НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: Архітектура, технології та інструменти розробки сучасних односторінкових (SPA) ВЕБ-додатків _____
_____

Виконав (-ла): студент (-ка) _IV_ курсу, групи _ДА-21__
(шифр групи)

Воронін Ігор Борисович_____
(прізвище, ім'я, по батькові) (підпис)

Керівник Романов Валерій Володимирович_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Британов О.А._____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування 7.050103, 8.050103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Вороніну Ігорю Борисовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Архітектура, технології та інструменти розробки сучасних односторінкових (SPA) ВЕБ-додатків
керівник проекту (роботи) Романов Валерій Володимирович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи) _____

1. Дослідити наступні елементи архітектури: клієнтська маршрутизація, модульність, шаблони, зв'язування.
2. Порівняння наступних фреймворків Backbone, Ember, Knockout, Angular 2.
3. Приклад односторінкового веб-додатку створеного за допомогою одного з порівняних фреймворків.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Дослідження концепції односторінкового веб-додатку
2. Дослідження типових особливостей архітектури

3. Огляд та порівняння інструментів, фреймворків та технологій
4. Дослідження особливостей розробки веб-додатку з використання Angular 2
5. Функціонально-вартісний аналіз програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Порівняння веб-сайтів, односторінкових веб-додатків, нативних додатків – плакат.
2. Життєвий цикл традиційного веб-сайту й односторінкового додатку – плакат.
3. Архітектура, ієрархія та зв'язок компонентів Angular 2 додатку – плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ			

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Дослідження концепції односторінкового веб-додатку	22.02.2016	
4	Дослідження типових особливостей архітектури	07.03.2016	
5	Огляд та порівняння інструментів, фреймворків та технологій	26.03.2016	
6	Дослідження особливостей розробки веб-додатку з використання Angular 2	10.04.2016	
7	Функціонально-вартісний аналіз програмного продукту	16.04.2016	
8	Підготовка графічного матеріалу	25.04.2016	
9	Оформлення дипломної роботи	31.05.2016	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ І.Б.Воронін
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ В.В.Романов
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

бакалаврської дипломної роботи Вороніна Ігоря Борисовича на тему: «Архітектура, технології та інструменти розробки сучасних односторінкових (SPA) ВЕБ-додатків»

Дипломна робота присвячена дослідженню концепції односторінкових веб-додатків, елементів типової архітектури, популярних шаблонів проектування, інструментів та фреймворків призначених для розробки таких веб-додатків. Також під час роботи були розглянуті існуючі односторінкові веб-додатки, детально досліджено один із фреймворків та набуто практичних навичок у створенні односторінкового додатку з його використанням.

Загальний обсяг роботи 85 сторінок, 33 рисунка, 8 таблиць, 19 бібліографічних найменувань, 1 додаток обсягом 14 сторінок.

Ключові слова: односторінковий веб-додаток, розробка веб-сайту, JavaScript фреймворки, шаблони проектування, архітектура.

АННОТАЦИЯ

бакалаврской дипломной работе Воронина Игоря Борисовича на тему: «Архитектура, технологии и инструменты разработки современных одностраничных (SPA) Веб-приложений»

Дипломная работа посвящена исследованию концепции одностраничных веб-приложений, элементов типичной архитектуры, популярных шаблонов проектирования, инструментов и фреймворков предназначенных для разработки таких веб-приложений. Также вовремя роботы были рассмотрены существующие одностраничные веб-приложения, детально исследован один из фреймворков и приобретены практические навыки в создании одностраничного веб-приложения с его использованием.

Общий объем работы 85 страниц, 33 рисунка, 8 таблиц, 19 библиографических наименований, 1 дополнение объёмом 14 страниц.

Ключевые слова: одностраничное веб-приложение, разработка веб-сайта, JavaScript фреймворки, шаблоны проектирования, архитектура.

ANNOTATION

Of a bachelor's degree work of Voronin Igor Borisovich on "Architecture, technology and instruments of modern Single Page WEB applications development»

Degree work is dedicated to the exploration of single-page application conception, elements of typical architecture, popular design patterns, tools and frameworks dedicated for such application developing. Also during this work existing single-page web applications were reviewed. Studied one of frameworks and acquired practical skills in creating single-page applications using it.

The total amount of work 85 pages, 33 figures, 8 tables, 19 bibliographical references, 1 application of 14 pages.

Tags: single-page web application, web-site developing, JavaScript frameworks, design patterns, architecture.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML – HyperText Markup Language

ПЗ – Програмне забезпечення

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

HTTP – HyperText Transfer Protocol

API – Application programming interface

Web – система доступу до пов'язаних між собою документів на різних комп'ютерах, підключених до Інтернету

HTML – HyperText Markup Language — Мова розмітки гіпертекстових документів

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

XML – eXtensible Markup Language (розширювана мова розмітки)

БД – База даних

SPA – (Single Page Application) Односторінковий додаток

MVC – Model-view-Controller

MVVM – Model-View-ViewModel

MVP – Model-View-Presenter

DOM – (Document Object Model – «об’єктна модель документа») прикладний програмний інтерфейс для роботи зі структурованими документами.

ВСТУП

Сьогодні веб технології швидко розвиваються. Платформонезалежність та доступність веб сайтів роблять їх дуже привабливими для розробників. Розвиток веб технологій сьогодні дозволяє будувати не лише веб сайти у звичайному сенсі цього слова, коли сайт складається зі сторінок, а й дуже інтерактивні сайти – веб-сервіси які можна називати повноцінними додатками. Такі веб-сервіси вже становлять значну конкуренцію нативним додаткам, що змушує розробників створювати онлайн версії популярних додатків.

Односторінковий веб-додаток (SPA) – це концепція веб-сайту, що робить його дуже інтерактивним й дуже схожим на звичайний додаток, зберігаючи при цьому всі переваги веб-сайту. Ідея SPA не нова, вона почала з'являтися як тільки з'явилися JavaScript та AJAX й веб-сайти почали набувати динаміки будучі вже не лише статичними сторінками. Мабуть кожному розробнику сайтів рано чи пізно приходили в голову ідеї про сайт-додаток, який завантажується лише один раз, а все інше відбувається за рахунок скриптів та асинхронних запитів. Проте об'єм роботи, який треба було зробити не був вартий цього, адже для створення веб-сайту за концепцією SPA треба прикласти значних зусиль. Полегшити розробку такого виду сайтів допоможе використання безліч технологій, фреймворків й інструментів, що були для цього розроблені в останній час. Використовуючи ці технології можна будувати чудові веб-додатки витрачаючи на це час порівняний з часом, необхідним на розробку традиційного нативного додатку.

Метою даної роботи є збір теоретичних відомостей про концепцію односторінкового веб-додатку та типові особливості їх архітектури, огляд існуючих популярних та відомих SPA додатків, технологій, що були використані при їх розробці, огляд інструментів та фреймворків, що використовуються при їх розробці, набуття практичних навичок у створенні односторінкового веб-додатку з використанням одного з фреймворків.

1. ДОСЛІДЖЕННЯ КОНЦЕПЦІЇ ОДНОСТОРІНКОВОГО ВЕБ-ДОДАТКУ

1.1 Огляд концепції

Розглянемо звичайний, традиційний сайт, наприклад Wikipedia. Відкриваючи статтю користувач чекає поки браузер зв'яжиться з сервером й останній обробить його запит. Потім браузер підвантажує різні ресурси необхідні для відображення сторінки, паралельно відтворюючі ті, що вже завантажились. Під час цього процесу користувач бачить як елементи сторінки скачуть й мерехтять, перераховуючи місце необхідне під зображення та блоки, що підвантажились пізніше. В залежності від пропускної здатності мережі та завантаженості сервера цей процес може тривати доволі довго, дратуючи користувача. Коли користувач захоче переглянути іншу статтю, відредагувати цю, або зробити ще якісь дії на цьому сайті, його чекає перезавантаження сторінки за новим URL. Частина ресурсів буде кешована браузером й завантаження нової сторінки має пройти швидше, але всі недоліки даного процесу прибрати все одно не вдасться.

Так працює більшість сайтів в інтернеті. Але з моменту створення Web пройшло багато років й багато змін, з'явилось багато технологій, JavaScript, AJAX та інші, сайти почали набувати інтерактивності й бути чимось більшим ніж просто сторінками. Зараз розробники мають достатньо технологій, щоб змінити поведінку традиційного сайту й зробити його схожим на нативний додаток операційної системи. Для цього всі зміни на сторінці веб-сайту роблять за допомогою скриптів, а запити до сервера використовують AJAX. Відкриваючи такий сайт користувачу доводиться лише раз завантажити його сторінку, а зміни на сторінці, які не потребують нових даних, виконуються швидше, оскілки не чекають обробки запитів на сервері. Така концепція веб-сайту отримала назву односторінковий додаток (Single Page Application). Все більше й більше нових сайтів створюють саме такими.

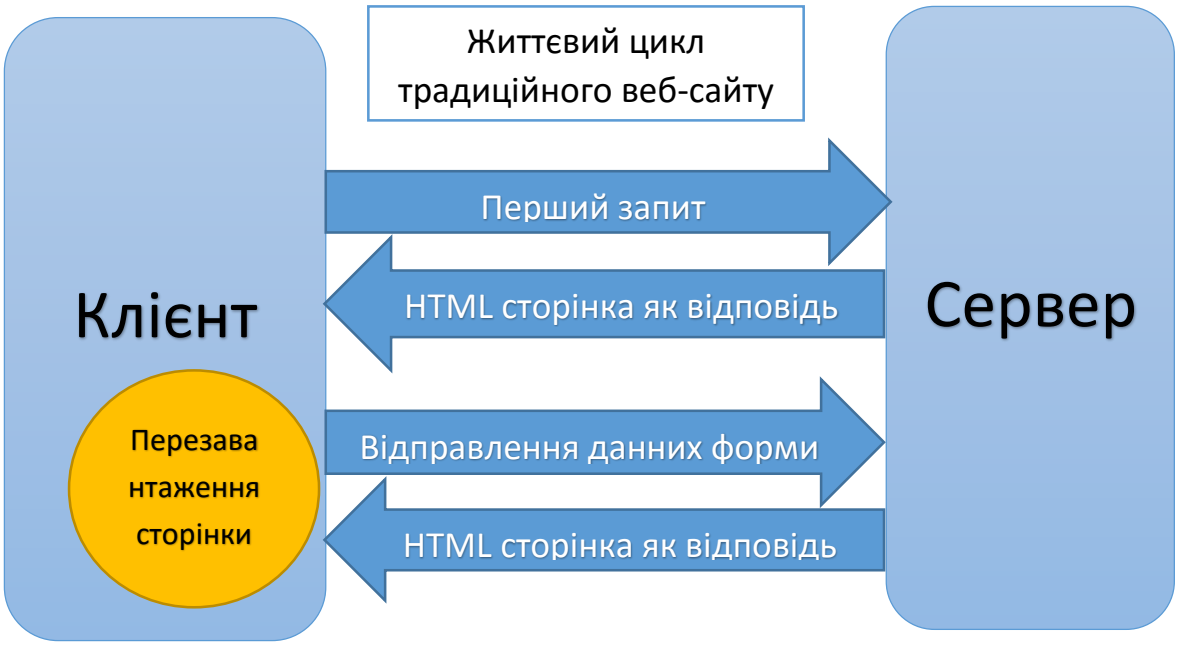

Рисунок 1- Життєвий цикл традиційного веб-сайту

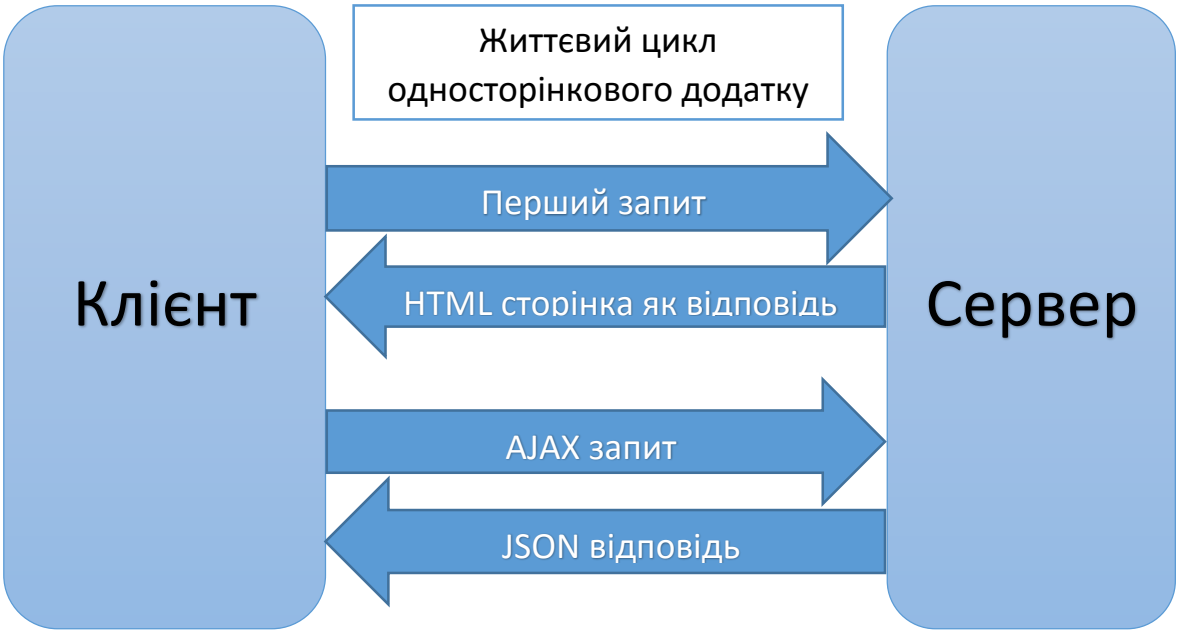

Рисунок 2- Життєвий цикл односторінкового додатку

Отже, незважаючи на назву, суть концепції односторінкового додатку полягає не в обмеженні сторінок веб-сайту, а в наданні сайту відчуття нативного додатка, шляхом позбавлення перезавантажень й перенесенні відтворюючої

логіки з сервера до клієнта, тим самим зменшуючи залежність роботи сайту від сервера та інтернет з'єднання.

1.2 Порівняння з традиційним веб-сайтом

Концепція односторінкового додатку виглядає дуже привабливо, проте вона теж має певні недоліки та переваги.

Переваги односторінкових додатків:

1. Насичений функціоналом інтерфейс
2. Швидка реакція інтерфейсу, завдяки відсутній необхідності звертатися до серверу при кожній дії.
3. Значне зменшення навантаження на сервер.
4. Значне спрощення логіки та складності серверу.
5. Схожість додатків з нативними додатками операційної системи.

Недоліки:

1. Підвищення навантаження на клієнт завдяки великої кількості JavaScript.
2. Складність розробки.

1.3 Порівняння з нативним додатком

Нативні додатки це те, на що намагається бути схожим односторінковий веб-додаток, тобто те до чого він прагне. Справедливо буде поставити питання «Чому не створити справжній нативний додаток?». Справа в тому, що нативні додатки також мають свої недоліки. Односторінковий веб-додаток намагається перейняти найкраще від обох, веб-сайту й нативного додатку. Однак, в певних випадках, використання звичайного нативного додатку все ще може бути вигіднішим, або й навіть необхідним. Порівняємо їх недоліки й переваги.

Переваги односторінкових веб-додатків:

1. Багатоплатформність.
2. Відсутність необхідності встановлення.

3. Не займає місце на жорсткому диску.
4. Можливість віддалених розрахунків.

Переваги нативних додатків:

1. Вільність у вибору мов програмування та інтерфейсів.
2. Продуктивність нативного коду.
3. Використання нативних можливостей платформи.
4. Доступ до будь якого обладнання комп’ютера.
5. Незалежність від інтернет з’єднання.

1.4 Підсумкова порівняльна таблиця

Таблиця 1 – Порівняння традиційного веб-сайту, односторінкового веб-додатку та нативного додатку.

	Веб-сайт	Односторінковий веб-додаток	Нативний додаток
Чуйність інтерфейсу	- маленька	+ велика	+ велика
Встановлення	+ не потрібно	+ не потрібно	- потрібно
Місце на ПЗУ	+ не потрібно	+ не потрібно	- потрібно
Інтернет з’єднання	-- дуже потрібно	- потрібно	+ не потрібно
Наявність серверу	-- дуже потрібно	- потрібно	+ не потрібно
Навантаження на клієнт	+ середнє	- велике	+ середнє
Доступ до обладнання клієнта та можливостей платформи	- не має	- не має	+ є
Багатоплатформність	+ чудова	+ чудова	- погана
Гнучкість у вибору мови й інструментів розробки	- погана	- погана	+ гарна

1.5 Приклади існуючих додатків

1.5.1 Онлайн сервіси Google

Google за час свого існування створив багато онлайн сервісів (рис. 3), більшість з яких цілком або майже цілком зроблені як односторінкові.

