    <div class="form-group">
                        <label for="title">Назва</label>
                        <input type="text" name="title" class="form-control"
id="title" #if(editing){value="#(post.title)"}/>
                    </div>

                    <div class="form-group">
                        <label for="text">Зміст</label>
                        <textarea rows = "5" name="text" class="form-control"
id="text">#if(editing){#(post.text)}</textarea><br>
                    </div>

                    <div class="form-group">
                        <label for="categories">Категорії</label>
                        <select name="categories[]" class="form-control"
id="categories" placeholder="Категорії" multiple="multiple">
                            #if(editing) {
                            #for(category in categories) {
                            <option value="#(category.name)"
selected="selected"> #(category.name)
                            </option> }
                            }
                        </select>
                    </div>

                    <div style="text-align:center">
                        <input class="btn btn-secondary" type="reset"
value="Очистити"> 
                            #if(editing){
                            <button type="submit" class="btn btn-
primary">Змінити</button>
                            } else {
                            <button type="submit" class="btn btn-
success">Створити</button>
                            }
                            </div>
                </form>
            </td>
        </tr>
    </tbody>
</table>
}
#embed("base")
```

Файл index.leaf

```
#set("content") {
#embed("postsTable")
}
#embed(“base")
```

Файл login.leaf

```
#set("content") {
<h1 style="text-align:center">#(title)</h1>
#if(loginError) {
<div class="alert alert-danger" role="alert">
    Помилка входу. Перевірте правильність вводу логіну або паролю.
</div> }
<form method="post">
    <div class="form-group">
```

```
        <label for="username">Логін</label>
        <input type="text" name="username" class="form-control"
id="username"/>
    </div>
    <div class="form-group">
        <label for="password">Пароль</label> <input type="password"
name="password" class="form-control" id="password"/>
    </div>
    <div style="text-align:center">
        <button type="submit" class="btn btn-success">Вхід</button> 
        <a class="btn btn-primary" href="/register"
role="button">Реєстрація</a>
    </div>
</form>
<div style="text-align:center">
    <a href="/login-google">
        <img class="mt-3" src="/images/btn_google_signin_dark_normal_web.png"
alt="Вхід з Google"></a><br>
    <a href="/forgottenPassword">Забули свій пароль?</a>
</div>
}
#embed("base")
```

Файл register.leaf

```
#set("content") {
<h1 style="text-align:center">#(title)</h1>

#if(message) {
<div class="alert alert-danger" role="alert"> Please fix the following
errors:<br /> #(message)
</div> }

<form method="post">
    <div class="form-group">
        <label for="name">Ім'я</label>
        <input type="text" name="name" class="form-control"
        id="name"/>
    </div>
    <div class="form-group">
        <label for="username">Логін</label>
        <input type="text" name="username" class="form-control"
        id="username"/>
    </div>

    <div class="form-group">
        <label for="emailAddress">Email</label>
        <input type="email" name="emailAddress" class="form-control"
        id="emailAddress"/>
    </div>

    <div class="form-group">
        <label for="password">Пароль</label> <input type="password"
name="password"
        class="form-control" id="password"/>
    </div>

    <div class="form-group">
        <label for="confirmPassword">Підтвердження паролю</label> <input
type="password" name="confirmPassword"
        class="form-control" id="confirmPassword"/> </div>
    <div style="text-align:center">
```

```
        <button type="submit" class="btn btn-
success">Зареєструватися</button>
    </div>
</form>
}
#embed("base")
```

Файл user.leaf

```
#set("content") {
<h1 style="text-align:center">#(user.name)</h1>
<h3 style="text-align:center">#(user.username)</h3>

<div style="text-align:center">
    #if(user.profilePicture) {
    #//<a
href="/users/#(user.id)/profilePicture/">#(user.profilePicture)</a><br>
    <img style="border-radius:8px; width:300px"
src="/users/#(user.id)/profilePicture" alt="#(user.name)">
        }
</div><br>

<div style="text-align:center">
    #if(authenticatedUser) {
    #if(user.profilePicture){
    <a class="btn btn-primary" href="/users/#(user.id)/addProfilePicture"
role="button">Змінити зображення</a>
    } else {
    <a class="btn btn-success" href="/users/#(user.id)/addProfilePicture"
role="button">Додати зображення</a>
    }}
</div><br>

#embed("postsTable")
}
#embed("base")
```

Файл event.leaf

```
#set("content") {
<table class="table table-bordered">
    <tbody>
        <tr>
            <td>
                <h4 style="text-align:center">#(event.date)</h4>
                <h1 style="text-align:center">#(event.start) -
#(event.end)</h1>
                <h3 style="text-align:center">#(event.title)</h3>
                <h4>#(event.type)<br>
                    #(event.text)<br>
                    <a href="#(event.link)">#(event.link)</a></h4>

                <p>Створено користувачем <a
href="/users/#(user.id)/">#(user.name)</a></p>
                <form method="post" action="/events/#(event.id)/delete"
style="text-align:center">
                    #if(userLoggedIn){
                    <a class="btn btn-primary"
href="/events/#(event.id)/edit" role="button">Змінити</a> 
                    <input class="btn btn-danger" type="submit"
value="Видалити" />
                    }
```

