ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004.415

# КВАЛІФІКАЦІЙНА РОБОТА

Тема **Розробка вебзастосунку агрегації розважальних подій по геолокаціях з використанням арі**

***121 «Інженерія Програмного Забезпечення»***

Спеціальність

(код і назва спеціальності)

Студент

*Гринішак С.В.*

(підпис) (дата) (розшифрування підпису)

Керівник проекту

*к.т.н. Ващишак С. П.*

Допускається до захисту

Завідувач кафедри

*д.т.н.,доц. Мануляк І.З.*

| Зм. | Арк. | № докум. | Підпис | Дата | Розробка вебзастосунку агрегації розважальних подій по геолокаціях з використанням арі | Літ. | Арк. | Аркушів |
|---|---|---|---|---|---|---|---|---|
| Розроб. | | Гринішак С. В. | | | | У | 5 | 75 |
| Перевір. | | Ващишак С.П. | | | | | | |
| Реценз. | | Дячишин І. М. | | | | УКД ІПЗс-2016 | | |
| Н. Контр. | | | | | | | | |
| Затверд. | | Мануляк І.З. | | | | | | |

**ЗМІСТ**

# ВСТУП

Інтенсивно набувають популярності так звані онлайн-сервіси агрегатори, які дозволяють людям отримати саме те, що їм потрібно, відсікаючи величезні обсяги непотрібних даних. Зважаючи на це, зростає час уведення, обробки та аналізу цих даних. Сфера розваг не є виключенням.

На даний час розроблена велика кількість найрізноманітніших програмних застосунків для обслуговування людських потреб. Щодня ми маємо змогу бачити величезний наплив реклами, промо-акцій та інших засобів, якими виробники програмного забезпечення намагаються збільшити кількість користувачів своїх програм. Вони відрізняються функціоналом, вартістю, сферою обслуговування та колом користувачів, що зацікавлені у користуванні цією програмою. Одними з найбільш відомих агрегаторів контенту в Україні є ukr.net - агрегує новини з величезної кількості інших веб-ресурсів, prom.ua - торгова платформа для бізнесу, де розміщують свої товари десятки а то і сотні інтернет-магазинів, hotline.ua – агрегатор даних про інтернет магазини, спеціалізується на товарах та порівнянні їх цін на різних платформах. Попри те, що з кожним роком агрегаторів стає все більше, деякі області людського життя вони все ще не покрили.

Веб-ресурс та програмна частина застосунку, описаного в цій роботі, покриває одну з цих областей – події. На сьогоднішній день існує невелика кількість агрегаторів подій, що допомагають користувачам отримати повний набір інформації про подію, яку вони хочуть відвідати, а таких зараз величезне різноманіття: концерти, вечірки, лекції, майстер-класи і ще безліч різних можливостей для цікавого проведення вільного часу. Та все ж існуючі програмні рішення далеко не повністю задовольняють потреби користувачів, наприклад мають не надто зручний для користувача інтерфейс, забирають його увагу непотрібними даними або ж не надають інформацію у зручному для аналізу

вигляді. Також на даний момент не існує єдиного агрегатора подій з усього світу з можливістю зручного їх перегляду.

Проаналізувавши деякі існуючі веб-додатки агрегатори подій можна зробити висновок, що найчастіше у них або немає API взагалі, або він не задовольняє вимогам кінцевого користувача. API для веб-проектів досить широко поширилися за останні пару років.

Сьогодні API є незамінним інструментом кожного веб-розробника і, більш того, всѐ більше і більше API дуже ефективно застосовуються в маркетингу для багатьох типів бізнесу. Веб API дозволяють розробникам створювати веб-сторінки і веб-додатки, використовуючи дані з декількох джерел відразу. Додатки, які використовують API, істотно виграють на тлі додатків свого типу.

**Метою роботи є**: створення сайту, який буде виводити користувачу події по його запиту, або місцезнаходженню, а також реалізувати функціонал форм реєстрації.

**Об’єкт дослідженнь:** процесс збереження подій розважальних заходів.

**Задачі дослідженнь:**

- проаналізувати існуючі вебзастосунки агрегації розважальних подій;
- вибрати технології розробки та архітектури веб-додатків;
- розробити загальну структуру вебзастосунку;
- розробити систему геолокації;
- розробити взаємодію з API.

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

БД (DB, Database) – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення, програмний застосунок;

API – прикладний програмний інтерфейс;

GPS – Система глобального позиціонування;

CLI – інтерфейс командного рядка;

HTTP – протокол передачі гіпер-текстових документів;

OAuth – (скорочення від англ. Open Authorization, відкрита авторизація);

Фреймворк – (Framework, каркас, платформа) ;

REST – передача репрезентативного стану;

JSON – запис об'єктів JavaScript;

# 1. ОГЛЯД НАЯВНИХ РІШЕНЬ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

## 1.1 Загальні відомості про веб-ресурси агрегатори та аналіз інформаційних розважальних порталів

Діяльність агрегаторів різних типів даних тісно пов’язана з розробкою та постійним покращенням функціоналу додатку для постійного притоку нових користувачів. Вміння збирати дані, надавати їх користувачам у найзручнішому вигляді, з постійно актуальною інформацією дозволяє на вищому якісному рівні підходити до розробки та просування власного агрегатора.

Якщо ви користуєтесь інтернетом давно, то пам’ятаєте епоху, коли в магазинах продавались паперові довідники і журнали, які містили каталоги сайтів. Також були і сайти-довідники – «жовті сторінки інтернету». Потім в українському та російськомовному сегменті інтернету набув популярності “Рамблер”, а його вже наздогнав і обігнав “Google”.

Але ще до пошукачів були агрегатори, сайти-каталоги. І вони з часом не зникли, лише підлаштувались під потреби своєї аудиторії. Хоча деякі з них майже не міняли дизайн за останні 15-20 років.

Сайт-агрегатор («мешап», «Mash-up», змішувати) – веб-додаток або інтернет сайт, що об'єднує дані з декількох джерел в один з єдиним призначеним для користувача інтерфейсом; наприклад, сайти-агрегатори відгуків різних кіно- чи інших критиків (Rotten Tomatoes і Metacritic), сайти для покупки авіаквитків з можливістю вибору найбільш зручної пропозиції, сайти для бронювання готелю.

На території сучасної України використовуються різні програмні застосунки агрегатори. Деякі з існуючих програмних рішень є загальновідомими у суспільстві, як от Uber чи Hotline. Але всі існуючі аналоги мають свої переваги та недоліки.

Через відсутність достатньої кількості аналогічних проектів будуть розглянуті веб-ресурси агрегатори  такі як 0342.ua та frankivsk-event.com .

Основними завданнями веб-сайту є:

- реклама продукції, послуг, ідей. Правильно зроблений веб-сайт із легкістю приведе клієнта до висновку про необхідність покупки товару, або послуг, або ідей, що пропагуються на ньому;
- продаж товарів, послуг, інформації, ідей. У сучасної людини немає багато часу для ходіння по магазинах. Тому можливість замовлення товарів і послуг, не відходячи від комп'ютера, значно розширює можливості і клієнта, і продавця;
- безкоштовне надання інформації або послуг. Насправді надання інформації або послуг  це засіб залучення відвідувачів до даного ресурсу для здобуття, наприклад, статистичної інформації або ж для показу реклами, якщо це рекламний майданчик;
- підтримка клієнтів.

Відповідно до завдань існують такі типи веб-сайтів:

- рекламні веб-сайти;
- веб-сайти-продавці;
- веб-сайти-альтруїсти;
- веб-сайти для підтримки.

Складові веб-сайту:

- дизайн сайту;
- контент сайту;
- код сайту.

На сьогоднішній день існують кілька етапів розробки веб-сайту, виділимо з них основні:

- проектування сайту;
- створення макетів сторінок;
- верстка сторінок і дизайнів;

– програмування;

– тестування та внесення коригувань;

– відкриття проекту на хостингу;

– обслуговування сайту.

Визначення структури сайту містить у собі планування розділів, системи навігації по сайті. Як буде виглядати меню сайту, де і як будуть розташовані ті або інші модулі. Відповіді на ці питання дозволяють уже на етапі проектування представити, як буде виглядати майбутній сайт .

На проектуванні сайту визначаються всі завдання, які ставляться перед командою, що розробляє проектування або, як ще кажуть, прототипування сайту. В результаті командної роботи, замовник отримує готовий прототип (макет) сайту, із зручною для користувача навігацією, продуманою структурою і концепцією дизайну, також повне ТЗ (технічне завдання).

Макет показує, як будуть виглядати різні сторінки, де буде розташовуватися інформація, де буде перебувати навігаційні й керуючі елементи тощо Нарешті, на основі макета проробляється зовнішній вигляд кожної категорії сторінок. Наприклад, розробляється дизайн сторінок з новинами, дизайн сторінок з інформацією про продукт тощо Фактично дизайнер вивчає технічне завдання і малює шаблони для всього, що там описано.

Інтерфейс сайту, який розроблений дизайнером, – це ще тільки макет остаточного інтерфейсу сайту. Фактично, він складається просто з набору малюнків. Для того, щоб його можна було використати в програмному продукті, потрібно провести верстку – розрізати макет інтерфейсу на складові його графічні компоненти і описати правила розташування всіх цих елементів на сторінці.

Структура сайту є найважливішим технічним інструментом з точки зору SEO. Неправильне побудова структури сайту значно ускладнює просування користувачів по сайту. Тому при розробці архітектури ресурсу, необхідно аналізувати розміщення кожного розділу і підрозділу, щоб все зробити грамотно, задовольнивши потреби користувача і відповівши на вимоги пошукових роботів.

На жаль, чіткого визначення якою має бути правильної структура не існує. Вона залежить від виду сайту, семантичного ядра і цільової аудиторії, тому завжди індивідуальна. Однак існують рекомендаційні типи структур, а також основні правила по її розробці. Структура сайту - це логічна побудова всіх сторінок ресурсу. Схема, за якою розподіляється шлях до папок, категорій, підкатегорій. З технічної точки зору, навігація ресурсу являє собою набір URL-ів, логічно вибудуваних в певній послідовності (рис 1.1). Структура взаємопов'язана з семантичним ядром. Саме воно говорить про те, які папки і документи повинні бути присутніми на сайті. Тому, зібравши семантику, вже можна зробити начерки схеми побудови кожного майбутнього URL-а .

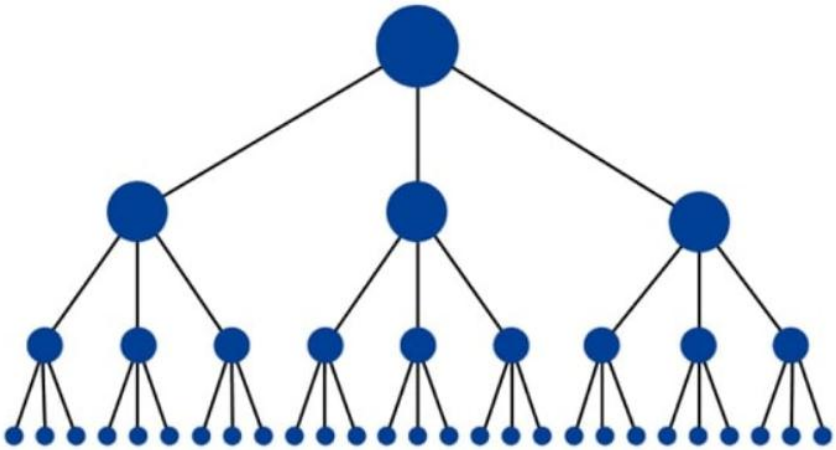

Рисунок 1.1 – Загальний вигляд структур веб-сайтів

Внутрішня структура сайту передбачає наявність логічних зв’язків між сторінками. Важливо, щоб користувач, який відвідав Ваш ресурс, знайшов потрібну інформацію не більше, ніж в 3 кліка. Фахівці також називають внутрішньою структурою особливості розміщення ресурсів і директорій на сервері.

Зовнішня структура повторює навігацію користувачем на сайті, використовується для спрощення користування сайтом. При правильно розробленій зовнішній структурі юзер отримує доступ до основного функціоналу

сайту з кожної сторінки. Звертаємо Вашу увагу, що пошукові роботи аналізують саме зовнішню побудова структури сайту.

Існує також кілька типів структури сайту, кожен з яких має свої особливості і рекомендований для використання при створенні конкретно інтернет-магазину або, наприклад, сайту-візитки.

Лінійна структура веб-сайту (рис 1.2). Найбільш проста і зрозуміла структура, в якій кожна сторінка посилається на попередню (або на Головну) і на наступну. Найчастіше така структура застосовується при розробці сайтів-візиток, головна мета яких – ознайомити користувача з інформацією, викладеною в певній послідовності. Специфіка даного типу в складності просування, оскільки вага сторінок переходить від Головної до останньої по спадаючій. Тому успішно просувати можна тільки Головну сторінку.


Рисунок 1.2 – Лінійна структура сайту

Деревовидна структура веб-сайту (рис 1.3).

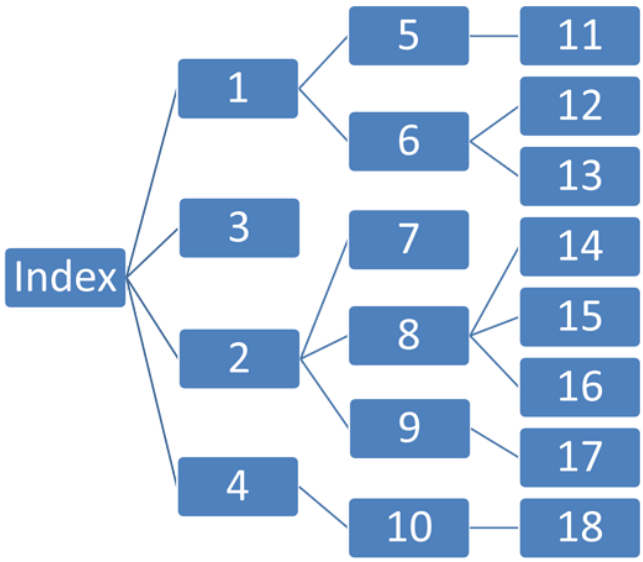


Рисунок 1.3 – Деревовидна структура сайту

Використовується найбільш часто, тому що вона сама оптимальна. Ідея застосування подібної структури заключається в тому,що у користувача є вибір і можливість як з головної сторінки  сайту, так і з любої іншої,перейти в любий розділ ,підрозділ і на конкретну сторінку (документ).

Виходячи з цих даних , можна вибрати відповідну організацію структури з декількох існуючих.

## 1.2 Огляд сайтів-аналогів інформаційного супроводу розважальних заходів

Для того, щоб чітко розуміти, що потрібно реалізувати, потрібно оглянути існуючі рішення. На даний час існує безліч веб-агрегаторів подій, які висвітлюють інформацію про події тощо. Сайти містять, як і просто необхідну інформацію про поточні події, так і про заплановані.

Для огляду було обрано сайти 0342 .ua та frankivsk-online .com.

Сайт 0342.ua є інформаційним ресурсом міста Івано-франківська який надає його жителям та гостям актуальну інформацію про найцікавіші та найпопулярніші події, а також довідкові дані про міську інфраструктуру. Веб-ресурс є відкритою для користувачів та безкоштовною онлайн платформою для організаторів різноманітних подій, івентів, концертів, вечірок та будь-яких інших видів заходів.

Головна сторінка сайту складається з наступних блоків:

- головна;
- новини;
- довідник;
- дозвілля;
- афіша.

Сторінка «Афіша» (рис 1.4) відображає всі події міста, про кожну подію подана така інформація:

- фото;

– назва події;
– тип події;
– дата проведення;
– місце проведення;
– кнопка «Купити» з посиланням на купівлю квитка.

