Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування факультету)

_____Кафедра обчислювальної техніки_____
(повна назва кафедри)

Пояснювальна записка

До магістерської кваліфікаційної роботи

_____магістр_____
(освітньо-кваліфікаційний рівень)

на тему_Сервіс для організації зустрічей клубу настільних ігор_____
_____
_____

Виконав магістрант, групи КІ-18м
напряму підготовки (спеціальності)
_____ 08-23.МКР.006.00.000 ПЗ_____
(шифр і назва напряму підготовки, спеціальності)

_____Ревера Д. В._____
(прізвище та ініціали)
Керівник __Гарнага В. А._____
(прізвище та ініціали)
Рецензент__Дудатьєв А. В._____
(прізвище та ініціали)

м. Вінниця - 2020 року

ДОДАТОК А — Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. Т. Б. Мартинюк

«___»_____2020 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Сервіс для організації зустрічей клубу настільних ігор

08-23.МКР.00.00.000 ПЗ

Науковий керівник:

_____Гарнага В. А._____

Магістрант групи КІ-18м

_____Ревера Д. В._____

Вінниця 2020

АНОТАЦІЯ

В даній магістерській роботі було розглянуто створення веб-ресурсу для організації зустрічей представників клубу настільних ігор. Даний ресурс допомагає користувачам створювати зустрічі, переглядати уже створені заплановані зустрічі, та підписуватись, на прийняття участі в них.

ANOTATION

In the current master's thesis was considered a web resource for manage the meeting for the representatives of the board games club. This resource helps users to create the meetings, view the meetings which ware already created and subscribe on them.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасний рівень інтернет соціалізації дає змогу нам знайти інформації, майже про все чого ми потребуємо. Існує багато ресурсних архітектур, які останнім часом все частіше зав'язується на швидких, так званих «лендінг-пейджах», які містять лише стислу інформацію, про певну галузь, а решту непохідної вам інформації можна знайти лише перейшовши за посиланням у мобільний додаток певного месседжеру, таких як Telegram, Viber і їм подібні.

І відповідно є ряд мінусів такої реалізації:

— зазвичай в подібного роду групах, канал, ботах, міститься інформація лише базуючась на чомусь, наприклад: якщо це канал, то там розміщена лише інформація, яку надав автор, а авторів таких не більше 4-5. У групах все навпаки, автором може бути кожен, і це має свою погану сторону, тому що, чим більше авторів, тим більше інформації, а чим більше інформації, тим важче її знайти;

— і ще одним мінусом, це ряд правил, котрі не порушують закони країн, але побудованій з власних міркувань компаній розробників, порушивши котрі, сам канал/група назавжди може втрати існуючу там інформацію.

Тому базуючись на цьому, для вузько направлених галузей, сайти, ще не втратили свою актуальність. По-перше, ти не залежиш від одного чи іншого типу месседжеру, для отримання, або надання інформації, тобі потрібен в цілому лише вихід в глобальну мережу інтернет та пристрій який має змогу відображати веб сторінку.

У даній магістерській роботі за основу була взята за основу одна і найпопулярніших на теперішній момент часу мова програмування JavaScript. Це універсальна мова, яку можна використовувати як для клієнтської частини, так і для серверної.

За основу для збереження даних була використана досить проста в експлуатації та освоєнні технологія MongoDB. Це новий підхід до збереження даних, який не використовує ні ключів, ні схем, що робить її використання ще простішим. Також має в собі надзвичайну особливість якісь дозволяє в одному документі, зберігати надзвичайно різні типи даних, чи то текстове поле, чи JSON об’єкт, або і хоча б *doc файл.

Ще однією особливістю роботи є система розміщення клієнтської та серверної частини в середовищі AWS.

AWS — Amazon Web Services, дозволяє створити свою віртуальну машину в будь якій кількості, на майже будь якій операційній системі, розміром до 30gb безкоштовно, що для клієнт/сервера, є більше ніж достатньо.

Зв'язок роботи з науковими програмами, планами, темами.

Тема магістерської роботи відповідає науковому напрямку досліджень кафедри обчислювальної техніки Вінницького національного технічного університету.

Робота виконана у рамках Національної стратегії розвитку освіти в Україні на період до 2021 р. (наказ Президента № 334/2012 від 25.06.2013 р.); плану наукової та навчально-методичної роботи кафедри обчислювальної техніки.

Мета та завдання дослідження. Метою дослідження даної магістерської кваліфікаційної роботи є створення веб ресурсу, для організації зустрічей, який буде зручним в доступі, за рахунок використання браузерних інтернет технологій, так швидкості, за рахунок розміщення на AWS.

Для успішного завершення даного проекту, необхідно опрацювати наступні пункти:

— провести аналіз сучасних технологій веб та серверної розробки;

— віднайти зручний спосіб збереження даних;

— розглянути більш вдалий метод розміщення роботи в глобальній мережі інтернет;

— організувати взаємо зв’язок між клієнт/серверною частиною, а тож між середовищем збереження даних;

— розробити веб додаток та переконатись в його роботі.

Об’єкт дослідження — це веб-серверна архітектура зі сховищем даних, на прикладі веб-сайту для організації зустрічей учасників клубу настільних ігор.

Предмет дослідження — це взаємозв’язок між частинами проекту, таким як клієнт, сервер, та база даних, швидкість їх роботи, та зручність у використані в сучасних реаліях обміну та збереженням інформації в глобальній мережі інтернет.

Методи дослідження. Дослідження, які були проведені під час виконання даної кваліфікаційної магістерської роботи, ґрунтуються на базисно відомих методах збереження інформації, та сучасних без технологіях.

Наукова новизна отриманих результатів полягає в удосконалені методу посилання запитів до бази даних та рівномірному навантаженні на кожну частину проекту, шляхом додавання, ще одного шару NodeJS, Express серверу, який слугує зв’язком між клієнтською частиною, та базою даних, що дає змогу зняти частину навантаження з клієнту.

Практичне значення одержаних результатів в тому, що було розроблено веб ресурс, який дасть змогу зручно організовувати зустрічі, для учасників клубу натільних ігор, з мінімальним витратами розробку проекту, а також є чудовою платформою для майбутнього розміщення реклами, та її монетизації.

Достовірність теоретичних положень кваліфікаційної роботи магістра базується на обробці та аналізу сучасного ринку технологій в даній галузі, чітко сформованим технічним та економічним планами, перегляді біль двох підходів до реалізації розробленого проекту, але різними шляхами та виборі кращого з них.

Особистий внесок магістранта. Усі напрацювання, що виконані в даній роботі, були опрацьовані самостійно магістром так, як і визначення найбільш вірного підходу до реалізації.

1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ВЕБ – РЕСУРСІВ.

1.1 Мова гіпертекстової розмітки HTML.

Мережа Інтернет — мережа глобального користування, і тому вся інформація в мережі повинна бути представлена на універсальній мові, який розуміли б всі користувачі. Мовою публікації, використовуваним в World Wide Web, є HTML. (HyperText Markup Language - мова розмітки гіпертексту)[1].

HTML — мова розмітки, що надає розробникам наступні можливості:

— представляти інформацію в мережі у вигляді електронних документів, з інформаційним вмістом у вигляді форматовано тексту, таблиць, списків, фотографій;

— включати в документи звукові фрагменти, відеокліпи, електроні таблиці і інші додатки, і елементи мультимедіа;

— здійснювати завантаження документів за допомогою активізації гіпертекстового посилання;

— розробляти форми для здійснення взаємодії з віддаленими службами (пошуковими роботами, онлайн магазинами і подібним)[1];

— розмітка документів полягає в тому, що документ видається в вигляді послідовності елементів.

Ключове слово тут — документ. Тобто, ми створюємо, редагуємо і переглядаємо документ в якійсь програмі, в даному випадку — в офісному редакторі. Якщо відкрити такий документ в простому текстовому редакторі, наприклад, в Блокноті, то ми побачимо безліч дивних символів і знаків. Ця каша із символів незрозуміла людині, але зрозуміла для комп'ютерів. Завдяки цьому у внутрішній мові, документ Word набуває певну структуру і вигляд в

самому редакторі, а документ постає перед нами у всій своїй красі з відформатованим текстом і картинками на своєму місці.

HTML — це мова розмітки документів для браузера. Word'ом тут виступає браузер, а документом - HTML сторінка. Це сама основа технології HTML, розуміння якої необхідно для того, щоб не плутати мову розмітки веб документів з мовами програмування. Назва говорить за себе - за допомогою HTML ми розмічуємо, де на сторінці буде показаний елемент, картинка або текст, і в якому порядку вони будуть слідувати один за одним.

HTML теги — це спеціальні команди для браузера. Вони кажуть йому, що, наприклад, слід вважати заголовком сторінки, а що абзацом.

Теги будуються за таким принципом: куточок <, потім ім'я тега, а потім куточок>, ось так: <ім'я тега>. Ім'я тега може складатися з англійських букв і цифр. Приклади тегів: <h1>, <p>, <b>.

Теги зазвичай пишуться парами - відкриває тег і відповідний йому закриваючий. Різниця між відкриває і закриває тегами в тому, що в закриває тезі після куточка <слеш /.

Наприклад, <p> — так виглядає відкритий тег p, а так — </ p> - так закритий. Все, що потрапляє між відкритим і закритим тегами, підпадає під впливданого тега.

Бувають теги, які не потрібно закривати, наприклад,
 або <img>.

Протягом перших п’яти років (1990-1995) HTML пройшов ряд змін і зазнав низки розширень, в основному, спочатку розміщувався в CERN, а потім в IETF.

Зі створенням W3C розробка HTML знову змінила місце проведення. Перша абонентна спроба розширення HTML у 1995 році, відома як HTML 3.0,

потім поступилася місцем більш прагматичному підходу, відомому як HTML 3.2, який був завершений у 1997 році.

Наступного року членство в W3C вирішило припинити розвиватися HTML і замість цього почати роботу над еквівалентом на основі XML, який називається XHTML. Ці зусилля розпочалися з переформулювання HTML4 в XML, відомого як XHTML 1.0, який не додав нових функцій, окрім нової серіалізації, і який був завершений у 2000 р. Після XHTML 1.0, фокус W3C перейшов до спрощення для інших робочих груп розширити XHTML під прапором модуляризації XHTML. Паралельно з цим W3C також працював над новою мовою, несумісною з попередніми мовами HTML та XHTML, називаючи її XHTML2.

Приблизно в той час, коли еволюція HTML була зупинена в 1998 році, частини API для HTML, розроблені постачальниками браузерів, були визначені та опубліковані під назвою DOM Level 1 (у 1998 р.) Та DOM Level 2 Core та DOM HTML 2 рівня (починаючи з 2000 р. Та завершився в 2003 році). Потім ці зусилля були вирішені. Деякі специфікації рівня DOM D3 були опубліковані у 2004 році, але робоча група була закрита до завершення всіх проектів рівня 3.

У 2003 році публікація XForms, технології, яка позиціонується як наступне покоління веб-форм, викликала нове зацікавлення розвиватися самим HTML, а не шукати заміни для нього. Цей інтерес викликаний усвідомленням того, що розгортання XML як веб-технології обмежується цілком новими технологіями (наприклад, RSS та пізнішим Atom), а не заміною існуючих розгорнутих технологій (наприклад, HTML).

Підтвердженням концепції, що свідчить про те, що можна було розширити форми HTML4, щоб забезпечити багато функцій, запроваджених XForms 1.0, не вимагаючи, щоб браузери реалізували двигуни візуалізації,

несумісні з існуючими веб-сторінками HTML, стали першим результатом цього відновленого інтересу. На цій ранній стадії, хоча проект вже був загальнодоступним, а вхід уже вимагався з усіх джерел, специфікація була лише під авторським правом Opera Software.

Ідея, що еволюцію HTML слід відновити, була протестована на семінарі W3C у 2004 році, де деякі принципи, що лежать в основі роботи HTML5 (описані нижче), а також вищезазначений ранній проект пропозиції, що охоплює лише функції, пов'язані з формами, були представлені W3C спільно Mozilla та Opera. Пропозицію було відхилено на тій підставі, що пропозиція суперечила раніше обраному напрямку розвитку мережі Інтернет; співробітники та члени W3C проголосували за продовження розробки заміни на основі XML.

Незабаром після цього Apple, Mozilla та Opera спільно заявили про намір продовжувати працювати над зусиллями під парасолькою нового місця під назвою WHATWG. Створено загальнодоступний список розсилки, і проект був переміщений на сайт WHATWG. Згодом авторські права були змінені для спільної власності всіх трьох постачальників та для того, щоб дозволити повторне використання специфікації.

WHATWG базувався на кількох основних принципах, зокрема, що технології повинні бути зворотно сумісними, що специфікації та реалізації повинні відповідати, навіть якщо це означає зміну специфікації, а не реалізацію, і що технічні характеристики повинні бути деталізовані, щоб реалізація могла досягти повна взаємодія без реверсивної інженерії.

Остання вимога, зокрема, вимагала, щоб область специфікації HTML5 включала те, що раніше було визначено у трьох окремих документах: HTML4, XHTML1 та DOM2 HTML. Це також означало включення значно більше деталей, ніж раніше вважалося нормою.

У 2006 році W3C виявив зацікавленість брати участь у розробці HTML5, а в 2007 році сформував робочу групу, яка заснувала роботу з WHATWG над розробкою специфікації HTML5. Apple, Mozilla та Opera дозволили W3C публікувати специфікацію під авторським правом W3C, зберігаючи версію з менш обмежувальною ліцензією на сайті WHATWG.