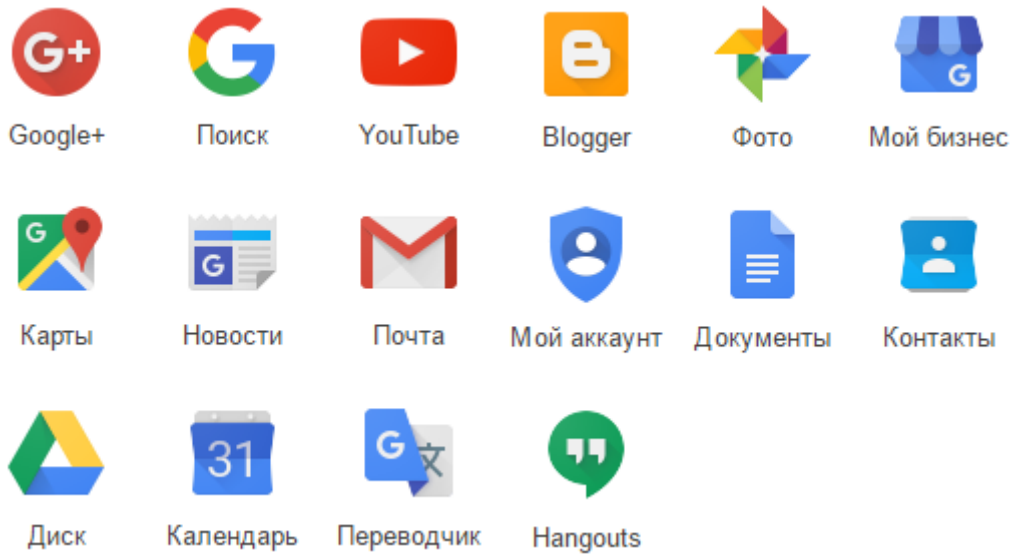

Рисунок 3 – Деякі онлайн сервіси Google

Google Mail або Gmail – веб інтерфейс до пошти від Google. Він був одним із найперших додатків виконаних за SPA концепцією. Додаток представляє із себе список електронних листів з каталогами та пошуком, також є можливість редагувати та переглядати листи у невеличкому вікні. Оскільки на момент створення Gmail не існувало майже ніяких відкритих фреймворків для SPA, додаток був написаний на чистому HTML та JavaScript без використання додаткових бібліотек, хіба що власних від Google. За час існування він оновлювався і зараз виглядає доволі сучасно.

Google вже запустила наступника під назвою Inbox, який теж э односторінковим додатком. Однак Gmail все ще користується популярністю. Довгий час успішної роботи доводить що використання концепції SPA було вдалим рішення. Про таке кажуть «перевірено часом».

Цей веб сервіс є гарним прикладом SPA, проте перейняти з його архітектури майже нічого не можливо оскільки ніяких бібліотек використано не було. Тому ми будемо розглядати інші, новіші веб-сайти.

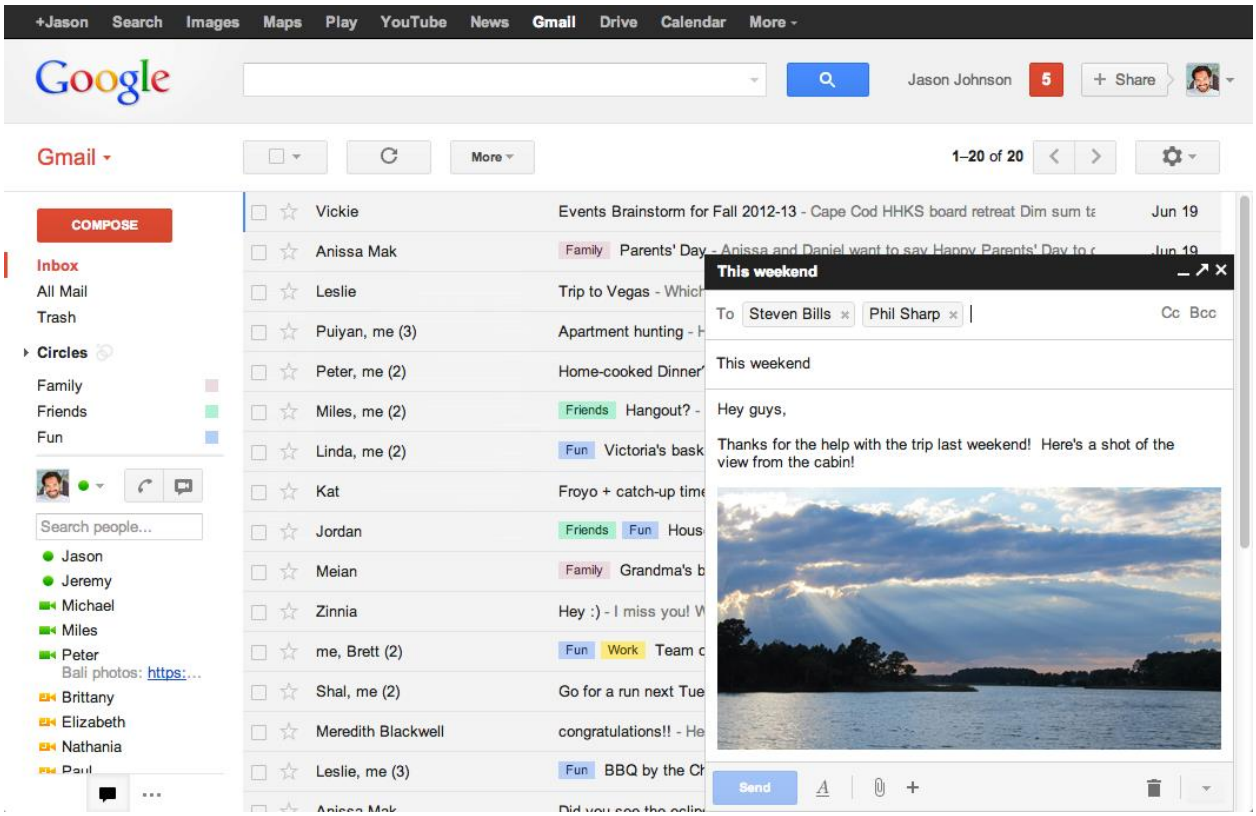

Рисунок 4 – Google Mail

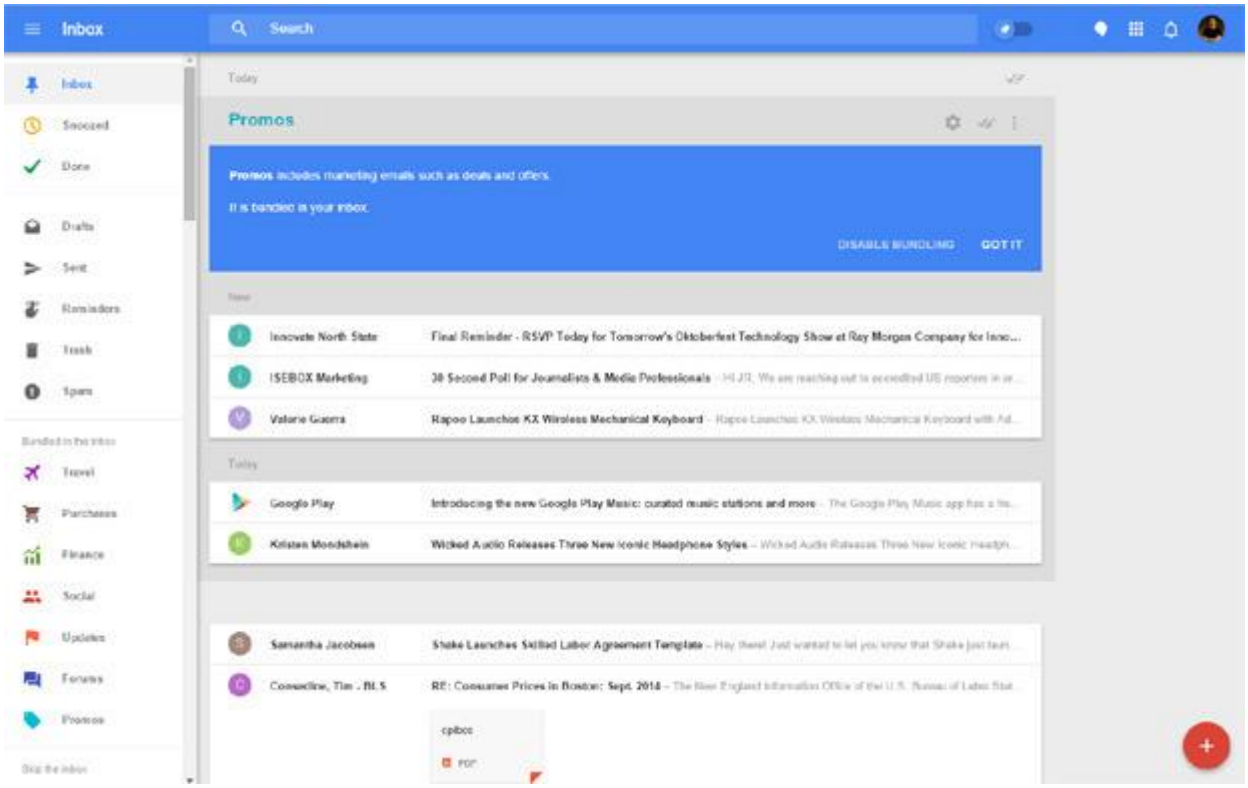

Рисунок 5 - Google Inbox

Наступний популярний онлайн сервіс яким користуються по всьому світі це Google Maps. Карти від Google дозволяють дивитися карти будь якого місця на планеті у схематичному або супутниковому вигляді з великої кількістю інформації. Також сервіс дозволяє будувати маршрути між точками використовуючи різні види транспорту та віртуально мандрувати вулицями в крупних містах.

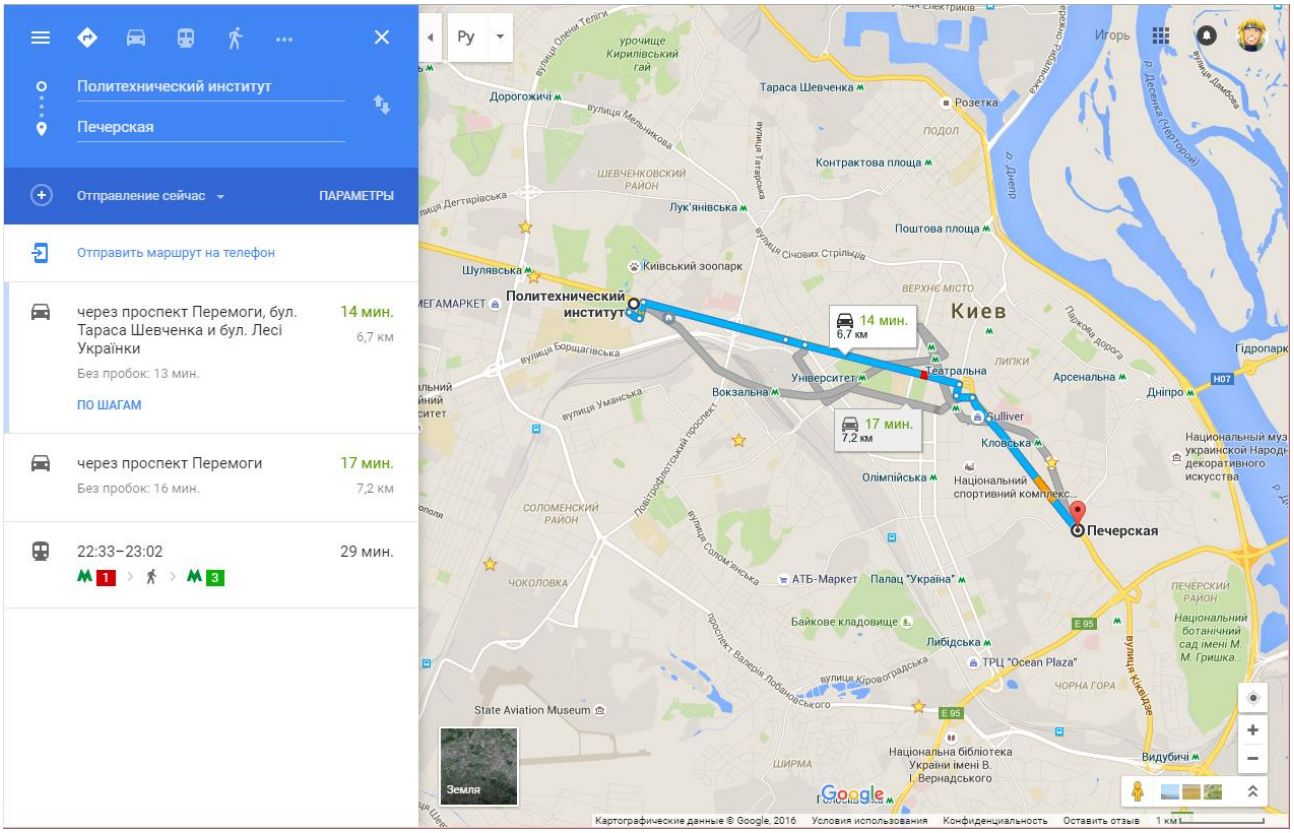

Рисунок 6 - Google Maps

1.5.2 Microsoft Office Online

Microsoft, так само як і Google, має багато односторінкових додатків серед своїх онлайн сервісів (рис. 7). Найцікавіші з них це онлайн версії популярних програм з офісного пакету Microsoft Office, такі як Word Online (рис. 9), Excel Online (рис. 10), PowerPoint Online (рис. 11) та сервіс хмарного сховища OneDrive (рис. 8). Всі ці додатки пов’язані між собою й дозволяють безпосередньо відкривати та редагувати офісні документи збережені в хмарному сховищі.

Рисунок 7 Онлайн сервіси Microsoft

Пакет необхідних кожному діловому чоловіку програм Microsoft Office завжди був пакетом нативних додатком який ставився одразу після встановлення Windows. Однак зараз кожен може скористатися знайомими офісними додатками просто відвідавши сайт з будь якого пристрою що має сучасний браузер. Звісно онлайн версія офісу має деякі обмеження й неповний функціонал, але для більшості задач вона прекрасно підійде. Також не можна не відмітити те, що вона є безкоштовною, на відміну від нативного пакету.