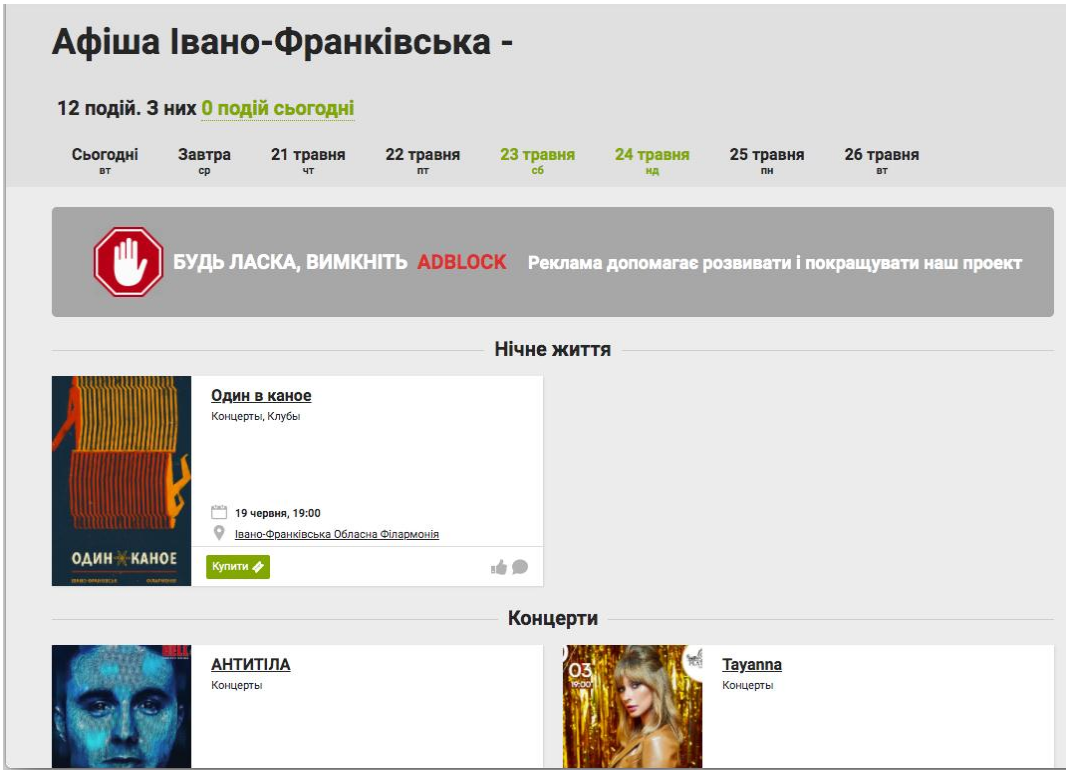


Рисунок 1.4 – Екран «Афіша»

При настисканні на одну з подій відбувається перехід на неї (рис 1.5). Сторінка, що містить детальну інформацію про подію складається з зображення та деталей про подію.

Про подію відображається наступна інформація:

– фото;
– місце проведення;
– вулиця;
– дата проведення;
– час проведення;

– посилання на купівлю квитків.


Рисунок 1.5 – Сторінка з детальним описом події

Сайт frankivsk-online надає можливість переглядати та користуватись інформацією не лише про події, а й про місця міста, наприклад бари, клуби, ресторани, музеї, кінотеатри, парки та торгово-розважальні центри. Сайт є багатосторінковим, тому кожна сторінка буде розглядатись окремо. Головна сторінка сайту (рис. 1.6) складається з наступних блоків:

– події;
– локації;
– публікації;
– пошук по сайту(заховане під знаком «Лупи»);
– посилання на купівлю квитків.

Також на екрані головної сторінки (рис 1.6) відображається список подій, про кожну подію подана наступна інформація:

– фото;
– тип події;
– дата;
– місце проведення;
– короткий опис події.

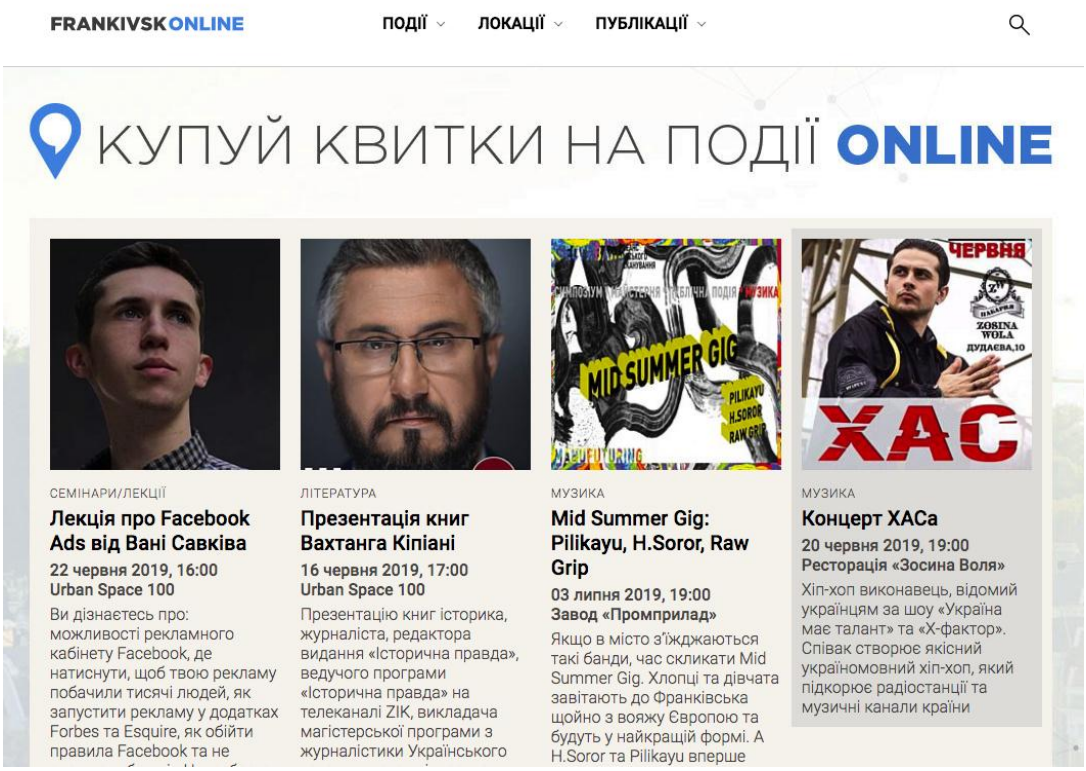


Рисунок 1.6 – Головна сторінка сайту

При натисканні на одну з подій відбувається перехід на екран з детальною інформацією про подію, (рис 1.7).

Сторінка містить детальну інформацію про подію, а саме:

– фото;
– назва події;
– дата проведення;
– час проведення;
– місце проведення;
– посилання на фейсбук події;

– вартість;

– опис.

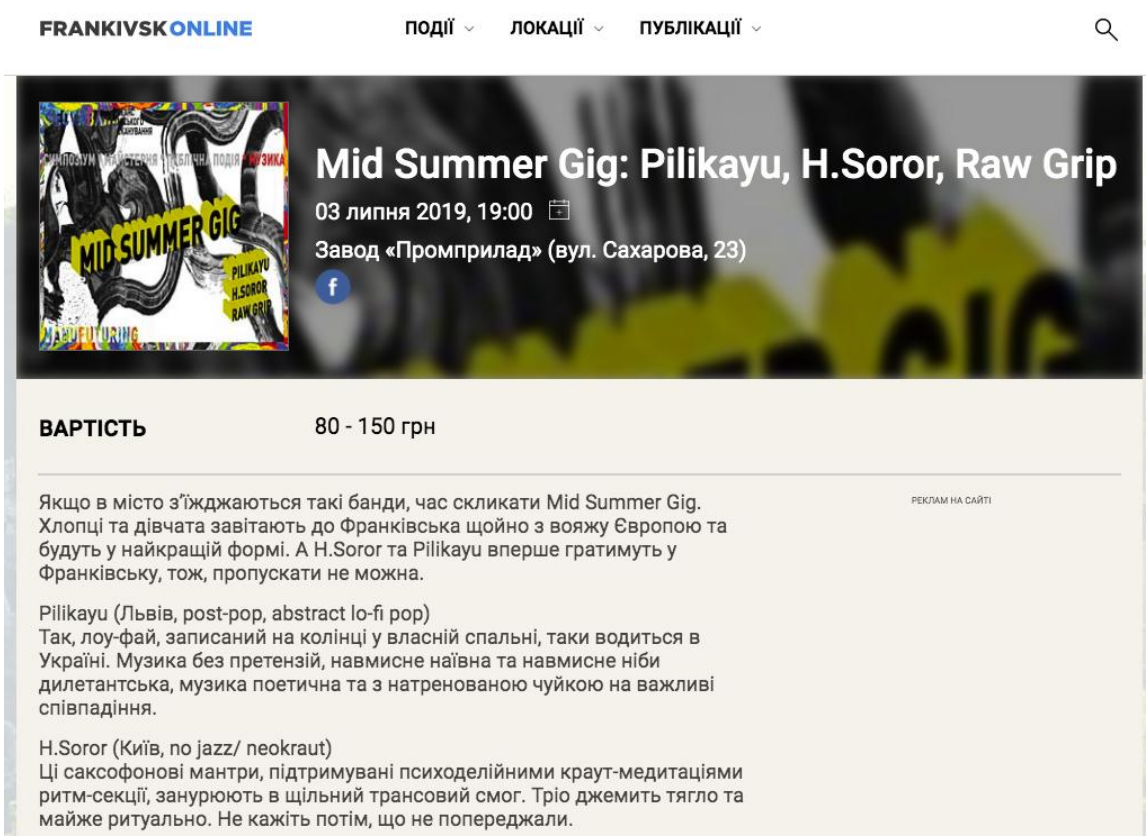


Рисунок 1.7 – Сторінка з детальною інформацією про подію.

Блок «Події»  при розгортанні має вигляд, який зображено на рис 1.8.

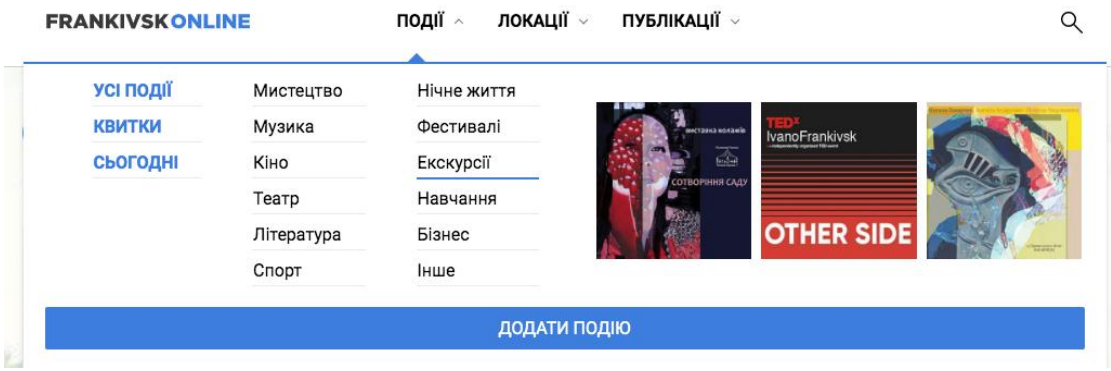


Рисунок 1.8 – Блок «Події»

Блок складається з наступних пунктів:

– усі події;

– посилання на купівлю квитків;

– сьогодні;

– додати подію.

До блока входять підгрупи подій, такі як: Мистецтво, Музика, Кіно, Театр, Література, Спорт, Нічне життя, Фестивалі, Екскурсії, Навчання, Бізнес, Інше.

Сторінка «УСІ ПОДІЇ» зображена на рис 1.9. Тут відображається список усіх подій, про кожну подію подана наступна інформація:

– фото;

– назва;

– дата та час проведення;

– місце проведення;

– короткий опис;

– ціна події;

– посилання на купівлю квитка.

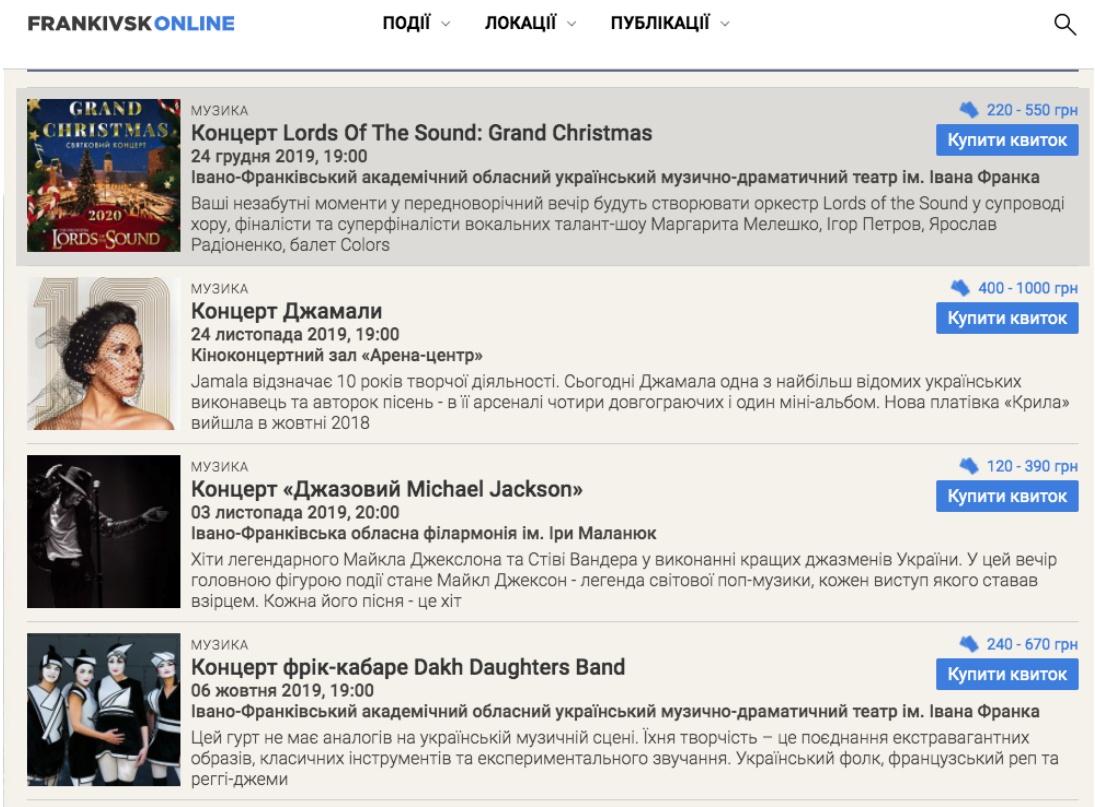


Рисунок 1.9 – Блок «УСІ ПОДІЇ»

Підводячи підсумки, можна констатувати, що на ринку є безліч сайтів які пропонують послуги агрегатора подій, але в них є декілька вагомих недоліків:

- контент більшості сайтів заповнюється вручну і може довгий час не обновлятись;
- банерна реклама;
- обмеженість: інформація на платформах стосуються тільки одного конкретного міста.

### 1.3 Постановка задачі

У даному розділі були розглянуті веб-ресурси які тим чи іншим чином збирають, впорядковують та показують користувачеві інформацію з різних джерел.

Отримана інформація була систематизована з метою використання в подальшій роботі для визначення вимог до програмної системи розроблюваного додатку. На основі дослідження даних програмних систем можна зробити висновок про їхні особливості, основні переваги та недоліки.

Отже, можна зробити висновок, що існуючі на даний момент в Україні та світі агрегатори інформації про події та місця не задовольняють усі потреби користувачів, мають не достатньо розвинений та зручний інтерфейс і велику кількість вагомих недоліків, що погіршують досвід користування.

Існуючі аналоги розвиваються повільно або ж взагалі не розвиваються через недостатню кількість користувачів та непродуманий інтерфейс.

Основними недоліками українських аналогів є:

- покриття додатками чи веб-ресурсами лише частини території України;
- відсутність мобільних версій проектів, що значно ускладнює роботу з ними;
- незручний інтерфейс;

– банерна реклама;
– низька якість та кількість контенту.

Основним завданням кваліфікаційної роботи є створення сайту, який буде виводити користувачу події по його запиту, або місцезнаходженню. А також реалізувати функціонал форм реєстрації.

Сайт повинен бути розробленим за допомогою стандартних методів та технологій веб-розробки. Основними етапами розробки повинні бути:

– створення макетів сторінок;
– верстка сторінок і дизайнів;
– програмування, розробка функціональних інструментів (для веб на стороні клієнта і сервера).

Сайт, для максимальної взаємодії з користувачем, повинен забезпечити швидке завантаження сторінок при великій кількості запитів.

Інтерфейс повинен бути максимально простим, та не має містити багато елементів.

Для роботи ресурсу не потрібні додаткові встановлення програмного забезпечення, або конфігурації браузера. Сайт повинен бути максимально простий та зрозумілий у користуванні .

Сайт повинен містити наступні блоки та сторінки:

– вхід;
– реєстрація;
– геолокація;
– пошуковий блок;
– детальніше;
– список подій.

Для цього необхідно вирішити такі задачі:

– проаналізувати існуючі вебзастосунки агрегації розважальних подій;
– вибрати технології розробки та архітектури веб-додатків;
– розробити загальну структуру вебзастосунку;

- розробити систему геолокації;
- розробити взаємодію з API.

# 2. ВИБІР ТЕХНОЛОГІЙ ТА РОЗРОБКА АРХІТЕКТУРИ САЙТУ

## 2.1 Вибір технологій розробки

Створити сайт можна багатьма способами, ось декілька з них.