Протягом кількох років обидві групи тоді працювали разом. Однак у 2011 році групи дійшли висновку, що вони мали різні цілі: W3C хотів опублікувати "готову" версію "HTML5", тоді як WHATWG хотів продовжувати працювати над Живим стандартом для HTML, постійно підтримуючи специфікацію а не заморожувати його у стані з відомими проблемами та додавати нові функції у міру необхідності розвитку платформи.[2]

1.2 Каскадні таблиці стилів CSS та SCSS.

Каскадні таблиці стилів, які в народі називають CSS, - це проста мова дизайну, призначена для спрощення процесу створення веб-сторінки презентабельними.

CSS обробляє зовнішній вигляд частиною веб-сторінки. За допомогою CSS ви можете керувати кольором тексту, стилем шрифтів, інтервалом між абзацами, розмірами та розміщенням стовпців, навіть фоновими зображеннями чи кольорами які в ньому використовуються, дизайном макета, варіаціями відображення для різних пристроїв та розмірами екрана а також різноманітні інші ефекти.[3]

CSS легко вивчити та зрозуміти, але він забезпечує потужний контроль над поданням HTML-документа. Найчастіше CSS поєднується з мовами розмітки HTML або XHTML.

Переваги CSS:

— CSS економить час - ви можете написати CSS один раз, а потім повторно використовувати той же аркуш на декількох HTML-сторінках. Також є можливість визначити стиль для кожного елемента HTML і застосувати його до якомога більшої кількості веб-сторінок;

— сторінки завантажуються швидше - якщо ви використовуєте CSS, вам не потрібно щоразу писати атрибути тегів HTML. Просто напишіть одне правило CSS тегу та застосуйте його до всіх випадків. Так менше коду означає швидше завантаження;

— простота обслуговування - щоб зробити глобальну зміну, просто змініть стиль, і всі елементи на всіх веб-сторінках будуть оновлені автоматично;

— покращені стилі до HTML - CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати набагато кращий вигляд для своєї веб-сторінки порівняно з атрибутами HTML;

— сумісність декількох пристроїв - аркуші стилів дозволяють оптимізувати вміст для більш ніж одного типу пристроїв. Використовуючи один і той же документ HTML, різні версії веб-сайту можуть бути представлені для портативних пристроїв, таких як КПК та мобільних телефонів або для друку;

— глобальні веб-стандарти — Тепер атрибути HTML застарівають, і рекомендується використовувати CSS. Тож добре почати використовувати CSS на всіх HTML-сторінках, щоб зробити їх сумісними з майбутніми браузерами;

CSS створюється та підтримується групою людей у рамках W3C, що називається Робоча група CSS. Робоча група CSS створює документи, що називаються специфікаціями. Коли специфікація була обговорена та офіційно ратифікована членами W3C, вона стає рекомендацією.

Ці ратифіковані специфікації називаються рекомендаціями, оскільки W3C не контролює фактичну реалізацію мови.[3]

SCSS - "діалект" мови SASS. А що таке SASS? SASS це мова схожий на HAML (вельми лаконічний шаблонізатор), але призначений для спрощення створення CSS-коду. Простіше кажучи, SASS це така мова, код якого спеціальної ruby-програмою транслюється в звичайний CSS код.

Синтаксис цієї мови дуже гнучкий, він враховує безліч дрібниць, які так бажані в CSS. Більш того, в ньому є навіть логіка (@if, each), математика (можна складати як числа, рядки, так і кольори).

Можливо, деякі можливості SCSS здадуться вам надмірними, але, на мій погляд, зайвими вони не будуть, залишаться "про запас".

Відмінність SCSS від SASS полягає в тому, що SCSS більше схожий на звичайний CSS код.

Окрім вищесказаного, є ряд корисних фіч:

— вкладені правила: ви можете вкладати CSS властивості, в кілька наборів дужок {}. Це зробить ваш CSS чистіше і зрозуміліше;

— змінні: в стандартному CSS теж є змінні, але змінні Sass куди більш потужний інструмент. Наприклад, ви можете використовувати змінні в циклах і генерувати значення властивостей динамічно. Також можна впроваджувати змінні в імена властивостей, наприклад так: property-name-N {...};

— краща реалізація операторів: ви можете підсумувати, віднімати, ділити і множити CSS значення. Sass реалізація більш інтуїтивна, ніж стандартний функціонал CSS calc ();

— функції: Sass дозволяє багаторазово використовувати CSS стилі, як функції;

— тригонометрія: крім базових операцій (+, -, *, /), SCSS дозволяє писати власні функції. Наприклад, функції sin і cos можна написати, використовуючи тільки синтаксис Sass / SCSS. Звичайно, вам знадобляться знання тригонометрії. Такі функція можуть знадобиться для створення анімації;

— зручний робочий процес: ви можете писати CSS, використовуючи конструкції, знайомі по іншим мовам: for-цикли, while-цикли, if-else. Але майте на увазі, це тільки препроцесор, а не повноцінна мова, Sass контролює генерацію властивостей і значень, а на виході ви отримуєте стандартний CSS.Міксини: дозволяють один раз створити набір правил, щоб потім використовувати їх багаторазово або змішувати з іншими правилами. Наприклад, міксини використовують для створення окремих тем макета.

Кожен веб-браузер використовує механізм компонування для візуалізації веб-сторінок, а підтримка функціональності CSS між ними не узгоджується. Оскільки браузери не аналізують ідеально CSS, було розроблено кілька методів кодування для націлювання на конкретні браузери з обхідними шляхами (загальновідомі як CSS-хаки або CSS-фільтри). Прийняттю нової функціональності в CSS може перешкоджати відсутність підтримки в основних браузерах. Наприклад, Internet Explorer повільно додав підтримку багатьох функцій CSS 3, що уповільнило використання цих функцій і пошкодило репутацію браузера серед розробників. [3] Щоб забезпечити постійний досвід для своїх користувачів, веб-розробники часто перевіряють свої сайти в різних операційних системах, браузерах та версіях браузера, збільшуючи час розробки та складність. Такі інструменти, як BrowserStack, були створені для зменшення складності підтримки цих середовищ.

На додаток до цих засобів тестування, багато сайтів підтримують списки підтримки браузера для певних властивостей CSS, включаючи CanIUse та Мережу розробників Mozilla. Крім того, CSS 3 визначає запити щодо функцій,

які надають директиву @supports, яка дозволить розробникам орієнтуватися на браузери з підтримкою певної функціональності безпосередньо в їх CSS. 20] CSS, який не підтримується старими браузерами, іноді також може бути виправлений за допомогою поліфайлів JavaScript, які є частинами коду JavaScript, розробленими для того, щоб браузери поводилися послідовно. Ці обхідні шляхи - і необхідність підтримки резервної функціональності - можуть ускладнювати проекти розвитку, і, отже, компанії часто визначають список версій браузерів, які вони будуть і не підтримуватимуть.

Оскільки веб-сайти приймають новіші стандарти коду, несумісні зі старими браузерами, ці браузери можуть бути відхилені від доступу до багатьох ресурсів в Інтернеті (іноді навмисно). [3] Багато найпопулярніших сайтів в Інтернеті не просто візуально деградують на старих браузерах через погану підтримку CSS, але взагалі не працюють, значною мірою через розвиток JavaScript та інших веб-технологій.

1.3 JavaScript

JavaScript (часто скорочується до JS) — це легка, інтерпретована, об'єктно-орієнтована мова з першокласними функціями, і найбільш відома як мова сценаріїв для веб-сторінок, але вона також використовується у багатьох середовищах, що не належать до браузера. Це заснована на прототипі багатомовна парадигмальна сценарна мова, яка є динамічною і підтримує об'єктно-орієнтований, імперативний та функціональний стилі програмування.

JavaScript працює на клієнтській стороні Інтернету, який може бути використаний для проектування / програмування поведінки веб-сторінок у разі події. JavaScript — це легкий у вивченні, а також потужний сценарій мови, широко використовується для контролю поведінки веб-сторінок.

Всупереч поширеній помилці, JavaScript не є "інтерпретованою Java". У двох словах, JavaScript — це динамічна мова сценаріїв, яка підтримує побудову об'єкта на основі прототипу. Базовий синтаксис навмисно схожий як на Java, так і на C ++, щоб зменшити кількість нових понять, необхідних для вивчення мови. Мовні конструкції, такі як, наприклад, if умови, for цикли і інше.

JavaScript може функціонувати як як процедурна, так і об'єктно-орієнтована мова. Об'єкти створюються програмно в JavaScript, шляхом приєднання методів та властивостей до інакше порожніх об'єктів під час виконання, на відміну від синтаксичних визначень класу, поширених у складених мовах, таких як C ++ та Java. Після побудови об'єкта він може бути використаний як база (або прототип) для створення подібних об'єктів.

Динамічні можливості JavaScript включають побудову об'єктів часу виконання, списки змінних параметрів, змінні функцій, створення динамічного сценарію (через eval), інтроспекцію об'єкта (через for ... in) та відновлення вихідного коду (програми JavaScript можуть декомпілювати функціональні органи назад у вихідний текст).[4]

JavaScript на стороні клієнта - це найпоширеніша форма мови. Сценарій повинен бути включений у HTML-документ або посилатися на нього, щоб код інтерпретувався браузером.

Це означає, що веб-сторінка не повинна бути статичним HTML, але вона може включати програми, які взаємодіють з користувачем, контролюють браузер та динамічно створюють вміст HTML.

Механізм клієнтського боку JavaScript надає багато переваг перед традиційними сценаріями CGI на стороні сервера. Наприклад, ви можете використовувати JavaScript, щоб перевірити, чи користувач ввів дійсну електронну адресу в поле форми.

Код JavaScript виконується, коли користувач подає форму, і лише якщо всі записи дійсні, вони будуть передані на Веб-сервер.

JavaScript може використовуватися для лову подій, ініційованих користувачем, таких як натискання кнопок, навігація посилання та інші дії, які користувач явно або неявно ініціює. [12]

Перший в історії JavaScript був створений Брендоном Ічем в Netscape, і з тих пір було оновлено відповідно до версій ECMA-262 Edition 5 та пізніших версій. Цей двигун, кодом названий SpiderMonkey, реалізований в C / C ++. Двигун Rhino, створений головним чином Норрісом Бойдом (також у Netscape) - це реалізація JavaScript, написана на Java. Як і SpiderMonkey, Rhino відповідає сумісності ECMA-262 Edition 5.

Кілька основних оптимізацій виконання, таких як TraceMonkey (Firefox 3.5), JägerMonkey (Firefox 4) та IonMonkey, були додані до двигуна JavaScript SpiderMonkey з часом. Завжди триває робота з підвищення ефективності виконання JavaScript.

Крім перерахованих вище реалізацій, є й інші популярні двигуни JavaScript, такі як:

— Google V8, який використовується в браузері Google Chrome і останніх версіях браузера Opera. Це також двигун, який використовує Node.js;

— JavaScriptCore (SquirrelFish / Nitro), який використовується в деяких браузерах WebKit, таких як Apple Safari;

— Carakan - у старих версіях Opera;

— двигун Chakra, що використовується в Internet Explorer (хоча мова, яку він реалізує, формально називається "JScript", щоб уникнути проблем із торговельною маркою).

Кожен з механізмів JavaScript Mozilla відкриває відкритий API, який розробники додатків можуть використовувати для інтеграції JavaScript у своє програмне забезпечення. На сьогодні найпоширенішим середовищем для роботи JavaScript є веб-браузери. Веб-браузери, як правило, використовують загальнодоступний API для створення об’єктів хоста, відповідальних за відображення DOM у JavaScript.

Інша поширена програма для JavaScript — це як сценарій (веб) серверна мова скриптів. Веб-сервер JavaScript відкриє об'єкти хоста, що представляють HTTP-запит та об’єкти відповіді, які потім можуть маніпулювати програмою JavaScript для динамічного генерування веб-сторінок. Node.js - популярний приклад цього.[6]

1.4 TypeScript

TypeScript — мова програмування, представлений Microsoft в 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript.

Розробником мови TypeScript є Андерс Хейлсберг (англ. Anders Hejlsberg), який створив раніше Turbo Pascal, Delphi і C #.

Специфікації мови відкриті і опубліковані в рамках угоди Open Web Foundation Specification Agreement (OWFa 1.0).

TypeScript є назад сумісним з JavaScript і компілюється в останній. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверної платформою Node.js. Код експериментального компілятора, який транслює TypeScript в JavaScript, поширюється під ліцензією Apache. Його розробка ведеться в публічному репозиторії через сервіс GitHub.

TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах), а також підтримкою підключення модулів, що покликане підвищити швидкість розробки, полегшити читаність, рефакторинг і повторне використання коду, допомогти здійснювати пошук помилок на етапі розробки і компіляції, і, можливо, прискорити виконання програм.

Планується, що в силу повної сумісності адаптація існуючих додатків на нову мову програмування може відбуватися поетапно, шляхом поступового визначення типів.

На момент релізу представлені файли для сприйняття розширеного синтаксису TypeScript для Vim і Emacs, а також плагін для Microsoft Visual Studio.

Одночасно з виходом специфікації розробники підготували файли з деклараціями статичних типів для деяких популярних JavaScript-бібліотек, серед яких jQuery.

TypeScript виник через передбачувані недоліків JavaScript в великомасштабних додатках як в Microsoft, так і у інших користувачів JavaScript. [10] Проблеми з розробкою складних програм на JavaScript привели до необхідності полегшення розробки компонентів мови.

Розробники TypeScript шукали рішення, яке не порушуватиме сумісність зі стандартом і його крос-платформної підтримкою. Знаючи, що тільки стандарт ECMAScript пропонує підтримку в майбутньому для програмування на базі класів (Class-based programming), TypeScript був заснований на цьому припущенні. Це призвело до створення компілятора JavaScript з набором синтаксичних мовних розширень, збільшеним на основі пропозиції, яке трансформує розширення в JavaScript. У цьому сенсі

TypeScript є уявленням про те, що очікувати від ECMAScript 6. Унікальний аспект не в реченні, а в додаванні в TypeScript статичної типізації, що дозволяє статично аналізувати мову, полегшуючи оснащення і IDE підтримку.