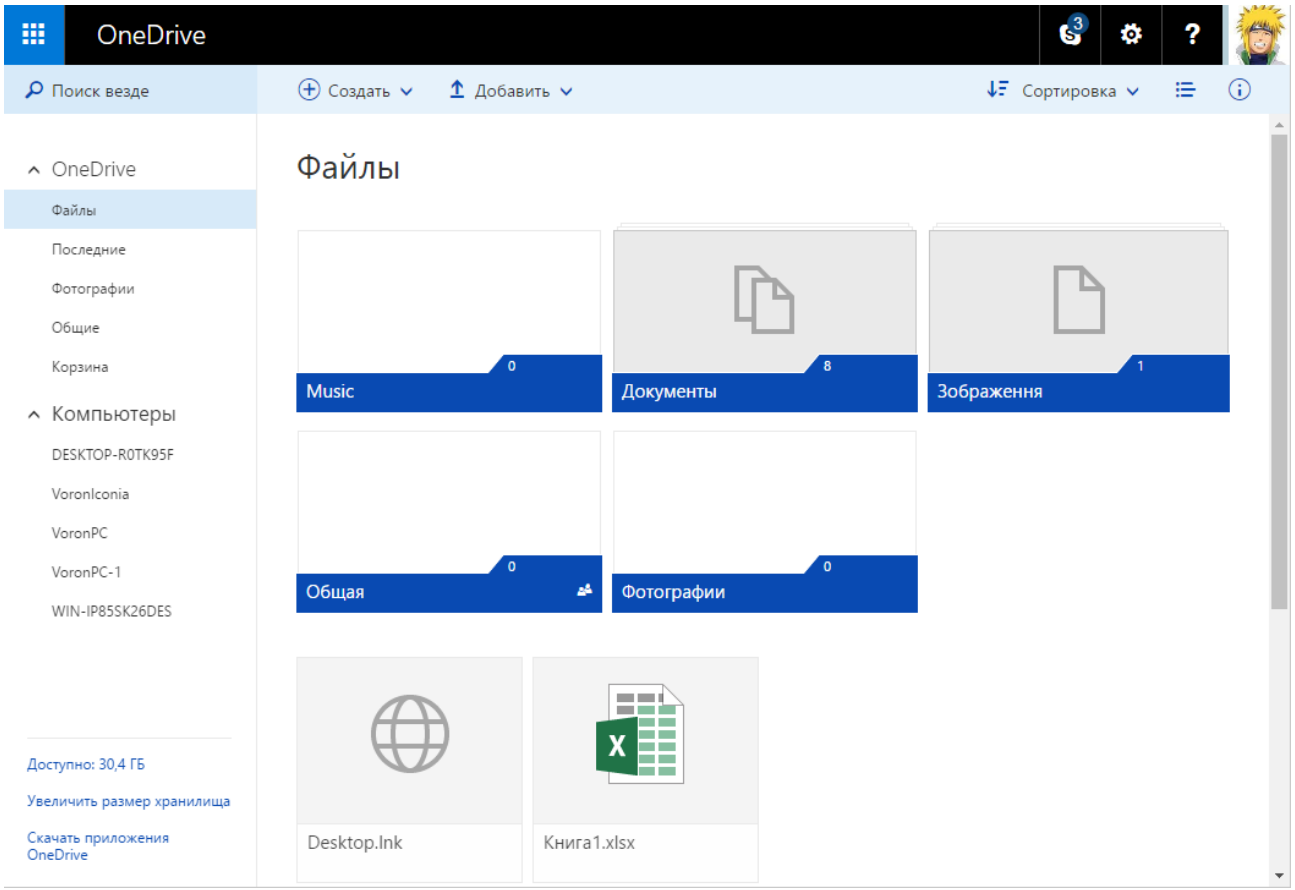

Рисунок 8 - OneDrive

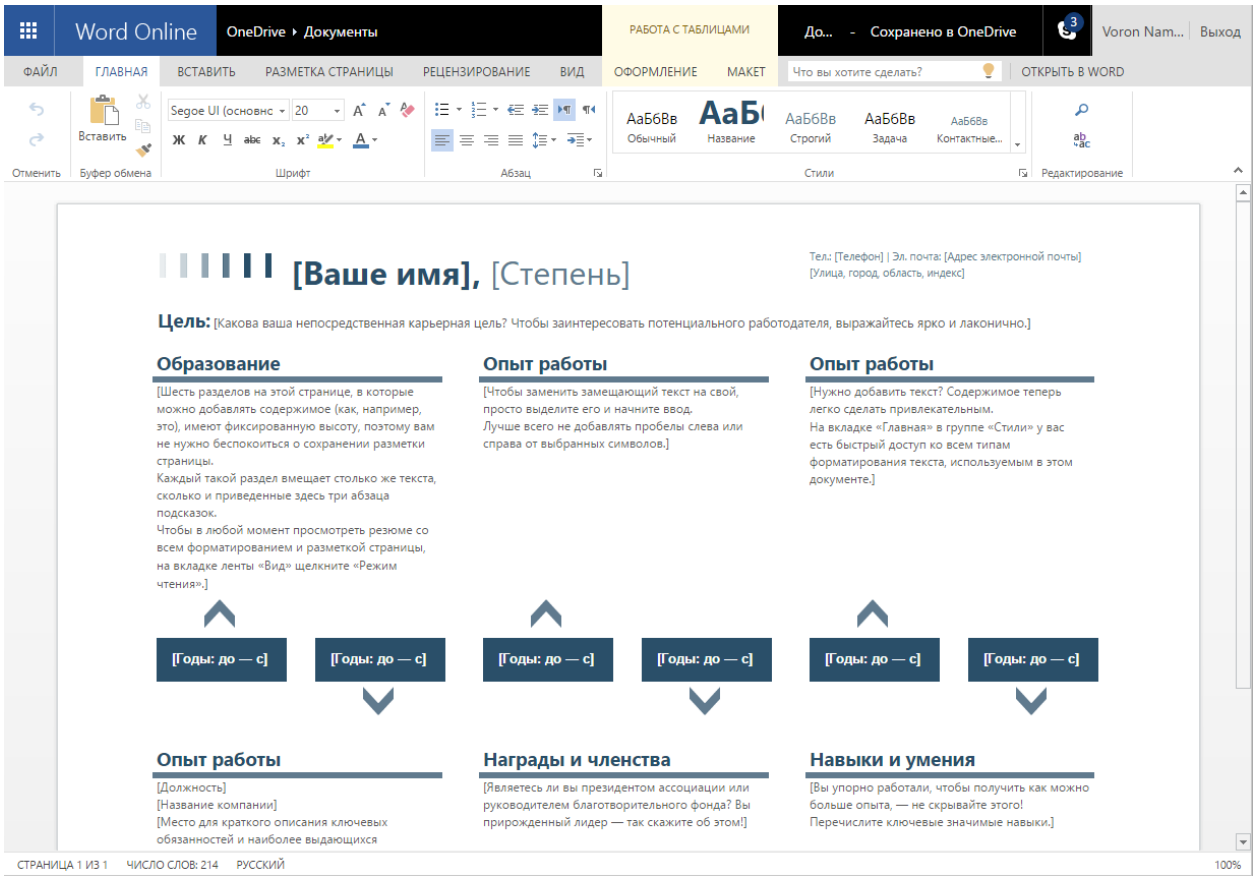

Рисунок 9 - Word Online

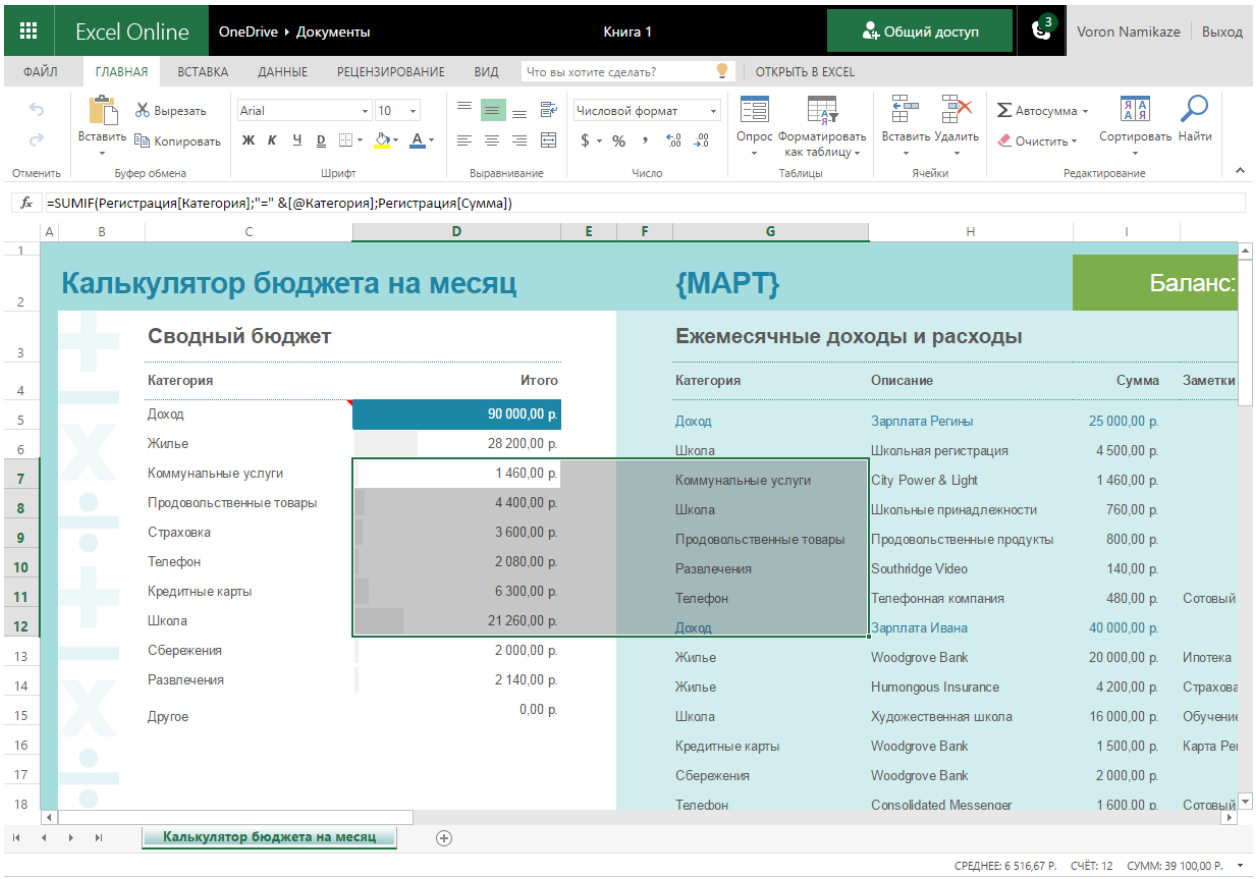

Рисунок 10 - Excel Online

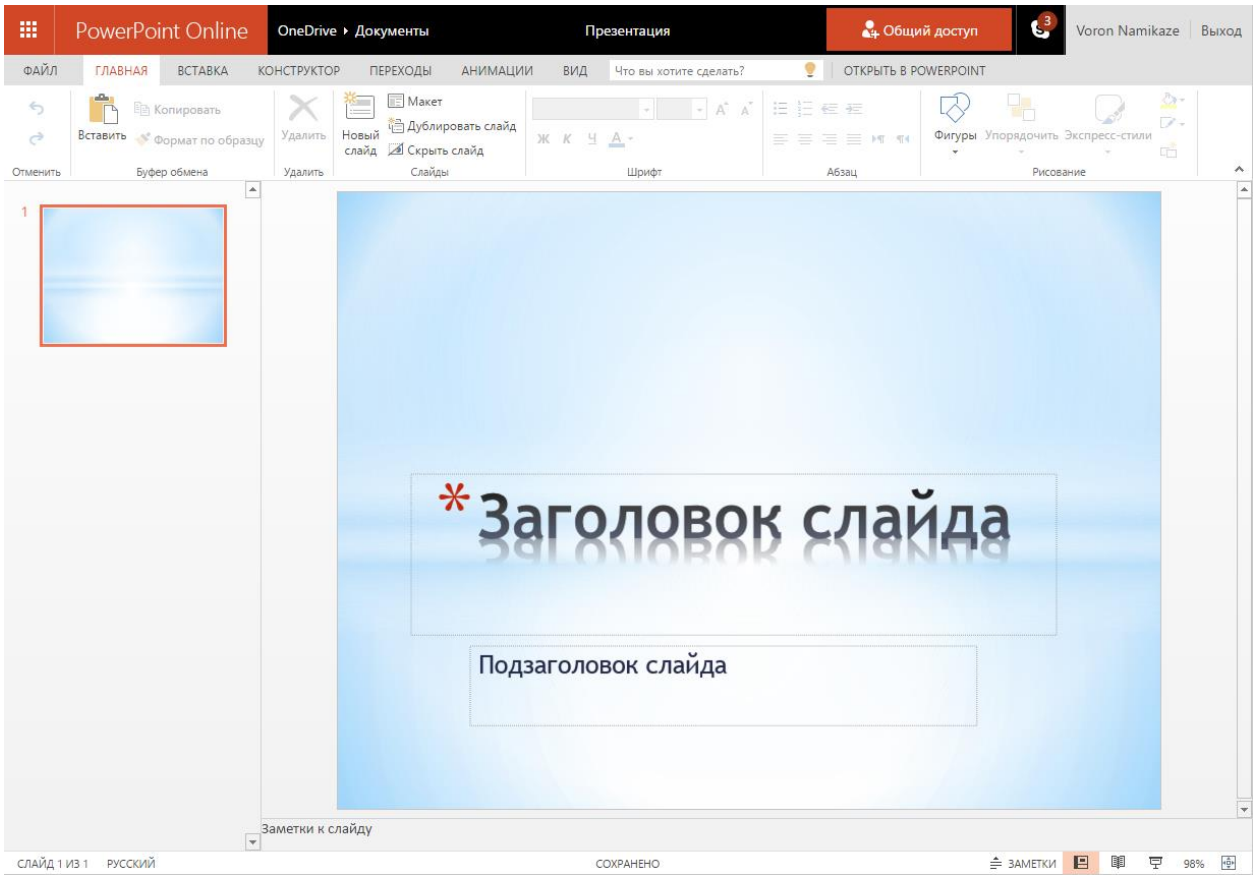

Рисунок 11 - PowerPoint Online

1.5.3 Azure Portal

Azure portal – це веб портал для керування хмаровою платформою Microsoft Azure. Цей веб сайт, що за виглядом нагадує самостійну операційну систему, насправді звичайний веб-сайт виконаний за концепцією SPA.

Заходячи на сайт користувач бачить анімацію завантаження поки не завантажаться основні необхідні для роботи додатки скрипти та ресурси. Після завантаження користувач опиняється на панелі з плитками що відображують цікаву йому інформацію (рис. 12). Зліва є панель що дозволяє керувати ресурсами. Замість відкриття нових вікон та вкладок у браузері при навігації по сайту, усі вікна відкриваються в середині сайту, один за одним й складають горизонтальну ленту вікон, що добре відображує їх дочірність (рис. 13). Цей дизайн горизонтально розташованих вікон нагадує леза бритви, тому розробники назвали вікна блейдами (англ. Blade – лезо).

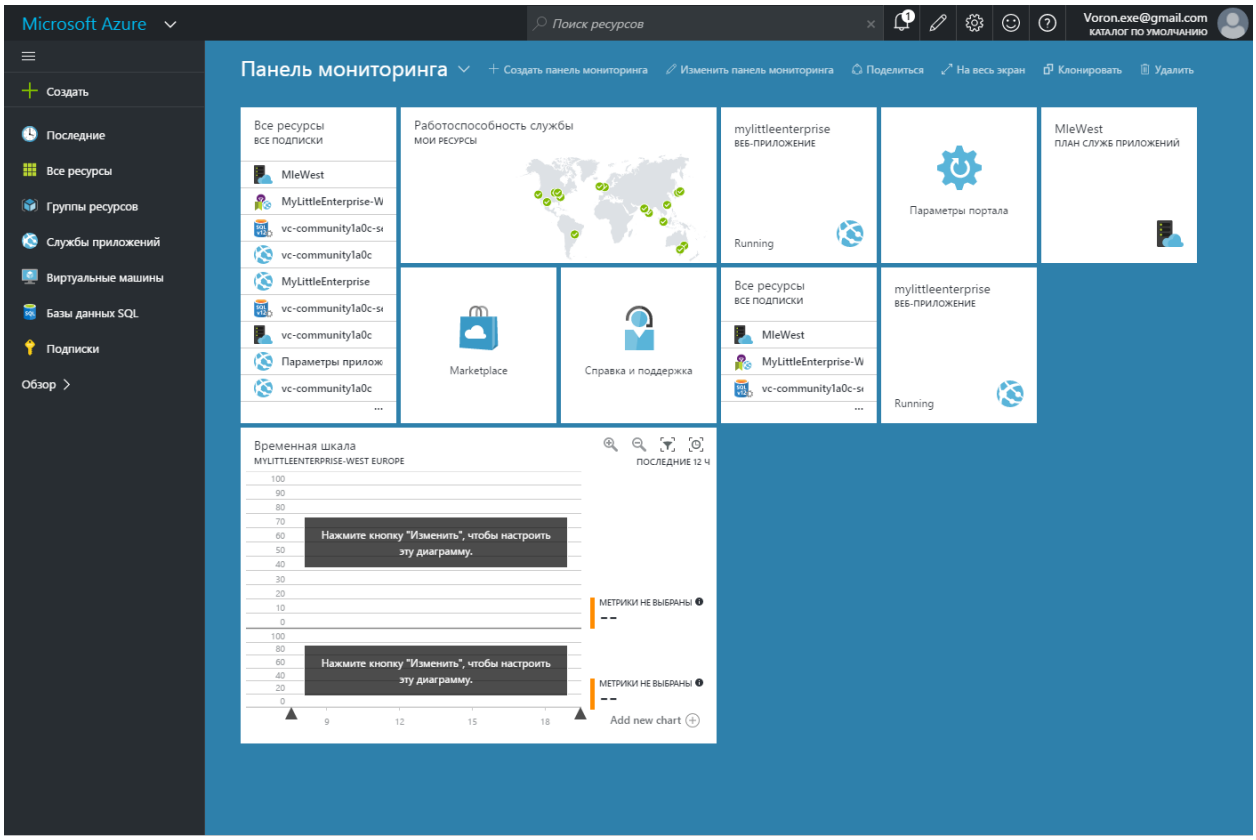

Рисунок 12 - Azure Portal Dashboard

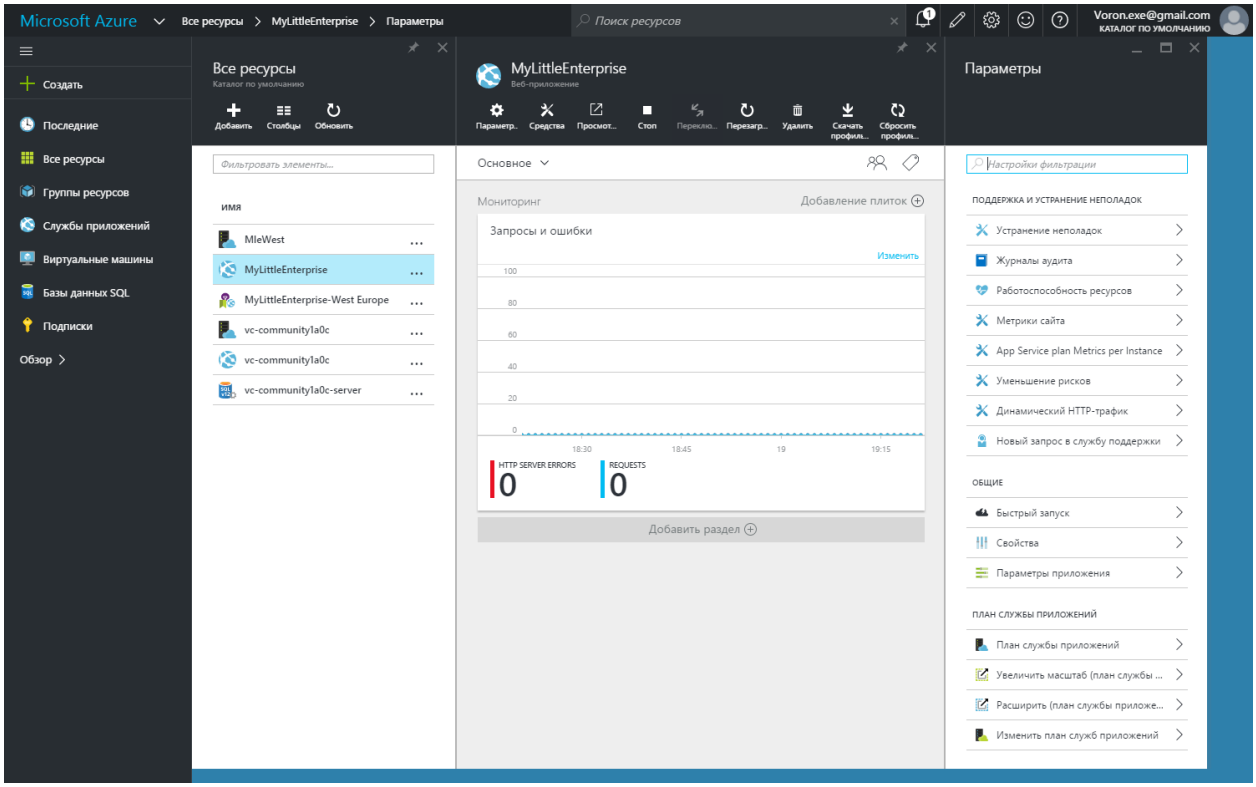

Рисунок 13 - Azure Portal Blades

Сайт надає величезну кількість можливостей керування платформою. Все відбувається швидко, оскільки все необхідне для роботи інтерфейсу завантажується одразу при вході на сайт. Додаткові ресурси підвантажуються асинхронно за допомогою скриптів, тому користувач майже не помічає цього.

Портал був розроблений у 2014 року на заміну старому порталу, архітектура якого не могла забезпечити зручне керування всіма ресурсами Microsoft Azure, кількість яких швидко зростала. Новий портал, завдяки модульності, має чудову масштабованість й зможе за незначних змін розширити функціонал у майбутньому.

Технології що були використані:

- TypeScript
- KnockoutJS
- LESS

2. ДОСЛІДЖЕННЯ ТИПОВИХ ОСОБЛИВОСТЕЙ АРХІТЕКТУРИ

2.1 Розподілення ролей

В традиційних веб-сайтах значна частина роботи по відображенню виконується сервером. Це призводить до написання складних та важких веб-серверів які мають справлятися зі значним навантаженням. В односторінкових додатках цю частину роботи виконують клієнти, що призводить до розвантаження сервера й залишає йому лише сервісну частину. Оскільки відображення та створення HTML сторінок, окрім головної, в односторінковому додатку відбувається на клієнті, для розміщення односторінкового додатку достатньо лише статичного веб-сервера та API сервера сервісу, якщо необхідно.