CMS (система керування вмістом) як і за допомогою мови розмітки HTML, мови для опису зовнішнього вигляду сторінок CSS та інших веб-орієнтованих мов програмування. Для опису використовується спеціалізована мова HTML мова розмітки гіпертекстових документів. Готові зверстані HTML шаблони далі використовуються в наступних етапах реалізації проекту.

На цій стадії графічна картинка нарізається на окремі елементи і з використанням технологій HTML і CSS трансформується в код, який можна переглядати за допомогою браузера.

HTML - це не мова програмування. Це форма збереження даних. Мова програмування відрізняється від мови розмітки тим, що мова програмування програмує дещо на виконання деяких дій, а мова розмітки готує певним чином деякий документ для того, щоб абияка програма могла його використовувати.

Елементи являють собою базові компоненти розмітки HTML. Кожен елемент має дві основні властивості: атрибути та зміст (контент). Існують певні настанови щодо кожного атрибута та контент елемента, які треба виконувати задля того, щоб HTML-документ був визнаний валідним.

У елемента є початковий тег, який має вигляд <element-name>, та кінцевий тег, який має вигляд </element-name>. Атрибути елемента записуються в початковому тегу одразу після назви елемента, контент елемента записується між його двома тегами.

Як і HTML, CSS не є справжньою мовою програмування. Це лише мова таблиць стилів, яка дозволяє задавати стилі обраним елементам у HTML документах.

Каскадна або блочна верстка прийшла на заміну табличній верстці веб-сторінок. Головна перевага блочної верстки - розділення змісту сторінки (даних) та їхньої візуальної презентації.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг - можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах тощо.).

Declaration (Визначення) - одне правило на зразок color: red; вказує, яку з властивостей елементу ви бажаєте стилізувати.

Properties (Властивості) - шляхи, якими ви можете стилізувати даний HTML елемент. (У цьому випадку, color — це властивість елементів p).

Selector (Селектор) - назва елементу HTML на початку правила. Селектор вибирає елемент чи елементи, які будуть стилізовані .

JavaScript - прототипна, об'єктно-орієнтована та динамічна мова програмування. Підтримує імперативний, об'єктно-орієнтований та функціональний стилі. Являється реалізацією мови ECMAScript.

Найбільш широке застосування мова отримала в браузерах як основний інструмент для створення сценаріїв для додавання інтерактивних елементів вебсторінок на стороні клієнта, а також надає можливість взаємодії з користувачем, керування браузером, асинхронного обміну даних з сервером, зміни структури та зовнішнього вигляду веб-сторінки.

Основними архітектурними рисами цієї мови є: автоматичне керування пам'яттю, динамічна типізація, прототипне програмування, використання функцій як об'єктів першого класу, прототипне наслідування.

JavaScript розроблялась під впливом багатьох мов програмування. Під час розробки однією з цілей було розробити інструмент схожим на Java і разом з цим простим для використання та розуміння людей, які не є програмістами. Мова JavaScript не знаходиться у власності будь-яких компаній чи організацій, що відрізняє її від ряду інших мов програмування, які використовуються у веб-розробці.

Javascript використовується для широкого кола задач, і для багатьох має уже готові фреймворки для розробки:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- програмування на стороні сервера (Node.js); 30 – стаціонарних застосунків (Electron, NW.js);
- мобільних додатків (React Native, Cordova); – сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів. Три ключові елементи об'єдналися в технології мови Javascript;
- робота з об’єктно-орієнтованістю – розробник напряму взаємодіє з певними об’єктами;
- створення односторінкових веб-додатків (React, AngularJS, Vue.js);
- динамічна типізація – тип даних не потрібно вказувати, він визначається компілятором;

Рушій JavaScript підключається до об'єктів свого середовища виконання (зазвичай, веб-переглядача) та надає можливість керування ними.

Мова JavaScript має стандартну бібліотеку об'єктів (таких як Array, Date та Math) і основний набір елементів мови програмування, таких як оператори, керівні структури та вирази.

Ядро JavaScript може бути розширене для різних потреб шляхом доповнення його додатковими об'єктами. Наприклад:

На стороні клієнта JavaScript розширює ядро мови, додаючи об'єкти керування переглядачем і його об'єктною моделлю документа — Document Object Model (DOM). Наприклад, клієнтські розширення дозволяють застосункам розміщувати елементи на HTML-формі та реагувати на дії користувача, такі як клацання миші, введення даних у форму і пересування сторінками.

На стороні сервера JavaScript розширює ядро мови шляхом додавання об'єктів, що стосуються роботи JavaScript на сервері. Наприклад, серверні розширення дозволяють застосункам взаємодіяти з базою даних, забезпечувати безперервність потоку інформації від одного запущеного застосунку до іншого, або виконувати маніпуляції з файлами на сервері.

Так як потрібно просто реалізувати сайт з великою швидкодією, для цього знадобиться мова програмування Ruby.

Ruby - це ретельно збалансована мова програмування. Її творець Юкихиро Мацумото (так само відомий як "Matz"), об'єднав частини його улюблених мов (Perl, Smalltalk, Eiffel, Ada і Lisp) щоб сформувати нову мову, в якій парадигма функціонального програмування збалансована принципами імперативного програмування.

Він часто повторював, що він "намагається зробити Ruby природною, але не простою" мовою, яка відображає життя.

Ґрунтуючись на цьому, він додає: Ruby простий на вигляд, але дуже складний всередині, подібно людському тілу.

Зростання популярності Ruby почалося з часу випуску публічної версії в 1995 році, Ruby привернув увагу програмістів зі всього світу. У 2006 році Ruby завоював масове визнання.

Спочатку Matz розглядав інші мови в пошуках ідеального синтаксису.

Також у найбільших містах по всьому світу активно діють групи користувачів Ruby , а конференції, присвячені Ruby , заповнені до межі.

Згадуючи свої дослідження, він говорив: “Мені потрібна була скриптова мова, яка була би більш потужна, ніж Perl, і більш об'єктно-орієнтована, ніж Python”.Однією з особливостей цієї мови є те що все в ній об'єкти:

```
5.times { print “ми любимо Ruby! Ruby -це прекрасно!” }
```

Для кожної частинки інформації або коду можуть бути визначені власні властивості і дії. В об'єктно-орієнтованому програмуванні властивості називаються змінними об'єкта, а дії - методами.

Найчастіше об'єктно орієнтований підхід Ruby може бути продемонстрований парою рядків коду, в яких здійснюється дія над числом.

У багатьох мовах числа та інші примітивні типи даних не є об'єктами. Ruby під впливом мови Smalltalk дозволяє задати методи і змінні об'єкта всім типам даних.

Це спрощує використання Ruby, так як правила застосовні до об'єктів – застосовні до всього Ruby.

Ruby дуже гнучка мова програмування, так як вона дозволяє його користувачам вільно змінювати його частини.

Наприклад, додавання виконується операцією плюс (+). Але, якщо ви хочете використовувати для цього більш читабельне слово plus – ви можете додати такий метод прямо в Numeric, внутрішній клас мови Ruby :

```
Class Numeric
     def plus(x)
          self.+(x)
     end
end
y = 5.plus 6
```

Блоки, по-справжньому виразна конструкція.

Блоки в Ruby також є відмінним джерелом гнучкості. Програміст може додати замикання до будь-якого методу, описуючи, як цей метод повинен діяти. Замикання називається блоком і є однією з найбільш популярних конструкцій, які

прийшли у світ Ruby зі світу імперативних мов програмування, таких як PHP або Visual Basic.

Створення блоків було навіяне функціональними мовами програмування. Matz говорив: "замиканнями в рубі я хотів віддати данину поваги культурі мови Lisp." На відміну від багатьох об'єктно-орієнтованих мов, Ruby навмисно надає лише одиночне спадкування. Але Ruby також надає концепцію модулів (званих Категоріями в Objective-C). Модулі – це колекції методів.

Класи можуть вільно вмішувати модуль і отримувати всі його методи. Наприклад, будь-який клас, який реалізує метод each, може підмішати модуль Enumerable, який додасть купу методів що використовують each для створення циклів,підключення модуля :

```
Class MyArray
     Include Enumerable
end
```

Крім основ Ruby наповнений іншими особливостями і конструкціями, і ось деякі з них:

- в Ruby є конструкції для обробки виключень як в Java або Python, які дозволяють простіше працювати з помилками;
- в Ruby представлений справжній mark-and-sweep (поміть та очисти) збирач сміття для всіх Ruby об'єктів. Не потрібно вручну відстежувати кількість посилань в інших бібліотеках. Як говорить Matz: "Це корисно для вашого здоров'я." ;
- ruby може довантажувати сторонні бібліотеки динамічно, якщо дозволяє операційна система;
- в Ruby реалізовані незалежні від операційної системи потоки. Таким чином, на будь-яких платформах, де ви запускаєте Ruby, ви також маєте можливість використовувати багатопоточність, не залежно від того, чи підтримує ця система потоки чи ні. Ви можете використовувати можливості багатопоточності навіть в MS-DOS! ;

– ruby відрізняється високою стійкістю: він був розроблений більшою частиною на GNU/Linux, але працює на багатьох типах UNIX, Mac OS X, Windows 95/98/Me/NT/2000/XP/Vista/8, DOS, BeOS, OS/2, тощо.

Для розробки додатку буде використано об'єктно-орієнтований програмний каркас (фреймворк) для створення веб-додатків, написаний на мові програмування Ruby. Цей каркас має назву Ruby on Rails.

Ruby on Rails — фреймворк, написаний на мові програмування Ruby. Ruby on Rails надає архітектурний зразок Model-View-Controller (модель-представлення-контролер) для веб-додатків, а також забезпечує їх інтеграцію з веб-сервером і сервером бази даних. Ruby on Rails є відкритим програмним забезпеченням і поширюється під ліцензією MIT.

Ruby on Rails був створений Давидом Хейнемейером Ханссоном на основі його роботи в компанії 37signals над засобом управління проектами Basecamp і випущений в липні 2004 року.

В RoR представленні такі принципи розробки:

– механізм повторного використання, що дозволяє мінімізувати дублювання коду, так званий Don’t repeat yourself принцип;
– за замовчуванням використовуються угоди по конфігурації, типові для більшості додатків (принцип Convention over configuration). Явна специфікація конфігурації потрібно тільки в нестандартних випадках.

Основними компонентами програм Ruby on Rails є модель, представлення і контролер. Ruby on Rails використовує REST-стиль побудови веб-додатків.

Модель надає іншим компонентам програми об'єктно-орієнтоване відображення даних (таких як каталог продукції або перелік замовлень). Об'єкти моделі можуть здійснювати завантаження і збереження даних в базі даних, а також реалізують бізнес-логіку.

Для зберігання об'єктів моделі в реляційній СУБД за замовчуванням в Rails 3 і вище використана бібліотека ActiveRecord. Конкуруючий аналог - DataMapper.

Існують плагіни для роботи з нереляційними базами даних, наприклад Mongoid для роботи з MongoDB.

Представлення створює користувальницький інтерфейс з використанням отриманих від контролера даних. Представлення також передає запити користувача на маніпуляцію даними в контролер (як правило, представлення не змінює безпосередньо модель).

В Ruby on Rails представлення описується за допомогою шаблонів ERB. Вони являють собою HTML-файли з додатковими включеннями фрагментів коду Ruby (Embedded Ruby або ERb). Результат, що генерується вбудованим кодом Ruby, включається в текст шаблону, після чого виходить сторінка HTML, яка повертається користувачу. Крім ERB можливо використовувати ще близько 20 шаблонизаторов, в тому числі Slim, Haml.

Контролер в Rails - це набір логіки, що запускається після отримання HTTP-запиту сервером. Контролер відповідає за виклик методів моделі і запускає формування представлення.

Відповідність інтернет-адреси з дією контролера (маршрут) задається у файлі config/routes.rb.

Контролером в Ruby on Rails є клас, наслідуваний від ActionController::Base. Відкриті методи контролера є так званими діями (actions). Action часто відповідає окремим представленням. Наприклад, за запитом користувача admin/list буде викликаний метод list класу AdminController і потім використано подання list.html.erb.

Веб-фреймворк RoR використовується у таких великих і відомих сайтах, як:

- twitter;
- shopify;
- cruncbase;
- groupon;
- bloomberg;
- heroku;

– hulu;
– github.

Дані реєстрації користувачів які будуть опрацьовуватись сайтом необхідно організовувати. Сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами називається базою даних. В загальному випадку базою даних можна вважати будь-який впорядкований набір даних.

Бази даних класифікують за різними критеріями. За моделлю організації даних розрізняють такі бази даних:

– ієрархічна. Ієрархічна база даних може бути представлена як дерево, що складається з об'єктів різних рівнів. Між об'єктами існують зв'язки типу «предок-нащадок». При цьому можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок;
– мережна. Така база даних подібна до ієрархічної, за винятком того, що кожен об'єкт може мати більше одного предка;
– реляційна. Реляційна база даних зберігає дані у вигляді таблиць. Найвживаніші СКБД використовують реляційну модель даних;
– об'єктно-орієнтована. У базі даних цього виду дані оформляють у вигляді моделей об'єктів.

Для розробки нашого сайту підійде реляційна база даних SQlite.

SQLite - є полегшеною версія реляційної системи керування базами даних SQL. У ній реалізована велика кількість можливостей стандарту SQL-92, а розроблена вона у вигляді бібліотеки. Дозволяється використовувати сирцевий код цієї системи у зв'язку з тим, що він знаходиться у відкритому доступі і розповсюджується за відкритою ліцензією. Кожен розробник при бажанні має змогу безоплатно користуватись усіма її можливостями без жодних обмежень з будь-якою ціллю.

Важливою рисою SQLite є той факт, що в ній не використовується клієнт-серверний принцип. Мається на увазі, що рушій SQLite не працює з застосунком в ролі окремого процесу, а надає змогу працювати з бібліотекою, яка компілюється у складі програми. Після успішної компіляції рушій стає її складовою частиною. У такий спосіб в ролі протоколу для обміну даними можуть використовуватись функції, доступ до яких надає інтерфейс (API) бібліотеки SQLite.

Таким чином зменшуються час відгуку застосунку, накладні витрати, а також стає можливим спрощення структури програми.

В коробці у системі є також виконуваний файл sqlite3, який є функціональною клієнтською частиною, яка надає можливість демонстрації реалізації функцій з основної бібліотеки. Клієнтська частина надає інтерфейс для роботи в командному рядку, і дозволяє звернення до файлу БД на основі типових функцій ОС.

Переваги SQLite:

- нульова конфігурація. SQLite не потрібно "встановлювати" перед його використанням. Процедура "установки" не існує. Не існує серверного процесу, який потрібно запускати, зупиняти або налаштовувати. Адміністратору не потрібно створювати новий екземпляр бази даних або призначати користувачам права доступу. SQLite не використовує файлів конфігурації. Нічого не потрібно робити, щоб повідомити системі, що SQLite працює. Не потрібно виконувати жодних дій для відновлення після аварії системи або збою живлення. Немає нічого для усунення несправностей. SQLite просто працює;
- SQLite не потрібен сервер, так як більшість двигунів БД були зроблені як окремий серверний процес. За допомогою цієї БД процес, який хоче отримати доступ до БД, читає і записує безпосередньо з файлів БД на диску. Не існує проміжного серверного процесу. Будь-яка програма, яка має доступ до диска, може використовувати SQLite;

- стабільний крос-платформний файл бази даних. Файл бази даних, написаний на одній машині, може бути скопійований і використаний на іншій машині з іншою архітектурою. Більшість інших двигунів баз даних SQL вимагають скидання та відновлення бази даних при переході від однієї платформи до іншої і часто під час оновлення до нової версії програмного забезпечення;
- відкрите джерело. Можна легко знайти сирцевий код безкоштовно онлайн.
- SQLite залишається однією з найбільш використовуваних систем БД у світі. Він сумісний практично з усіма ОС і є більш-менш промисловим стандартом.

Недоліки SQLite:

- SQLite використовується для обробки звернень HTTP з низьким та середнім трафіком;
- у більшості випадків розмір БД обмежений 2 Гб.

Atom - розроблений компанією «GitHub» вільний текстовий редактор і редактор коду, який може використовуватися як самодостатнє рішення, так і у ролі технологічного стека для побудови різних спеціалізованих рішень. Зокрема, на платформі Atom побудовані середовище розробки «Visual Studio Code» від компанії «Microsoft» і «Nuclide» від «Facebook».

Проект був представлений компанією GitHub у лютому 2014 року. Сирцевий код проекту поширюється під ліцензією «MIT ».

Atom надає засоби крос-платформового редагування коду, включає вбудований пакетний менеджер і інтерфейс навігації файловою системою, надає засоби для одночасної спільної роботи з кодом, має інтелектуальну систему автодоповнення вводу, надає режими сумісності з Vim і Emacs, підтримує API для розробки розширень. Кілька файлів можуть бути відкриті в різних вкладках і одночасно відображені з використанням вертикального або горизонтального розбиття панелей.