TypeScript додає підтримку наступних функцій: класи, модулі і синтаксис функції стрілкою (стрілочних функцій), як вони були запропоновані в стандарті ECMAScript 6.

TypeScript це — розширення мови ECMAScript 5. Додані наступні опції:

— анотації типів і перевірка їх узгодження на етапі компіляції [en] висновок типів;

— класи;

— інтерфейси;

— прераховуючі типи;

— домішка;

— узагальнене програмування;

— модулі;

— скорочений синтаксис «стрілок» для анонімних функцій;

— розширені можливості пошуку і параметри за замовчуванням кортежі.

Синтаксично, TypeScript дуже схожий на JScript .NET, чергову реалізацію Microsoft мовного стандарту ECMA-262, що забезпечує підтримку статичної типізації та класичних об'єктно-орієнтованих можливостей мови, таких як класи, спадкування, інтерфейси і простору імен.

TypeScript є надбудовою над JavaScript. Таким чином, програма JavaScript також є правильною програмою TypeScript, і програми TypeScript можуть легко включати JavaScript. TypeScript компілює ES3-сумісний JavaScript.

За замовчуванням компілюється ECMAScript 3, як переважної стандарт; також є можливість створювати конструкції, які використовуються в ECMAScript 5.

З TypeScript можна використовувати існуючий JavaScript-код, включати популярні бібліотеки JavaScript, і викликати TypeScript-код, згенерований з інших JavaScript. Оголошення типів для цих бібліотек поставляються разом з вихідним кодом.

1.5 NodeJS

Node.js представляє середу виконання коду на JavaScript, яка побудована на основі движка JavaScript Chrome V8, який дозволяє транслювати виклики на мові JavaScript в машинний код. Node.js перш за все призначений для створення серверних додатків на мові JavaScript. Хоча також існують проекти по написанню десктопних додатків (Electron) і навіть зі створення коду для мікроконтролерів. Але перш за все ми говоримо про Node.js, як про платформу для створення веб-додатків.

Node.js не є новою платформою із срібними кулями, яка буде домінувати у світі веб-розробок. Натомість це платформа, яка задовольняє певну потребу. І розуміти це абсолютно важливо. Ви точно не хочете використовувати Node.js для операцій, що вимагають процесора; насправді, використання його для важких обчислень скасує майже всі його переваги. Там, де Node дійсно світить, це створення швидких масштабованих мережевих додатків, оскільки воно здатне обробляти величезну кількість одночасних з'єднань з високою пропускною здатністю, що прирівнюється до високої масштабованості.

Він є відкритим проектом, вихідні коди якого можна подивитися на github.com.

Як це працює під кришкою, досить цікаво. Порівняно з традиційними методами веб-сервісу, коли кожне з'єднання (запит) породжує новий потік, займаючи системну оперативну пам’ять і врешті-решт збільшуючи обсяг

оперативної пам’яті, Node.js працює на одному потоці, використовуючи неблокуючий I / O виклики, що дозволяє йому підтримувати десятки тисяч одночасних з'єднань, що проходять у циклі подій.

Важливими подіями в розвитку платформи стала поява Atomics і SharedArrayBuffer в Node.js 9, а так само worker_threads в Node.js 10.5 (і значного розвитку в Node.js 12). Це дозволило створювати багатопотокові паралельні додатки, реалізовувати примітиви паралельного програмування і працювати з пам'яттю.

Node.js це — Виконавча JavaScript. Що ж це означає, і як працює?

Оточення Node.js включає все, що вам потрібно для виконання програми, написаної на JavaScript.

Раніше ви могли запустити JavaScript тільки в браузері, але одного разу розробники розширили його, і тепер ви можете запускати JS на своєму комп'ютері в якості окремого додатка. Так з'явився Node.js.

Тепер ви можете зробити набагато більше з JavaScript, ніж просто інтерактивні веб-сайти.

Тепер у JavaScript є можливість робити те, що можуть робити інші скриптові мови програмування, такі як Python.

Обидва - браузерні JavaScript і Node.js запускаються в середовищі виконання V8. Цей движок використовує ваш JS код, і перетворює його в більш швидкий машинний код. Машинний — низькорівневий код, який комп'ютер може запускати без необхідності спочатку його інтерпретувати.

NPM — Це бібліотеки, побудовані спільнотою. Вони вирішать більшість часто зустрічаються проблем. npm (менеджер пакетів Node) містить пакети, які ви можете використовувати в своїх додатках, щоб зробити вашу розробку більш швидкою і ефективною.

Require виконує три функції:

— завантажує модулі, що поставляються в комплекті з Node.js, наприклад з файлової системи або HTTP, з API Node.js;

— завантажує сторонні бібліотеки, такі як Express і Mongoose, які ви встановлюєте з npm;

— дозволяє створювати власні файли і ділити проект на модулі.

Require — це функція, і вона приймає параметр «шлях» і повертає module.exports.

Node модулі це — багаторазово використовувані блоки коду, існування яких не випадково не впливає на інший код.

Ви можете написати свої власні модулі та використовувати їх в різних додатках. Node.js має набір вбудованих модулів, що ви можете використовувати без спеціальної установки.

V8 - движок з відкритим вихідним кодом, написаний на C ++.

JavaScript -> V8 (C ++) -> машинний код

V8 реалізує сценарій ECMAScript, як зазначено в ECMA-262. ECMAScript був створений Ecma International для стандартизації JavaScript.

V8 може працювати автономно або може бути вбудований в будь-який додаток C ++. Завдяки цьому, Ви можете написати свій власний код на C ++, і зробити його доступним для JavaScript.

Події — це все, що сталося в нашій додатку, і на що ми можемо відповісти.

У Node є два типи подій:

— системні події: ядро C ++ з бібліотеки libuv. (Наприклад, закінчення читання файлу);

— призначені для користувача події: ядро JavaScript.

1.6 Angular

Angular — це платформа та основа для побудови односторінкових клієнтських додатків за допомогою HTML та TypeScript. Angular розробка виконується на мові TypeScript. Він реалізує основну та додаткову функціональність як набір бібліотек TypeScript, які ви імпортуєте у свої програми.

Архітектура програми Angular спирається на певні фундаментальні концепції. Основними будівельними блоками є NgModules, які забезпечують контекст компіляції компонентів. NgModules збирають відповідний код у функціональні набори.

Angular додаток визначається набором NgModules. У додатку завжди є принаймні кореневий модуль, який дозволяє завантажувати, і, як правило, має ще багато функціональних модулів.

Компоненти визначають представлення даних, що представляють собою набори елементів екрану, які Angular може вибирати та змінювати відповідно до логіки та даних вашої програми.

Компоненти використовують сервіси, які надають певні функціональні можливості, безпосередньо не пов'язані з відображенням. Провайдери можуть бути введені в компоненти як залежності, що робить ваш код модульним, багаторазовим та ефективним.

І компоненти, і сервіси — це просто класи, в яких є декоратори, які позначають їх тип та надають метадані, які вказують Angular, як ними користуватися.

Метадані класу компонентів асоціюють його з шаблоном, який визначає відображення в браузері. Шаблон поєднує звичайний HTML з Angular

директивами та розміткою, що дозволяють Angular змінювати HTML, перш ніж відбудеться відображення.

Метадані для класу провайдера надають інформацію, яку потребує Angular, щоб зробити її доступною для компонентів через введення залежності.

Компоненти програми зазвичай визначають багато шаблонів, розташованих ієрархічно. Angular надає послугу маршрутизатора, щоб допомогти вам визначити шляхи навігації серед компонентів.

Маршрутизатор забезпечує складні навігаційні можливості в браузері.

1.7 MongoDB

MongoDB — документоорієнтованих система управління базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Написана на мові C ++. Використовується в веб-розробці, зокрема, в рамках JavaScript-орієнтованого стека MEAN.

Система підтримує ad-hoc-запити: вони можуть повертати конкретні поля документів і призначені для користувача JavaScript-функції. Підтримується пошук за регулярними виразами. Є підтримка індексів.

Система може працювати з набором реплік, тобто містити дві або більше копії даних на різних вузлах. Кожен екземпляр набору реплік може в будь-який момент виступати в ролі основної або допоміжної репліки. Всі операції запису і читання за замовчуванням здійснюються з основною реплікою. Допоміжні репліки підтримують в актуальному стані копії даних. У разі, коли основна репліка дає збій, набір реплік проводить вибір, яка з реплік повинна стати основною. Другорядні репліки можуть додатково бути джерелом для операцій читання.

Система масштабується горизонтально, використовуючи техніку сегментування (англ. Sharding) об'єктів баз даних - розподіл їх частин з різних вузлів кластера. Адміністратор вибирає ключ сегментування, який визначає, за яким критерієм дані будуть рознесені по вузлах (в залежності від значень хеша ключа сегментування). Завдяки тому, що кожен вузол кластера може приймати запити, забезпечується балансування навантаження.

Система може бути використана в якості файлового сховища з балансуванням навантаження і репликацией даних (функція Grid File System; поставляється разом з драйверами MongoDB). Надаються програмні засоби для роботи з файлами і їх вмістом. GridFS використовується в плагінах для Nginx і lighttpd. GridFS розділяє файл на частини і зберігає кожну частину як окремий документ..

Може працювати відповідно до парадигми MapReduce. У фреймворку для агрегації є аналог SQL-інструкції GROUP BY. Оператори агрегації можуть бути пов'язані в конвеєр подібно UNIX-Конвейр. Фреймворк так само має оператор $ lookup для зв'язки документів при вивантаженні і статистичні операції такі як середньоквадратичне відхилення.

Підтримується JavaScript в запитах, функціях агрегації (наприклад, в MapReduce).

MongoDB підтримує колекції з фіксованим розміром. Такі колекції зберігають порядок вставки і після досягнення заданого розміру поводяться як кільцевої буфер.

У червні 2018 року (у версії 4.0) додана підтримка транзакцій, які відповідають вимогам ACID.

У MongoDB є офіційні драйвери для основних мов програмування (Ci, C ++, C #, Erlang, Golang, Haskell, J #, Java, JavaScript, Lisp, Perl, PHP, Python,

Ruby, Delphi, Scala). Існує також велика кількість неофіційних або підтримуваних спільнотою драйверів для інших мов програмування і фреймворків.

Основним інтерфейсом до бази даних була командна оболонка «mongo». З версії MongoDB 3.2 в якості графічної оболонки поставляється «MongoDB Compass». Існують продукти і сторонні проекти, які пропонують інструменти з GUI для адміністрування і перегляду даних.

База даних MongoDB підходить для наступних застосувань:

— зберігання та реєстрація подій;

— системи управління документами і контентом;

— електронна комерція;

— ігри;

— дані моніторингу, датчиків;

— мобільні додатки;

— сховище операційних даних веб-сторінок (наприклад, зберігання коментарів, оцінок, профілів користувачів, сеанси користувачів).

1.8 Amazon Web Services

Amazon Web Services (AWS) — це найпоширеніша в світі хмарна платформа з широкими можливостями, що надає більше 175 повнофункціональних сервісів для центрів обробки даних по всій планеті. Мільйони клієнтів, в тому числі стартапи, які стали лідерами за швидкістю зростання, найбільші корпорації і передові урядові установи, використовують AWS для зниження витрат, підвищення гнучкості і прискореного впровадження інновацій.

EC2 дозволяє запускати вже заздалегідь сконфігуровані сервери з попередньо встановленими ОС: Amazon Linux, Red Hat EL, Suse ES, Windows 2008, Oracle EL, Вибір операційних систем виглядає так:

Так само можливо створювати свої образи (AMI - Amazon Machine Image) і використовувати будь-який Linux. Наша платформа використовує Debian Squeez як основну систему, але, звичайно ж ми можемо запустити і працювати практично на будь-якому дистрибутиві Linux, наприклад CentOS або Ubuntu. Так само ми підтримуємо RHEL і Suse ES.

Є можливість налаштувати захист доступу до серверів. EC2 інстанси об'єднуються в групи безпеки (Security Groups) з можливістю обмеження доступу по портам з IP або підмереж.

Балансування навантаження і автомасштабирование є дуже важливими функціями EC2. Ви можете створити правила при яких стане можливо автоматично збільшити кількість серверів, наприклад, якщо один або декілька серверів не справляються з навантаженням. Контроль за здоров'ям серверів веде ще один сервіс AWS - Amazon Cloud Watch. За допомогою цього сервісу можна створювати різного роду перевірки - checks - за допомогою яких контролюються найважливіші показники роботи ОС.

Amazon Elastic Compute Cloud (Amazon EC2) — веб-сервіс, який надає обчислювальні потужності в хмарі. Простий веб-інтерфейс сервісу дозволяє отримати доступ до обчислювальних потужностей і налаштувати з мінімальними витратами ресурсів. Він надає користувачам повний контроль над обчислювальними ресурсами, а також доступну середовище для роботи. Сервіс скорочує час, необхідний для отримання та завантаження нового сервера.

— створити Amazon Machine Image (AMI), який буде містити ваші програми, бібліотеки, дані і пов'язані з ними конфігураційні параметри. Або використовувати заздалегідь налаштовані шаблони образів для роботи;

— завантажити AMI в Amazon S3. Amazon EC2 надає інструменти, для зберігання AMI. Amazon S3 забезпечує безпечне, надійне і швидке сховище для зберігання образів;

— використовувати Amazon EC2 Веб-сервіс для настройки безпеки і мережевого доступу;

— вибирати тип (и) операційної системи, який вам необхідний, запустити, завершити, або контролювати кілька AMI в міру необхідності, використовуючи API Веб-сервісу, або різних інструментів управління, які передбачені;

— визначити необхідність працювати в декількох місцях, використовувати статичний IP або інші варіанти;

— платити тільки за ресурси, які ви збираєтеся споживати, такі як час або передача даних.