2.1.1 Традиційний веб-сайт

Клієнт (браузер):

1. Відтворення HTML
2. Деякі незначні скрипти

Сервер:

1. Створення HTML
2. Маршрутизація
3. Логіка роботи графічного інтерфейсу
4. API сервісу

2.1.2 Односторінковий додаток

Клієнт (браузер):

1. Створення HTML
2. Маршрутизація
3. Логіка роботи графічного інтерфейсу

Сервер:

1. Статичний веб-сервер

2. API сервісу

2.2 Інфраструктура

2.2.1 Клієнтська маршрутизація

Як нам відомо, односторінкові додатки мають одну сторінку, тобто один URL. Всі інші зміни та стани сторінки відображуються станом JavaScript змінних та можливо сховищ, таких як Cookie та LocalStorage. Перезавантажив сторінку або відкрив її в іншому браузері чи на іншому пристрої, ми отримаємо нову сторінку в начальному стані. Щоб зберігати стани при перезавантаженні можна використовувати сховища згадані вище, але це не дозволить нам перенести стан до іншого браузера чи пристрою. Звісно в невеличких додатків це може і не знадобитися, але в додатках с багатьма станами, представленнями, вікнами та іншим, це важлива функція. В традиційних сайтах для цього використовується різні сторінки кожна з яких має свій URL. Оскільки суть концепції односторінкового додатку полягає не в обмеженні сторінок, а в позбавленні перезавантаження сайту при змінах, ми можемо мати сторінки але щоб уникнути перезавантаження при зміні сторінки, треба використовувати відокремлену частину URL, наприклад символом «#». Зміни URL записуються в історію браузера й є основою для браузерної навігації. Таким чином використовуючи URL для стану сторінки ми можемо користуватися кнопкою «Назад» у браузері.

Отже реалізація сумісної маршрутизації (серверної та клієнтської) стає ще одною важливою задачею побудови односторінкового додатка.

2.2.2 Компоненти

Односторінковий додаток містить велику кількість скриптів та HTML. Це потребує гарної структуризації інакше підтримка та масштабування, можливо навіть й розробка, стануть неможливими. В традиційних сайтах структурної одиницею є сторінки. В односторінкових додатках доводиться робити структуризації іншими способами, зокрема розділенням на компоненти.

Компонент - це певний об’єкт, який має ресурси й поведінку, він може залежати він інших компонентів або містити їх. Розділення на компоненти – дуже популярний підхід який використовується у багатьох фреймворках графічного інтерфейсу.

2.2.3 Модульність

Оскільки, як було зазначено раніше, односторінкові додатки мають велику кількість ресурсів, скриптів й HTML, перше завантаження додатку може стати надто довгим. Для зменшення цього слід реалізувати динамічне підвантаження ресурсів, коли вони потрібні.

2.2.4 API модуль

Оскільки додатку все ще необхідно зв’язуватися з сервером для виконання складних або пов’язаних з базою даних операцій, гарним рішенням може стати створення окремого компонента, який відповідає за виконання операцій на сервері. Більшість таких операцій потребують авторизації, її також можна реалізувати як частину компонента.

2.2.5 Інверсія управління

З ростом кількості бібліотек та фреймворків, що використовуються при створенні односторінкових додатків, все більше набуває сенсу застосування такого принципу побудови програми як «інверсія управління», який відокремлює створення залежностей клієнта від власної логіки клієнта, що дозволяє компонентам бути слабко зв'язаними.

Згідно принципу інверсії управління, якщо в нас є клієнт що використовує певний сервіс, то він має робити це не безпосередньо, а використовуючи посередника.

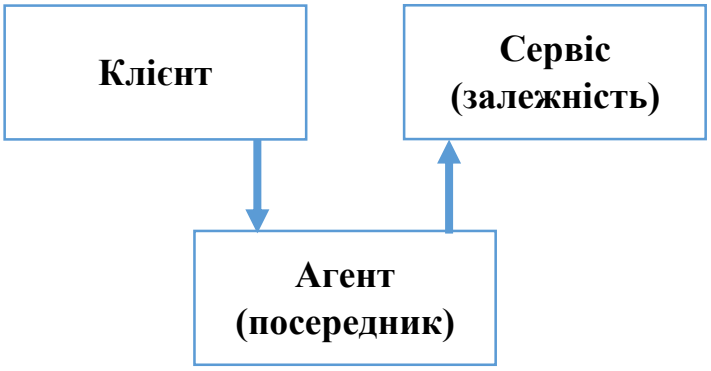

Рисунок 14 - Принцип інверсії управління

Існує три реалізації цього принципу: впровадження залежностей, фабрика та локатор сервісів.

Зараз Angular 2 - це єдиний фреймворк що нав'язує використання даного шаблону при розробці.

2.3 Графічний інтерфейс

Як відомо вигляд веб-сайту надає HTML розмітка сторінки. В традиційних сайтах HTML генерується сервером під час обробки запита. Але в односторінкових додатках бажано уникати зайвих запитів до сервера, тому окрім основного HTML сторінки, який ми отримуємо під час відкриття додатка, весь інший HTML створюється скриптами сторінки на стороні клієнта. За час існування Web для створення HTML на сервері було розроблено багато фреймворків та технологій які полегшують цей процес. Однак зараз виникла необхідність мати такі технології на стороні клієнта. Вже існують багато бібліотек які спрошують процес створення графічного інтерфейсу скриптами. Використання цих фреймворків - необхідність для швидкого створення великого односторінкового додатка який буде зручно підтримувати та масштабувати.

2.3.1 Шаблони

Одною з задач створення графічного інтерфейсу на будь якому фреймворку та й загалом в програмуванні є задача повторного використання та структуризації коду. Гарним рішенням цієї задачі є використання шаблонів.

Шаблони дозволять мати один HTML шаблон для кожного подібного компонента на сторінці змінюючи в ньому лише необхідні часті відповідно особливостям екземпляра. Можливості шаблонів залежать від фреймворка який буде обраний для написання односторінкового додатка.

Шаблони підтримуються багатьма фреймворками, зокрема:

1. Angular/Angular2
2. Knockout
3. Ember

2.3.2 Зв’язування

Наступною задачею створення графічного інтерфейсу є синхронізація або зв’язування між даними та їх відображенням. Розрізняють два види зв’язування:

1. Одностороннє зв’язування – при зміні джерела змінюється ціль зв’язування.
2. Двостороннє зв’язування – при зміні джерела змінюється ціль зв’язування й навпаки.

При великої кількості елементів що потребують зв’язування краще використовувати готові фреймворки. Вони дозволять зменшити кількість коду й вирішити складні задачі зв’язування такі як залежності між зв’язуваннями та розповсюдження змін по всьому дереву зв’язувань.

Зв’язування підтримуються багатьма фреймворками, зокрема:

1. Angular/Angular2
2. Knockout
3. Ember

2.4 Шаблони MVC та MVVM

При розробці графічного інтерфейсу часто використовують популярні шаблони MVC або його більш новий варіант MVVM.

2.4.1 Model-View-Controller

Концепція MVC була створена дуже давно у 80-ті роки минулого сторіччя й була пов'язана з графічним інтерфейсом того часу. Мета MVC – розділити код на три рівні за своїми обов’язками:

1. Модель (Model) – представляє данні й бізнес-логіку предметної області.
2. Представлення (View) – відображує модель
3. Контроллер (Controller) – приймає введення й оновлює модель

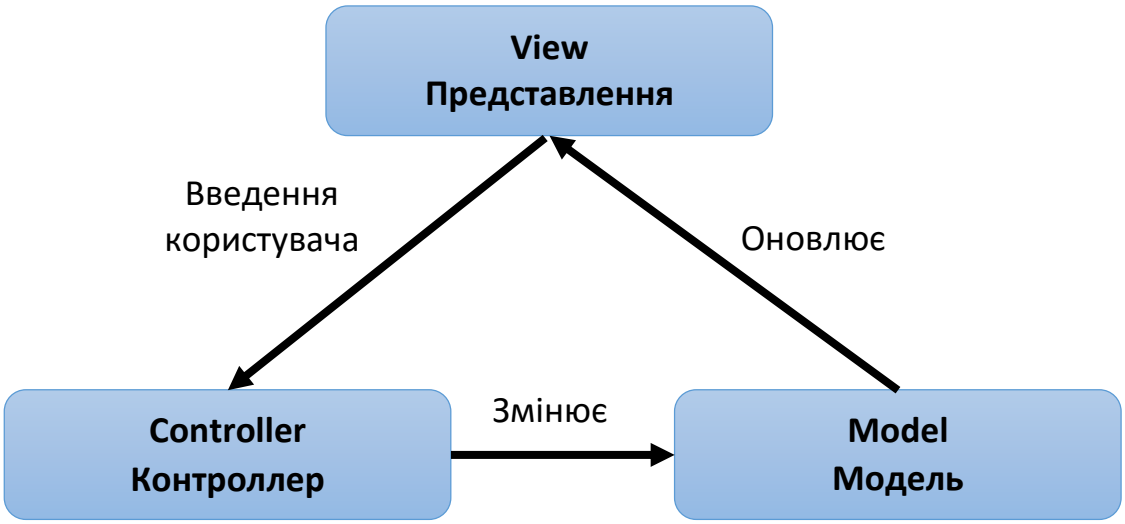

Рисунок 15 – MVC

2.4.2 Model-View-ViewModel

MVVM – більш сучасна версія MVC яка більш підходить під сучасні особливості розробки графічного інтерфейсу. В MVVM контролер відповідає як за введення користувача так і за оновлення представлення та має назву модель представлення (ViewModel):

1. Модель (Model) – представляє данні й бізнес-логіку предметної області.
2. Модель представлення (ViewModel) – абстрактне відображення представлення.

3. Представлення (View) відображає модель представлення й посилає користувацьке введення моделі представлення.

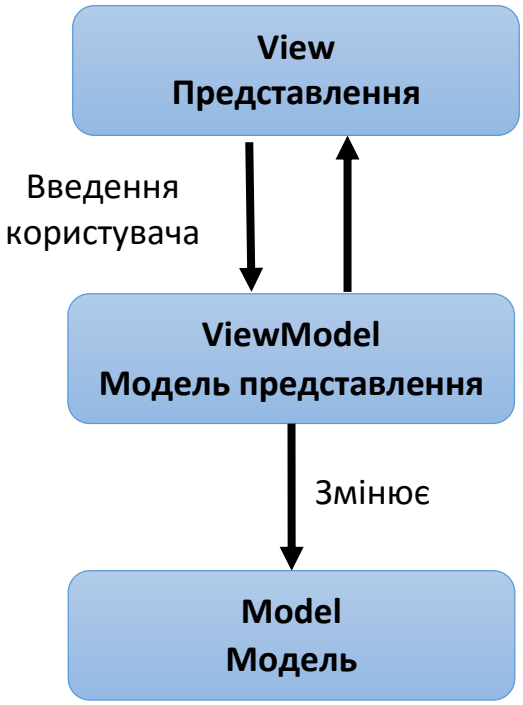

Рисунок 16 – MVVM

2.5 Висновки до розділу

Дослідив типові особливості архітектури односторінкових додатків можна зробити висновки що вона переймає особливості архітектури графічних інтерфейсів нативних додатків. Велика кількість шаблонів проектування нативних додатків після невеликої адаптації знаходить використання при розробці односторінкових веб-додатків.

3. ОГЛЯД ТА ПОРІВНЯННЯ ІНСТРУМЕНТІВ, ФРЕЙМВОРКІВ ТА ТЕХНОЛОГІЙ

3.1 Backbone

Рисунок 17 - Логотип Backbone.js

Backbone – JavaScript фреймворк який допоможе в створенні односторінкових додатків вирішуючи деякі основні проблеми що виникають при їх розробці.

Основні можливості:

1. Моделі зі зв’язування за ключом й користувацькими подіями
2. Відкритий сирцевий код (розповсюджується за MIT ліцензією)
3. Колекції з великим набором методів для роботи з перерахованими сутностями
4. Представлення з декларативною обробкою подій
5. Інтеграція за існуючим RESFUL JSON API
6. Клієнтська маршрутизація

3.2 Ember

Рисунок 18 - Логотип Ember.js

Ember – JavaScript фреймворк який спрощує створення масштабованих односторінкових додатків. Представляє з себе каркас що реалізує MVC шаблон.

Основні можливості:

1. Клієнтська маршрутизація
2. Відкритий сирцевий код (розповсюджується за MIT ліцензією)
3. Моделі які містять данні що відповідають поточному стану додатку
4. Контролери використовуються щоб надати моделі логіку відображення
5. Бібліотека Ember Data для зв'язку з сервером й отримання JSON даних.
6. Шаблони (прототипування) на мові HTMLBars

3.3 Knockout

Рисунок 19 - Логотип Knockout.js

Knockout – JavaScript фреймворк для створення візуального інтерфейсу який використовує шаблон MVVM.

Основні особливості:

1. Безкоштовний
2. Відкритий сирцевий код (розповсюджується за MIT ліцензією)
3. Написаний на чистому JavaScript
4. Легкий (54 кб мінімізований)
5. Не має залежностей
6. Підтримується всіма сучасними браузерами (IE 6+, Firefox 3.5+, Chrome, Opera, Safari (desktop/mobile))

7. Повністю задокументований

Основні можливості:

1. Декларативне зв’язування
2. Автоматичне оновлення візуального інтерфейсу при оновленні даних моделі
3. Відстежування залежностей
4. Шаблони (прототипування)

3.4 Angular 2

Рисунок 20 - Логотип Angular 2

Angular 2 – нова версія популярного фреймворка для розробки односторінкових додатків від Google. Angular 2 ще знаходиться у стадії розробки, проте вже користується великою популярністю, оскільки пропонує повноцінну інфраструктуру додатка та рішення багатьом проблемам, які не вирішують інші фреймворки.

Серед його можливостей:

1. Зв’язування
2. Відкритий сирцевий код (розповсюджується за MIT ліцензією)
3. Шаблони (прототипування)
4. Компонента архітектура додатка
5. Впровадження залежностей
6. Клієнтська маршрутизація

Angular 2 найновіший й найбільший за можливостями фреймворк, він не тільки допомагає в розробці односторінкового додатку а й пропонує його цілу інфраструктуру з використанням багатьох інших нових додаткових технологій таких як TypeScript й ECMAScript6. Також Angular 2 інтегрував в собі багато дрібних бібліотек які вирішили проблеми першої версії, наприклад Zone.js.

Для детального розглядання можливостей фреймворку та отримання практичних навичок у створенні односторінкових додатків з його допомогою зручно використовувати приклади з офіційного сайту https://angular.io/.

3.5 Висновки до розділу

Зараз існує достатньо фреймворків для комфортної розробки односторінкового додатку. Кожен з них має свої недоліки й переваги. Вибір треба роботи в залежності від вимог та цілей до розроблюваного продукту.

Наступна підсумкова таблиця показує що Angular 2 володіє найбільшою кількістю функцій та можливостей, проте, на момент написання роботи, він ще знаходиться у розробці, що робить його ризикованим вибором для серйозних проектів. Незважаючи на це, та приймаючи до уваги що мета цієї роботи в дослідженні, Angular 2 э припустим вибором.

Таблиця 2 - Порівняння фреймворків

	Backbone	Ember	Knockout	Angular 2
Перший випуск	2010	2011	2015	Ще в розробці
Ліцензія	MIT	MIT	MIT	MIT
Основний шаблон	MVP	MVC	MVVM	MVC
Зв’язування	Ні	Так	Так	Так
Шаблони (прототипування)	Ні	Так	Так	Так
Клієнтська маршрутизація	Так	Так	Ні	Так
Інверсія управління	Ні	Ні	Ні	Так
Відкритий сирцевий код	Так	Так	Так	Так

Таким чином можна зробити висновок що Angular 2 э дуже перспективним й тому стає найкращим вибором для подальшого детального розгляду й надбання практичних навичок в будуванні односторінкового веб-додатку.

4. ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РОЗРОБКИ ВЕБ-ДОДАТКУ З ВИКОРИСТАННЯМ ANGULAR 2

4.1 Підготовка прикладу

В ході роботи, на базі офіційних прикладів, був складений робочий приклад веб-додатку на якому будуть розглянуті особливості розробки односторінкового веб-додатку за допомогою фреймворку Angular 2.

4.1.1 Встановлення програмного забезпечення

Як було зазначено вище, фреймворк Angular 2 пропонує розробнику повну архітектуру та структуру веб-додатку. У якості серверної технології Angular 2 пропонує використовувати платформу Node.js засновану на JavaScript рушії 8.

Встановити її можна використовуючи інсталятор з офіційного сайту https://nodejs.org/. Для запуску прикладу необхідно мати встановлені версії Node.js 5.x.x та npm 3.x.x або новіше. Щоб перевірити версії встановленого Node.js та npm потрібно ввести в консолі команди які зображені на рисунку 21.

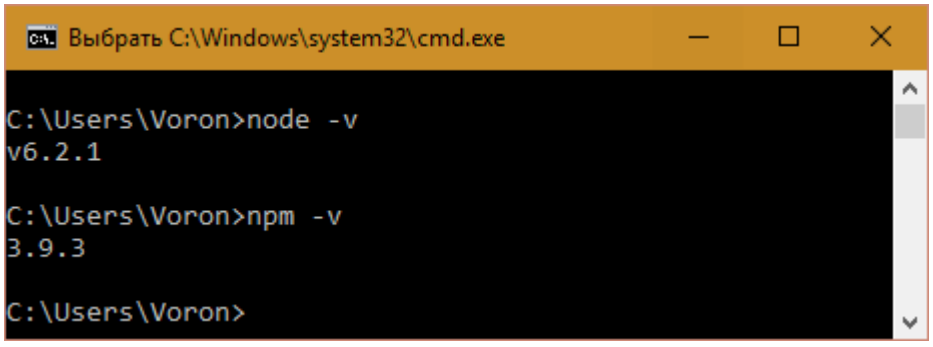

Рисунок 21 - Команди перевірки встановленої версії Node.js та npm

Інші залежності можна буде автоматично встановлювати завдяки вбудованому менеджеру пакетів npm.