Основу Atom становить компонент Electron (раніше Atom Shell), що представляє собою засноване на Chromium і Node.js ядро, поверх якого реалізований редактор.

Electron поставляється у формі самодостатнього фреймворку, який можна використовувати для створення довільних користувацьких застосунків, логіка роботи якого визначається на JavaScript, HTML і CSS, а функціональність може бути розширена через систему доповнень.

Наприклад, Atom надає вбудований файловий менеджер і гнучкі засоби пошуку файлів, які неможливо реалізувати при використанні звичайних веб-застосунків.

Web API - це прикладний програмний інтерфейс (Application programming interface, API) для веб-сервера. API визначає функціональність, яку надає програма (модуль, бібліотека), при цьому API дозволяє абстрагуватися від того, як саме ця функціональність реалізована. API може розглядатися як абстракція над функціональністю і реалізацією компонентів.

Іншими словами, API посилається на набір функцій, вбудованих в додатки, які можуть бути використані іншими додатками (або сам по собі), щоб взаємодіяти з додатком.

API є відмінним способом надійно і безпечно розкрити функціональність програми для зовнішніх додатків. Всі функціональні можливості, доступні зовнішнім додаткам, обмежуються наданим API.

Чим обумовлена популярність, а тепер вже і необхідність API?

1. У 2017 році до 75% інтернет трафіка припадає на мобільні пристрої. Безліч мобільних додатків для різних сервісів працюють при використанні API цих самих сервісів. Це дозволяє стандартизувати та уніфікувати процес розробки додатків.

2.Тенденції розвитку open-source, тобто, відкритого програмного забезпечення. Якщо у вашого сервісу склалася певна аудиторія, яка користується ним, чому б не обернути це собі на користь?

За допомогою API користувачі при бажанні зможуть створити нові клієнти для вашого додатку, нові сервіси на його основі, розкрити нові його грані.

3. Максимальне розділення frontend (зовнішнє представлення, інтерфейс користувача) та backend (його програмна реалізація). Це значно спрощує розробку додатків. API-орієнтований веб-додаток - це веб-додаток, в якому весь або більшу частину функціоналу реалізується через виклики API. Наприклад, якщо ви маєте увійти в систему, то ви відправляєте свої дані через функцію API. Результатом виконання цієї функції буде або авторизація, або повідомлення про помилку.

Однією з переваг створення API-орієнтованих веб-додатків, є те, що це допоможе побудувати функціональність, яка може бути використана будь-яким пристроєм, будь то браузер, мобільний телефон, планшет або навіть настільний додаток.

Все, що потрібно зробити - це створити API таким чином, щоб всі ці пристрої могли взаємодіяти з ним. Таким чином можна створити централізований додаток, який може приймати виклики з будь-якого пристрою, який є у користувача.

Принцип роботи API зображений на рис 2.1.

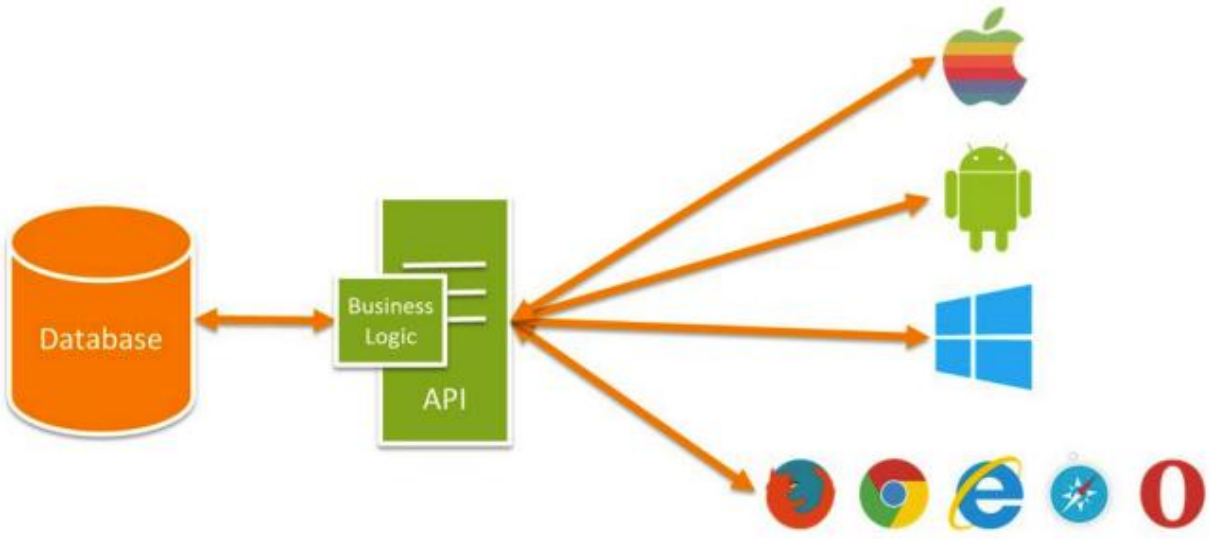


Рисунок 2.1 – Принцип роботи Web API

Як бачимо з рисунку 2.1, користувач може користуватися продуктом з будьякої платформи та використовуючи будь-який браузер чи інший програмний продукт, не відчуваючи змін.

API інкапсулює від кінцевого користувача бізнеслогіку системи, надаючи йому зручний користувацький інтерфейс та реалізуючи взаємодію з базою даних «під капотом». Такий принцип роботи дозволяє повністю відділити програмну логіку від інтерфейсу користувача та забезпечити доступ до системи незалежно від того, з якого пристрою користувач намагається взаємодіяти. До того ж, така архітектура є дуже зручною для масштабування програмного забезпечення.

При збільшенні кількості користувачів та створенні нових програм-клієнтів чи інтеграції у нові веб-сервіси бізнес-логіку системи не доведеться змінювати, оскільки API надає зручну оболонку для взаємодії для кожного клієнта.

Визначимо основні вимоги, які має задовольняти якісний та ефективний API:

1. API повинен мати стандартизований та зручний формат запитів. Крім того, для підвищення швидкодії системи слід використовувати засоби кешування запитів. В свою чергу, відповіді сервера повинні мати явне чи неявне позначення як кешовані чи некешовані з метою попередження отримання клієнтами застарілих або невірних даних у відповідь на подальші запити .

2. API повинен мати стандартизований та зручний формат відповідей та підтримувати різні формати представлення даних. Це дозволить забезпечити кросплатформенність системи, розширити сфери використання продукту, полегшить його популяризацію та інтеграцію в інші веб-сервіси.

3. Має бути передбачена можливість повернення помилок виконання запиту. Причому помилки мають бути чітко описані, щоб не тільки користувач знав, що йому необхідно зробити, але й ви легко орієнтувалися, коли користувач надсилає вам запит для вирішення проблеми. Але необхідно також уникати зайвої надлишковості, щоб не заплутувати користувачів.

4. Забезпечення захищеності та конфіденційності системи. API має бути захищений від таких загроз як DDOS-атаки, csrf-атаки.

5. Масштабованість. API має бути структурований і спроектований так, щоб була можливість легко масштабувати систему або розширити її функціонал без погіршення швидкості та ефективності роботи. Розглянемо детальніше роботу API на прикладі ресурсу Github. Його API дозволяє отримати інформацію про користувача, його аватар, читачів, його репозиторії та детальну інформацію про них.

Стандартний запит до API має наступний вигляд:

curl https://api.github.com/users/hrunishak

В такому випадку ми отримаємо наступну відповідь (рис. 2.2).


```
{
  "login": "hrunishak",
  "id": 36689953,
  "node_id": "MDQ6VXNlcjM2Njg5OTUz",
  "avatar_url": "https://avatars3.githubusercontent.com/u/36689953?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/hrunishak",
  "html_url": "https://github.com/hrunishak",
  "followers_url": "https://api.github.com/users/hrunishak/followers",
  "following_url": "https://api.github.com/users/hrunishak/following{/other_user}",
  "gists_url": "https://api.github.com/users/hrunishak/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/hrunishak/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/hrunishak/subscriptions",
  "organizations_url": "https://api.github.com/users/hrunishak/orgs",
  "repos_url": "https://api.github.com/users/hrunishak/repos",
  "events_url": "https://api.github.com/users/hrunishak/events{/privacy}",
  "received_events_url": "https://api.github.com/users/hrunishak/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Sergiy",
  "company": null,
  "blog": "",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": null,
  "twitter_username": null,
  "public_repos": 1,
  "public_gists": 0,
  "followers": 0,
  "following": 1,
  "created_at": "2018-02-21T06:51:22Z",
  "updated_at": "2020-04-01T22:09:49Z"
}
```


Рисунок 2.2 – Приклад відповіді на запит до API

Незважаючи на численні труднощі, пов'язані з реалізацією програмних систем в середовищі всесвітньої павутини WorldWideWeb, створення рішень на

платформі Веб ось уже більше десяти років утворює найбільш перспективно і динамічно розвивається напрямок сучасної індустрії розробки додатків.

Додатки подібні до Gmail, Google Maps, Flickr залишили в минулому звичну для розробників модель «тонкого» клієнта, і змусили повною мірою працювати стек технологій DHTML (JavaScript / DOM / CSS), закладений в стандартній архітектурі сучасного веб-клієнта. Разом з тим не можна не помітити, що створення таких додатків є дуже трудомісткою, складною в технічному плані роботою.

REST (Representationalstatetransfer) - це стиль архітектури програмного забезпечення для розподілених систем, таких як WorldWideWeb, яка, як правило, використовується для побудови вебслужб.

Термін REST був введений у 2000 році Роєм Філдінгом, одним з авторів HTTP-протоколу.

Системи, що підтримують REST, називаються RESTful-системами. У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL.

Кожна URL в свою чергу має строго заданий формат. Відсутність додаткових внутрішніх прошарків означає передачу даних в тому ж вигляді, що і самі дані. Тобто ми не загортаємо дані в XML, як це робить SOAP і XML-RPC, не використовуємо AMF, як це робить Flash і т.д. Просто віддаємо самі дані. Але це може спричинити за собою проблеми пов'язані з безпекою передачі даних.

REST-сервіси це просто HTTP-запити, а значить, легко контролюються штатними засобами вирівнювання навантаження звичайних пристроїв управління трафіком.

Моніторинг використання REST-сервісів теж значно простіший і часто менш дорогий, оскільки моніторинг на базі URI вже став сдандартною технологією.

Також пропонуються як відповідні комерційні продукти, так і рішення з відкритим вихідним кодом.

REST-сервіси простіше реалізувати, оскільки вони базуються на добре відомих Web-протоколах і не вимагають від розробника вивчення WSDL, SOAP і маси інших WS-специфікацій, що використовуються для управління і забезпечення безпеки SOAP-сервісів.

Оскільки технологія REST базується на HTTP, то вона несе на собі відпечаток ненадійності цього протоколу і неможливості збереження стану (його stateless-характеру).

Доступно декілька онлайн-сервісів, які демонструють найважливіші події, що дозволяють людям легко знайти наступний соціальний привід у своєму населеному пункті. Цікаво, що деякі платформи, що перелічують події, відкрили свої API (інтерфейси програмування прикладних програм), щоб дозволити розробникам отримати доступ до їх функцій та інтегрувати їх у додатки, популярні API подій зображено на рис 2.3.

Щоб знайти найкращий, ми переглянули декілька API подій на основі наступних чотирьох основних критеріїв:

- особливості API. Ми оцінили відмінні риси кожного з API;
- популярність. Ми перевірили популярність служб подій. Це допоможе вам дізнатися ті, які варті інтеграції у вашу програму;
- ціна. Ми розглянули вартість використання кожного з API для демонстрації та керування подіями;
- простота використання. Ми перевірили простоту включення API в додатки.

Врешті-решт наступний список топ-10 найкращих API подій (рис 2.3).

Після ретельного аналізу підходящих до завдання API, було обрано використовувати EventfulAPI.

| API | Особливості API | Популярність | Ціна |
|---|---|---|---|
| Eventbrite API | Створюйте та видаляйте події, керуйте класами квитків та знижками, створюйте запитання щодо замовлення квитків, створюйте та оновлюйте місця | Використовується 700 000 організаторів заходів та приймає 2 мільйони заходів щорічно | Безкоштовно |
| Meetup API | Створіть події, знайдіть події, видаліть події, оновіть події, перелічіть події | 40 мільйонів зареєстрованих користувачів | Безкоштовно |
| Ticketmaster API | Перелічіть події, квитки, місця та платежі, пошукові події | Використовується 12 000 організаторів заходів | Безкоштовно |
| Events API | Доступ до зведених подій, пошукових подій, отримання квитків | Не популярний | Безкоштовно |
| Eventful API | Створюйте та керуйте подіями, знаходьте та відстежуйте події, діліться подіями | 20 мільйонів зареєстрованих користувачів | Безкоштовно |
| SeatGeek API | Пошук подій, пошук місця подій, сортування подій, отримання результатів подій | Списки 120 000 подій наживо | Безкоштовно |
| Trade Events API | Перелічіть події для американських експортерів, пошукові події | Не популярний | Безкоштовно |
| PredictHQ API | Доступ до ранжуваних подій у різних категоріях, сортування подій, зберігання локальної копії подій | Список 20 мільйонів подій | Безкоштовні та різноманітні платні плани |
| 31Events API | Надсилайте події до календаря одержувачів, створюйте події, редагуйте події, видаляйте події | Не популярний | $ 10 на місяць |
| WooCommerce Events Plugin API | Створюйте події, продайте квитки та пропуски, керуйте подіями | Не популярний | Безкоштовно |

Рисунок 2.3 – таблиця порівняння API подій

Отже, для розробки був обраний наступний стек технологій:

- HTML;
- CSS;
- JavaScript;
- Ruby;
- Ruby on Rails;
- Atom;
- Eventful API;
- SQlite.

## 2.2 Вибір архітектури веб-додатку

Одним із архітектурних шаблонів програмного забезпечення є архітектура клієнт-сервер (рис 2.4). Клієнт-сервер є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними.

Основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

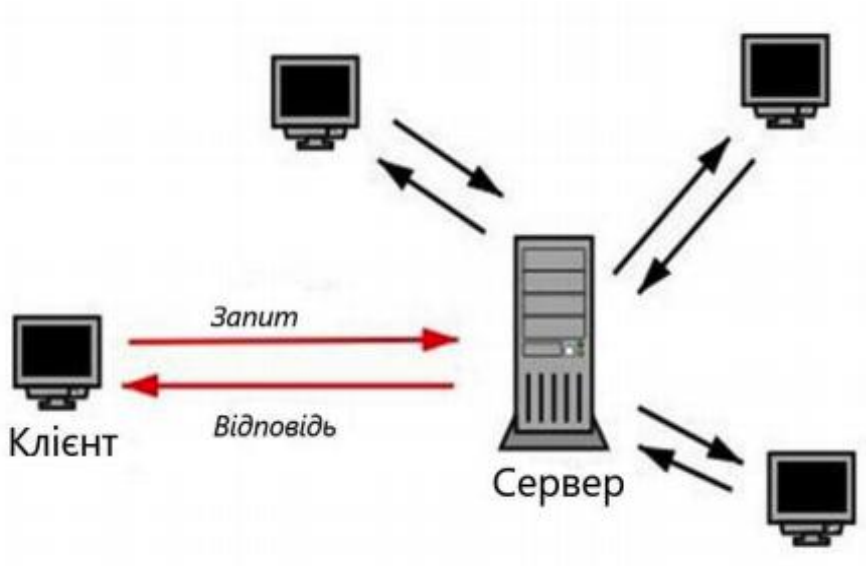


Рисунок 2.4 – Клієнт-серверна архітектура

Схожою з клієнт-серверною трьохрівневою архітектурою є шаблон MVC (рис. 2.5).

MVC - це схема розподілення рівня даних додатку, інтерфейсу користувача та логіки додатку на три окремих та незалежних компонента. Таким чином,

модифікація кожного з трьох компонентів може відбуватися незалежно, це дає переваги у портативності та масштабованості системи .

Розглянемо компоненти MVC детальніше:

- модель (Model) – надає дані для обробки та відображення і реагує на команди контролера, змінюючи власний стан;
- представлення (View) – відповідає за відображення даних, які надає модель користувачу, реагуючи на зміни у моделі;
- контролер (Controller) – реагує на дії користувача та оповіщає модель про необхідність змін.

Основною ідеєю застосування цього підходу є розмежування бізнес-логіки (Model) від її візуального вигляду – представлення (View). За рахунок такого розмежування виконується принцип повторного використання коду. Крім того, це дуже корисно в ситуаціях, коли користувач повинен мати змогу бачити одні і ті ж дані одночасно, але у різних контекстах або з різних точок зору.