Офіційно запущений в 2006 році, Amazon Web Services надає онлайн послуги для інших веб - сайтів або клієнтських додатків. Більшість з цих послуг не піддаються безпосередньо кінцевим користувачам, але замість того, щоб запропонувати функціональні можливості, що і інші розробники можуть використовувати в своїх додатках. Пропозиції Amazon Web Services 'доступні через HTTP, використовуючи REST архітектурний стиль і SOAP - протокол. Всі послуги тарифікуються на основі використання, але як використання вимірюється для виставлення рахунків залежить від сервісу.

2.ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір Angular, як фреймворк для реалізації клієнтської частини

Angular дозволяє вам з "коробки" створювати великі і складні за частиною бізнес-логіки додатка. Angular було повним переосмисленням AngularJS, напевно, це було найболючіше, але воно того варте, сам фреймворк став куди чистіше і гнучкіше, більш enterprise-подібним і з цієї точки зору має високу масштабованість.

Які плюси можна виділити:

— підтримка Google, Microsoft;

— інструменти розробника (CLI);

— єдина структура проекту;

— TypeScript з "коробки" (ви можете писати строго типізований код);

— пеактивний програмування з RxJS;

— єдиний фреймворк з Dependency Injection з "коробки";

— шаблони, засновані на розширенні HTML;

— кросбраузерності Shadow DOM з коробки (або його емуляція);

— кросбраузерності підтримка HTTP, WebSockets, Service Workers;

— не потрібно нічого додатково налаштовувати. Більше ніяких обгорток;

— більш сучасний фреймворк, ніж AngularJS (на рівні React, Vue);

— велике ком'юніті.

Щоб залишатися чесними, варто виділити і мінуси:

— вище поріг входження через Observable (RxJS) і Dependency Injeciton;

— щоб все працювало добре і швидко потрібно витрачати час на додаткові оптимізації (він не супер швидкий, за замовчуванням, але швидше AngularJS у багато разів і з кожною новою версією стає все швидше);

— насправді, якщо ви плануєте розробляти велику enterprise-додаток, то в цьому випадку у вас немає архітектури для управління станом з "коробки" - потрібно додавати Mobx, Redux, CQRS / CQS або інший state-менеджер, щоб потім не зламати собі мозок ;

— Angular-Univesal має багато підводних каменів;

— динамічне створення компонентів виявляється нетривіальною завданням.

Насправді, всі ці мінуси нівелюються власним досвідом розробника. Все, що вам доведеться дізнатися в Angular для розробки продуктивних і швидко працюють додатків будь-якого рівня складності, описано в нижче наступних концепціях:

Form Builder — для розробки воістину складних форм, вам слід знати реактивні форми, точніше принципово забути про декларативні форми. Ось один з хороших прикладів (реактивна форма + валідація);

Change Detection — так як Angular за замовчуванням використовує двостороннє зв'язування моделі даних, то при роботі з великим об'ємом таких даних ваші програми будуть працювати повільніше, тому в деяких випадках варто подбати про правильну стратегію виявлення змін. Ви можете подивитися на різні OpenSource проекти: PrimeNG, Angular Material, Clarity UI, Angular Bootstrap і інші, всі вони використовують ChangeDetection.OnPush.

Templating — синтаксис шаблонів з точки зору абстракції змінився не дуже сильно в порівнянні з AngularJS, тобто ми так само можемо написати умови, цикли, зв'язати модель даних та інше. Все, що вам слід добре зрозуміти

і розібратися в Angular шаблонах - що таке структурні і декларативні директиви, а також що таке Input-параметри і Output-події.

Routing — напевно, це одне з основних явищ у розробці веб-додатків. Тут просто важливо розуміти, що маршрутизація так само, як і компоненти, має свій життєвий цикл, розуміючи це, ви можете писати воістину круті додатки. Ще варто відзначити: якщо на якійсь із маршрутів ви вішаєте модуль, а не компонент, який відповідає за відображення сторінки по цьому маршруту, то модуль буде довантажувати на сторінці на вимогу.

Annotations — до слова, багато новачків не знають, але варто відзначити. Декоратори, які використовуються в достатку при написанні додатків на Angular, не є якоюсь жорсткою магією TypeScript. Декоратори є специфікацією EcmaScript і коли браузери почнуть підтримувати їх, вони будуть нативної виконуватися в browser runtime. Насправді, декоратори вельми корисні і забезпечують досить високою читаністю ваш код. Один із прикладів — це валідація моделей даних з використанням декораторів або десеріалізацію / сериализация даних.

Observables — насправді, тут варто відзначити тільки те, що незабаром Observables будуть специфікацією EcmaScript і все це буде нативної підтримуватися в браузерах. З точки зору теорії, якщо розкрити поняття Observer (спостерігач) — це поведінковий шаблон проектування. Також відомий як «підлеглі» (Dependents). Створює механізм у класу, який дозволяє отримувати примірника об'єкта цього класу оповіщення від інших об'єктів про зміну їх стану, тим самим спостерігаючи за ними.

Shadow DOM — це засіб для створення окремого DOM-дерева всередині елемента, яке не видно зовні без застосування спеціальних методів, є специфікацією W3C. Грубо кажучи, це зручний спосіб створення ізольованих і переіспользуемих веб-компонентів. Чисто технічно, якби браузери вже

сьогодні підтримували багато концепції, які використовує Angular, нам не потрібні були б транспайлери і інші системи збирання, все, що ми писали на Angular, працювало б уже нативної.

Також було використано декілька основних бібліотек:

Material Design — це мова дизайну для веб-і мобільних додатків, який був розроблений Google в 2014 році. Material Design спрощує розробникам настройку UI, зберігаючи при цьому зручний інтерфейс додатків. З Material Design ви отримуєте добре організований формат і гнучкість, щоб висловити свій бренд і стиль.

Якщо ви хочете отримати уявлення про Material Design, ознайомтеся з дизайном домашньої сторінки Houzz, Gmail, Google Play та інші серві Google які в 2016 році виграли нагороду Google Play. Ця програма використовує Material Design, щоб реалізувати численні функції в обмеженому просторі мобільних пристроїв. Це дає хороший досвід для користувачів, дозволяючи їм переглядати і переміщатися по додатку, не відчуваючи себе перевантаженим. Це дійсно одна з кращих програм Material Design.

RxJS — це бібліотека для реактивного програмування, яка дозволить зручно організувати роботу з подіями і асинхронним кодом, писати складну логіку декларативно. RxJS активно використовується в фреймворку Angular, а також з Vue (Vue-rx) і лежить в основі реалізації middleware для Redux (redux-observable) для React.

Http — Для взаємодії з сервером і відправлення запитів по протоколу HTTP застосовується клас HttpClient. Цей клас визначає ряд методів для відправки різного роду запитів: GET, POST, PUT, DELETE. Даний клас побудований поверх стандартного об'єкта в JavaScript - XMLHttpRequest.

2.2 Вибір Exxpress, як платформи для побудови back-end частини

Express — це фреймворк для Node.js, який реалізовує шар функцій, необхідних для створення ефективних програм і API. Його використання значно скорочує написання коду, а, значить, зменшується витрачається на розробку час. Він є мінімалістичний і гнучкий фреймворк для веб-додатків, який надає широкий набір функцій для створення однієї або безлічі сторінок і гібридних веб-додатків.

Express, як добре відомо, розвивається своїм шляхом, на відміну від інших фреймворків, багато в чому спираються на Rails, але також багато запозичив з іншого Ruby-фреймворка під назвою Sinatra. Концепція проста: фреймворк надає достатньо можливостей для запуску і роботи «на льоту», не вимагаючи багато часу на підготовку.

У цьому керівництві ми будемо використовувати Express в якості основного інструменту для отримання веб-додатки і запуску його з сервером, підтримки маршрутів, сторінки помилок, ведення логів і т. д.

Це легка платформа веб-додатків, яка допоможе організувати ваше веб-додаток в архітектурі MVC на стороні сервера. Ви можете використовувати різні варіанти для мови шаблонів (наприклад, EJS, Jade і Dust.js).

Потім ви можете використовувати базу даних, наприклад MongoDB з Mongoose (для моделювання), щоб забезпечити бекенда для вашого застосування Node.js. Express.js в основному допомагає вам управляти всім, від маршрутів, обробляти запити і уявлення.

2.3 Вибір MongoDB, як сховище даних.

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні об'єктно-реляційних баз даних.

З часів динозаврів було звичайною справою зберігати всі дані в реляційних базах даних (MS SQL, MySQL, Oracle, PostgresSQL). При цьому було не так важливо, а чи підходять реляційні бази даних для зберігання даного типу даних чи ні.

На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати.

Але, навіть враховуючи всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних - в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (Бісон) або скорочення від binary JSON.

BSON дозволяє працювати з даними швидше: швидше виконується пошук і обробка. Хоча треба зазначити, що BSON на відміну від зберігання даних в форматі JSON має невеликий недолік: в цілому дані в JSON-форматі

займають менше місця, ніж в форматі BSON, з іншого боку, даний недолік з лишком окупається швидкістю.

MongoDB написана на C ++, тому її легко перенести на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити вихідний код і самому скомпілювати MongoDB, але рекомендується використовувати бібліотеки з офсайта.

Якщо реляційні бази даних зберігають рядки, то MongoDB зберігає документи. На відміну від рядків документи можуть зберігати складну за структурою інформацію. Документ можна уявити як сховище ключів і значень.

Ключ являє просту мітку, з яким асоційоване певний шматок даних.

Однак при всіх відмінностях є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СУБД зустрічається таке поняття як первинний ключ. Це поняття описує якийсь стовпець, який має унікальні значення. У MongoDB для кожного документа є унікальний ідентифікатор, який називається _id. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Кожному ключу зіставляється певне значення. Але тут також треба враховувати одну особливість: якщо в реляційних базах є чітко окреслена структура, де є поля, і якщо якесь поле не має значення, йому (в залежності від налаштувань конкретної бд) можна привласнити значення NULL. У MongoDB все інакше. Якщо якомусь ключ не порівнювати значення, то цей ключ просто опускається в документі і не вживається.

Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані

об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Система зберігання даних в MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Всі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один з вторинних вузлів стає головним.

Відсутність жорсткої схеми бази даних і в зв'язку з цим потреби при щонайменшій зміні концепції зберігання даних створити заново цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про перестворення бази даних і витрачати час на побудову складних запитів.

2.4 AWS Instances, як область розміщення клієнт/серверної частини.

Amazon EC2 — це істинно віртуальне середовище обчислень, що дозволяє використовувати інтерфейси веб-сервісів для запуску інстанси з різними операційними системами, завантажувати на них ваші програми, управляти дозволами на доступ до мережі і запускати образи, використовуючи стільки систем, скільки вам потрібно.

Користуватися Amazon EC2 досить просто, для цього необхідно наступне:

Виберіть попередньо налаштований образ машини Amazon (AMI) або створіть власний образ (AMI), що містить ваші програми, бібліотеки, дані і пов'язані з ними параметри конфігурації. Крім того, ви можете створити образ AMI, що містить ваші програми, бібліотеки, дані і пов'язані з ними параметри конфігурації.

Налаштуйте систему безпеки і доступ до мережі в своєму інстанси Amazon EC2.

Виберіть необхідні типи інстанси, після чого ви зможете запускати, припиняти дію і відстежувати необхідну кількість інстанси способу AMI за допомогою API веб-сервісів або різних наданих інструментів управління.

Визначте, чи потрібно запускати інстанси в декількох місцях розташування, використовувати кінцеві точки зі статичними IP-адресами або прикріплювати постійні блокові сховища до ваших інстанси.

Платіть тільки за реально спожиті ресурси, наприклад за інстанси-годинник або за обсяги переданих даних.

Продукт Amazon EC2 має потужні функції для створення масштабованих відмовостійких додатків корпоративного класу.

Інстанси без встановленої операційної системи

Інстанси Amazon EC2 без встановленої операційної системи надають додаткам прямий доступ до процесора і пам'яті базового сервера. Ці інстанси оптимально підходять для робочих навантажень, що вимагають доступу до наборів апаратних функцій (наприклад, Intel® VT-x), або для додатків, які необхідно запускати в невіртуалізірованних середовищах для забезпечення вимог ліцензування або підтримки. Інстанси без встановленої операційної системи створені на основі системи Nitro - набору компонентів апаратної розвантаження і апаратного захисту AWS, який надає безпечні і високопродуктивні мережеві ресурси і ресурси сховища для інстанси EC2. Інстанси без встановленої операційної системи теж є інстанси EC2, а значить, забезпечують таку ж високу безпеку, надійність, еластичність ресурсів і підтримку різних операційних систем і пакетів ПО, як і інші віртуальні інстанси EC2. Інстанси без встановленої операційної системи також можна

використовувати з сервісами AWS, наприклад з Amazon Virtual Private Cloud (VPC), Elastic Block Store (EBS) і Elastic Load Balancing (ELB) і іншими.

Оптимізація продуктивності і вартості обчислень за допомогою груп інстанси Amazon EC2

Використання груп інстанси в Amazon EC2 дозволяє за один виклик API виділити обчислювальні ресурси з використанням різних типів інстанси EC2, кількох зон доступності та різних схем придбання. Це дозволяє оптимізувати масштаб, продуктивність і витрати. Сервіс дозволяє вказати обсяг ресурсів на вимогу і спотових ресурсів, які будуть запускатися в рамках групи інстанси EC2. Можна також визначити бажані типи інстанси і налаштувати масштабування ресурсів на основі ядер, обсягу пам'яті або кількості інстанси.