4.1.2 Структура каталогів та файли

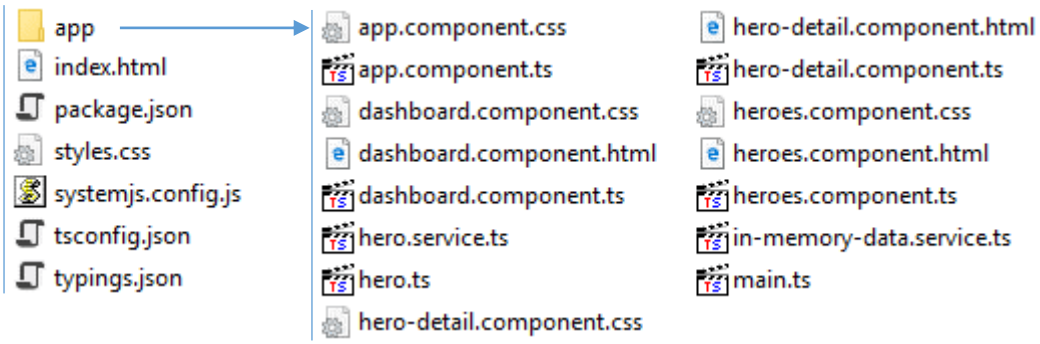

Рисунок 22 - Файлова структура прикладу

Лістинг всіх файлів прикладу наведений в додатку А.

Файли відповідні за роботу серверної частини:

- «package.json» – файл пакетного менеджеру npm, який містить перелік та версії залежностей які необхідні для роботи прикладу, а також деякі команди для збірки та запуску серверу.
- «tsconfig.json» – конфігурація компілятору TypeScript.
- «typings.json» - ідентифікує TypeScript визначення.
- «systemjs.config.js» - конфігурація завантажувача модулів SystemJS.

Файли відповідні за роботу клієнтської частини:

- «index.html» - головна сторінка веб-додатку яка містить лише основні теги та скрипти які асинхронно завантажать потрібні ресурси для повного відображення додатку.
- «main.ts» - містить код який завантажує інфраструктуру фреймворка Angular 2, сервіси та запускає додаток з головний компонентом «AppComponent». За це відповідає функція «bootstrap».
- «heroes.ts» - містить TypeScript код класу «Hero».
- «hero.service.ts» - містить TypeScript код сервісу «HeroService».

- «in-memory-data.service.ts» - містить TypeScript код сервісу «InMemoryDataService», який надає набір даних для прикладу.
- «styles.css» - містить загальні спільні стилі необхідні для роботи додатку.

Кожен компонент «AppComponent», «HeroesComponent», «HeroDetailComponent» та «DashboardComponent» має «*.ts» файл який містить його TypeScript код, та може мати «*.html» та «*.css» файли що містять HTML шаблон та CSS стилі компонента відповідно. За рекомендованими нормами кодування Angular 2 компоненти мають містити у назві «component».

4.1.3 Збірка та запуск

Керування Node.js та npm відбувається через консоль. Отже перш за все необхідно відкрити консоль та перейти до теки з проектом.

Для першого запуску, або після внесення змін у файл «package.json», треба виконати процедуру відновлення залежностей в ході якою будуть завантажені потрібні версії всіх необхідних залежностей. Для виконання цієї процедури необхідно в консолі ввести команду «npm install».

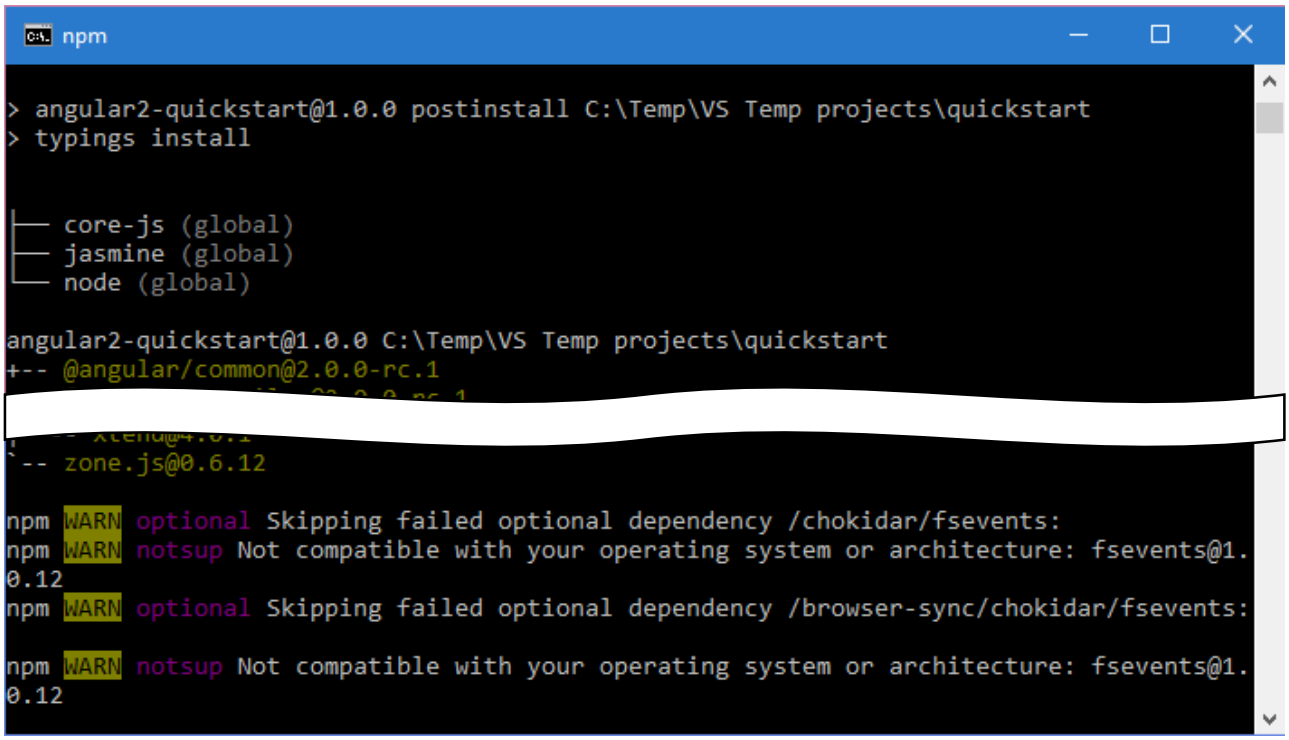

Рисунок 23 - Результат роботи команди «npm install»

Для запуска серверу необхідно в консолі ввести команду «npm start».

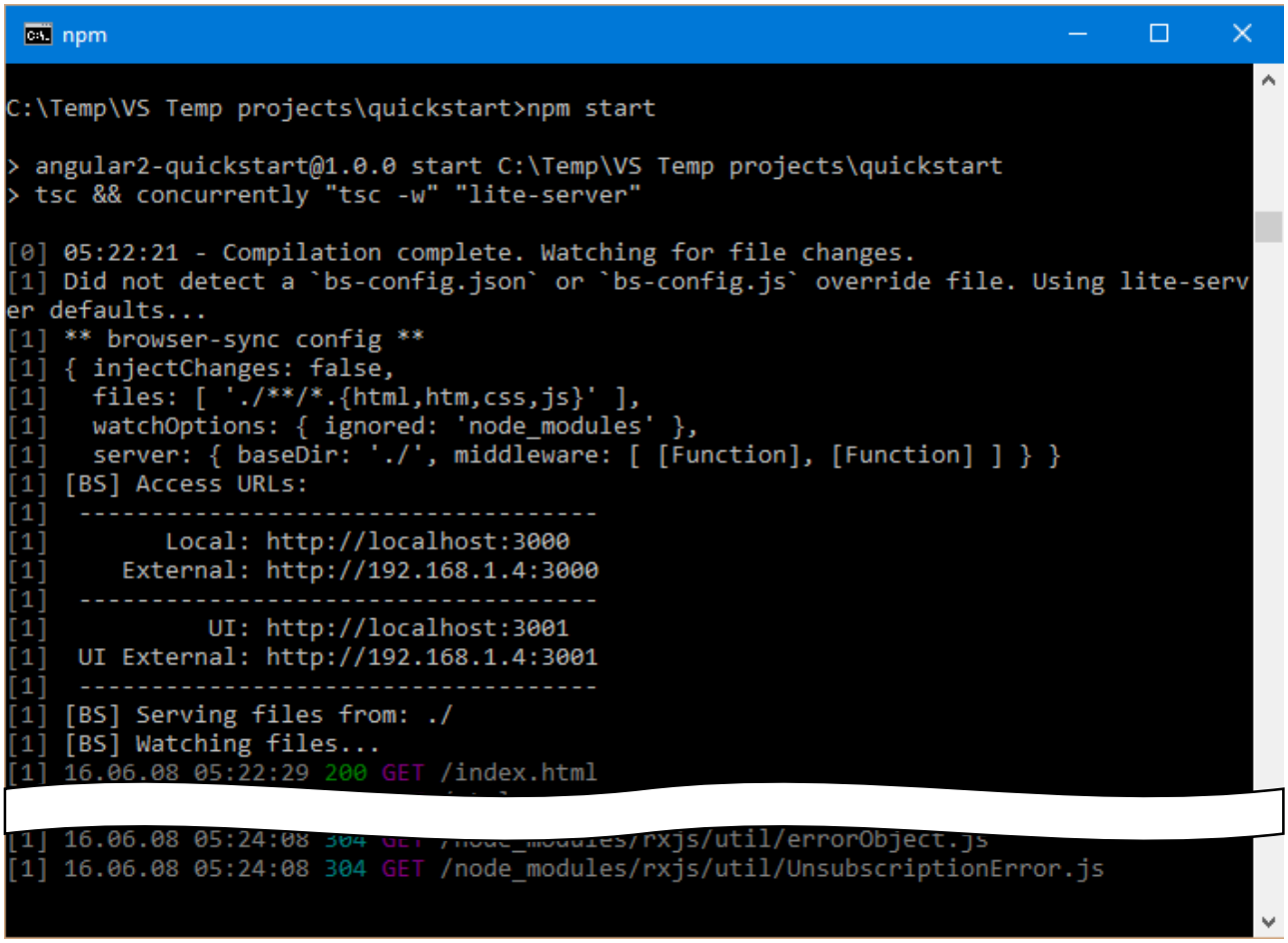

Рисунок 24 – Вивід консолі при успішному запуску серверу

Після успішного запуска сервера веб-сайт буде доступним за адресами які виведені у консолі. Також за додатковими адресами, які теж виведені у консолі, буде доступний графічний інтерфейс керування сервером.

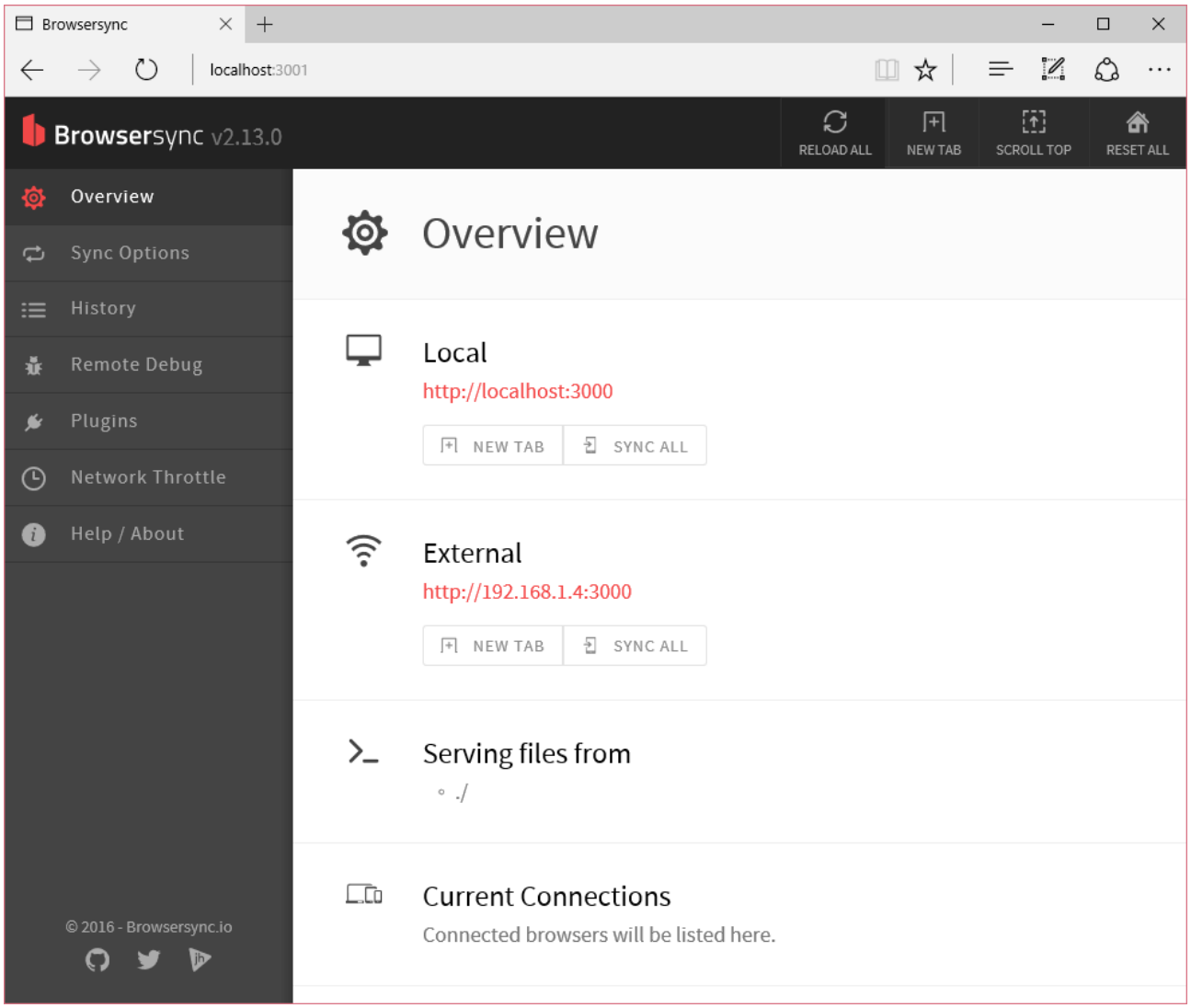

Рисунок 25 - Графічний інтерфейс керування сервером

4.1.4 Дослідження додатку

Додаток-приклад складається з кореневого компоненту «AppComponent», що містить налаштування клієнтської маршрутизації, та є контейнером для компонентів що задають поточне представлення, тобто вигляд додатку:

- «DashboardComponent» - компонент-представлення за промовчанням. Містить кнопки навігації до компонента-представлення «HeroesComponent» та список найліпших героїв

клікнувши на яких відкриється компонент-представлення «HeroDetailComponent» з деталями про героя.

- «HeroesComponent» - компонент-представлення що містить кнопку навігації до компонента-представлення «DashboardComponent» та список всіх героїв клікаючи на яких можна переглянути їх деталі компонентом-представленням «HeroDetailComponent».
- «HeroDetailComponent» - компонент-представлення що відображує та надає можливість змінити деталі певного героя та містить кнопки повернення до попереднього представлення та до компонентів-представлення «DashboardComponent» або «HeroesComponent».

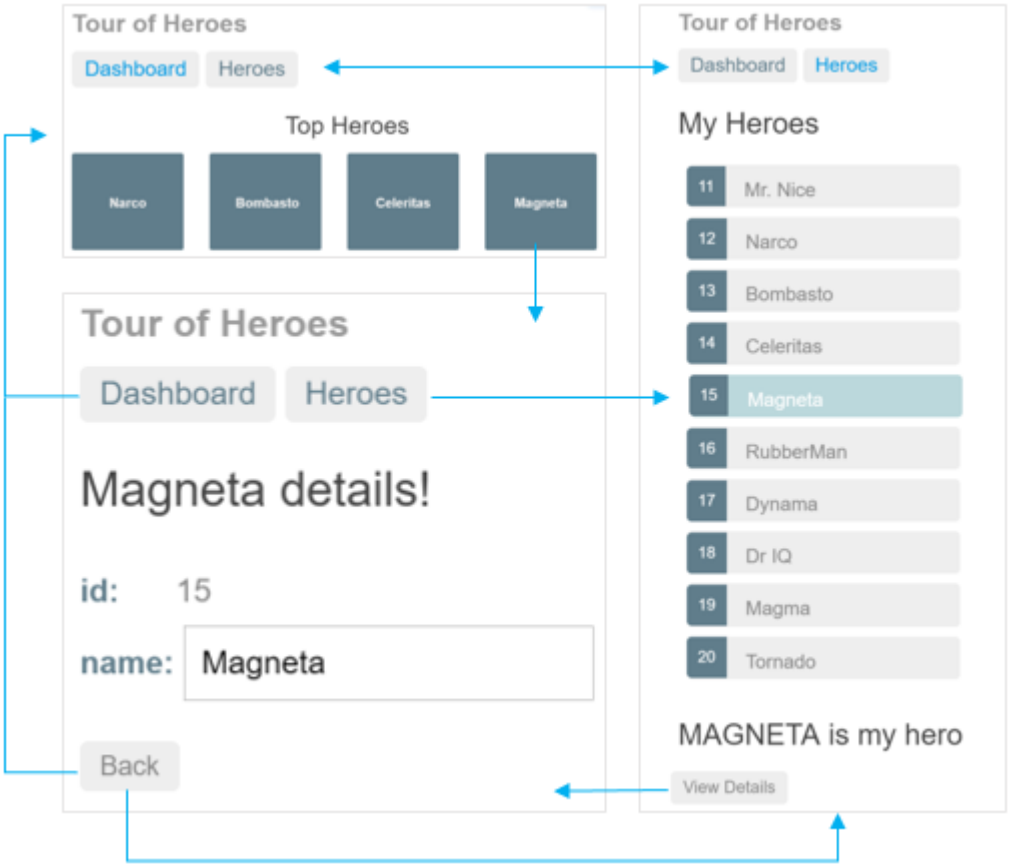

Рисунок 26 - Навігація по додатку

4.2 Модулі та TypeScript

Angular 2 рекомендує використання модульного підходу, а саме модулів ES2015 які мають гарну підтримку у TypeScript, хоча це не э необхідністю. Використання TypeScript теж може бути замінено звичайним JavaScript якщо необхідно, адже будь який TypeScript код компілюється у JavaScript. Використання модулів та TypeScript значно спрощує написання, структуризацію та підтримку коду тому відмовлятися від них не бажано.