А також:

- не вносячи змін до реалізації моделі, до неї можна приєднати декілька представлень. Для прикладу, дані можуть бути одночасно представлені у вигляді тексту, графіка, таблиці чи гістограми. Причому на саму модель це ніяк не буде впливати ;
- не вносячи змін до реалізації представлення, можна змінити обробку дій користувача, таких як введення даних, жести миші. Для цього достатньо використати новий контролер або внести зміни до існуючого;
- бізнес-логіка та візуальне представлення програми можуть розроблятися повністю незалежно один від одного. Розробники програмної логіки можуть навіть не мати уяви про те, яке представлення буде використовуватися.

Структура архітектурного шаблону MVC зображена на рис 2.5.

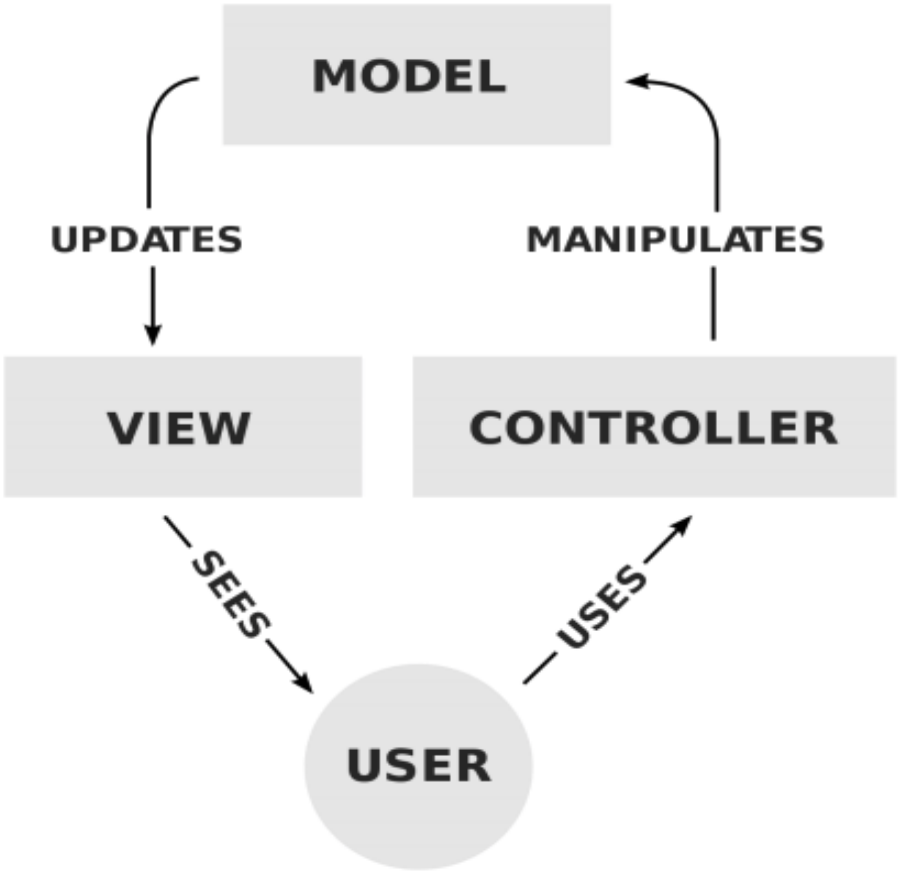


Рисунок 2.5 – Загальна схема MVC

Модель надає дані та методи, які дозволяють працювати з ними, наприклад, запити до бази даних, перевірка на коректність отриманих даних. Модель є незалежною від представлення, вона зовсім не знає, як ці дані візуалізувати. Також вона повністю незалежна від контролера і не має спільних точок взаємодії з користувачем. Вона просто надає доступ до даних та керування ними.

Представлення, у свою чергу, відповідає за отримання необхідних даних від моделі та надсилає їх користувачеві. Представлення не має змоги впливати на стан моделі, спілкуючись з нею напряму. Також представлення не обробляє дані, які ввів користувач.

Контролер забезпечує зв’язок між користувачем та системою. Він вирішує, які методи мають виконуватися у випадку тієї чи іншої дії користувача. Для реалізації необхідної дії він використовує модель та представлення.

MVC є загальним шаблоном і не має строгої реалізації. Немає загальноприйнятого визначеного місця, де повинна розміщуватися бізнес-логіка додатку. Вона може знаходитись як у контролері, так і у моделі. Деякі фреймворки чітко задають, де має знаходитись логіка програми, інші не мають таких установок. Валідація даних, введених користувачем, також не має чітко визначеного місця розташування. Якщо валідація проста, вона може знаходитись навіть у представленні. Але зазвичай вона все-таки знаходиться у контролері або моделі.

В залежності від складності архітектурного рішення для реалізації схеми Model-View-Controller можуть використовуватися наступні шаблони проектування:

- «спостерігач»;
- «стратегія»;
- «компонувальник».

Типовою реалізацією шаблона «компонувальник» є те, - коли представлення відмежоване від моделі шляхом встановлення між ними протоколу взаємодії, який використовує «апарат подій». «Апарат подій» - це позначення «подіями» певних ситуацій, які виникають в ході виконання програми та розсилання повідомлень про них всім, хто підписався на їх отримання. При кожній зміні внутрішніх даних у моделі, яке позначене як «подія», вона оповіщає про це всі представлення (Views), які підписані на цю подію. При отриманні оповіщення представлення оновлює свій стан.

Використання шаблону «стратегія» означає, що при обробці реакції користувача саме представлення обирає, який контролер буде обробляти цю реакцію та забезпечувати той чи інший зв’язок з моделлю. Цей шаблон також має модифікацію, яка називається «команда».

Шаблон «компонувальник» використовується переважно для можливості однотипного спілкування з підоб’єктами складно-складового виду ієрархії.

Дозволяє користувачам будувати складні структури з простіших компонентів та може згрупувати дрібні компоненти для формування ще більших.

Модель MVC може існувати в активному та пасивному вигляді. В такому випадку модель (Model) програми містить лише сукупність методів для доступу до даних, а вся програмна логіка додатку міститься у контролері .

В об’єктно-орієнтованому програмуванні застосовується переважно активна модель Model-View-Controller, де модель окрім методів для доступу до СУБД містить також і бізнес-логіку програми. Крім того, моделі можуть інкапсулювати в себе й інші моделі.

Разом з цим контролери тримаються «тонкими» - тобто, відповідають лише за:

- прийом запиту від користувача;
- аналіз отриманого запиту;
- вибір наступної дії системи у відповідності з результатами аналізу (для прикладу, передача запиту іншим елементам моделі).

Таким чином контролер виконує лише функцію зв’язуючої ланки (glue layer) між окремими компонентами інформаційної системи.

### 2.3 Загальна структура сторінок сайту

Сайт складається з інформативних сторінок та сторінок на яких користувач зможе здійснювати певні дії (рис 2.6), наприклад зареєструватись як користувач (рис 2.8).

До інформативних сторінок належать наступні сторінки:

- головна сторінка;
- сторінка зі списком подій та їх інформацією.

До сторінок дій належать:

- сторінка реєстрації учасника.


Рисунок 2.6 – Загальна структура сторінок сайту

Також на усіх сторінках буде відображено навігаційне меню сайту разом з формою для пошуку подій, знаходження геолокації, та посиланням на реєстрацію користувача (рис 2.7).

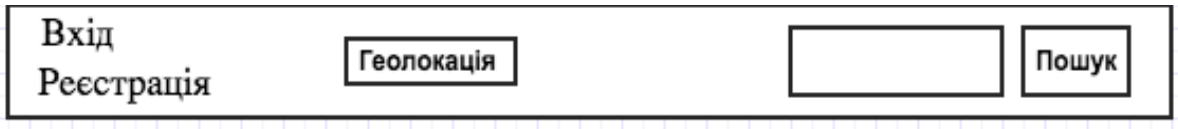


Рисунок 2.7 – Навігаційне меню сайту

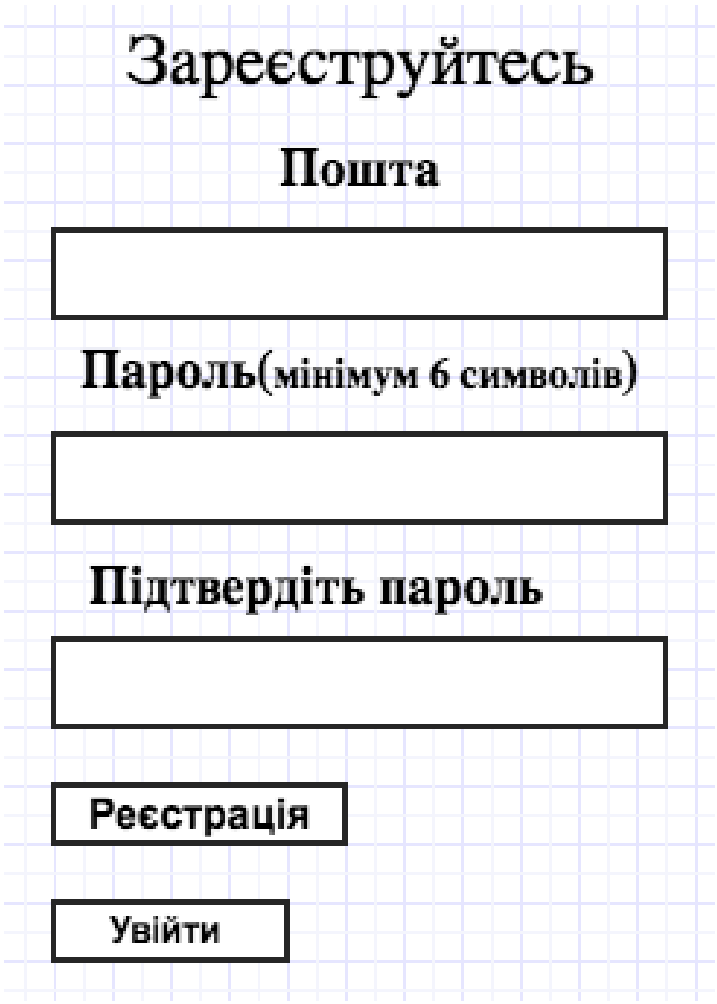


Рисунок 2.8 – Форма реєстрації користувача

Сторінка реєстрації користувача крім меню містить форму реєстрації. Усі поля форми є обов’язковими.

Перелік полів:

- електронна пошта (текстове);
- пароль (текстове).

Сторінка зі списком подій містить в собі блоки (рис 2.9), та кнопку деталі з випадаючим вікном (рис 2.10).


Рисунок 2.9 – Блок з подіями

У блоках з подіями відображається наступна інформація:

– назва події;
– зображення;
– короткий опис;
– дата проведення.


Рисунок 2.10 – Блок з детальною інформацією про подію

У блоці деталі відображається наступна інформація:

- назва події;
- повний опис події;
- дата проведення;
- місце проведення;
- посилання на сайт з подією.

У розробці програмного забезпечення та системному проектуванні для опису поведінки системи, як вона відповідає на зовнішні запити, використовують Use Case (варіант використання).

У системному проектуванні різновиди використання застосовуються на більш високому рівні ніж при розробці програмного забезпечення, часто представляючи цілі зацікавлених осіб або місії. На стадії аналізу вимог різновиди використання можуть бути перетворені на ряд детальних вимог і задокументовані за допомогою діаграм вимог SysML або інших подібних механізмів.

Кожен різновид використання зосереджується на описі того, як досягти мети або завдання. Для більшості програмних проектів це означає, що потрібно безліч різновидів використання щоб визначити необхідний набір властивостей нової системи. Ступінь формальності програмного проекту і його стадії буде впливати на необхідний рівень деталізації, для кожного різновиду використання.

Різновиди використання не можна плутати з поняттям властивостей системи (Feature). Різновид використання може бути пов'язаний з однією або більше властивістю системи, і властивість може бути пов'язана з одним або більше різновидом використання.

Різновид використання визначає взаємодії між зовнішніми агентами і системою, спрямовані на досягнення мети. Актор (Actor) являє собою роль, яку грає людина або річ, взаємодіючи з системою. У випадку нашого веб-сайту буде розглянуто 2 ролі:

- користувач;
- відвідувач.

Різновиди використання розглядають систему як «чорний ящик», і взаємодії з системою, включаючи системні відповіді, описуються з точки зору зовнішнього спостерігача. Це - навмисна політика, тому що це змушує автора зосередитися на тому, що система повинна зробити, а не, як це повинно бути зроблено, і дозволяє уникати створення припущень про те, як функціональні можливості будуть реалізовані.

Різновиди використання можуть бути описані на абстрактному рівні (діловий різновид використання, іноді званий ключовим різновидом використання), або на системному рівні (системний різновид використання).

Діловий різновид використання не зачіпає технологій, розглядає систему як «чорний ящик» і описує бізнес-процес, який використовується діловими акторами (людьми, або системами, зовнішніми до бізнесу) для досягнення своїх цілей (наприклад, обробка оплати, схвалення авансового звіту, управління корпоративними нерухомим майном). Діловий різновид використання описує процес, цінний для ділового агента, описує що саме робить процес .

Системний різновид використання зазвичай описується на рівні функцій системи (наприклад, створіть ваучер), і визначає функцію або сервіс, що надаються системою для користувача. Системний різновид використання описує що актор може зробити взаємодіючи з системою. З цієї причини рекомендується, щоб системні випадки використання починалися з дієслова (наприклад, створіть ваучер, виберіть платежі, скасуйте ваучер) .

Для опису Use Case-ів системи буде використано системний різновид, з огляду на наявний функціонал для більшості ролей.

Use Case повинен:

- описувати що саме система має зробити, щоб актор досяг своєї мети;
- не торкатися деталей реалізації;
- мати достатній рівень деталізації;
- не описувати інтерфейси та екрани. Це робиться під час дизайну користувальницького інтерфейсу.

В уніфікованій мові моделювання відносини між усіма або частиною різновидів використання й акторами представлені у вигляді діаграми різновиду використання або діаграмах .

На діаграмах різновидів використання в UML різновид відображається у вигляді еліпса. Всередині еліпса або під ним вказується ім'я елемента.

До різновиду використання в UML застосовують наступні види відносин:

- асоціація (Association) - Може вказувати на те, що актор ініціює відповідний варіант використання;
- розширення (Extend) - Різновид відносини залежності між базовим варіантом використання та його спеціальним випадком;
- включення (Include) - Визначає взаємозв'язок базового варіанту використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов;
- узагальнення (Generalization) - моделює відповідну спільність ролей.

На рисунку 2.11 зображено відповідності між акторами та їхніми цілями.

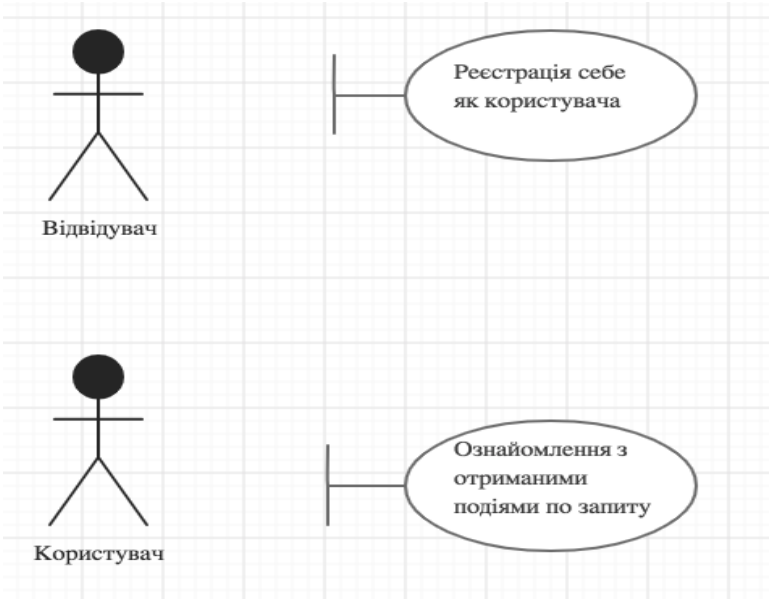


Рисунок 2.11 – Актори та їх цілі

Беручи за основу для побудови Use Case рисунок з описом цілей акторів, побудовано UML-діаграми (рис. 2.12).