Застосування Amazon EC2 Auto Scaling забезпечує доступ до всіх можливостей груп інстанси EC2: виділення і автоматичне масштабування обчислювальних ресурсів з використанням будь-яких типів інстанси EC2, зон доступності та способів придбання інстанси виконується в єдиній групі Auto Scaling.

Тепер можна перевести свої інстанси Amazon EC2, зарезервовані за допомогою Amazon EBS, в режим гібернації, а потім відновити їх роботу. Ця функція особливо корисна при роботі з додатками, початкове завантаження яких займає певний час і які зберігають стан в пам'яті (ОЗУ). Гибернация забезпечує всі переваги зупинки і запуску системи. Крім того, дані в пам'яті ОЗУ також зберігаються між сеансами. При перекладі інстанси в сплячий режим плата за користування інстанси не стягується. Зберігання даних сплачується за стандартними розцінками EBS. Додаткові відомості про глибокого сну і підтримувані типи інстанси і ОС.

Клієнти, яким потрібна велика обчислювальна потужність для розрахунків з плаваючою комою, зможуть використовувати обчислювальні

інстанси GPU Compute загального призначення наступного покоління від AWS - інстанси Amazon EC2 P3, в яких використовується до 8 графічних процесорів NVIDIA® V100 з ядрами Tensor. Інстанси P3 забезпечують продуктивність обчислень з плаваючою комою до 1 петафлопса в режимі змішаної точності, до 125 терафлопс в режимі одинарної точності і до 62 терафлопс в режимі подвійної точності. Внутрішнє з'єднання NVLink другого покоління, що забезпечує передачу даних зі швидкістю 300 ГБ / с, дозволяє здійснювати обмін даними між графічними процесорами з високою швидкістю і низькими затримками. Інстанси P3 також підтримують до 96 віртуальних ЦПУ на базі спеціальних процесорів Intel, 768 ГБ DRAM і 100 Гбіт / с виділеної сукупної пропускної здатності мережі з використанням еластичного мережевого адаптера (ENA). Інстанси P3 ідеально підходять для робочих навантажень, пов'язаних з машинним навчанням, високопродуктивними обчисленнями, обчислювальної гідродинаміки, фінансової інженерією, сейсмічним аналізом, молекулярним моделюванням, геноміки та рендерингом.

Використання інстанси GPU Graphics стане перевагою для клієнтів, яким потрібна можливість обробки великого обсягу графічних даних. G3, інстанси GPU Graphics поточного покоління, забезпечує доступ до графічних процесорів NVIDIA Tesla M60, кожен з яких має до 2048 паралельних процесорних ядер, 8 гігабайт графічної пам'яті і апаратний кодировщик, що підтримує до 10 потоків H.265 (HEVC) 1080p30 і до 18 потоків H.264 1080p30. При використанні останніх версій драйверів такі графічні процесори підтримують технології OpenGL, DirectX, CUDA, OpenCL і Capture SDK (раніше відомий як GRID SDK). Інстанси GPU Graphics ідеально підходять для тривимірних візуалізацій, тривимірного рендеринга, створення віддалених робочих станцій для обробки графіки, запуску додатків потокової передачі, кодування відео і інших графічних робочих навантажень на стороні сервера.

Клієнти, яким необхідно забезпечити високошвидкісний доступ для операцій випадкового введення-виведення даних з малими затримками, можуть використовувати інстанси високопродуктивного сховища. Інстанси високопродуктивного сховища - це тип інстанси Amazon EC2, який здатний забезпечити для клієнтів швидкість, яка перевищує 3 мільйони випадкових операцій введення-виведення в секунду. Інстанси високопродуктивного сховища I3 і I3en побудовані на твердотільних накопичувачах на базі NVMe і ідеально підходять для високопродуктивних баз даних NoSQL, транзакційних систем і робочих навантажень Elastic Search. Інстанси високопродуктивного сховища також забезпечують послідовну пропускну здатність носіїв до 16 ГБ / с. Цього цілком достатньо для виконання будь-яких робочих навантажень, пов'язаних з аналітикою.

Функція Optimize CPUs надає більш широкі можливості управління інстанси Amazon EC2. По-перше, ви можете при запуску нових інстанси вказати власне кількість віртуальних процесорів, щоб зменшити витрати на ліцензування на основі кількості віртуальних процесорів. По-друге, можна відключити технологію багатопоточності для тих робочих навантажень, які в достатній мірі обробляються однопоточні ЦПУ, наприклад для створення додатків для певних високопродуктивних обчислень (HPC).

Еластичні IP-адреси - це статичні IP-адреси, призначені для динамічних хмарних обчислень. Еластичний IP-адреса пов'язаний з вашим аккаунтом, а не з конкретним інстанси. Ви можете контролювати цю адресу до тих пір, поки явно не відмовитеся від нього. Проте, на відміну від традиційних статичних IP-адрес еластичні IP-адреси дозволяють маскувати збої інстанси або зони доступності шляхом програмного перерозподілу публічних IP будь-якого інстанси в ваш обліковий запис. Замість того щоб чекати, коли фахівець по роботі з даними змінить настройки хоста або замінить його, або чекати, коли DNS поширюється серед всіх ваших клієнтів, за допомогою Amazon EC2 можна

швидко вирішити проблеми, пов'язані з вашим інстанси або програмним забезпеченням, перепризначити еластичний IP- адреса для нового інстанси. Крім того, при необхідності ви можете налаштувати зворотний запис DNS будь-якого з ваших еластичних IP-адрес, заповнивши відповідну форму.

2.5 Ознайомлення з роботою проекту.

Як тільки вперше користувач відкриває сайт, його зустрічає стартова сторінка, яка містить в собі дві кнопки «Реєстрація» та «Авторизація».

Рис. 2.1 — «Стартова сторінка»

Відповідно якщо користувач немає облікового запису, йому необхідно створити натиснувши клавішу «Реєстрація». Після натискання відкриється форма для реєстрації нового користувача [Рис. 2].

Після успішного створення користувач може ввійти на сайт за допомого кнопки «Авторизація». Після її натискання відриється діалогове вікно для вводу імені користувача та паролю. [Рис. 3].

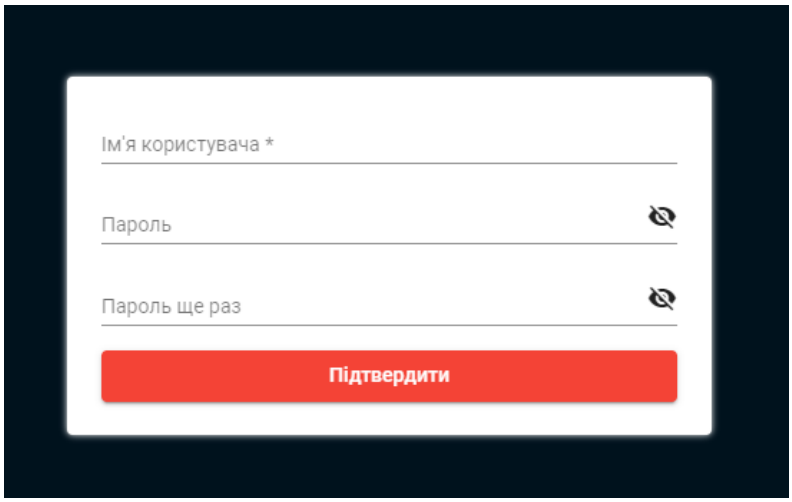

Рис. 2.2 — «Вікно реєстрації»

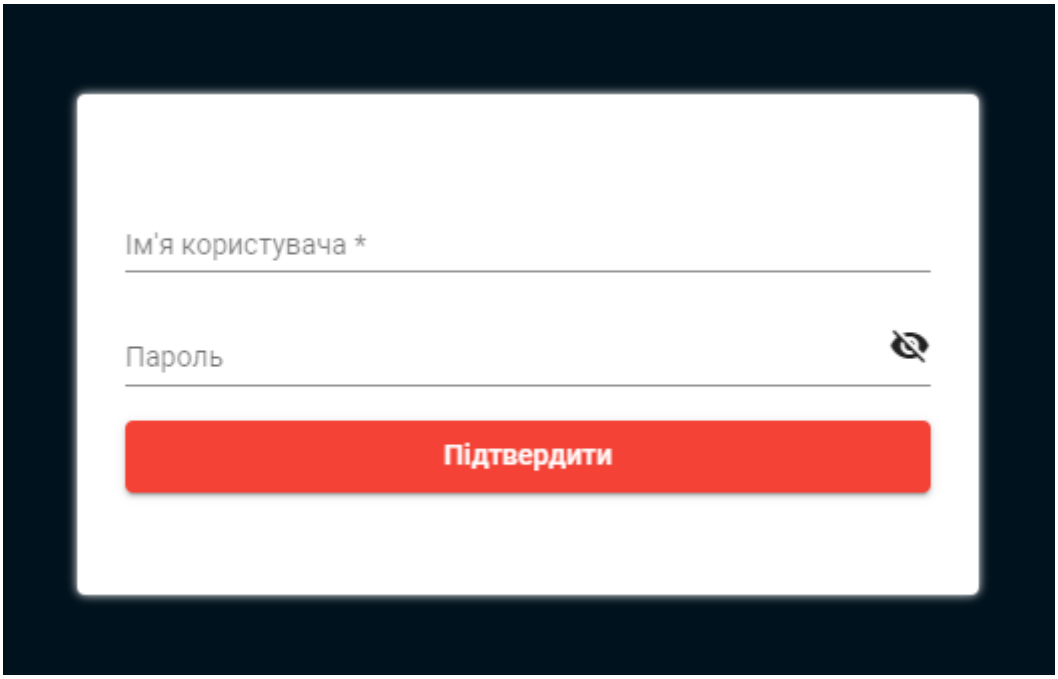

Рис. 2.3 — «Вікно авторизації»

Далі, після успішного завершення авторизації, користувача буде переправлено до головної сторінки сайту. На ній поділено в дві колонки будуть відображатись загальні зустрічі, та зустрічі на якій підписався користувач [Рис. 4]. Зверніть увагу, що якщо користувач вже підписався за зустріч, повторно він цього зробити не може.

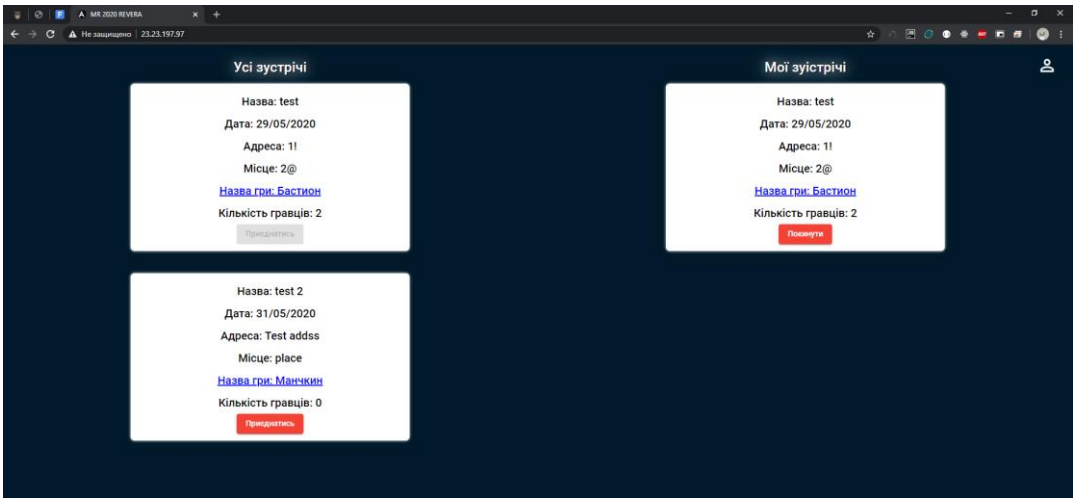

Рис. 2.4 — «Головна сторінка»

В переліку зустрічі ми бачимо, назву, дату проведення зустрічі, адресу, заклад, посилання на сайт де можна придбати гру, та кількість учасників.