Модулі це аналог простору імен в таких мовах програмування як C# та C++. Вони мають внутрішній та зовнішній код (класи, функції, значення), тобто код яким можуть користуватися інші модулі. Для позначення зовнішнього коду що експортує модуль використовується модифікатор «export», наприклад:

```
export class AppComponent { }
```

Для того, щоб підключити частину іншого модулю треба використовувати наступний синтаксис:

```
import { AppComponent } from './app.component';
```

Оскільки ES2015 модулі будуть підгримуватись браузерами в майбутньому, для підтримки їх в сучасних версіях браузерів необхідно додатково використовувати такий менеджер модулів як SystemJS.

4.3 Структурні частини Angular 2 веб-додатка

Веб-додаток Angular 2 структурно складається з компонентів, директив та сервісів. Компоненти контролюють певну частину DOM-елементів та містять логіку їх відображення й поведінки. Директиви, на відміну від компонентів, містять лише поведінку. Сервіси створені для зручного впровадження одного екземпляра класу в компонентах та директивах додатку які від них залежать, тобто сервіси реалізують шаблон проектування «Одинак».

Для позначення класу компонентом, директивою або сервісом Angular 2 використовуються метадані, а саме функція-декоратор, яка з необхідними параметрами розташовується перед класом.

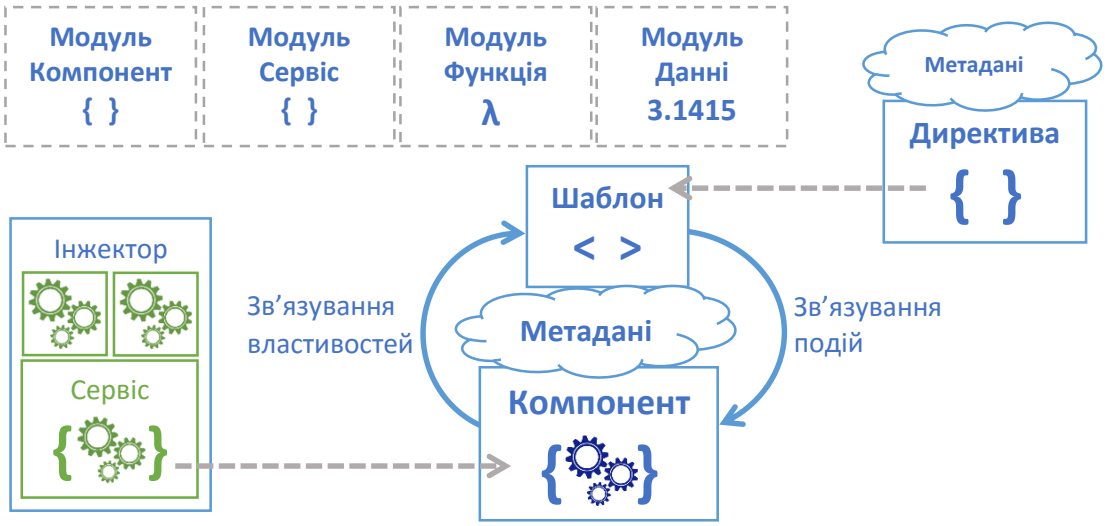

Рисунок 27 – Архітектура Angular 2 додатку

Компоненти можуть використовувати інші компоненти як дочірні, складаючи зручну ієрархію елементів додатку. Зв'язування так само працюють й між батьківськими та дочірніми компонентами.

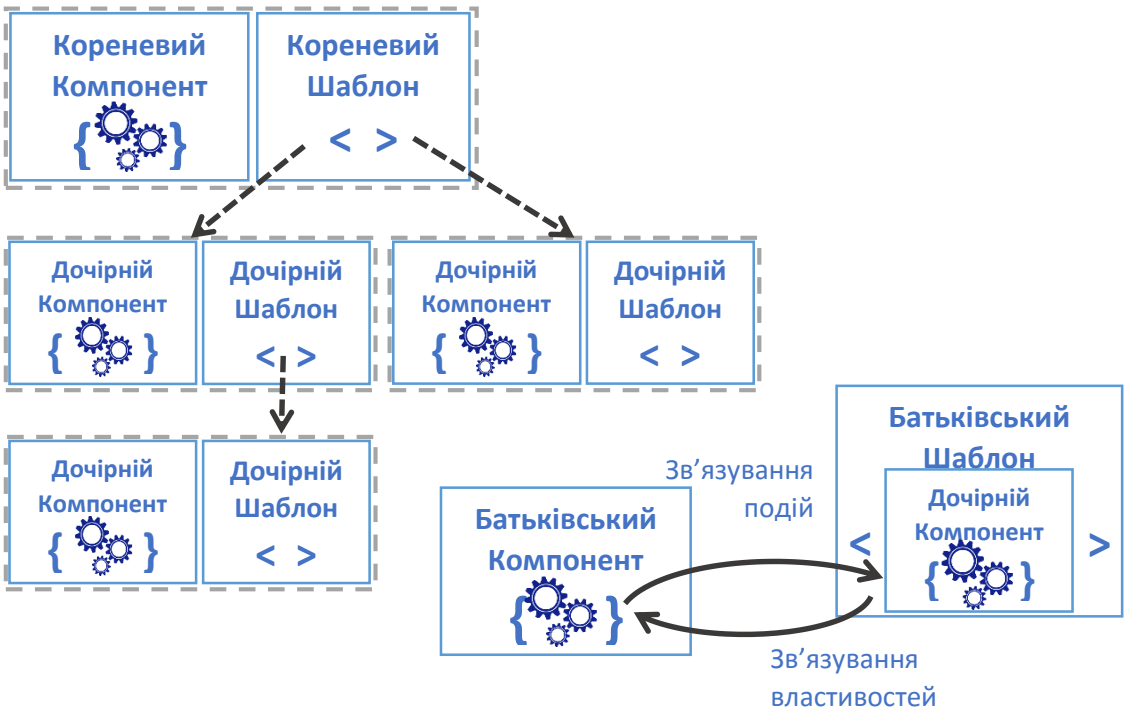

Рисунок 28 - Ієрархія та зв'язок компонентів Angular 2 додатку

4.3.1 Компоненти

Для того щоб створити компонент треба обернути клас який буде містити поведінку компонента у функцію-декоратор «@Component».

Розглянемо наступний приклад компоненту «HeroesComponent».

```
@Component({
    selector: 'hero-list',
    templateUrl: 'app/hero-list.component.html',
    directives: [HeroDetailComponent],
    providers: [HeroService]
})
export class HeroesComponent { ... }
```

Деякі параметри декоратору компонента:

1. «selector» - задає css селектор за яким фреймворк буде вибирати html елементи, що будуть відтворювати компонент, тобто будуть контейнерами для компонента. Селектор з прикладу вказує фреймворку відтворити кожен тег «<hero-list></hero-list>» як компонент «HeroesComponent».
2. «templateUrl» - задає посилання на html шаблон що буде відтворений компонентом. Для невеличких шаблонів зручно використовувати альтернативний параметр «template».
3. «directives» - задає модулі які використовуються у компоненті.
4. «providers» - задає необхідні компоненту сервіси, екземпляри реалізації яких фреймворк передасть у конструктор.

4.3.2 Директиви

Директиви поділяють на структурні, які додають, видаляють або змінюють елементи DOM та атрибутні, які змінюють вигляд або поведінку існуючих html елементів. Зазвичай для створення доданку достатньо використання вбудованих директив. Деякі популярні директиви:

1. «*ngFor» - структурна директива що інструктує фреймворк створити DOM елемент для кожного елементу з вказаного списку. Приклад:

   ```
   <div *ngFor="let hero of heroes"></div>
   ```

2. «*ngIf» - структурна директива що інструктує фреймворк створити DOM елемент лише за певної умови. Приклад:

```
<hero-detail *ngIf="selectedHero"></hero-detail>
```

3. «[(…)]=”…”» - атрибутна директива що декларує двостороннє зв’язування.

Приклад:

```
<input [(ngModel)]="hero.name">
```

Для того щоб створити власну директиву треба обернути клас який буде містити поведінку директиви у функцію-декоратор «@Directive».

4.3.3 Сервіси

Для того щоб створити реалізацію сервісу треба обернути клас який буде містити поведінку сервісу у функцію-декоратор «@Injectable».

Приклад реалізації сервісу ведення логів:

```
@Injectable()
export class Logger {
    log(msg: any) { console.log(msg); }
    error(msg: any) { console.error(msg); }
    warn(msg: any) { console.warn(msg); }
}
```

Приклад використання цього сервісу у компоненті:

```
@Component({
    providers: [HeroService]
})
export class HeroesComponent {

    constructor(private service: HeroService) {
        service.log("HeroesComponent created");
    }
}
```

4.4 Шаблони

Angular 2 має велику кількість вбудованих директив що дозволяють гнучко та зручно керувати виглядом компонентів завдяки шаблонам. При необхідності користувач може створити власні директиви.

4.4.1 Директиви зв`язування

Angular 2 дозволяє зв’язувати властивості, атрибути та події елементів шаблону компонента з властивостями та функціями компонента та дочірніх компонентів. Зв’язування відбувається декларативно через використанням спеціального синтаксису у шаблоні:

1. Інтерполяція - дозволяє одностороннє оновлювати властивості та вміст DOM-елементів при оновленні значення компонента. Приклад:
```
<div>{{hero.name}}</div>
```
2. Зв’язування властивостей - дозволяє одностороннє зв’язати властивості DOM-елементів та дочірніх компонентів з властивостями компонента. Приклад:
```
<hero-detail [hero]="selectedHero"> </hero-detail>
```
3. Зв`язування подій - дозволяє зв’язати події DOM-елементів з методами компонента. Приклад:
```
<div (click)="selectHero(hero)"></div>
```
4. Двостороннє зв’язування – дозволяє двостороннє зв’язати властивості DOM-елементів та дочірніх компонентів з компонентом. Приклад:
```
<input [(ngModel)]="hero.name">
```
5. Зв`язування атрибуту – дозволяє задати значення атрибуту DOM-елементу при його створенні. Приклад:
```
<button [attr.aria-label]="help">help</button>
```
6. Зв`язування класу – дозволяє включати та виключати клас з DOM-елементу в залежності від значення виразу. Приклад:
```
<div [class.special]="isSpecial">Special</div>
```
7. Зв`язування стилю – дозволяє включати та виключати стиль з DOM-елементу в залежності від значення виразу. Приклад:
```
<button [style.color]="isSpecial ? 'red' : 'green'">
```

4.4.2 Структурні директиви

Структурні директиви дозволяють додавати, видаляти, повторювати або змінювати елементи DOM в залежності від значень певних змінних, властивостей компоненту або їх виразів.

1. Умовна директива – дозволяє додавати та прибирати елемент DOM в залежності від значення логічного виразу. Приклад:
```
<hero-detail *ngIf="isActive"></hero-detail>
```

2. Директива перемикач – дозволяє змінювати елемент DOM в залежності від значення виразу. Приклад:

```
<span [ngSwitch]="toeChoice">
    <span *ngSwitchWhen="'Eenie'">Eenie</span>
    <span *ngSwitchWhen="'Meanie'">Meanie</span>
    <span *ngSwitchWhen="'Miney'">Miney</span>
    <span *ngSwitchWhen="'Moe'">Moe</span>
    <span *ngSwitchDefault>other</span>
</span>
```

3. Директива повтору – дозволяє створити елемент DOM для кожного елементу списку. Приклад:

```
<hero-detail *ngFor="let hero of heroes" [hero]="hero"></hero-detail>
```

4.5 Клієнтська маршрутизація

Для організації клієнтської маршрутизації в Angular 2 існує спеціальний компонент «ComponentRouter» який дозволяє зробити маршрутизацію основану на компонентах. Для його роботи необхідно виконати декілька кроків:

1. Додати «<base href="/" >» до «<head>…</head>» секції головної сторінки «index.html».
2. Підключити наступні модулі:

```
import { RouteConfig, ROUTER_DIRECTIVES, ROUTER_PROVIDERS } from '@angular/router-
deprecated';
```

3. До компонента, що буде використовувати «ComponentRouter» необхідно додати наступні директиви й сервіси:

```
@Component({
  …
  directives: [ROUTER_DIRECTIVES],
  providers: [ROUTER_PROVIDERS]
})
```

4. Додати конфігурацію маршрутів та компонентів що будуть їм відповідати:

```
@RouteConfig([
    {
        path: '/heroes',
        name: 'Heroes',
        component: HeroesComponent
    }
])
```

5. Додати тег «<router-outlet></router-outlet>» до шаблону компоненту який буде використовувати «ComponentRouter».

Після цього при навігації по адресу «/heroes» компонент маршрутизації завантажить компонент «HeroesComponent» у DOM-елемент «<router-outlet></router-outlet>».

4.6 Висновки до розділу

В даному розділі були досліджені особливості розробки з використанням фреймворку Angular 2 на прикладі складеному з офіційних прикладів фреймворку. Також була розглянута переважна більшість основних можливостей фреймворка та як їх використовувати. Детально описана структура додатку, необхідне для його роботи програмне забезпечення та процес створення й запуску додатку.

Можна зробити висновок що фреймворк Angular 2 є дуже гнучкий й зручним та володіє великою кількістю можливостей для створення сучасних односторінкових веб-додатків. Найбільшим недоліком фреймворку є те, що на момент написання роботи він ще знаходиться у стадії розробки, що робить ризикованим його використання в серйозних проектах, проте дає надії на розширення його можливостей в завершеній версії.

5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для демонстрації концепції односторінкових веб-додатків. Прототипи ігор були розробленні на за допомогою Unity та Construct2

Програмні продукти призначено для використання на персональних комп’ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність

скороченням кроків, що не впливають на цінність і відповідно витрат.

- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
- забезпечувати високу швидкість обробки даних та відклик користувачеві у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем;
- забезпечувати можливість зручного масштабування та обслуговування;
- передбачати мінімальні витрати на впровадження програмного продукту.

5.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір серверної технології;

F_2 – вибір клієнтського фреймворка;

F_3 – вибір клієнтського менеджеру модулів.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) ASP.NET;

б) PHP;

в) Node.js

Функція F_2:

а) Angular 2

б) Backbone

в) Ember

Функція F_3:

а) Webpack

б) Browserify

в) SystemJS

5.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 29). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 3).

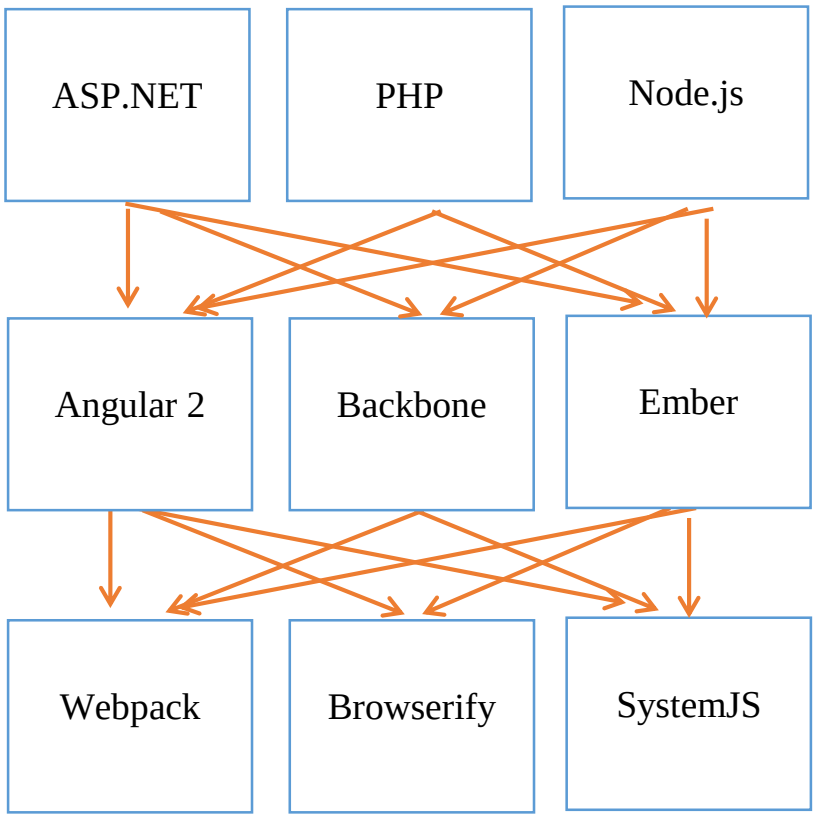

Рисунок 29 - Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 3 - Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Великі можливості платформи та зручне масштабування	Прив’язка до платформи .Net
	Б	Низький поріг входу для новачків,	Важче відладити
	В	Кросплатформеність, спільна мова для клієнта та сервера	Погана швидкість та можливості відладки
F2	А	Потужний найновіший фреймворк з готовою архітектурою	Продукт ще знаходиться в стадії розробки
	Б	Бібліотека не потребує певної архітектури, використовує інші відомі бібліотеки	Найстаріший з обраних варіантів
	В	Зручність, надійність	Не використовую найсучасніші технології
F3	*А*	Багаті можливості налаштування та пакування	Необхідність складного налаштування
	Б	Динамічне оновлення скриптів при змінах, зручність розробки	Втрачає популярність на користь більш новим бібліотекам, не підтримує майбутній формат модулів
	В	Реалізовую синтаксис який в майбутньому буде підтримуватися браузера без використання додаткових бібліотек	Погана можливість налаштування

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки на вивчення трьох серверних технологій потрібно надто багато часу, обираємо лише 2 варіанти А та Б .