```
                </form>
            </td>
        </tr>
    </tbody>
</table>
}
#embed("base")
```

Файл createEvent.leaf

```
#set("content") {
<table class="table table-bordered">
    <tbody>
        <tr>
            <td>
                <h1 style="text-align:center">#(title)</h1>
                <form method="post">
                    <div class="form-group">
                        <label for="date">Дата</label>
                        <input type="date" name="date" class="form-control"
id="date" #if(editing){value="#(event.date)"}/>
                    </div>
                    <div class="form-group">
                        <label for="title">Час початку</label>
                        <input type="time" name="start" class="form-control"
id="start"  #if(editing){value="#(event.start)"}/>
                    </div>
                    <div class="form-group">
                        <label for="title">Час закінчення</label>
                        <input type="time" name="end" class="form-control"
id="end" #if(editing){value="#(event.end)"}/>
                    </div>
                    <div class="form-group">
                        <label for="title">Назва</label>
                        <input type="text" name="title" class="form-control"
id="title" #if(editing){value="#(event.title)"}/>
                    </div>
                    <div class="form-group">
                        <label for="text">Тип</label>
                        <input type="text" name="type" class="form-control"
id="type" #if(editing){value="#(event.type)"}/>
                    </div>
                    <div class="form-group">
                        <label for="text">Коментар</label>
                        <input type="text" name="text" class="form-control"
id="text" #if(editing){value="#(event.text)"}/>
                    </div>
                    <div class="form-group">
                        <label for="link">Посилання</label>
                        <input type="text" name="link" class="form-control"
id="link" #if(editing){value="#(event.link)"}/>
                    </div>
                    <div style="text-align:center">
                        <input class="btn btn-secondary" type="reset"
value="Очистити"> 
                            #if(editing){
                            <button type="submit" class="btn btn-
primary">Змінити</button>
                            } else {
                            <button type="submit" class="btn btn-
success">Створити</button>
                            }
```

```
                            </div>
                </form>
            </td>
        </tr>
    </tbody>
</table>
}
#embed("base")
```

Файл allEvents.leaf

```
#set("content") {
<table class="table">
    <thead class="thead-dark">
        <tr>
            <th>Всі події</th>
        </tr>
    </thead>
    <tbody>
    </tbody>
</table>
#if(count(events) > 0) {
<div style="text-align:center">
    <form method="get" action="/events/create">
        <button type="submit" class="btn btn-success">Створити подію</button>
    </form><br>
</div>
#for(event in events) {
<table class="table table-bordered">
    <tbody>
        <tr>
            <td style="text-align:center">
                <h6>#(event.date)</h6>
                <h3>#(event.start) - #(event.end)</h3>
                <p style="text-align:left">#(event.title)<br>
                <i>#(event.type)</i><br>
                #(event.text)<br>
                <a href="#(event.link)">#(event.link)</><br>
                    <form method="post" action="/events/#(event.id)/delete">
                        <a class="btn btn-info" href="/events/#(event.id)"
role="button">Детальніше</a> 
                        #if(userLoggedIn){
                        <a class="btn btn-primary"
href="/events/#(event.id)/edit" role="button">Змінити</a> 
                        <input class="btn btn-danger" type="submit"
value="Видалити"/>
                        }
                    </form>
                    </td>
        </tr>
    </tbody>
</table>
}
} else {
<h4 style="text-align:center">Тут ще не має подій!</h4>
<div style="text-align:center">
    <form method="get" action="/events/create">
        <button type="submit" class="btn btn-success">Створити подію</button>
    </form><br>
</div>
}
}
```

```
#embed("base")
```

Папка scripts

Файл cookies.js

```
function cookiesConfirmed() {
  $('#cookie-footer').hide();
  var d = new Date();
  d.setTime(d.getTime() + (365*24*60*60*1000));
  var expires = "expires="+ d.toUTCString();
  document.cookie = "cookies-accepted=true;" + expires;
}
```

Файл createPost.js

```
$.ajax({
  url: "/api/categories/",
  type: "GET",
  contentType: "application/json; charset=utf-8"
}).then(function (response) {
  var dataToReturn = [];
  for (var i=0; i < response.length; i++) {
    var tagToTransform = response[i];
    var newCategory = {
      id: tagToTransform["name"],
      text: tagToTransform["name"]
    };
    dataToReturn.push(newCategory);
  }
  $("#categories").select2({
    placeholder: "Додати категорію",
    tags: true,
    tokenSeparators: [','],
    data: dataToReturn
  });
});
```

Папка styles

Файл style.css