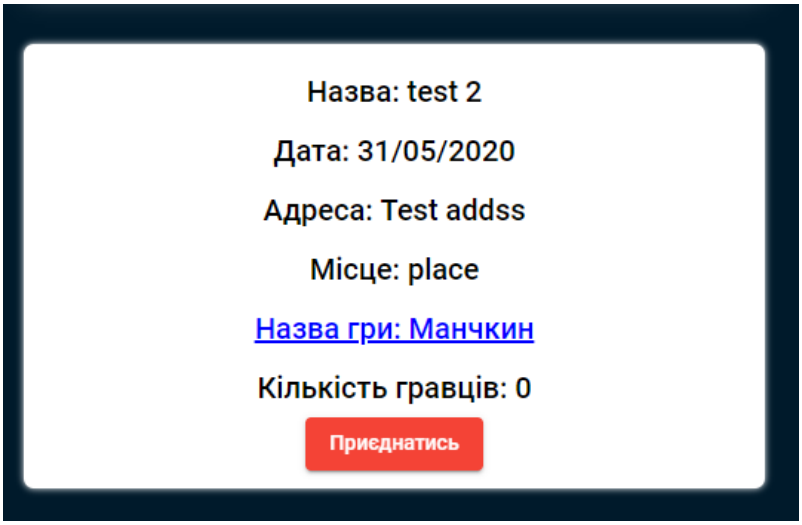

Рис.2.5 — «Детальний огляд»

Також, будь який користувач має змогу створити свою власну зустріч, натиснувши на ярлик свого профілю і натиснувши кнопку «Створити зустріч» [Рис .6]

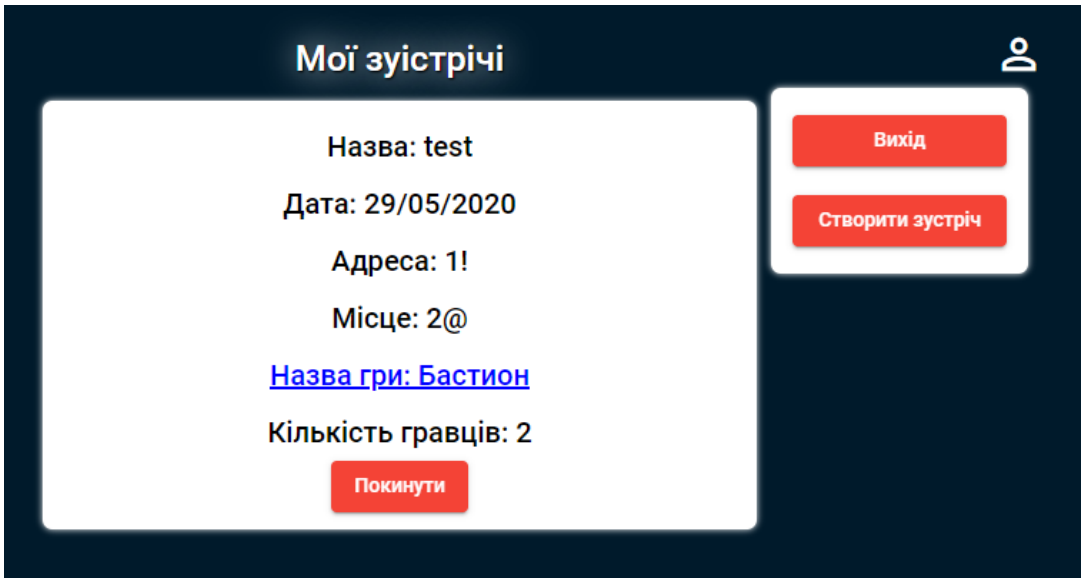

Рис. 2.6 — «Меню користувача»

Після натискання на кнопку «Створити зустріч», відкриється форма створення зустрічі.

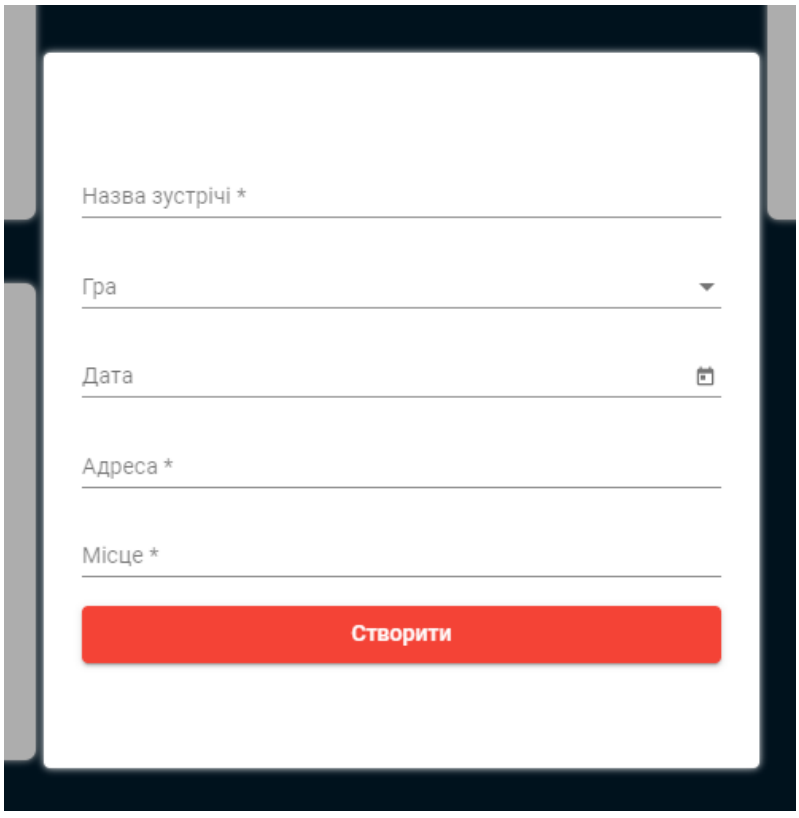

Рис. 2.7 — «Вікно створення нової зустрічі»

Заповнивши усі відповідні поля, та натиснути кнопку «Створити», зустріч потрапить в кінець списку «Усі зустрічі».

Рис. 2.8 — «Результат створення зустрічі»

3. ЕТАПИ РОЗРОБКИ

3.1 Front-end реалізація.

Для початку потрібно було встановити останні nodejs та angular cli ініціалізувати проект. Nodejs можна встановити скачавши .msi інсталятор з сайту, а angular project командами через cmd. По-перше, потрібно встановити angular-cli використовуючи npm (node pack manager), ввести в cmd команду “npm install -g @angular/cli”, далі створити новий проект командою “ng new [app-name]”. Після завершення потрібно зайти в папку і запустити проект командою “ng s”. Якщо все виконалось без проблем ми, побачимо наступне:

```
C:\Users\revera\Documents\MR\pj-to-show>ng s
Compiling @angular/core : es2015 as esm2015
Compiling @angular/animations : es2015 as esm2015
Compiling @angular/compiler/testing : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/animations/browser : es2015 as esm2015
Compiling @angular/core/testing : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/animations/browser/testing : es2015 as esm2015
Compiling @angular/common/testing : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @angular/platform-browser/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/platform-browser/animations : es2015 as esm2015
Compiling @angular/common/http/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015
Compiling @angular/router/testing : es2015 as esm2015

chunk {main} main.js, main.js.map (main) 60.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.7 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3 MB [initial] [rendered]
Date: 2020-05-28T19:19:28.083Z - Hash: 1209d42f9db1f7bcb6c1 - Time: 6969ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.
```

Рис. 3.1 — «Консоль створення нового проекту»

І відкривши в веб браузері http://localhost:4200 ми побачимо стартову сторінку нового angular проекту. Далі відразу встановимо Material UI для спрощення роботи зі стилями та елементами сторінки, можна зробити це командою “ng add @angular/material”.

І приступимо до створення стартової сторінки проекту, в є набір команд які дозволяють генерувати компоненти, сервіси і тд, автоматично. Створимо компонент header командою “ng g c header”, буде створено наступне:

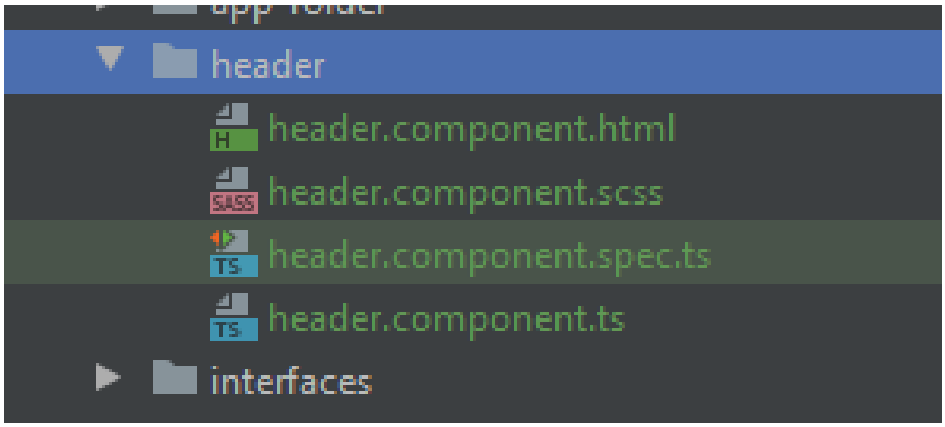

Рис. 3.2 — «Директорія компоненту»

Файл з розширенням .html [Додаток А] вміщає в собі розмітку компонента header мовою html, .scss [Додаток Б] таблицю стилів мовою css, а точніше його вдосконаленим родичом scss. Файл .spec.ts це файл для unit тестів, допомагає уникнути помилкового втручання в код в майбутній розробці і в файлі .ts [Додаток В] зберігається сама логіка компоненту.

Для того щоб створити дизайн сайту, треба мати гарне уявлення та смак тому стартова сторінка проекту виглядає так:

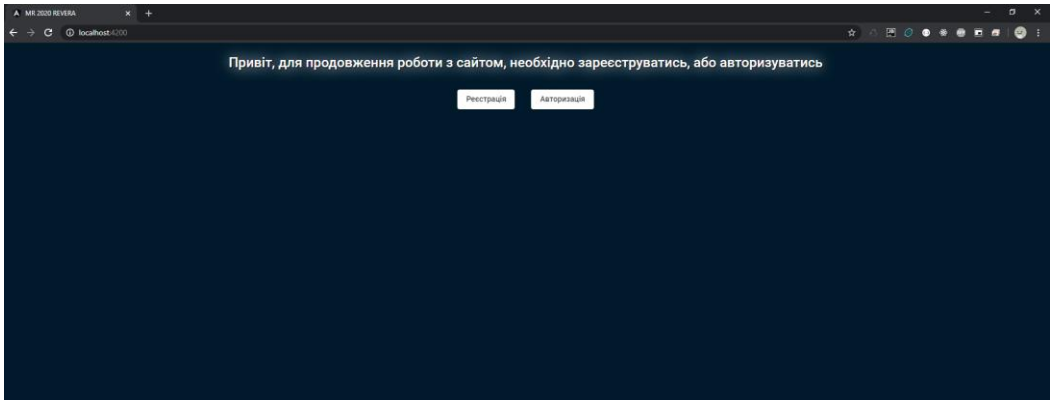

Рис. 3.3 — «Стартова сторінка»

Під час того як користувач авторизується проект записує в localStorage зміну з ім’ям користувача, відповідно натискає кнопку для виходу, змінна видаляється і коли тільки завантажується проект, він перевіряє, чи є ця змінна в localStorage, якщо є відразу відкриває головну сторінку, якщо – ні відкриває

стартову. Це реалізовано за допомогою auth сервісу, під час завантаження сторінки, реалізовано це так:

```
public getUserState(): boolean {
  return this.authService.loggedIg();
}
```

Рис. 3.4 — код з файлу header-component.ts

```
<div class="user-menu" *ngIf="getUserState()">
  <mat-icon aria-hidden="false" (click)="hide = !hide" aria-label="">perm_identity</mat-icon>
  <div *ngIf="!hide" class="menu-popup">
    <button mat-raised-button color="warn" (click)="onClick( data: 'logOut')">Вихід</button>
    <button mat-raised-button color="warn" (click)="createMeeting()">Створити зустріч</button>
  </div>
</div>
<span class="auth-false" *ngIf="!getUserState()">
  Привіт, для продовження роботи з сайтом, необхідно зареєструватись, або авторизуватись
</span>
<div class="start-buttons" *ngIf="!getUserState()">
  <button mat-raised-button (click)="onClick( data: 'reg')">Реєстрація</button>
  <button mat-raised-button (click)="onClick( data: 'auth')">Авторизація</button>
</div>
```

Рис. 3.5 — код с файлу header-component.html

```
loggedIg() {
  const state = localStorage.getItem( key: 'loginState');
  return state && state.length > 0 && state === 'true';
}
```

Рис. 3.6 — код с файлу auth.service.ts

Далі підемо трохи стисліше. При натисканні на копку «Реєстрація», відкриється модальне вікно реєстрацію, виглядає воно так:

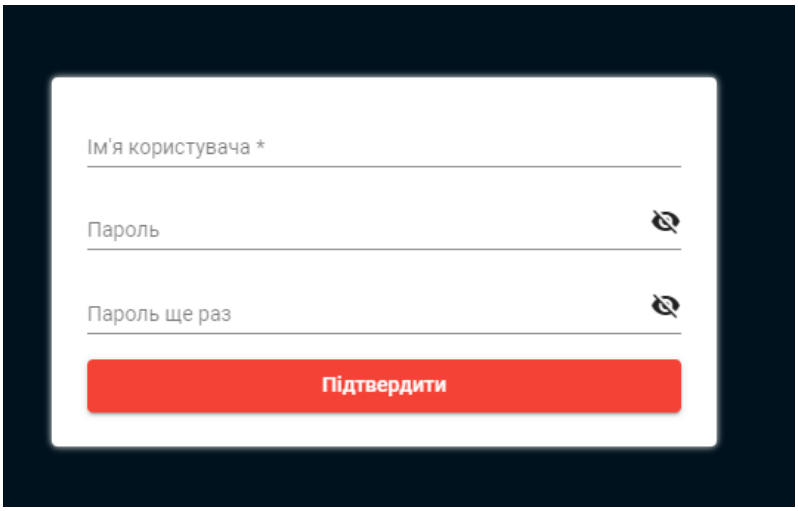

Рис. 3.7 — «Вікно реєстрації»

Вікно приймає в себе параметри:

— ім’я користувача;

— пароль;

— підтвердження паролю.

Так ж є перевірки на коректність заповнених даних та інформування користувача про це:

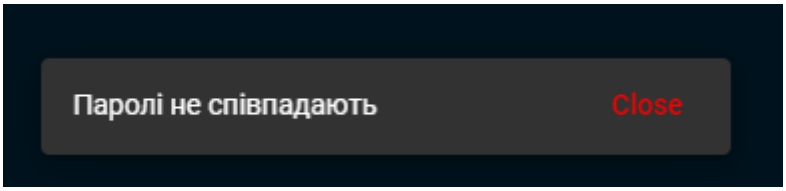

Рис. 3.8 — «Інформаційне повідомлення»

Після успішної реєстрації, та авторизації, ми потравляємо на головну сторінку в якій відображаються активні зустрічі, та власні зустрічі користувача. Користувач також має змогу створити нову зустріч підписатись на прийняття участі у зустрічі так відповідно відписатись від нього.