Функція *F2:*

Так само вибираємо лише 2 фреймворки, варіанти А та В .

Функція *F3:*

Варіант А надто складний налаштуванні, варіант Б застарілий й втрачає популярність, тому залишимо тільки варіант В.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3в
2. F1б – F2в – F3в

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.2 Обґрунтування системи параметрів ПП

5.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри: На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування серверної технології;
- *X2* – об’єм пам’яті для збереження даних;
- *X3* – час обробки даних;
- *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій мови програмування залежно від обраної серверної технології.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

5.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.

Таблиця 4 - Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки запитів користувача	X3	мс	200	100	50
Потенційний об’єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 4 будуються графічні характеристики параметрів – рис. 30 – рис. 33.

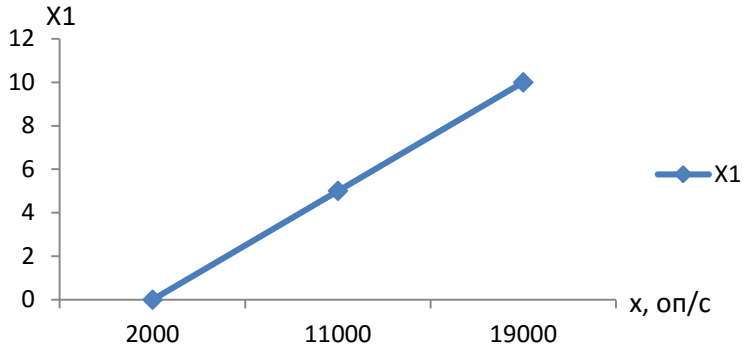

Рисунок 30 - X1, швидкодія мови програмування

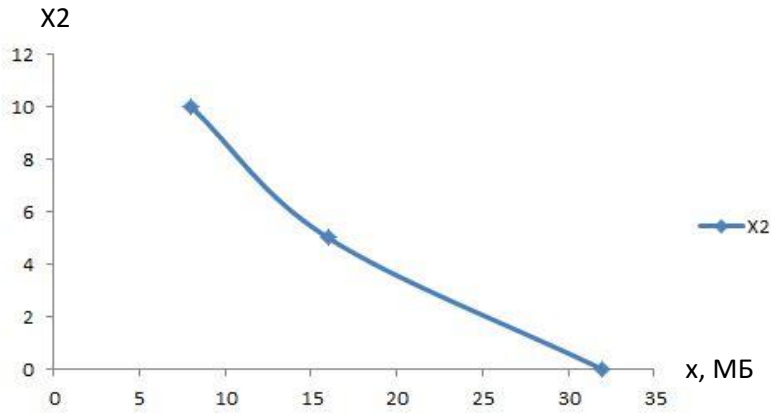

Рисунок 31 - X2, об’єм пам’яті для збереження даних

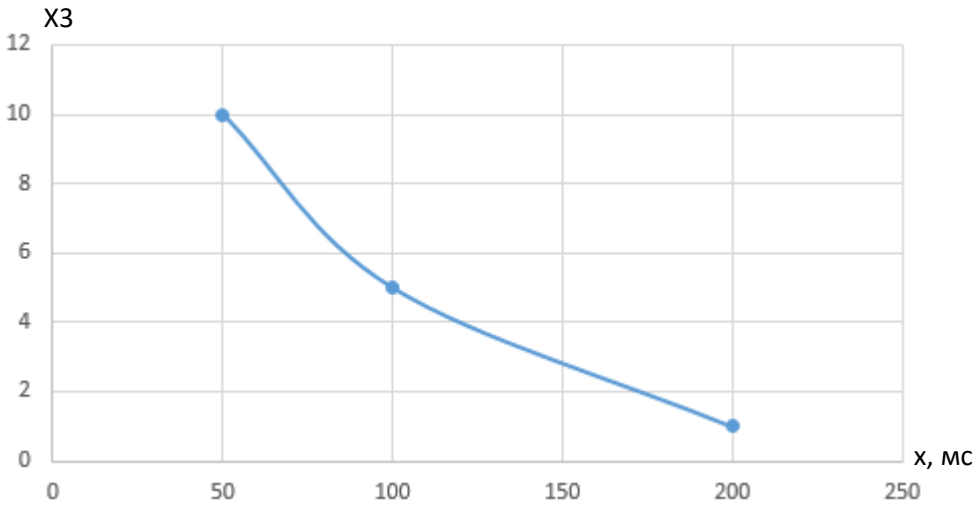

Рисунок 32 - X3, час обробки запитів користувача

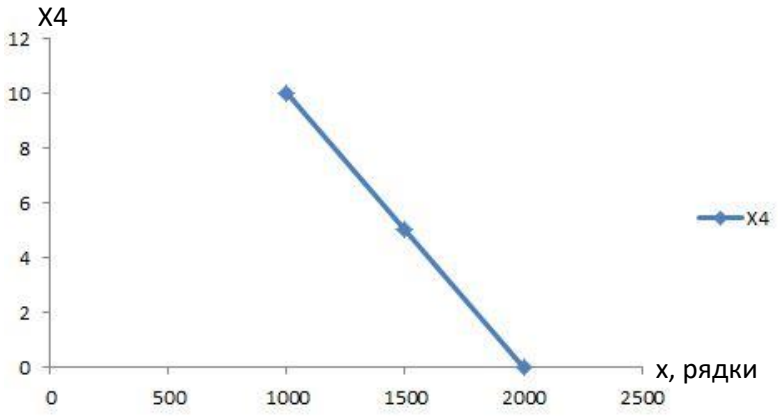

Рисунок 33 – X4, потенційний об’єм програмного коду

5.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.

Таблиця 5 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	3	2	2	1	1	2	2	13	-4,5	20,25
X2	Об’єм пам’яті для збереження даних	Мб	1	1	1	2	2	1	1	9	-8,5	72,25
X3	Час обробки запитів користувача	Мс	2	3	3	3	3	3	4	21	3,5	12,25
X4	Потенційний об’єм програмного коду	кількість строк коду	4	4	4	4	4	4	3	27	9,5	20,25
	Разом		10	10	10	10	10	10	10	70	0	195

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 195.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 195}{7^2(4^3 - 4)} = 0{,}8 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.

Таблиця 6 - Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	>	>	<	<	<	0,5
X1 і X3	<	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 7, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 7 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	0,5	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X2	1,5	1,0	1,5	1,5	5,5	0,344	21,25	0,36	77,875	0,361
X3	0,5	0,5	1,0	1,5	3,5	0,219	12,25	0,208	44,875	0,207
X4	0,5	0,5	0,5	1,0	2,5	0,156	9,25	0,157	34,125	0,158
Всього:					16	1	59	1	216	1

5.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*потенційний об’єм програмного коду) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 2000 або варіанту б) 1500

Абсолютне значення параметра *X4 (*час обробки запитів користувача) обрано не найкращим (не мінімальним), тобто це значення відповідає або варіанту а) 50 мс або варіанту б) 100 мс

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

Таблиця 8 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри, що беруть участь у реалізації функцій	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	100	5	0,274	1,37
F2	А	X2	500	6,7	0,361	2,42
F3	А	X3	1500	2,5	0,158	0,395
	Б		800	6,4	0,158	1,01
	А	X4	50	7,5	0,207	1,55
	Б		100	5,5	0,207	1,14

За даними з таблиці 8 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$К_{К1}$ = 1,37 + 2,42 + 1, 55 + 0,395 = 5,725

$К_{К2}$ = 1,37 + 2,42 + 1,14 + 1,01 = 5,95

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \quad (5.1)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість

дорівнює: $Т_Р$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90·1.7·0.8 = 122.4 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$Т_2$ = 27 · 0.9 · 0.8 = 19.44 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (122.4 + 19.44 + 4.8 + 19.44) · 8 = 1328,64 людино-годин;

$Т_{II}$ = (122.4 + 19.44 + 6.91 + 19.44) · 8 = 1345.52 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 6000 грн., один фінансовий аналітик з окладом 9000грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{6000 + 9000}{2 \cdot 21 \cdot 8} = 44{,}64 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д$,

де $С_ч$– величина погодинної оплати праці програміста; $Т_і$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 44,64 \cdot 1328.64 \cdot 1.2 = 71172,59$ грн.

II. $С_{ЗП} = 44,64 \cdot 1345.52 \cdot 1.2 = 72076,82$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 71172,59 \cdot 0.22 = 24429$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.3677 = 67281,38 \cdot 0.22 = 24739$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 6000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 6000 \cdot 0,2 = 16800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 16800 \cdot (1 + 0.2) = 20280$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 20280 \cdot 0,22 = 4461,6$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 8000 = 2300$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,9733 \cdot 2,0218 = 523.83$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0,67 = 5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС} = 17280 + 6353,86 + 2300 + 460 + 523.83 + 5360 = 33883,55$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 33883,55/1706,4 = 20,68$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$С_М = С_{М\text{-}Г} \cdot Т$

I. $С_М = 20,68 \cdot 1328,64 = 27476$ грн.;

II. $С_М = 20,68 \cdot 1345.52 = 27825$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 71172,59 \cdot 0,67 = 47685,63$ грн.;

II. $С_Н = 72076,82 \cdot 0,67 = 48291,47$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП} = 71172,59 + 4461,6 + 27476 + 47685,63 = 150795,82$ грн.;

II. $С_{ПП} = 72076,82 + 4461,6 + 27825 + 48291,47 = 152654,89$ грн.;

5.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$К_{ТЕРj} = К_{Кj} / С_{Фj}$,

$К_{ТЕР1} = 5,325 / 150795,82 = 0,35 \cdot 10^{-4}$;

$К_{ТЕР2} = 6,35 / 152654,89 = 0,42 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,42 \cdot 10^{-4}$.

5.6 Висновки до розділу

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,42 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Серверна технологія ASP.NET;
- Клієнтський фреймворк Angular 2;
- Клієнтський менеджер модулів SystemJS.

Обраний варіант є оптимальний й найкраще відповідає вимогам поставленим в ході цієї роботи.

ВИСНОВКИ

В ході даної дипломної роботи, в першому розділі було детально досліджено популярну сьогодні концепцію веб-сайту “односторінковий додаток” та ключові елементи що її складають, наведені приклади сучасних односторінкових додатків. Проаналізувавши недоліки та переваги концепції у порівнянні з традиційним веб-сайтом та нативним додатком було встановлено, що на сьогодні нативні додатки мають перевагу у продуктивності, мережевою незалежності та більших можливостях. Односторінкові веб-додатки в свою чергу мають переваги в багатоплатформності й відсутній необхідності встановлення. Однак у майбутньому межи між ними будуть все більш розмиватися.

Також було встановлено що концепція проста й була придумана вже давно, проте набула популярності тільки нещодавно, оскільки з’явились інструменти, фреймворки та технології що дозволяють значно зменшити великий об’єм що необхідний для реалізації даної концепції та значно спростити процес розробки.

Досліджені в другому розділі типові особливості архітектури односторінкового веб-додатку відображують тенденцію до адаптації вже відомих шаблонів проектування та рішень що використовуються при розробці нативних додатків. Це ще раз підкреслює мету концепції односторінкових веб-додатків, що полягає у надаванні веб-сайтам властивостей нативних додатків.

Встановлено, що архітектура односторінкового додатку має певні необхідні елементи, проте велика кількість підходів не дозволяють створити єдиний правильний варіант. Вибір технології, архітектурних рішень та підходів дуже залежить від задач які буде вирішувати додаток та його складності.

Оглянуті в третьому розділі фреймворки та технології дозволяють зробити висновок про зростання інтересу до розробки веб-сайтів даного типу в останні декілька років. Нові фреймворки з’являються кожні пару років й значно спрощують процес розробки зменшуючи об’єму коду, структуруючи його та адаптуючи відомі рішення з інших платформ. На основі порівняння декількох

популярних фреймворків, «Angular 2» був обраний як найбільш сучасний та перспективний.

У четвертому розділі був повністю досліджений процес створення односторінкового веб-додатку з використання обраного фреймворку «Angular 2». Було розглянуто необхідне для роботи фреймворка програмне забезпечення та процес його встановлення. Детально розібраний приклад складений з офіційних прикладів роботи з фреймворком. Також була розглянута переважна більшість основних можливостей фреймворка та досліджено як їх використовувати. Детально описана структура додатку, процес створення й запуску додатку. Розробка з використанням даного фреймворку була зручна й комфортна. Його можливостей цілком достатньо для створення досить складних додатків. Як було зазначено в роботі, головним його недоліком є те, що на момент написання роботи він ще знаходиться у стадії розробки, що робить ризикованим його використання в серйозних проектах.

Набуті в ході даної роботи знання концепції, архітектури та практичні навички роботи з фреймворком «Angular 2» були використанні в розробці інтерфейсу у вигляді односторінкового веб-додатку для позадипломного проекту , мета якого полягає у створенні системи обліку документів та товарів невеликого підприємства роздрібної торгівлі.

Підсумовуючи можна сказати що концепція односторінкових додатків має значні переваги і гарну перспективу розвитку у майбутньому. Це підтверджується значним використанням її при створенні нових веб-сайтів та великою кількістю існуючих. Кількість інструментів, фреймворків та технологій основним використанням яких є розробка односторінкових веб-додатків швидко росте. Це суттєво спрощує створення нових додатків, проте створення великого додатку сьогодні все ще залишається складною задачею.

ПЕРЕЛІК ПОСИЛАНЬ

1. Одностраничные приложения: создание современных адаптивных веб-приложений с помощью ASP.NET. – Режим доступу : http://www.interface.ru/home.asp?artId=35504/. – Дата доступу : 22.05.2016.
2. UNDER THE HOOD OF THE NEW AZURE PORTAL. – Режим доступу : http://jbeckwith.com/2014/09/20/how-the-azure-portal-works/. – Дата доступу : 27.04.2016.
3. Single page apps in depth. – Режим доступу : http://singlepageappbook.com/goal.html. – Дата доступу : 27.04.2016.
4. Що таке SPA або односторінковий портал. – Режим доступу : http://www.calabonga.net/blog/post/141. – Дата доступу : 27.04.2016.
5. Офіційний сайт Knockout. – Режим доступу http://knockoutjs.com/. – Дата доступу : 27.04.2016.
6. Офіційний сайт AngularJS. – Режим доступу https://angularjs.org/– Дата доступу : 27.04.2016.
7. Офіційний сайт jQuery. – Режим доступу https://jquery.com/. – Дата доступу : 27.04.2016.
8. Офіційний сайт Backbone. – Режим доступу http://backbonejs.org/. – Дата доступу : 27.04.2016.
9. Офіційний сайт TypeScript. – Режим доступу https://www.typescriptlang.org/– Дата доступу : 27.04.2016.
10. Seshardi S., AngularJS: Up and Running. // Seshadri S., Green B. - O'Reilly Media, 2014 – 322 с.
11. Williamson K., Learning AngularJS. // Williamson K. - O'Reilly Media, 2015 – 212 с.
12. Gechev M., Switching to Angular 2 // Gechev M. - Packt Publishing, 2016 – 254 с.
13. Herrington J., Learning AngularJS. // Herrington J. - Packt Publishing, 2015 – 235 с.