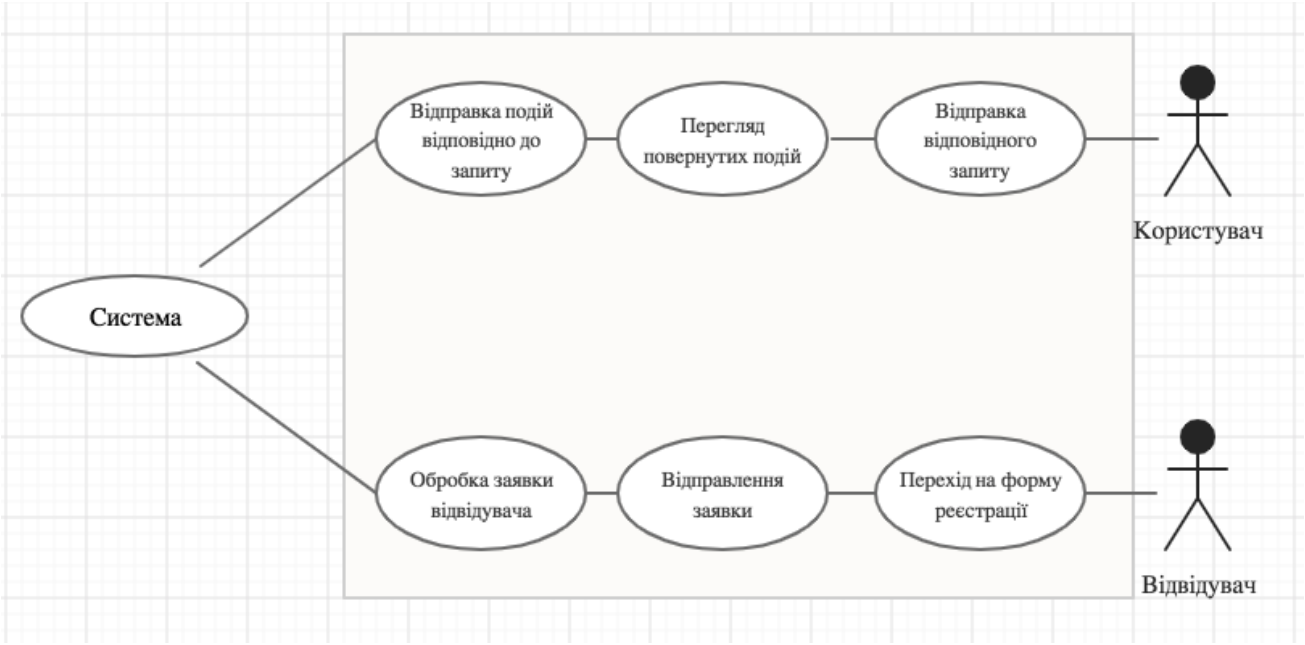


Рисунок 2.12 – UML-діаграми

На рис. 2.12 зображено діаграму Use Case-ів потенційних користувачів сайту, відповідно до їхнії вимог.

Історія користувача (user story) - це одне чи більше речень, звичайною мовою предметної області, які описують чого користувач хоче досягти. Історії користувача використовуються в гнучких методологіях для з'ясування базових функцій що будуть реалізовуватись. Кожна історія користувача достатньо коротка і записується на карточках приблизно 7 на 12 сантиметрів, що гарантує те, що вона не стане занадто великою. Історії користувача пишуться споживачами програмного забезпечення і є основним інструментом їх впливу на розробку програми. Вони описують вимоги у простий та точний спосіб .

Історії користувача - швидкий спосіб оперування вимогами користувача, без необхідності застосування занадто формалізованих документів, та виконання адміністративних задач пов'язаних з опрацюванням цих документів. Наміром з яким використовують історії користувача є швидше та менш накладне реагування

на швидко змінювані вимоги реального світу. Приклад user story зображений на рис. 2.13.

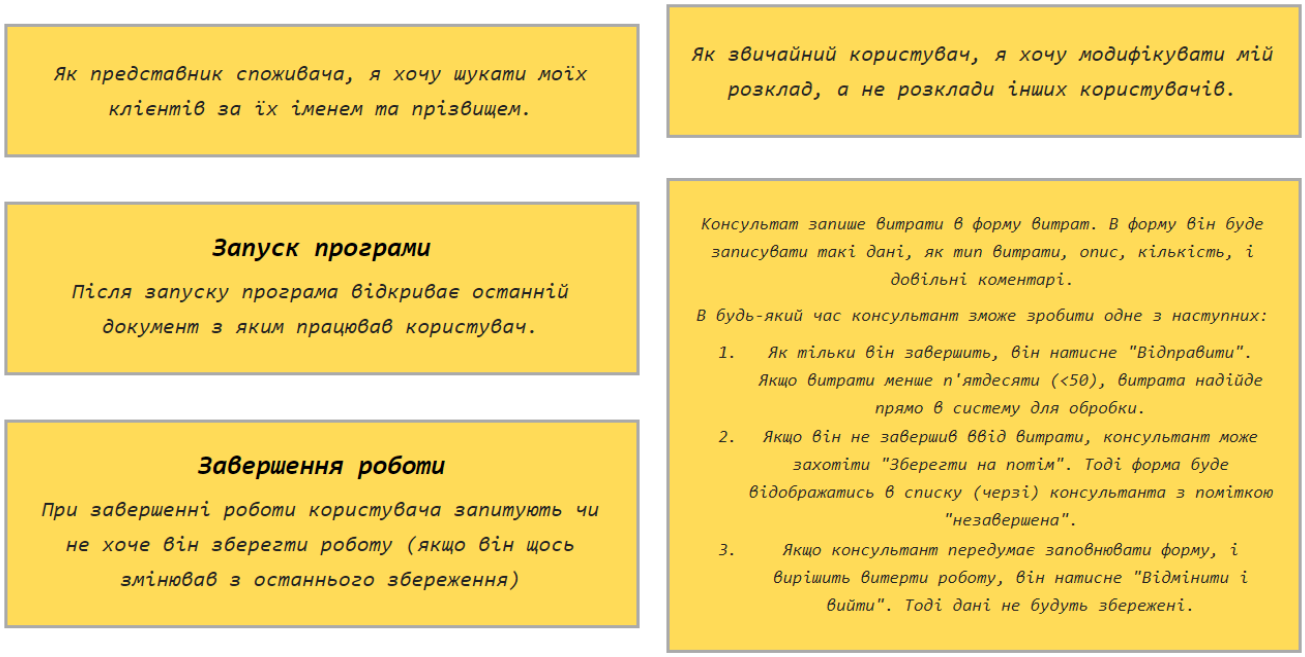


Рисунок 2.13 – Приклад user story

Усі user story розроблені за поданим прикладом та описують детальні кроки роботи користувачів на сайті. User story звичайного відвідувача сторінки сайту зображена на рис. 2.14.

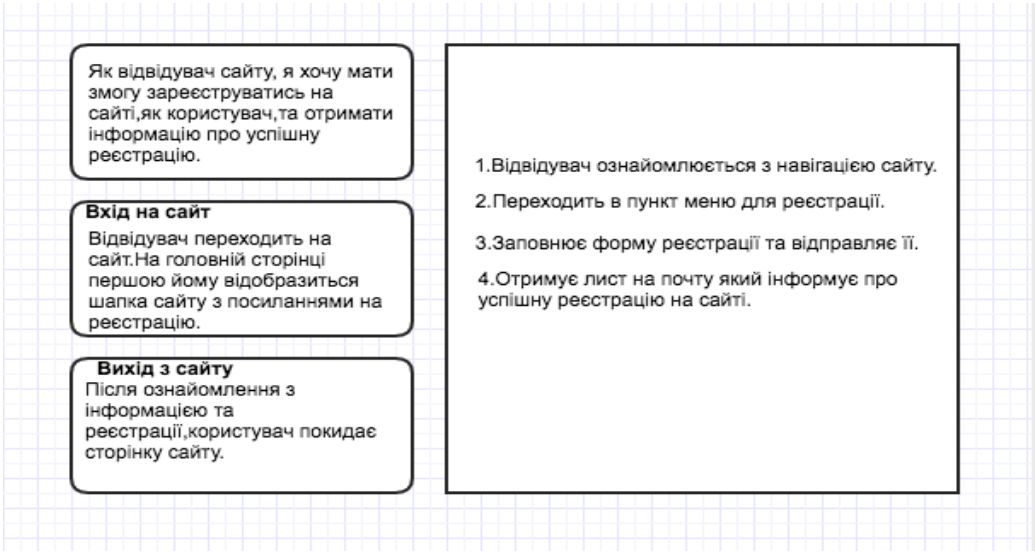


Рисунок 2.14 – User story відвідувача сайту

Дана user story описує роботу відвідувача , який хоче зареєструватись на сайті. User story користувача зображена на рис. 2.15.

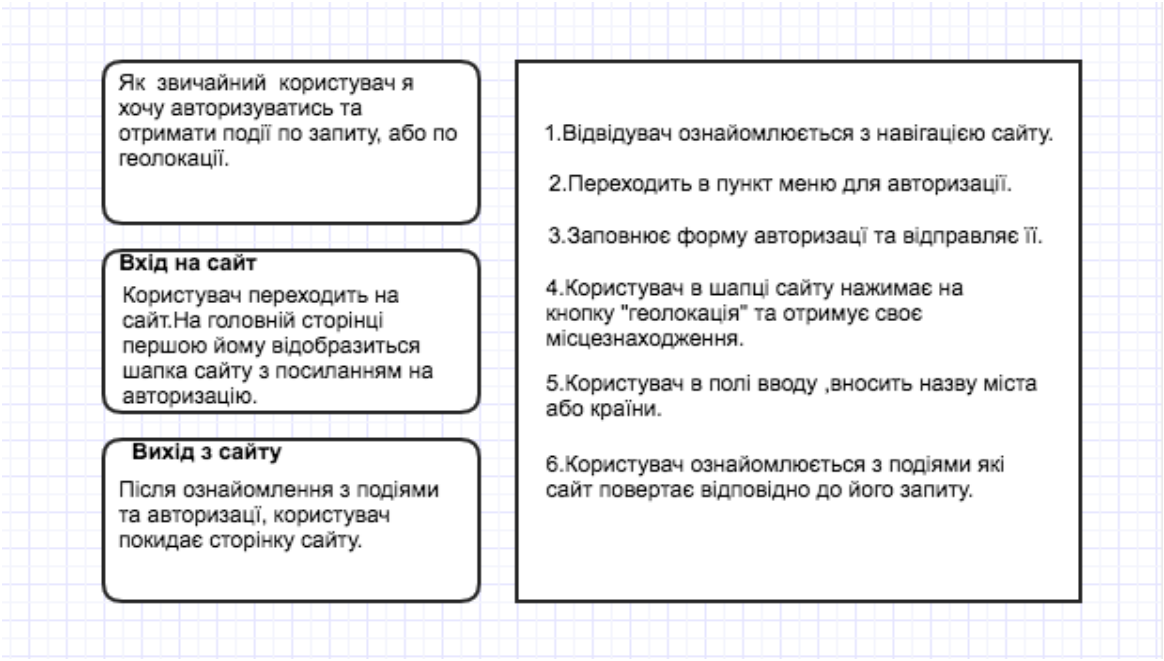


Рисунок 2.15 – User story користувача

Дана user story описує роботу користувача, який хоче авторизуватись та отримати події по відповідному запиту.

# 3. РОЗРОБКА ВЕБ ЗАСТОСУНКУ АГРЕГАЦІЇ РОЗВАЖАЛЬНИХ ПОДІЙ

## 3.1 Реалізація взаємодії з API

Для прискорення розробки взаємодії з API, ми будемо використовувати гем eventful.

Це SDK що позволяє зручно взаємодіяти з даним програмним інтерфейсом взаємодії. Інтеграцію гема розподілено на два етапи: встановлення та налаштування.

Для встановлення потрібно вказати гем в Gemfile:

```
gem 'evenfulapi'
```

Для встановлення гему потрібно виконати команду:

```
bundle install
```

Після встановлення гему потрібно підключити його в нашому контроллері, для початку створимо його. Заходимо в консоль, в папку з нашим проектом та прописуємо таку команду:

```
rails g controller events
```

Автоматично створиться контроллер з назвою city. Тепер нам потрібно підключити гем в контроллер. Заходимо в контроллер city і вверху прописуємо:

```
require 'eventful/api'
```

Після чого створюємо метод:

```
def search
end
```

В методі ініціалізовуємо наш обєкт Eventful::API та прописуємо ключ (який можна дістати на сайті API після реєстрації):

```
eventful = Eventful::API.new 'наш ключ'
```

@location - наша змінна в яку ми будемо передавати наше місце знаходження, або любий запит, що нас цікавить:

```
@location = (params[:country])
```

Після чого викликається REST метод /GET , формується наш URL де

:page_size - це максимальна кількість подій яку може вернути наш запит.

:location - це місце з якого ми хочемо повернути події і в нього ми передаємо нашу змінну @location, та повертаємо масив даних.

```
@results = eventful.call 'events/search'
                          :location => @location,
                          :page_size => 60
```

Змінна @resulst містить наші дані. Далі ми перебираємо циклом подііі та присвоюємо ключі до потрібних нам значень:

```
@results['events']['event'].each do |event|
event ={
         :title => event["title"],
         :url => event["url"],
         :start_date_time => event["start_time"],
         :venue_address => event['venue_address'],
         :city_name => event['city_name'],
         :description=>event['description'],
         :image => event['image'] && event['image']['medium'] &&
event['image']['medium']['url'],
         :place =>event['venue_name']
                          }
                          @events << event
end
```

Події будуть повернуті в форматі багатовимірного масиву XML/JSON в залежності від вказання параметру. Gem Evenful зразу парсить інформацію, тому нам не потрібно перетворювати JSON в Ruby hash, за нас цю роботу виконав SDK.

### 3.2 Розробка системи геолокації

Для релізації пошуку місце розташування використовується інтерфейс Geolocation представляє можливість програмно отримувати місце розташування пристрою.

Він надає веб-запит до позиції користувача. Це також допомагає сайтам і додаткам надавати результати, ґрунтуючись на отриманих даних.

Об'єкт з цим інтерфейсом виходить за допомогою властивості NavigatorGeolocation.geolocation і реалізується об'єктом Navigator.

Інтерфейс Geolocation не наслідує ніяких методів.

Метод який визначає місцерозташування пристрою і повертає об’єкт Position з даними:

`Geolocation.getCurrentPosition() Secure context`

Другий метод long значення, представляє заново створену callback-функцію, яка викликається при зміні місцерозташування пристрою:

`Geolocation.watchPosition() Secure context`

Третій метод видаляє опрацьовувач, створений з допомогою watchPosition():

`Geolocation.clearWatch() Secure context`

Буває таке що браузери можуть не працювати з інтерфейсом Geolocation, тому на сайті є таблиця сумісності щоб дізнатись чи ваш браузер буде підтримувати даний інтерфейс (рис 3.1).

| | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari |
|---|---|---|---|---|---|---|
| Geolocation | 5 | 12 | 3.5 * | 9 | 10.6 | 5 |
| clearWatch | 5 | 12 | 3.5 | 9 | 10.6 | 5 |
| getCurrentPosition | 5 | 12 | 3.5 | 9 | 10.6 | 5 |
| Secure context required | 50 | ≤79 | 55 | Нет | 37 | Да |
| watchPosition | 5 | 12 | 3.5 | 9 | 10.6 | 5 |

Рисунок 3.1 – Сумісність з ПК версіями браузерів

API геолокації доступний через об’єкт navigator.geolocation.

Якщо об’єкт існує, функції оприділення місцерозташування доступні. Ви зможете провірити це слідуючим образом (рис 3.2):

```
if ("geolocation" in navigator) {
  /* Місцерозташування доступне */
} else {
  /* Місцерозташування НЕ доступно */
}
```

Рисунок 3.2 – Провірка функціїї визначення місцезнаходження

Щоб отримати поточне місце розташування користувача, ви повинні викликати метод getCurrentPosition ().

Це ініціює асинхронний запит для виявлення місця розташування користувача, і запитує апаратні засоби позиціонування, щоб отримати останню актуальну інформацію. Коли розташування визначено, виконується callback. За бажанням ви можете вказати другу callback функцію для обробки помилки, яка запуститься в разі помилки.

Третій, опціональний параметр - об'єкт з опціями, де ви можете налаштувати максимальне значення даних, що повертаються, час очікування відповіді на запит, і, при бажанні, точність даних, що повертаються.

За замовчуванням getCurrentPosition () намагається повернути результат так швидко, як це можливо, за рахунок чого дає не надто точний результат.

Це може бути корисно, якщо вам потрібно швидко отримати відповідь, при цьому не важлива точність. Пристрої з GPS, наприклад, можуть намагатися скорегувати дані GPS близько хвилини і навіть більше, тому на самому початку можуть повернутися менш точні дані (місце розташування IP або wifi-мережі), отримані getCurrentPosition (). JavaScript код визначення координат (рис 3.6).

GetCurrentPosition () і watchPosition () приймають callback-функцію при успіху, необов'язкову callback-функцію при помилці і необов'язковий об'єкт PositionOptions.