Із цікавого був також побудований сервіс для шифрування паролю [Додаток Г], коли створюється новий користувач, пароль відправляється на сервер, а звідти зберігається в базу зашифрованому вигляді, виглядає це так:

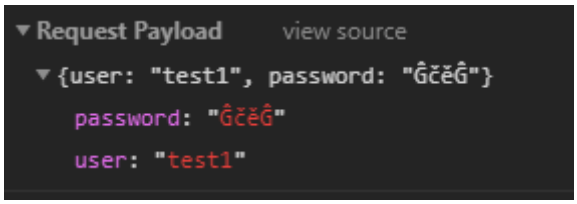

Рис. 3.9 — «Приклад шифрування»

При створенні нової зустрічі в вікні створення можна вибрати із падаючого списку, попереднього внесені в базу даних ігри, при відкритті модального вікна відбувається запит на сервер, який повертає список ігор, а також при створенні інтегрований сучасний календар на базі material UI (рис. 18)

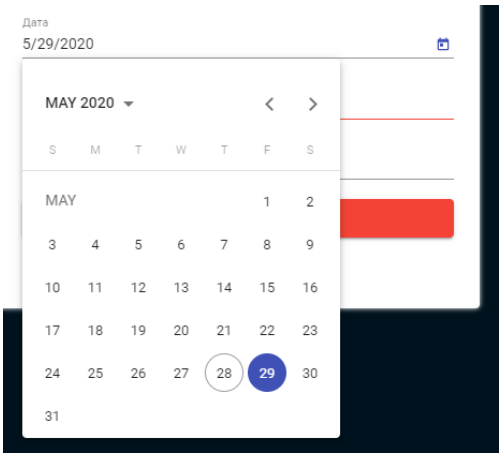

Рис. 3.10 — «Вибір дати»

3.2 Back-end реалізація.

В якості серверної частини був використаний nodeJS з фреймворком express. Для початку роботи нам потрібно створити директорію серверу та за допомогою терміналу виконати команду “npm init”. Далі необхідно

встановити сам фреймворк командою “npm i express” і створити головний файл проекту, в нашому випадку “index.js” та підключити в нього фреймворк, виглядає це так:

Рис. 3.11— «Головний файл сервісу»

Далі був створений конфіг в файл в якому будуть зберігатись креденшили підключення до бази даних, та збережені роути на коллекції.

Відповідно були створені методи для роботи з базою даних [Додаток Д],

в проекті були використані методи типу find, для отримання цілої колекції, а бо пошуку по ній за допомогою параметрів переданих в це метод.

Метод insert для занесення даних в базу, і метод update для їх оновлення.

Також методи для роботи з клієнтською частиною [Додаток Е], реалізовані за допомогою express і трансляцією подій, кожен з методів “слухає”, коли до нього звернеться клієнт, і як тільки це відбулось, роби запит до бази даних і повертає дані, або в випадки помилки – її опис.

3.3 Створення MongoDB.

Для початку необхідно було створити новий обліковий запис на сайті mongodb.com, далі відповідно стартовій інструкції, внести адресу свого серверу в так званий «білий список», після чого спробувати зі сторони серверу підключатись до бази даних.

Після успішного підключення, до облікового запису, була створена сама база з назвою “MR”, а також колекції для зберігання так званих документів, “users”, “meetings”, “games”

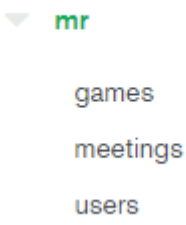

Рис. 3.12 — «База та колекції»

MongoDB дає нам змогу зберігати дані у будь-якому зручному для нас форматі, документи з зустрічами виглядає ось так:

Рис. 3.13 — «Приклад документу»

Ключ з _id це підхід створення об’єктів задекларований самою функціональністю MongoDB, для того, щоб кожен документ мав особистий ключ, далі поля, name, date, address, place, game зберігаються типом string, поле visitors як масив і поле id як number. Це доводить чудову гнучкість для збереження даних в системі MongoDB.

3.4 Розміщення клієнт/серверних частин на віддалених машинах.

Для того, щоб проект був глобально доступний, необхідно розмістити його віртуальних машинах, для цього було обрано AWS базуючись на

стабільності роботи даного ресурсу, і безкоштовному створенні віртуальних ubuntu машин об’ємом до 30gb. В даному є один мінус, це використання доменного імені, яке на AWS мінімально коштує 12$.

Перше, що потрібно зробити, це створити обліковий запис AWS, далі перейти на вкладку ec2 і створити новий інстанс, після проходження по інструкції і успішного завершення, ми маємо змогу підключитись до віртуальної машини, та розпочати її налаштування.

Для уникнення, проблем з коннекшином, AWS дає змогу задати для інстансів ElasticIP, це означає, що які б зміни не виковзались в мережевій системі віртуальній машині, доступитись до є, ви завжди матимете змогу, по одній конкретній IP адресі.

Першим етапом було, залиття клієнтської частини на віртуальну машину, для цього потрібно на віддаленій машині встановити apache2 – це лінукс утиліта бля рендеру html сторінок, виконується командою apt і apache2.

Підключення до віртуальних маниш, виконується за допомогою програми putty, а також має тип шифрування ssh-key, який гегепнується при створені віртуальних машини.

Для того, щоб залити на віртуальну машину файли клієнтської частини було використано програму FileZilla — це програма для FTP обміну файлами.

Попередньо збілджені файли клієнтської частини за допомогою angular команди “ng build” були перенесені на віртуальну машину в папку home/var/www – це парка яку використовує apache2, для рендеру веб-сторінок. І тепер сайт доступний за лінкою http://23.23.197.97/.

Схожа процедура виконалась для серверної частини, файли були завантажені на іншу, попередньо створену віртуальну машину. Далі на машині потрібно було встановити nodejs, після чого в папці проекту було виконано

команду npm i, для того аби встановити всі відповідні проекту бібліотеки та залежності, я прописані в файлі проекту package.json.

Також додатково була встановлена бібліотека pm2, яка дозволяє підтримувати сервер в живому стані, навіть після закриття терміналу, тому що, звичайна node джоба, має життєвий цикл, лише поки вікритий термінал.

Тепер проект повністю розгорнутий і доступний віддалено.

4. ТЕХНОЛОГІЧНИЙ АУДИТ РОЗРОБКИ САЙТУ ДЛЯ ОРГАНІЗАЦІЇ ЗУСТРІЧЕЙ КЛУБУ НАСТІЛЬНИХ ІГОР

4.1 Оцінювання комерційного потенціалу розробки проекту по організації зустрічей

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності по створенню сайту для організації зустрічей учасників клубу настільних ігор.

Аудит здійснюється двома, або трьома експертами, в нашому випадку – двома.

Оцінювання комерційного потенціалу проекту було здійснено за дванадцятою критеріями, наведеними в таблиці 4.1.

Таблиця 4.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка.

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
1	2	3	4	5	6
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці

1	2	3	4	5	6
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці

1	2	3	4	5	6
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки заносимо в табл. 4.2.

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки.

Критерії	Експерти	
	1	2
1	4	4
2	2	2
3	4	3
4	4	4
5	3	4
6	3	4

Продовження таблиці

7	3	3
8	4	3
9	4	4
10	4	4
11	4	4
12	4	4
Сума балів	43	43
Середньоарифметична сума балів *СБ*	*СБ* = 43	

Оцінювання виявило, що рівень розробки є високим, так як середня кількість балів складає 43.

Реалізація не потребує стороннього втручання, окрім фахівців в даній галузі. Крім того база проекту дозволяє гнучко її замінити її галузь, якщо попередня втратила актуальність, Зміна потребує долучення лише одного фахівця, за затрати часу не виходять за поріг двох тижнів.

4.2 Прогнозування витрат на виконання науково-дослідної роботи по розробці проекту по організації зустрічей

Прогнозування витрат проходить в три етапи:

Етап 1. розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи.

1) Основна заробітна плата розробників розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \text{ [грн]},$$

де М — місячний посадовий оклад розробника [грн];

$Т_р$ — число робочих днів в місяці, $Т_р = 20$;

t — число робочих днів роботи розробника.

Розрахунок витрат за формулою 4.1:

$$З_о = \frac{18000}{20} \cdot 15 = 13500 \text{ (грн)},$$

Зроблені розрахунки заносимо до таблиці 4.3.

Таблиця 4.3 — Витрати на заробітну плату розробників

Найменування посади виконавця	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на оплату праці, грн.
Програмний інженер	20000	1000	20	20000
Всього				$\sum З_о = 20000$

2) Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою.

$$З_р = \sum_{1}^{n} t_i \cdot С_i \text{ [грн]}.$$

де t_i — норма часу (трудомісткість) на виконання конкретної роботи, годин;

n — кількість робіт по видах та розрядах;

$С_і$ — погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу.

$$С_і = \frac{М_м \cdot К_і}{Т_р \cdot Т_{зм}} \left[\frac{грн}{год}\right],$$

де $М_м$ — розмір мінімальної заробітної плати за місяць, грн.;

з 01.01.20 р. 4723 грн.;

$К_і$ — тарифний коефіцієнт робітника відповідного розряду. Величина чинних тарифних коефіцієнтів робітників відповідних розрядів для бюджетної сфери наведена в таблиці 4.4:

Таблиця 4.4 — Значення тарифних коефіцієнтів робітників відповідних розрядів.

Розряд	1	2	3	4	5	6	7	8
$К_і$	1,00	1,09	1,18	1,27	1,36	1,45	1,54	1,64

$Т_р$ — число робочих днів у місяці;

$Т_{зм}$ — тривалість зміни, зазвичай $Т_{зм} = 8$ годин.

Таблиця 4.5 — Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н.-гд	Розряд роботи	Коефіцієнт	Погодинна тарифна ставка	Величина оплати, грн.
Налагоджувальні (організація робочої мережі)	16	7	1,54	45,45	727
Всього					$З_р = 727$

3) Додаткова заробітна плата ($З_д$) всіх розробників та робітників, які приймали участь в дослідженні.

Додаткову заробітну плату розраховуємо як 10...12% від суми основної заробітної плати всіх розробників та робітників. Приймаємо додаткову заробітну плату 10% від основної:

$$З_д = 0{,}10 \cdot (З_о + З_р).$$

$$З_д = \frac{(20000 + 727) \cdot 10}{100\%} = 2072{,}70 \text{ (грн)}.$$

4) Ставка єдиного соціального внеску на загальнообов'язкове державне соціальне страхування, %. $\beta = 22\%$.

5) Амортизація обладнання, комп'ютерів та приміщень А, які використовувались під час дослідження.

У спрощеному вигляді амортизаційні відрахування по кожному виду обладнання та приміщенням можуть бути розраховані за формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ [грн]},$$

де Ц — загальна балансова вартість даного виду обладнання (приміщень), грн.;

$Н_а$ — річна норма амортизаційних відрахувань. $Н_а = (10 \ldots 25)\%$. Т — термін використання обладнання (приміщень), місяці.

Проведені розрахунки амортизаційних відрахувань були занесені до таблиці 4.6.

Таблиця 4.6 — Розрахунок амортизаційних відрахувань

Найменування комплектуючих	Балансова вартість, грн.	Норма амортизації, %	Термін використання міс.	Величина амортизаційних відрахувань, грн.
Комп´ютер працівника	15000	20	1	249
Всього				$А = 249$

6) Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \ [грн],$$

де $Н_i$ — витрати матеріалу і-го найменування, шт.;

$Ц_i$ — вартість матеріалу і-го найменування, грн./шт.;

$К_i$ — коефіцієнт транспортних витрат, $К_i = (1{,}1 \dots 1{,}15)$. Обираємо $К_i = 1{,}15$;

$В_i$ — маса відходів матеріалу і-го найменування, шт.;

$Ц_в$ — ціна відходів матеріалу і-го найменування, грн/шт.;

n — кількість видів матеріалів.

Проведені розрахунки витрат на матеріали внесені до таблицю 4.7.

Таблиця 4.7 — Витрати на матеріали

№	Найменування матеріалу, марка, тип, сорт	Од. виміру	Витрачено на виріб	Ціна за матеріал, грн.	Загальна вартість
1.	Ліцензійні ключі до середовищ розробки	шт	2	1500	3000
2	Віртуальна машина для розташування середовища розробки	шт	2	500	1000
3	Доменне ім’я	шт	2	320	640
Всього, грн.					4640

7) Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \text{ [грн]},$$

де В – вартість 1 кВт-год. електроенергії, в 2020 році $В = 2{,}4$ грн./кВт;

П – установлена потужність обладнання, кВт. Ф – фактична кількість годин роботи обладнання, годин. $К_п$ – коефіцієнт використання потужності, $К_п < 1$ [37]. Обираємо $К_п = 0{,}9$.

Отже, витрати на силову електроенергію становлять:

$$В_е = 2{,}4 \cdot 0{,}2 \cdot 140 \cdot 0{,}9 = 60{,}5 \text{ (грн)}.$$

8) Інші витрати Він охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати Ів можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (З_о + З_р) \cdot (100\% \dots 300\%) \text{ [грн]},$$

$$В_{ін} = \frac{(20000 + 727) \cdot 100\%}{100\%} = 20727 \text{ (грн)}.$$

9) Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В.