14. Black C., Building a Single Page Web Application with Knockout.js // Black C., Ly D. - Packt Publishing, 2014 – 152 с.
15. Monteiro F., Learning Single-page Web Application Developmen // Monteiro F. - Packt Publishing, 2014 – 214 с.
16. Козловский П., Разработка веб-приложений с использованием AngularJS // Козловский П., Дарвин П. - ДМК Пресс, 2014 – 394 с.
17. Knol A., Dependency Injection with AngularJS // Knol A. - Packt Publishing, 2015 – с. 78
18. Фримен А., jQuery 2.0 для профессионалов // Фримен А. – Вильямс, 2014 – с. 1040
19. Миковски М., Разработка одностраничных веб-приложений // Миковски М. - ДМК Пресс, 2014 – с. 512

ДОДАТОК А

Лістинг файлів серверної частини:

Лістинг файлу «package.json»:

```
{
  "name": "angular2-quickstart",
  "version": "1.0.0",
  "description": "QuickStart package.json from the documentation, supplemented with testing
support",
  "scripts": {
    "start": "tsc && concurrently \"tsc -w\" \"lite-server\" ",
    "docker-build": "docker build -t ng2-quickstart .",
    "docker": "npm run docker-build && docker run -it --rm -p 3000:3000 -p 3001:3001 ng2-
quickstart",
    "e2e": "tsc && concurrently \"http-server\" \"protractor protractor.config.js\"",
    "lint": "tslint ./app/**/*.ts -t verbose",
    "lite": "lite-server",
    "postinstall": "typings install",
    "test": "tsc && concurrently \"tsc -w\" \"karma start karma.conf.js\"",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "typings": "typings",
    "webdriver:update": "webdriver-manager update"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@angular/common":  "2.0.0-rc.1",
    "@angular/compiler":  "2.0.0-rc.1",
    "@angular/core":  "2.0.0-rc.1",
    "@angular/http":  "2.0.0-rc.1",
    "@angular/platform-browser":  "2.0.0-rc.1",
    "@angular/platform-browser-dynamic":  "2.0.0-rc.1",
    "@angular/router":  "2.0.0-rc.1",
    "@angular/router-deprecated":  "2.0.0-rc.1",
    "@angular/upgrade":  "2.0.0-rc.1",

    "systemjs": "0.19.27",
    "core-js": "^2.4.0",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "zone.js": "^0.6.12",

    "angular2-in-memory-web-api": "0.0.10",
    "bootstrap": "^3.3.6"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.10",
    "typings": "^1.0.4",

    "canonical-path": "0.0.2",
    "http-server": "^0.9.0",
    "tslint": "^3.7.4",
    "lodash": "^4.11.1",
    "jasmine-core": "~2.4.1",
    "karma": "^0.13.22",
    "karma-chrome-launcher": "^0.2.3",
    "karma-cli": "^0.1.2",
```

```
    "karma-htmlfile-reporter": "^0.2.2",
    "karma-jasmine": "^0.3.8",
    "protractor": "^3.3.0",
    "rimraf": "^2.5.2"
  },
  "repository": {}
}
```

Лістинг файлу «tsconfig.json»:

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true
  }
}
```

Лістинг файлу «typings.json»:

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160317120654",
    "jasmine": "registry:dt/jasmine#2.2.0+20160505161446",
    "node": "registry:dt/node#4.0.0+20160509154515"
  }
}
```

Лістинг файлу «systemjs.config.js»:

```
/**
 * System configuration for Angular 2 apps
 * Adjust as necessary for your application needs.
 */
(function(global) {

  // map tells the System loader where to look for things
  var map = {
    'app':                        'app', // 'dist',

    '@angular':                   'node_modules/@angular',
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',
    'rxjs':                       'node_modules/rxjs'
  };

  // packages tells the System loader how to load when no filename and/or no extension
  var packages = {
    'app':                        { main: 'main.js',  defaultExtension: 'js' },
    'rxjs':                       { defaultExtension: 'js' },
    'angular2-in-memory-web-api': { main: 'index.js', defaultExtension: 'js' },
  };

  var ngPackageNames = [
    'common',
    'compiler',
    'core',
    'http',
    'platform-browser',
```

```
    'platform-browser-dynamic',
    'router',
    'router-deprecated',
    'upgrade',
  ];

  // Individual files (~300 requests):
  function packIndex(pkgName) {
    packages['@angular/'+pkgName] = { main: 'index.js', defaultExtension: 'js' };
  }

  // Bundled (~40 requests):
  function packUmd(pkgName) {
    packages['@angular/'+pkgName] = { main: pkgName + '.umd.js', defaultExtension: 'js' };
  };

  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;

  // Add package entries for angular packages
  ngPackageNames.forEach(setPackageConfig);

  var config = {
    map: map,
    packages: packages
  }

  System.config(config);

})(this);
```

Лістинг файлів кліентської частини:

Лістинг файлу «index.html»:

```
<!DOCTYPE html>
<html>
  <head>
    <script>document.write('<base href="' + document.location + '" />');</script>
    <title>Angular 2 Tour of Heroes</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="styles.css">

    <!-- Polyfill(s) for older browsers -->
    <script src="https://npmcdn.com/core-js/client/shim.min.js"></script>

    <script src="https://npmcdn.com/zone.js@0.6.12?main=browser"></script>
    <script src="https://npmcdn.com/reflect-metadata@0.1.3"></script>
    <script src="https://npmcdn.com/systemjs@0.19.27/dist/system.src.js"></script>

    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

Лістинг файлу «main.ts»:

```
import { HTTP_PROVIDERS } from '@angular/http';

import { AppComponent }   from './app.component';

/*
bootstrap(AppComponent, [ HTTP_PROVIDERS ]);
 */
bootstrap(AppComponent, [
    HTTP_PROVIDERS,
    provide(XHRBackend, { useClass: InMemoryBackendService }), // in-mem server
    provide(SEED_DATA,  { useClass: InMemoryDataService })     // in-mem server data
]);
```

Лістинг файлу «app.component.ts»:

```
import { Component } from '@angular/core';
import { RouteConfig, ROUTER_DIRECTIVES, ROUTER_PROVIDERS } from '@angular/router-
deprecated';

import { DashboardComponent }  from './dashboard.component';
import { HeroesComponent }     from './heroes.component';
import { HeroDetailComponent } from './hero-detail.component';
import { HeroService }         from './hero.service';

@Component({
  selector: 'my-app',

  template: `
    <h1>{{title}}</h1>
    <nav>
      <a [routerLink]="['Dashboard']">Dashboard</a>
      <a [routerLink]="['Heroes']">Heroes</a>
    </nav>
    <router-outlet></router-outlet>
  `,
  styleUrls: ['app/app.component.css'],
  directives: [ROUTER_DIRECTIVES],
  providers: [
    ROUTER_PROVIDERS,
    HeroService,
  ]
})
@RouteConfig([
  { path: '/dashboard',  name: 'Dashboard',  component: DashboardComponent, useAsDefault:
true },
  { path: '/detail/:id', name: 'HeroDetail', component: HeroDetailComponent },
  { path: '/heroes',     name: 'Heroes',     component: HeroesComponent }
])
export class AppComponent {
  title = 'Tour of Heroes';
}
```

Лістинг файлу «dashboard.component.html»:

```
<h3>Top Heroes</h3>
<div class="grid grid-pad">
  <div *ngFor="let hero of heroes" (click)="gotoDetail(hero)" class="col-1-4">
    <div class="module hero">
      <h4>{{hero.name}}</h4>
    </div>
  </div>
</div>
```

Лістинг файлу «dashboard.component.ts»:

```
import { Component, OnInit } from '@angular/core';
```

```
import { Router }           from '@angular/router-deprecated';

import { Hero }        from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-dashboard',
  templateUrl: 'app/dashboard.component.html',
  styleUrls: ['app/dashboard.component.css']
})
export class DashboardComponent implements OnInit {

  heroes: Hero[] = [];

  constructor(
    private router: Router,
    private heroService: HeroService) {
  }

  ngOnInit() {
    this.heroService.getHeroes()
      .then(heroes => this.heroes = heroes.slice(1,5));
  }

  gotoDetail(hero: Hero) {
    let link = ['HeroDetail', { id: hero.id }];
    this.router.navigate(link);
  }
}
```

Лістинг файлу «hero-detail.component.html»:

```
<div *ngIf="hero">
  <h2>{{hero.name}} details!</h2>
  <div>
    <label>id: </label>{{hero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="hero.name" placeholder="name" />
   </div>
  <button (click)="goBack()">Back</button>
  <button (click)="save()">Save</button>
</div>
```

Лістинг файлу «hero-detail.component.ts»:

```
import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { RouteParams } from '@angular/router-deprecated';

import { Hero }        from './hero';
import { HeroService } from './hero.service';

@Component({
  selector: 'my-hero-detail',
  templateUrl: 'app/hero-detail.component.html',
  styleUrls: ['app/hero-detail.component.css']
})
export class HeroDetailComponent implements OnInit {
  @Input() hero: Hero;
  @Output() close = new EventEmitter();
  error: any;
  navigated = false; // true if navigated here

  constructor(
    private heroService: HeroService,
    private routeParams: RouteParams) {
```

```
  }

  ngOnInit() {
    if (this.routeParams.get('id') !== null) {
      let id = +this.routeParams.get('id');
      this.navigated = true;
      this.heroService.getHero(id)
          .then(hero => this.hero = hero);
    } else {
      this.navigated = false;
      this.hero = new Hero();
    }
  }
  save() {
    this.heroService
        .save(this.hero)
        .then(hero => {
          this.hero = hero; // saved hero, w/ id if new
          this.goBack(hero);
        })
        .catch(error => this.error = error); // TODO: Display error message
  }
  goBack(savedHero: Hero = null) {
    this.close.emit(savedHero);
    if (this.navigated) { window.history.back(); }
  }
}
```

Лістинг файлу «heroes.component.html»:

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="let hero of heroes" (click)="onSelect(hero)" [class.selected]="hero ===
selectedHero">
    <span class="hero-element">
      <span class="badge">{{hero.id}}</span> {{hero.name}}
    </span>
    <button class="delete-button" (click)="delete(hero, $event)">Delete</button>
  </li>
</ul>

<button (click)="addHero()">Add New Hero</button>
<div *ngIf="addingHero">
  <my-hero-detail (close)="close($event)"></my-hero-detail>
</div>

<div *ngIf="selectedHero">
  <h2>
    {{selectedHero.name | uppercase}} is my hero
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>
```

Лістинг файлу «heros.component.ts»:

```
import { Component, OnInit } from '@angular/core';
import { Router }            from '@angular/router-deprecated';

import { Hero }                from './hero';
import { HeroService }         from './hero.service';
import { HeroDetailComponent } from './hero-detail.component';

@Component({
  selector: 'my-heroes',
  templateUrl: 'app/heroes.component.html',
  styleUrls:  ['app/heroes.component.css'],
```

```
  directives: [HeroDetailComponent]
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;
  addingHero = false;
  error: any;

  constructor(
    private router: Router,
    private heroService: HeroService) { }

  getHeroes() {
    this.heroService
        .getHeroes()
        .then(heroes => this.heroes = heroes)
        .catch(error => this.error = error); // TODO: Display error message
  }

  addHero() {
    this.addingHero = true;
    this.selectedHero = null;
  }

  close(savedHero: Hero) {
    this.addingHero = false;
    if (savedHero) { this.getHeroes(); }
  }

  delete(hero: Hero, event: any) {
    event.stopPropagation();
    this.heroService
        .delete(hero)
        .then(res => {
          this.heroes = this.heroes.filter(h => h !== hero);
          if (this.selectedHero === hero) { this.selectedHero = null; }
        })
        .catch(error => this.error = error); // TODO: Display error message
  }

  ngOnInit() {
    this.getHeroes();
  }

  onSelect(hero: Hero) {
    this.selectedHero = hero;
    this.addingHero = false;
  }

  gotoDetail() {
    this.router.navigate(['HeroDetail', { id: this.selectedHero.id }]);
  }
}
```

Лістинг файлу «heroes.ts»:

```
export class Hero {
  id: number;
  name: string;
}
```

Лістинг файлу «hero.service.ts»:

```
import { Injectable }    from '@angular/core';
import { Headers, Http } from '@angular/http';
```

```
import 'rxjs/add/operator/toPromise';

import { Hero } from './hero';

@Injectable()
export class HeroService {

  private heroesUrl = 'app/heroes';  // URL to web api

  constructor(private http: Http) { }

  getHeroes(): Promise<Hero[]> {
    return this.http.get(this.heroesUrl)
               .toPromise()
               .then(response => response.json().data)
               .catch(this.handleError);
  }

  getHero(id: number) {
    return this.getHeroes()
               .then(heroes => heroes.filter(hero => hero.id === id)[0]);
  }

  save(hero: Hero): Promise<Hero>  {
    if (hero.id) {
      return this.put(hero);
    }
    return this.post(hero);
  }

  delete(hero: Hero) {
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');

    let url = `${this.heroesUrl}/${hero.id}`;

    return this.http
               .delete(url, headers)
               .toPromise()
               .catch(this.handleError);
  }

  // Add new Hero
  private post(hero: Hero): Promise<Hero> {
    let headers = new Headers({
      'Content-Type': 'application/json'});

    return this.http
               .post(this.heroesUrl, JSON.stringify(hero), {headers: headers})
               .toPromise()
               .then(res => res.json().data)
               .catch(this.handleError);
  }

  // Update existing Hero
  private put(hero: Hero) {
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');

    let url = `${this.heroesUrl}/${hero.id}`;

    return this.http
               .put(url, JSON.stringify(hero), {headers: headers})
               .toPromise()
```

```
              .then(() => hero)
              .catch(this.handleError);
  }

  private handleError(error: any) {
    console.error('An error occurred', error);
    return Promise.reject(error.message || error);
  }
}
```

Лістинг файлу «in-memory-data.service.ts»:

```
export class InMemoryDataService {
  createDb() {
    let heroes = [
      {id: 11, name: 'Mr. Nice'},
      {id: 12, name: 'Narco'},
      {id: 13, name: 'Bombasto'},
      {id: 14, name: 'Celeritas'},
      {id: 15, name: 'Magneta'},
      {id: 16, name: 'RubberMan'},
      {id: 17, name: 'Dynama'},
      {id: 18, name: 'Dr IQ'},
      {id: 19, name: 'Magma'},
      {id: 20, name: 'Tornado'}
    ];
    return {heroes};
  }
}
```

Лістинг файлів стилів css:

Лістинг файлу «styles.css»:

```
/* Master Styles */
h1 {
    color: #369;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 250%;
}

h2, h3 {
    color: #444;
    font-family: Arial, Helvetica, sans-serif;
    font-weight: lighter;
}

body {
    margin: 2em;
}

body, input[text], button {
    color: #888;
    font-family: Cambria, Georgia;
}

a {
    cursor: pointer;
    cursor: hand;
}

button {
    font-family: Arial;
    background-color: #eee;
    border: none;
    padding: 5px 10px;
```

```
    border-radius: 4px;
    cursor: pointer;
    cursor: hand;
}

    button:hover {
        background-color: #cfd8dc;
    }

    button:disabled {
        background-color: #eee;
        color: #aaa;
        cursor: auto;
    }

/* Navigation link styles */
nav a {
    padding: 5px 10px;
    text-decoration: none;
    margin-top: 10px;
    display: inline-block;
    background-color: #eee;
    border-radius: 4px;
}

    nav a:visited, a:link {
        color: #607D8B;
    }

    nav a:hover {
        color: #039be5;
        background-color: #CFD8DC;
    }

    nav a.router-link-active {
        color: #039be5;
    }

/* items class */
.items {
    margin: 0 0 2em 0;
    list-style-type: none;
    padding: 0;
    width: 24em;
}

    .items li {
        cursor: pointer;
        position: relative;
        left: 0;
        background-color: #EEE;
        margin: .5em;
        padding: .3em 0;
        height: 1.6em;
        border-radius: 4px;
    }

        .items li:hover {
            color: #607D8B;
            background-color: #DDD;
            left: .1em;
        }

        .items li.selected:hover {
```

```
            background-color: #BBD8DC;
            color: white;
        }

    .items .text {
        position: relative;
        top: -3px;
    }

.items {
    margin: 0 0 2em 0;
    list-style-type: none;
    padding: 0;
    width: 24em;
}

    .items li {
        cursor: pointer;
        position: relative;
        left: 0;
        background-color: #EEE;
        margin: .5em;
        padding: .3em 0;
        height: 1.6em;
        border-radius: 4px;
    }

        .items li:hover {
            color: #607D8B;
            background-color: #DDD;
            left: .1em;
        }

        .items li.selected {
            background-color: #CFD8DC;
            color: white;
        }

            .items li.selected:hover {
                background-color: #BBD8DC;
            }

    .items .text {
        position: relative;
        top: -3px;
    }

    .items .badge {
        display: inline-block;
        font-size: small;
        color: white;
        padding: 0.8em 0.7em 0 0.7em;
        background-color: #607D8B;
        line-height: 1em;
        position: relative;
        left: -1px;
        top: -4px;
        height: 1.8em;
        margin-right: .8em;
        border-radius: 4px 0 0 4px;
    }

/* everywhere else */
* {
```

```
    font-family: Arial, Helvetica, sans-serif;
}
```

Лістинг файлу «app.component.css»:

```
h1 {
    font-size: 1.2em;
    color: #999;
    margin-bottom: 0;
}

h2 {
    font-size: 2em;
    margin-top: 0;
    padding-top: 0;
}

nav a {
    padding: 5px 10px;
    text-decoration: none;
    margin-top: 10px;
    display: inline-block;
    background-color: #eee;
    border-radius: 4px;
}

    nav a:visited, a:link {
        color: #607D8B;
    }

    nav a:hover {
        color: #039be5;
        background-color: #CFD8DC;
    }

    nav a.router-link-active {
        color: #039be5;
    }
```

Лістинг файлу «dashboard.component.css»:

```
[class*='col-'] {
    float: left;
}

*, *:after, *:before {
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}

h3 {
    text-align: center;
    margin-bottom: 0;
}

[class*='col-'] {
    padding-right: 20px;
    padding-bottom: 20px;
}

    [class*='col-']:last-of-type {
        padding-right: 0;
```

```
    }

.grid {
    margin: 0;
}

.col-1-4 {
    width: 25%;
}

.module {
    padding: 20px;
    text-align: center;
    color: #eee;
    max-height: 120px;
    min-width: 120px;
    background-color: #607D8B;
    border-radius: 2px;
}

h4 {
    position: relative;
}

.module:hover {
    background-color: #EEE;
    cursor: pointer;
    color: #607d8b;
}

.grid-pad {
    padding: 10px 0;
}

    .grid-pad > [class*='col-']:last-of-type {
        padding-right: 20px;
    }

@media (max-width: 600px) {
    .module {
        font-size: 10px;
        max-height: 75px;
    }
}

@media (max-width: 1024px) {
    .grid {
        margin: 0;
    }

    .module {
        min-width: 60px;
    }
}
```

Лістинг файлу «heroes.component.css»:

```
.selected {
    background-color: #CFD8DC !important;
    color: white;
}

.heroes {
    margin: 0 0 2em 0;
```

```
    list-style-type: none;
    padding: 0;
    width: 15em;
}

    .heroes li {
        cursor: pointer;
        position: relative;
        left: 0;
        background-color: #EEE;
        margin: .5em;
        padding: .3em 0;
        height: 1.6em;
        border-radius: 4px;
    }

        .heroes li:hover {
            color: #607D8B;
            background-color: #DDD;
            left: .1em;
        }

        .heroes li.selected:hover {
            background-color: #BBD8DC !important;
            color: white;
        }

    .heroes .text {
        position: relative;
        top: -3px;
    }

    .heroes .badge {
        display: inline-block;
        font-size: small;
        color: white;
        padding: 0.8em 0.7em 0 0.7em;
        background-color: #607D8B;
        line-height: 1em;
        position: relative;
        left: -1px;
        top: -4px;
        height: 1.8em;
        margin-right: .8em;
        border-radius: 4px 0 0 4px;
    }

button {
    font-family: Arial;
    background-color: #eee;
    border: none;
    padding: 5px 10px;
    border-radius: 4px;
    cursor: pointer;
    cursor: hand;
}

    button:hover {
        background-color: #cfd8dc;
    }
```

Лістинг файлу «hero-detail.component.css»:

```
label {
  display: inline-block;
  width: 3em;
  margin: .5em 0;
  color: #607D8B;
  font-weight: bold;
}
input {
  height: 2em;
  font-size: 1em;
  padding-left: .4em;
}
button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer; cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #ccc;
  cursor: auto;
}
```