```
#cookie-footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  height: 60px;
  line-height: 60px;
  background-color: #f5f5f5; }
```

Додаток В

Перевірка роботи API знаходиться в таблиці Г.1 .

Таблиця В.1 – Перевірка роботи API

Дія	Показати всіх користувачів
Request	**GET** /api/users

Response Body	[
	{
	"id": "1922B0BA-57AC-42D4-BE1E-E44F52701410",
	"name": "Admin",
	"username": "admin"
	},
	{
	"id": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26",
	"name": "Putivec Nastya",
	"username": "putiva"
	},
	{
	"id": "791D1A2B-ABB9-4D93-8614-DE76CA1CF344",
	"name": "Александр Геннадьевич",
	"username": "krivodubag@gmail.com"
	}
	]

Дія	Показати корстувача за ID
Request	**GET** /api/users/4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26

Response Body	{
	"id": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26",
	"name": "Putivec Nastya",
	"username": "putiva"
	}

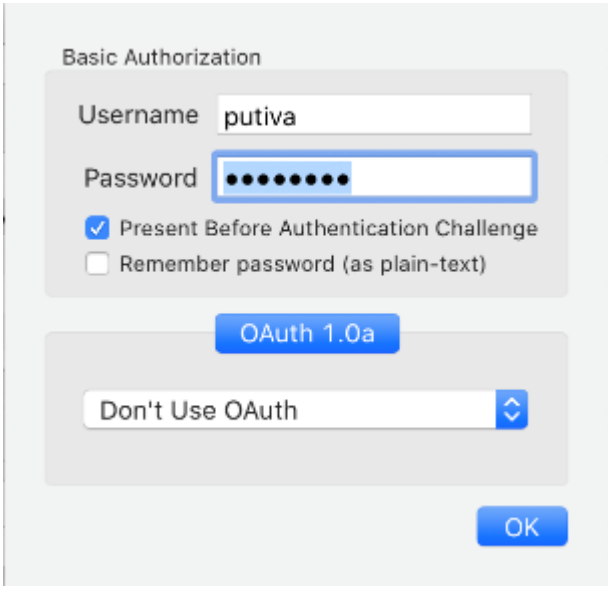
Дія	Додати користувача
Request	**POST** /api/users

Parameters:	{
	"email": "alice@gmail.com",
	"name": "Alice",
	"password": "password",
	"username": "alice"
	}

Response Body	{
	"id": "B3E2DBDD-95FF-4B0C-AB30-29A351DE7F62",
	"name": "Alice",
	"username": "alice"
	}

Дія	Показати всі оголошення
Request	**GET** /api/postss

Response Body	[
	{
	"id": 1,
	"title": "Строки переддипломної практики для спец. \"Комп'ютерні науки\"",
	"content": "Термін захисту звітів з практики: з 27 квітня по 03 травня 2020 року.",
	"userID": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26"
	},
	{
	"id": 2,
	"title": "Захист переддипломної практики",
	"content": "Надсилайте на o.protsenko@cs.sumdu.edu.ua",
	"userID": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26"
	}
	]

Дія	Увійти в систему
Request	**POST** /api/users/login
Parameters	

Response Body	{
	"id": "DBAB0246-4D6E-49DD-AF96-0296707F84A6",
	"token": "bQnPAqOsTfY0F4RHy2b3jw==",
	"userID": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26"
	}

Дія	Створити оголошення (потрібен токен авторизації)
Request	**POST** /api/postss
Headers & Body	Authorization: Bearer bQnPAqOsTfY0F4RHy2b3jw==

Parameters	{
	"title": "Назва оголошення 1",
	"content": "Зміст оголошення 1",
	}

Response Body	<pre>{
	    "id": 3,
	    "title": "Назва оголошення 1",
	    "content": "Зміст оголошення 1",
	    "userID": "4CE0D2F3-AB69-483C-9EE6-4A2B74AD9D26"
	}</pre>

Дія	Показати всі категорії
Request	**GET** /api/tags

Response Body	<pre>[
	    {
	        "id": 1,
	        "name": "Практика"
	    },
	    {
	        "id": 2,
	        "name": "4й курс"
	    },
	    {
	        "id": 3,
	        "name": "Бази даних"
	    }
	]</pre>

Дія	Створити категорію (потрібен токен авторизації)
Request	**POST** /api/tags
Headers & Body	Authorization: Bearer IorsCgVFCFvfrDPSWIGA9g==

Parameters:	<pre>{
	  "name": "Комп'ютерна графіка"
	}</pre>

Дія	Показати всі категорії оголошення 2
Request	**GET** /api/postss/2/tags

Response Body	<pre>[
	    {
	        "id": 1,
	        "name": "Практика"
	    },
	    {
	        "id": 2,
	        "name": "4й курс"
	    }
	]</pre>

Дія	Додати категорію 3 до оголошення 2
Request	**POST** /api/postss/2/tags/3
Headers & Body	Authorization: Bearer IorsCgVFCFvfrDPSWIGA9g==
Response Headers	HTTP/1.1 201 Created
Response Body	

Дія	Зміна оголошення
Request	**PUT** /api/postss/3
Headers & Body	Authorization: Bearer bQnPAqOsTfY0F4RHy2b3jw==

Response Body	{
	"title": "Змінена назва оголошення 1",
	"content": "Змінений зміст оголошення 1",
	}

Дія	Видалити оголошення
Request	**DELETE** /api/postss/3 Authorization: Bearer bQnPAqOsTfY0F4RHy2b3jw==
Response Headers	HTTP/1.1 204 No Content
Response Body	