В = 20000 + 727 + 2072,70 + 249 + 4640 + 60,50 = 27749,20

Етап 2. Розрахунок загальних витрат на виконання даної роботи. Розрахунок загальних витрат здійснюється у тому випадку, коли дипломник виконує тільки певну частину даної роботи. У подальшому ця наукова робота буде продовжена.

Тоді загальна вартість всієї наукової роботи визначається за $В_{заг}$ визначається за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \text{ [грн]},$$

Де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях.

$$В_{заг} = \frac{27749,20}{0,8} = 34686,50 \text{ (грн)}.$$

Етап 3. прогнозування загальних витрат на виконання та впровадження результатів виконаної наукової роботи. Прогнозування загальних витрат ЗВ на

виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \text{ [грн]},$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Так, якщо розробка знаходиться:

$$ЗВ = \frac{44428,91}{0,9} = 38540,55 \text{ (грн)}.$$

4.3 Прогнозування комерційних ефектів від реалізації результатів дослідження

Кількісне прогнозування, вигоди, зиску, які можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

Для продовження необхідно:

а) вказати, скільки часу займе виконання даної наукової роботи та впровадження її результатів.

Візьмемо до уваги, що виконання наукової роботи та впровадження її результатів займе 2-3 місяці.

б) зазначити, коли саме (після впровадження виконаної наукової роботи) очікуються основні позитивні результати від впровадження дослідження.

Приймаємо, що основні позитивні результати від впровадження дослідження очікуються після 6 місяців з моменту впровадження.

в) назвати ці позитивні результати та кількісно їх оцінити по роках.

В умовах ринку узагальнюючим позитивним результатом, що його отримує товариство від впровадження результатів дослідження, є збільшення бюджету товариства, а також збільшення кількості учасників.

Для оцінки та прогнозування зростання чистого бюджету товариства можливі два основні випадки. Скористаємося **1-м випадком,** коли можна прямо оцінити зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_{1}^{n} (\Delta П_я \cdot N + П_я \Delta N)_i$$

де $\Delta П_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ — основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо що в результаті впровадження наукових досліджень за допомогою реклами кількість учасників товариства зросла, що дозволяє

проводити більшу кількість зустрічей за рік з 10 до 15, за другий рік з 15 до 30, за третій рік, з 30 до 40, а плату за організацію підвищити на 200 грн.

Орієнтовно: поповнення бюджету товариства до впровадження проекту складало близько 15000 грн, кількість зустрічей на рік на 10.

Тоді збільшення чистого прибутку товариства $\Delta П_i$ протягом першого року складе:

$$\Delta П_1 = 200 * 10 \ + (1500 + 200) * 5 = 10500$$

Збільшення чистого прибутку товариства $\Delta П_i$ протягом другого року (відносно базового року, тобто року до впровадження результатів наукового дослідження) складе:

$$\Delta П_2 = 200 * 10 \ + (1500 + 200) * (5 + 15) = 36000$$

Збільшення чистого прибутку товариства $\Delta П_i$ протягом третього року (відносно базового року, тобто року до впровадження результатів наукового дослідження) складе:

$$\Delta П_2 = 200 * 10 \ + (1500 + 200) * (5 + 15 \ + \ 10) = 53000$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Під час розрахунку вкладених інвестицій було опрацьовано наступні кроки:

Крок 1. Розраховують теперішню вартість інвестицій PV, що вкладають в наукове дослідження. Такою вартістю можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, розраховану раніше за формулою 4.12, тобто будемо вважати, що $ЗВ = PV = 38540{,}55$ (грн).

Крок 2. Розраховують очікуване збільшення прибутку $\Delta П_i$, що його отримає товариство від впровадження результатів наукового дослідження, для кожного із років, починаючи з першого року впровадження. Таке збільшення прибутку також було розраховане раніше за формулою. За даною формулою: $\Delta П_1 = 10500(грн),\ \Delta П_2 = 36000(грн), \Delta П_3 = 53000(грн).$

Рисунок 4.1, характеризує рух платежів (інвестицій та додаткових прибутків) має такий вигляд.

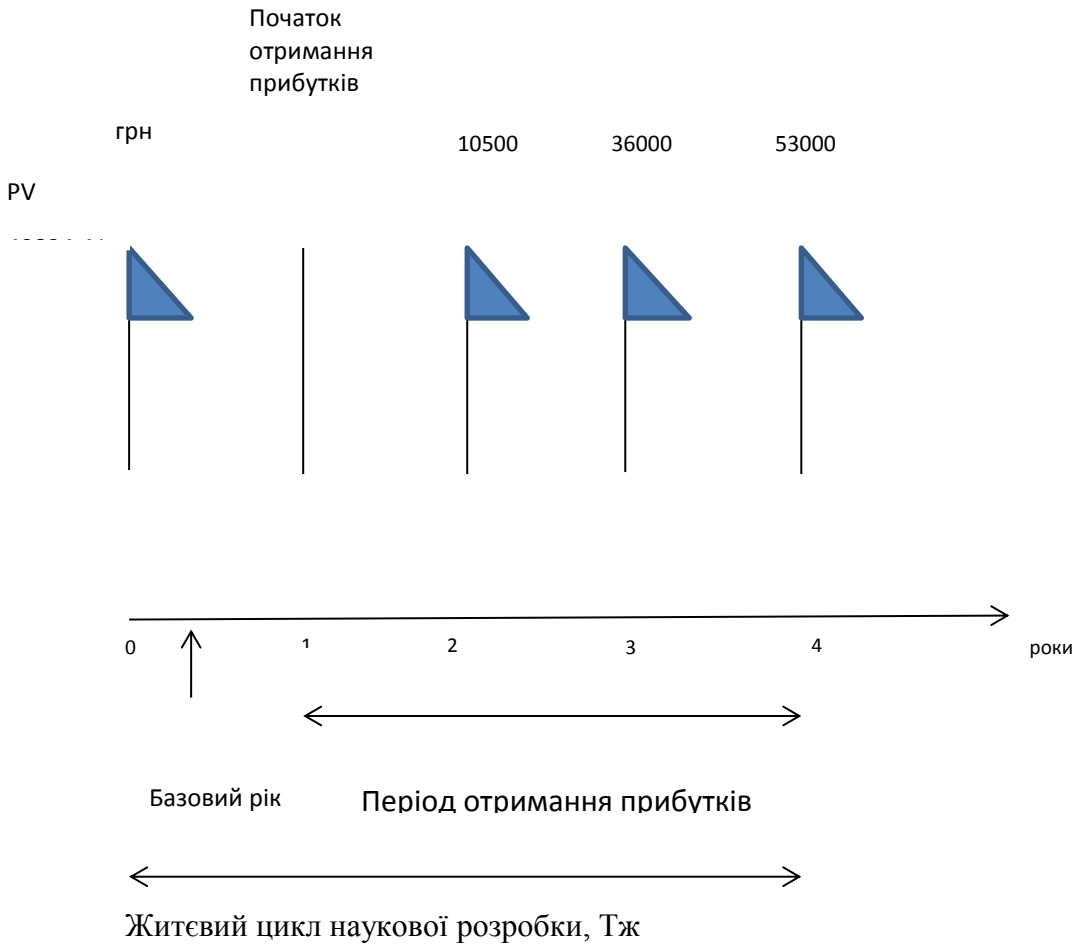

Рисунок 4.1 — Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР.

Крок 3. Для спрощення подальших розрахунків будують вісь часу, на яку наносять всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

Крок 4. Розраховують абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$Е_{абс} = (ПП - PV)\ [грн],$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає товариство (організація) від реалізації результатів наукового дослідження, грн.; PV — теперішня вартість інвестицій $PV = ЗВ$, грн.

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{Т} \frac{\Delta П_i}{(1+\tau)^t}\ [грн],$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн.;

Т — період часу, протягом якого виявляються результати впровадження НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівня 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки «0».

$$ПП = \frac{10500}{(1+0{,}1)^2} + \frac{36000}{(1+0{,}1)^3} + \frac{53000}{(1+0{,}1)^4} = 71924{,}6\ (грн).$$

Тоді:

$$\text{Е}_{\text{абс}} = (71924{,}6 - 38540{,}55) = 33384{,}04 \text{ (грн)}.$$

Оскільки $\text{Е}_{\text{абс}} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток.

Крок 5. Розраховують відносну (щорічну) ефективність вкладених в наукове дослідження інвестицій $\text{Е}_{\text{в}}$ за формулою:

$$\text{Е}_{\text{в}} = \sqrt[\text{Т}_{\text{ж}}]{1 + \frac{\text{Е}_{\text{абс}}}{PV}} - 1,$$

де $\text{Е}_{\text{абс}}$ — абсолютна ефективність вкладених інвестицій, грн.;

PV — теперішня вартість інвестицій $PV = \text{ЗВ}$, грн.;

$\text{Т}_{\text{ж}}$ — життєвий цикл наукової розробки, роки.

$$\text{Е}_{\text{в}} = \sqrt[4]{1 + \frac{33384{,}02}{38540{,}55}} - 1 = 1{,}17$$

Відносна (щорічна) ефективність вкладених інвестицій становить 117%.

Далі ефективність вкладених інвестицій потрібно порівняти з мінімальною (бар'єрною) ставкою дисконтування $\tau_{\text{мін}}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{\text{мін}}$ визначається за формулою:

$$\tau_{\text{мін}} = d + f,$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; $d = (0{,}14 \dots 0{,}2)$;

f — показник, що характеризує ризикованість вкладень, зазвичай величина $f = (0{,}05 \dots 0{,}1)$.

$$\tau_{\text{мін}} = 0{,}2 + 0{,}1 = 0{,}3 \text{ або } 30\%.$$

З розрахунків випливає $E_в = 117\% > \tau_{\text{мін}} = 30\%$, тому потенційний інвестор буде зацікавлений у фінансуванні даного наукового дослідження, а не у вкладенні коштів на депозит у комерційному банку.

Крок 6. Розраховують термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ за формулою:

$$T_{ок} = \frac{1}{E_в} \text{ [років]},$$

$$T_{ок} = \frac{1}{1{,}17} = 0{,}8 \text{ (року)}.$$

Вразовуючи термін окупності вкладених у реалізацію наукового проекту інвестицій знаходиться в допустимих межах $T_{ок} = 0{,}8 \text{ року} < 3 \ldots 5 -$ ти років, то фінансування даної наукової розробки є доцільним.

4.5 Заключення до технологічного аудиту

Опрацювавши підрахунки, по даному проекту, можна зробити висновок, що є доцільним впровадження розробки.

Спочатку було розглянуто витрати на розробку сайту, включаючи затрати на продукцію, для комфортного володіння проектом. Сума витрат склала 38540,55 грн.

Наступне, що було розглянуто, це прогнозування комерційних ефектів, від реалізації розробки, тобто прибуток підприємства після інтеграції.

В останньому розділі опрацьовані показники, які доводять, що проект вартий уваги та фінансування, адже його впровадження, забезпечую, значний примноження коштів, за досить кроткий проміжок часу (0,8 року).

ВИСНОВКИ

В даній магістерській роботі було розроблено веб-ресурс, для організації зустрічей учасників клубу настільних ігор, який допомагає швидко та зручно створити, або проглянути інформацію про уже існуючі зустрічі та підписайсь на прийняття участі в них, маючи лише пристрій з підключенням до глобальної мережі інтернет та додатком для відображення веб сторінок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Байков В. Интернет. Поиск информации и продвижение сайтов; Книга по Требованию - Москва, 2012. - 288 с.

2. Венедюхин Александр , Воробьев Андрей Создание сайтов (+ CD-ROM); Эксмо - Москва, 2011. - 528 с.

3. Гарднер Л., Григсби Д. Разработка веб-сайтов для мобильных устройств; Питер - Москва, 2013. - 448 с.

4. Дакетт Джон HTML и CSS. Разработка и дизайн веб-сайтов (+ CDROM); Эксмо - Москва, 2013. - 480 с.

5. Дронов Владимир HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов; БХВ-Петербург - Москва, 2011. - 416 с.

6. Дронов, Владимир Macromedia Dreamweaver 4: разработка Web-сайтов; М.: БХВ - Москва, 2014. - 608 с.

7. Костин С. П. Самоучитель создания Web-сайтов; Триумф - Москва, 2009. - 176 с.

8. Кристофер Б. Джонс 140 технологий раскрутки сайтов; Рид Групп - Москва, 2011. - 352 с.

9. Лавдей Ланс , Нихаус Сандра Проектирование прибыльных веб-сайтов; Манн, Иванов и Фербер - Москва, 2011. - 256 с.

10. Митчелл, Скотт 5 проектов Web-сайтов от фотоальбома до магазина; М.: НТ Пресс - Москва, 2013. - 224 с.

11. Тероу Шэри Видимость в Интернете. Поисковая оптимизация сайтов; Символ-Плюс - , 2009. - 288 с.

12. Фрейен Бен HTML5 и CSS3.Разработка сайтов для любых браузеров и устройств; Питер - Москва, 2014. - 304 с. 21. Фридман В. А., Александров А. В., Сергеев Г. Г., Костин С. П. Строительство Web-сайтов (+ CD-ROM); Триумф - Москва, 2011. - 288 с.

13. Хуторской А. В., Орешко А. П. Технология создания сайтов. 10-11 классы; Дрофа - Москва, 2011. - 256 с.

14. Чебыкин Ростислав Разработка и оформление текстового содержания сайтов; БХВ-Петербург - Москва, 2014. - 528 с.

15. Энж Эрик , Спенсер Стефан , Фишкин Рэнд , Стрикчиола Джесси SEO икусство раскрутки сайтов; БХВ-Петербург - Москва, 2011. - 592 с.