Цей об'єкт дозволяє вам включити можливість визначення позиції з високою точністю, вказати максимальний час кешування значення позиції (при повторних запитах, поки час не вийшов, вам будуть повертатися кешовані значення; після браузер буде запитувати актуальні дані).

Також в об’єкті налаштування відклику потрібно вказати значення, яке встановлює інтервал - як часто браузер повинен намагатися отримати дані про місцезнаходження, перш ніж вийде час.

Виклик watchPosition може виглядає наступним чином (рис 3.3):

```
function geo_success(position) {
  do_something(position.coords.latitude, position.coords.longitude);
}

function geo_error() {
  alert("Извините, нет доступной позиции.");
}

var geo_options = {
  enableHighAccuracy: true,
  maximumAge        : 30000,
  timeout           : 27000
};

var wpid = navigator.geolocation.watchPosition(geo_success, geo_error, geo_options);
```

Рисунок 3.3 – приклад виклику watchPosition

Місцезнаходження користувача міститься в екземплярі об'єкта GeolocationPosition, що містить усередині екземпляр іншого об'єкта - GeolocationCoordinates.

Екземпляр GeolocationPosition містить тільки дві речі, властивість coords, всередині якого GeolocationCoordinates і властивість timestamp, всередині якого екземпляр DOMTimeStamp, що надає мітку часу, створену при отриманні дані.

Екземпляр GeolocationCoordinates містить деяку кількість властивостей, двоє з яких ви будете найчастіше використовувати: latitude і longitude, які допоможуть вам відобразити отриману позицію на карті. Тому багато callback-функції з успішним отриманням позиції виглядають дуже просто (рис 3.4):

```
function success(position) {
  const latitude  = position.coords.latitude;
  const longitude = position.coords.longitude;
```

Рисунок 3.4 – Приклад виклику callback функції

Callback-функція для помилок (рис 3.5), якщо вона була передана в getCurrentPosition () або watchPosition (), очікує екземпляр об'єкта GeolocationPositionError як перший аргумент.

Він буде містити дві властивості, code, який вкаже на те, яка саме помилка сталася і зрозумілий для людини message, що описує значення поля code.

```
function errorCallback(error) {
  alert('ERROR(' + error.code + '): ' + error.message);
};
```

Рисунок 3.5 – приклад функціїї обробки помилок

Код визначення координат об’єкта наведено на рис. 3.6.

```
<script>
  var x = document.getElementById("coordinate");
   var y = document.getElementById("json");

   function getLocation() {
     if (navigator.geolocation) {
       navigator.geolocation.getCurrentPosition(showPosition);
     } else {
       x.innerHTML = "Geolocation is not supported by this browser.";
     }
   }
   function showPosition(position) {
     x.innerHTML =
       "Latitude: " +
       position.coords.latitude +
       "<br>Longitude: " +
       position.coords.longitude;
```

Рисунок 3.6 – JavaScript код визначення координат

Тепер нам відомі широта та довгота, щоб визначити місто в якому знаходиться користувач потрібно відправити запит в API ,щоб той в свою чергу повернув назву міста (рис 3.7).

```
      var latitude = "latitude=" + position.coords.latitude;
      var longitude = "&longitude=" + position.coords.longitude;
      var query = latitude + longitude + "&localityLanguage=en";
      const Http = new XMLHttpRequest();
      var bigdatacloud_api =
        "https://api.bigdatacloud.net/data/reverse-geocode-client?";
      bigdatacloud_api += query;
      Http.open("GET", bigdatacloud_api);
      Http.send();
      Http.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
          var myObj = JSON.parse(this.responseText);
          console.log(myObj);
          y.innerHTML += "" + myObj.locality ;
```

Рисунок 3.7 – JavaScript код визначення міста по координатам

Після того як користувач натисне на кнопку Геолокація, виконається JavaScript код, браузер запросить дозвіл на отримання даних про ваше місцерозташування, та видасть такі результати (рис 3.8).

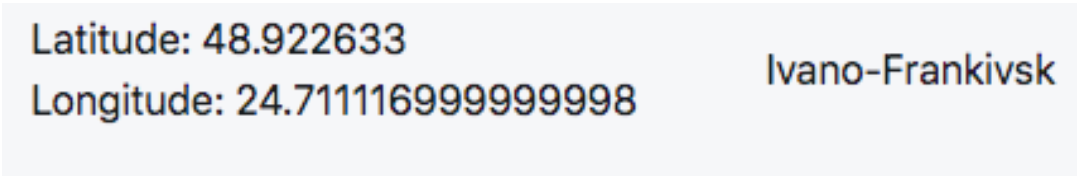


Рисунок 3.8 – Результат роботи JavaScript скрипта

### 3.3 Розробка системи реєстрації

Один з кращих гемів для аутентифікації в rails-додатках є Devise. Devise– це ruby–гем, що надає можливості для аутентифікації в rails-додатках. Devise працює в зв'язці з гемом Warden, який в свою чергу надає сам механізм для аутентифікації в rack-базованих ruby-додатках. Основними особливості Devise є:

- розроблений на Rack;
- є довершеним MVC-рішенням, розробленим на Rails;
- дозволяє вхід в систему по декількох моделях одночасно;
- модульний, тобто використовує тільки те, що вам дійсно необхідно.

Devise потрібен для реєстрації користувачів та, авторизації адміністратора в адмінпанель. Інтеграцію гема розподілено на два етапи: встановлення та налаштування.

Для встановлення потрібно вказати гем в Gemfile, можна також вказати точну версію для використання . Наприклад:

```
gem 'devise', '3.2.2'
```

Для встановлення гему потрібно виконати команду:

```
bundle install
```

Як залежностей будуть встановлені додатково наступні геми:

- warden – сполучне ПЗ, яке надає можливість аутентифікації для Rack-додатків;
- orm_adapter – надає єдину точку входу для використання основних функцій Ruby ORMs;

– bcrypt-ruby – надає просту обгортку для роботи з паролями. В основі лежить криптографічна хеш-функція bcrypt ();
– thread_safe – надає потоко-безпечні колекції і утиліти для Ruby;
– railties – внутрішні компоненти Rails, такі як завантажувачі додатків, плагіни, генератори і rake-завдання.

Devise має в своєму арсеналі зручні генератори. Виконати інсталяцію Devise можна шляхом запуску наступного генератора:

rails generate devise:install

Цей генератор встановить ініціалізатор, в якому описані всі конфігураційні налаштування Devise, необхідні для роботи, а також файл з базовою локаллю (англійська мова). Також інсталятор пропонує виконати базове налаштування.

Так як будуть використовуватись сповіщення, які надходитимуть на електронну пошту, після реєстрації нового користувача необхідно задати налаштування мейлера (відправника пошти) для кожного середовища виконання. Для середовища розробки необхідно додати наступний рядок в файл config/environments/development.rb:

```
config.action_mailer.default_url_options = { :host => 'localhost:3000' }
```

Після входу користувача в систему, реєстрації, підтвердження аккаунта або поновлення пароля, Devise буде шукати шлях для подальшого перенаправлення.

За замовчуванням виконає перенаправлення до user_root_path, якщо той існує. В іншому випадку Devise виконає перенаправлення до root_path. Тому цей шлях повинен бути обов'язково визначений у програмі. config/routes.rb повинен містити рядки такого вигляду:

```
root 'home#index'
```

Можна налаштувати файли вигляду під свої потреби. Для цього необхідно їх скопіювати з гема до свого додатку шляхом запуску наступної команди:

```
rails generate devise:views
```

В директорії app/views/devise появляться всі використовувані гемом файли вигляду. Ці файли можна налаштувати за своїм бажанням, під загальний стиль додатку.

Devise має в своєму арсеналі 10 модулів. За замовчуванням підключено 6 модулів. Можна відредагувати цей список. Опис всіх доступних модулів нижче:

1. Database Authenticatable: надає можливість входу в систему на основі зашифрованого і збереженого в базі даних пароля. Вхід може бути виконаний за допомогою відправки POST-запиту або за допомогою HTTP Basic Authentication.
2. Omniauthable: додає підтримку Omniauth.
3. Confirmable: дозволяє відправляти лист з інструкціями для підтвердження аккаунта, створеного під час реєстрації.
4. Recoverable: дозволяє відновлювати забутий пароль. Відправляє інструкції по відновленню на пошту.
5. Registerable: керує реєстрацією користувачів, дозволяє редагувати і видаляти акаунти.
6. Rememberable: дозволяє запам'ятовувати користувачів на основі cookies. Керує створенням і видаленням токенів.
7. Trackable: веде статистику кількості входів, враховує час і IT-адреси.
8. Timeoutable: відповідає за тривалість сесії активності користувача в системі;
9. Validatable: надає інструменти валідації e-mail і пароля. Модуль може бути легко налаштований, можете визначити власні валідатори.
10. Lockable: блокує акаунт після зазначеної в налаштуваннях кількості невдалих спроб авторизації. Аккаунт може бути розблоковано за допомогою email або через певний період часу:
    - email (Електронна пошта);
    - encrypted_password (Зашифрований пароль );
    - reset_password_token (Токен для зміни паролю);

– reset_password_sent_at (Дата відправлення зміни паролю);
– remember_created_at (Дата збереження cookie);
– sign_in_count (Дата входу в акаунт);
– current_sign_in_at (Дата поточної авторизації);
– last_sign_in_at (Дата останнього входу в акаунт);
– current_sign_in_ip (Поточна IP-адреса авторизації);
– last_sign_in_ip (Остання IP-адреса авторизації).

Для користувачів потрібно за коментувати усі модулі крім Registerable, для того, щоб вони могли тільки зареєструватись (рис 3.9). Також потрібно додати в моделі відповідні поля, які є обов'язковими для реєстрації. Для налаштування форми реєстрації потрібно відкрити згенерований шаблон форми реєстрації та виправити його.

Також для відправки повідомлень на електронну пошту потрібен Action Mailer. За допомогою нього можна розсилати повідомлення з додатку використовуючи класи і відображення розсильника.

У конкретного класу, який буде наслідувати Action Mailer та реалізовувати свої функції для розсилання повідомлень, існує і ряд необхідних методів для роботи.

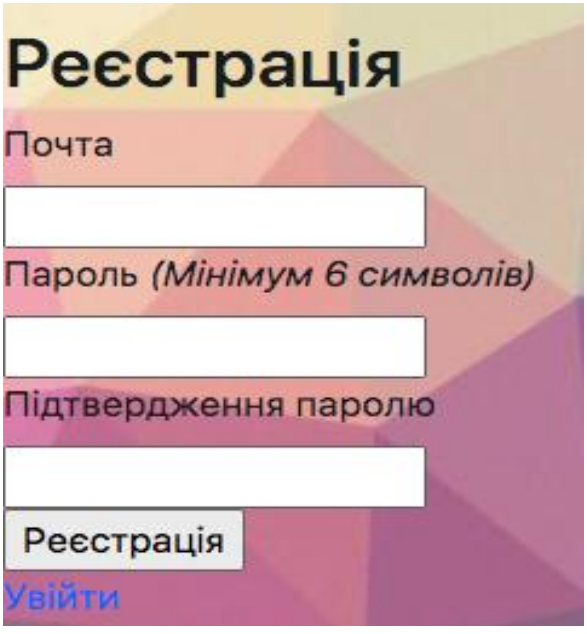


Рисунок 3.9 – Вигляд форми реєстрації користувачів

### 3.4 Розробка шаблонів інтерфейсу та сторінок сайту

Засобом зручної взаємодії користувача з інформаційною системою, в даному випадку з сайтом, є інтерфейс користувача. Сукупність засобів для обробки та відбиття інформації, якнайбільше пристосованих для зручності користувача; у графічних системах інтерфейс користувача, втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, ярлики, шрифти тощо), доступністю багатокористувацьких налаштувань.

Виділяють наступні основні види інтерфейсів:

- інтерфейс прямої обробки, це назва загального класу інтерфейсів користувача, який дозволяє користувачам маніпулювати наданими їм об'єктами, з використанням дій, котрі принаймні, відповідають фізичному світу;
- графічні інтерфейси користувача (GUI) приймають вхідні дані за допомогою таких пристроїв, як комп'ютерна клавіатура та миша, й забезпечують графічний висновок на моніторі комп'ютера. У GUI-дизайні, широко використовуються як мінімум, два різні принципи: об'єктно-орієнтовані інтерфейси користувача(OOUI) й інтерфейси, орієнтовані на додатки;
- веб-інтерфейси користувача або веб-інтерфейси користувача (WUI), які приймають вхідні дані та забезпечують виведення, створенням веб-сторінок, які передаються Інтернетом і проглядаються користувачем за допомогою програми веб-браузера. У нових втіленнях, використовуються: PHP, Java, JavaScript, AJAX, Apache Flex, NET Framework, або подібні технології для забезпечення керування у

дійсному часі , що усуває потребу відновлення традиційного веб-браузера на основі HTML;

- адміністративні веб-інтерфейси для веб-серверів, серверів і мережевих комп'ютерів, часто називаються панелями керування;
- сенсорні екрани — дисплеї, які приймають введення, дотиком пальців або стилусом. Використовуються у все більшій кількості мобільних пристроїв і багатьох засобах точок продажу, промислових процесах і машинах, пристроях самообслуговування тощо;
- апаратні інтерфейси — фізичні, просторові інтерфейси, присутні на виробах у повсякденному житті від тостерів, до приладових панелей автомобілів чи кабін літаків. Вони, як правило, є поєднанням ручок, кнопок, слайдерів, перемикачів і сенсорних екранів;
- інтерфейси командного рядка, де користувач надає вхідний сигнал, введенням командного рядка за допомогою комп'ютерної клавіатури, а система забезпечує виведення, шляхом відбиття тексту на моніторі комп'ютера.

З розвитком DHTML та JavaScript набув популярності підхід до розробки інтерфейсної частини веб-застосунків, названий AJAX. Серцем технології є здатність веб-сторінки зініціювати запит до веб-сервера і отримати потрібні дані, так щоб інтерфейс не перезавантажував сторінку цілком, а лише довантажують необхідні дані і змінив потрібні частини сторінки, що робить їх більш інтерактивними і продуктивними.

Веб-інтерфейси зручні тим, що дають можливість вести спільну роботу співробітникам, які не перебувають в одному офісі. Веб-інтерфейс дає можливість універсального віддаленого доступу до служб та пристроїв, у цьому технології практично нема альтернатив.

Але водночас, оскільки такий інтерфейс доступний усім, постають серйозні питання забезпечення безпеки, зокрема автентифікація та авторизація користувачів, шифрування переданих даних від сторонніх очей, модерація вмісту

тощо. Однією з основних вимог до веб-інтерфейсів є їх однаковий зовнішній вигляд і однакова функціональність при роботі в різних браузерах.

Для взаємодії з інформацією на сайті, потрібно розробити веб-інтерфейс. Класичним і найпопулярнішим методом створення веб-інтерфейсів є використання HTML із застосуванням CSS і JavaScript, як правило за допомогою скриптових мов на стороні сервера.

Проте різна реалізація HTML, CSS, DOM і інших специфікацій в браузерах викликає проблеми при розробці веб-застосунків і їхньої подальшої підтримки.

Стандартна мова розмітки для створення веб-сторінок і веб-додатків є HTML. HTML (Hypertext Markup Language — мова гіпертекстової розмітки) служить для опису Web-сторінки, що зберігається у виді звичайного текстового файлу з розширенням *.htm або *.html.

Головна мета HTML — описати формат вмісту Web-сторінки, він описується з допомогою дескрипторів (tag) HTML. Дескриптори визначають способи форматування тексту, служать розпізнавальними знаками зображень або таблиць, дозволяють зв'язувати слова або фрази з іншими документами в Internet.

Але так як, більшість програмістів використовує інфраструктуру програмних рішень, що полегшують розробку складних систем. Так звані фреймворки, які можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проекту та написання коду. То для розмітки, задання стилів та роботи з додатком будуть використані наступні технології:

- ERB;
- bootstrap.

Bootstrap - це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-стайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків.

Для початку роботи з Bootstrap потрібно встановити гем bootstrap, додавши наступний код в Gemfile:

```
gem "bootstrap", "~> 2.0"
```

В Rails є узгодження імен відображень. Як правило, ім'я відображення збігається з відповідною дією контролера.

Наприклад, індекс контролера в статті articles_controller.rb буде використовувати файл в index.html.erb в директорії app/views/articles. Повний HTML, повернений клієнтом, складається з комбінації цього файлу, шаблону макета, який обгортає його, і всіх партіалів (partials), які можуть бути використані.

Шаблони розміщені в спеціально відведених директоріях як partials layouts.

Для об’єднання та мінізації або зтискання асетів JavaScript і CSS, Rails надає фреймворк Asset Pipeline. Asset Pipeline увімкнутий в проекті по замовчуванню. Це дозволяє автоматично комбінувати асети програми з асетами інших гемів.

Для роботи з Asset Pipeline потрібно налаштовувати файли манфести, щоб фреймворк розумів, які файли враховувати, але для зручності, файли також розміщаються у директоріях з іменем відповідного типу асету (javascript, css) та відповідного контроллера. Першою особливістю файлопровода є з'єднання ассет, що може зменшити кількість запитів, необхідних браузеру для відображення сторінки. Браузери обмежені в кількості запитів, які вони можуть виконати паралельно, тому менша кількість запитів може означати більш швидке завантаження вашої програми. Другою особливістю файлопровода є мінімізація або стиснення Ассет. Для файлів CSS це виконується шляхом видалення пробілів і коментарів. Для JavaScript можуть бути застосовані більш складні процеси. Можна вибирати з набору вбудованих опцій або визначити свої.Третьою особливістю файлопровода є те, що він дозволяє писати ці Ассет на мові більш високого рівня з подальшою прекомпіляції до фактичного Ассет. Мови за замовчуванням включають Sass для CSS, CoffeeScript для JavaScript і ERB для обох.

Далі розглянемо код шапки сайту (рис 3.10).

```
<body>
<div class="container">
    <nav class="navbar navbar-light bg-light justify-content-between">
    <ul class="nav navbar-nav">
      <li class="nav-item active">
            <% if user_signed_in? %>
             <%= current_user.email %>
            <li class="nav-item ">
              <%= link_to "Вихід", destroy_user_session_path, method: :delete, class: "nav-link" %>
              </li>
            <% else %>
            <li class="nav-item ">
              <%= link_to "Вхід", new_user_session_path,class: "nav navbar-nav pull-xs " %>
              </li>
              <li class="nav-item ">
                <%= link_to "Реєстрація", new_user_registration_path,class: "nav navbar-nav pull-xs" %>
                  </li>
                   <% end %>
                </ul>
```

Рисунок 3.10 – код шаблону усіх сторінок сайту

В application_layout також реалізована кнопка геолокації та поле для вводу (рис 3.11).

```
<button type="button" class="btn btn-outline-success my-2 my-sm-0" onclick="getLocation()">
  Геолокація
</button>
<p id="coordinate"></p>

<p id="json"></p>
      <%= form_with(url: search_path, method: 'get', local: true ,class: 'form-inline') do %>
        <form class="form-inline">
         <%= text_field_tag :country, nil, placeholder: 'місто або країна', class: 'form-control mr-sm-2' %>
         <%= button_tag 'Пошук', class: 'btn btn-outline-success my-2 my-sm-0' %>
        </form>

      <% end %>
```

Рисунок 3.11 – Код форми вводу та кнопки геолокації

application_layout (рис 3.12) є головним шаблоном усіх сторінок сайту, який по замовчуванню інтегрується на всіх сторінках. Тому мета теги, стилі, java script, та заголовки описуються тут.


Рисунок 3.12 – Вигляд шапки сайту

Після того як користувач авторизується він зможе ввести місто чи країну для пошуку подій. Відповідний запит формується та відправляється, після чого нам прийдуть дані з арі які ми повинні оформити за допомогою bootstrap в блок (рис 3.14), та у випадаюче вікно блоку (рис 3.15).

Далі розглянемо частину коду вилучення даних з масиву подій (рис 3.13).

```
<% if @results %>

 <% @events.each do |event| %>

          <%title = event[:title]%>
          <%url = event[:url]%>
          <% if event[:image] %>
           <% returned_image = event[:image].split('') %>
           <% image = returned_image[0..100].join() %>
         <% else %>
           <% image = "

         <% end %>
         <% if event[:description] %>
         <% returned_description = event[:description].split('') %>
         <% description = returned_description[0..100].join() %>
       <% else %>
         <% description = "no description provided" %>
       <% end %>
```

Рисунок 3.13 – скрипт для перебору та форматування получених даних

Інколи трапляється що не всі дані про подію заповнені, або вони не коректні. При спробі вивести ці дані сайт буде видавати помилки, щоб уникнути цього скрипт перевіряє чи присвоюються якійсь значення змінній, якщо нічого немає тоді змінним присвоюються наперед задані значення.

```
        <div class="card border-primary mb-3 " style="width: 22rem;">
          <div class="card-header">
<h5 class="card-title"><%= title%></h5>
          </div>
          <img src="<%= image %>" class="card-img-top" alt="...">
          <div class="card-body">

            <p class="card-text"><%= description... %></p>
            <div class="dropdown">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="d
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    </div>

  </div>
```

Рисунок 3.14 – HTML код оформлення получених даних  в блок

```
          <div class="dropdown">
    <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdow
      Деталі
    </button>
    <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
      <span class="dropdown-item-text"><h3><%=title %></h3>
  </span>
      <span class="dropdown-item-text"><%=f_description %>
  </span>
  <span class="dropdown-item-text"><h4>Дата проведення: </h4>  <%=date_time %>
</span>
<span class="dropdown-item-text"><h4>Місце проведеннння: </h4>  <%=place %>
</span>
<a class="dropdown-item" href="<%=url%>">як потрапити?</a>

    </div>
```

Рисунок 3.15 – HTML код випадаючого вікна з  інформацією про подію

Вигляд сторінки з подіями у Івано-Франківську зображено на рис 3.17.

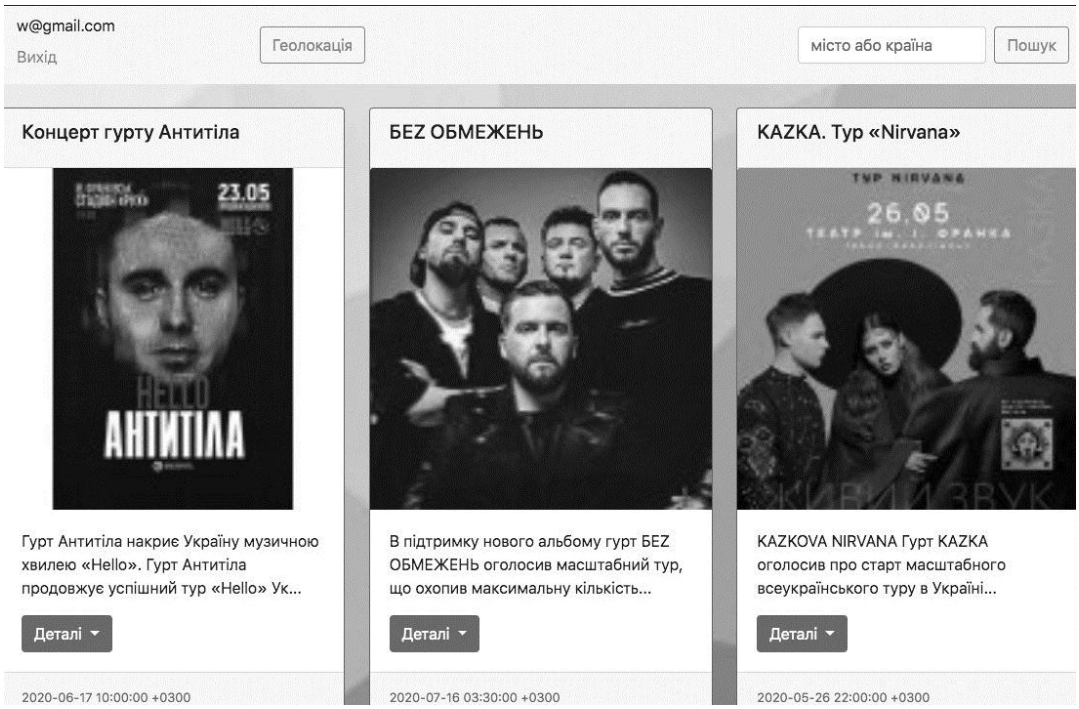


Рисунок 3.17 – Вигляд інформаційної сторінки з  подіями

В кожному блоці подій присутнє випадаюче вікно з деталями про подію (рис 3.18).

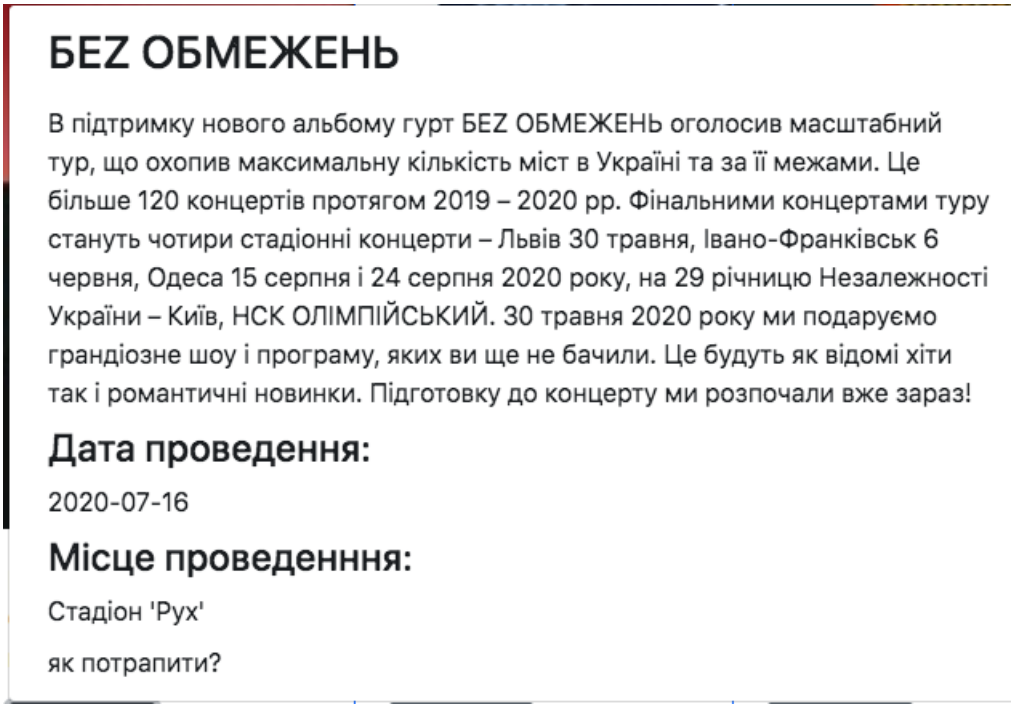


Рисунок 3.18 – Вигляд блоку “Деталі” списку подій

# ВИСНОВКИ

Завданням кваліфікаційної роботи була розробка веб-ресурсу «агрегатора подій», яка надавала б можливість користувачам переглядати події в усьому світі, виконувати операції пошуку та фільтрації за заданими параметрами.

Після проведення аналізу існуючих програмних аналогів, було зроблено висновок, що система має бути розроблена у вигляді веб-застосунку, що повинна мати зручний та зрозумілий інтерфейс, який мінімізуватиме помилки користувачів, та надаватиме надійну широку варіативність дій користувачам та організаторам.

За результатами проведеного аналізу засобів реалізації, а саме мов програмування, бібліотек та фреймворків було розроблено веб-додаток на мові програмування Ruby та його фреймворку Ruby on Rails за архітектурою шаблону MVC. При цьому було вирішено задачі геолокації та взаємодії з API.

Розроблений додаток надає:

- можливість реєструватися і входити на сайт;
- можливість переглядати події, та дізнаватись деталі про них;
- користувацький інтерфейс, спроектований за сучасними стандартами UI та UX.

Розробку програмного забезпечення виконано у повному обсязі та у відповідності до сформованих вимог. Використання розробленого веб-застосунку забезпечить користувачам комфортний та зручний сервіс для планування дозвілля, що є простим у керуванні та доступним у різних куточках світу.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Структура веб-сайту // [Електронний ресурс] – Режим доступу: https://naurok.com.ua/urok-struktura-veb-saytiv-riznovidi-veb-saytiv-riznovidi-veb-storinok-33309.html.
2. Github. Devise // [Електронний ресурс] – Режим доступу: https://github.com/plataformatec/devise.
3. Github. Rails admin // [Електронний ресурс] – Режим доступу: https://github.com/sferik/rails_admin.
4. Github. Ruby on Rails: руководство по стилю оформления // [Електронний ресурс] – Режим доступу: https://github.com/arbox/ruby-style-guide/blob/master/README-ruRU.md.
5. INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0) . Draft Federal Information Processing Standards Publication 183 ,1993 December 21.
6. MDN web docs – Вступ в JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/JavaScript/Guide/Introduction.
7. MDN web docs - Основи CSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Learn/CSS_basics.
8. Mongo DB // [Електронний ресурс] – Режим доступу: https://www.mongodb.com/.
9. Wikipedia. MVC // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/модель-вид-контроллер.
10. Wikipedia. UML // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language.
11. Wikipedia. Сценарій використання // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Сценарій використання.
12. Ашманов И.С., Продвижение сайта в поисковых системах / И.С. Ашманов, А.А. Иванов. – М.: Вильямс, 2007. – 304 с. 2.

13. Гото К. Веб-редизайн / К. Гото, Э. Котлер – М.: Символ-Плюс, 2006. – 416 с.

14. Гото Келлі Веб-редизайн, 2-е видання / Котлер Емілі. – СПб.: Символ Плюс, 206. – 416 с.

15. Завадський І. О. Основи баз даних / І. О. Завадський. – Київ: ПП І.О. Завадський, 2011. – 192 с.

16. Кантор Марри. Управление программными проектами. Издательство: Вильямс, 2002 г. — 176 стр.

17. Петлюшкин А.В., HTML в Web-дизайне. – СПб.: БВХ-Петербург, 2004. – 400 с.: ил.

18. Постановка задачи при проектировании сайта. – Режим доступу: http://site-manager.ru/webmasters/lib/110/.

19. Програмування по-українськи - А що таке HTML? [Електронний ресурс] – Режим доступу до ресурсу: http://programming.in.ua/web-design/html/73-html-introdution.html.

20. Прототип сайту – як він допомагає оцінювати і розробляти сайти [Електронний ресурс] – Режим доступу до ресурсу: http://evergreens.com.ua/ua/articles/prototype-site.html.

21. Севостьянов И.О., Поисковая оптимизация. Практическое руководство по продвижению сайта в Интернете / И.О. Севостьянов. – СПб.: Питер, 2010. – 240 с. 9.

22. Современный учебник Javascript Введение в – AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/ajax-intro.

23. Типи сайтів // [Електронний ресурс] – Режим доступу: https://internetdevels.ua/blog/most-common-types-of-websites .

24. Уэйншенк С. Интуитивный веб-дизайн / С. Уэйншенк – М.:, Эксмо, 2011. – 160 с.

25. Фрейн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн – СПб.: Питер, 2014. – 304 с.

26. Ken Pugh Interface Oriented DesignWith Patterns. / Publisher: Pragmatic Bookshelf, July 2006 – 240p.

27. Eric Elliott Programming JavaScript ApplicationsRobust Web Architecture With Node, HTML5, and Modern JS Libraries. / Publisher: O'Reilly Media, February 2013. – 300p.

28. API в веб-приложениях : [Електронний ресурс] – режим доступу до ресурсу : https://mkdev.me/posts/chto-takoe-api-v-veb-prilozheniyah-i-zachem-on-nuzhen.

29. Working with Web API : [Електронний ресурс] – режим доступу до ресурсу : https://launchschool.com/books/working_with_apis

30. PSR-7: HTTP message interfaces: [Електронний ресурс] – режим доступу до ресурсу : http://www.php-fig.org/psr/psr-7/.

31. SOAP и REST[Электронный ресурс]/ Л. Черняк. – 2003. – Режим доступа: http://www.osp.ru/os/2003/09/183376/

32. Интеграция приложний в WWW[Электронный ресурс]– Режим доступа:http://www.4stud.info/networking/web-services.html

33. API[Электронный ресурс]– Режим доступа: http://progschool.ru/products/webapi/

34. . Learn API testing [Электронный ресурс]– Режим доступа: 52 http://www.guru99.com/api-testing.html

35. SOAP-vs-REST [Электронный ресурс] – Режим доступа: http://www.alliancetek.com/Blog/post/2012/10/12/SOAP-vs-REST.aspx

36. OpenLibraryBooks API[Электронный ресурс] – Режим доступа: http://openlibrary.org/dev/docs/api/books

37. Web services. API. [Электронный ресурс] – Режим доступа:http://research.cs.wisc.edu/htcondor/manual/v7.6/4_5Applicat ion_Program.html

38. Kevin MakiceTwitter API: Up and RunningLearn How to Build Applications with the Twitter API / Publisher: O'Reilly Media, March 2009. – 416p.

39. James McGovernJava Web Services Architecture / James McGovern, Sameer Tyagi, Michael Stevens, Sunil Mathew / Publisher: Elsevier, May 2003. – 832p.

40. Stephen T. The Art of Software ArchitectureDesign Methods and Techniques / Publisher: WileyReleased, April 2003. – 336p.