Національний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

**КУРСОВА РОБОТА**

з дисципліни «Об'єктно-орієнтоване програмування»

Тема: «Розробка кулінарного веб-сайту для пошуку, перегляду та додавання рецептів»

Студента \_\_2\_\_ курсу АІ-222 групи

Спеціальності 122 – «Комп’ютерні науки»

\_\_\_\_\_\_\_\_\_Елькамалі І. Т.

(прізвище та ініціали)

Керівник ст.викл, к.т.н. Годовіченко М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Одеса – 2024

ЗМІСТ

[ЗАВДАННЯ НА КУРСОВУ РОБОТУ 3](#_Toc165974793)

[ВСТУП 6](#_Toc165974794)

[1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО КЛАСИ ТА ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ В JAVASCRIPT 7](#_Toc165974795)

[1.1 Теоретичні відомості 7](#_Toc165974796)

[2 ПРОГРАМНА РЕАЛІЗАЦІЯ ООП У ПРОЕКТІ 8](#_Toc165974797)

[2.1 Програмна реалізація ООП 8](#_Toc165974798)

[3 ІНСТРУКЦІЯ КОРИСТУВАЧА 19](#_Toc165974799)

[3.1 Інструкція користувача 19](#_Toc165974800)

[ВИСНОВКИ 23](#_Toc165974801)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 24](#_Toc165974802)

[ДОДАТОК А 25](#_Toc165974803)

Національний університет «Одеська політехніка»

Інститут комп'ютерних систем

Кафедра інформаційних систем

ЗАВДАННЯ НА КУРСОВУ РОБОТУ

1. Розробка інтерфейсу для пошуку рецептів за назвою, інгредієнтами або категоріями.

2. Перегляд рецептів з можливістю відображення детальної інформації про кожен рецепт.

3. Додавання нових рецептів користувачами.

4. Зберігання рецептів та їх характеристик у локальному сховищі.

5. Реалізація можливості додавання рецептів до обраного для подальшого перегляду.

6. Реалізація пагінації для навігації між списками рецептів.

7. Взаємодія з локальним сховищем для ефективного зберігання та обробки інформації.

8. Тестування та відлагодження різних функцій для забезпечення коректної роботи веб-сайту.

АНОТАЦІЯ

У веб-додатку Forkify реалізована архітектура Model-View-Controller (MVC) для зручного та ефективного управління логікою програми та її представленням. Модель включає в себе дані про рецепти та виготовлені страви. Представлення відповідає за відображення цих даних та взаємодію з користувачем через інтерфейс. Контролер координує взаємодію між моделлю та представленням, обробляючи вхідні дані та виконуючи необхідні дії. Окрім цього, веб-сайт Forkify реалізований з використанням об'єктно-орієнтованого підходу (ООП) на мові JavaScript для покращення читабельності, підтримки та розширюваності коду. Це дозволяє легко управляти складними взаємозв'язками між об'єктами та забезпечити модульність та повторне використання коду. У веб-додатку Forkify реалізовані функції, що дозволяють користувачам шукати, переглядати, зберігати рецепти та створювати свої рецепти. Забезпечено інтерактивність із користувачем через взаємодію з інтерфейсом та можливість редагування рецептів.

ABSTRACT

This term paper presents the architecture of the Forkify website, showcasing the Model-View-Controller (MVC) approach and Object-Oriented Programming (OOP) principles implemented in JavaScript. The website serves as a platform for managing recipes and cooking enthusiasts' interactions. The MVC architecture separates the application into three interconnected components:

Model: The model encompasses the data structure of recipes, storing information such as recipe name, ingredients, cooking instructions, and user interactions.  
View: The view component handles the presentation layer, responsible for rendering the user interface elements and displaying recipe information to users in an intuitive and visually appealing manner. Controller: Acting as an intermediary between the model and the view, the controller manages user input and application logic. It handles user requests, processes data from the model, and updates the view accordingly.

Additionally, the website employs Object-Oriented Programming (OOP) principles in JavaScript to enhance code readability, maintainability, and scalability. This approach facilitates the management of complex interactions between objects and promotes code modularity and reusability.

ВСТУП

Цільова мета курсової роботи полягає у поглибленні та закріпленні знань, які студенти здобули протягом курсу «Об'єктно-орієнтоване програмування», а також в розвитку навичок з вибору оптимального представлення початкових даних та вдосконаленні технік використання засобів тестування і налагодження програми. Для досягнення цієї мети використані технології, такі як JavaScript, для розробки веб-сайту нашого проекту Forkify, а також Vite для забезпечення ефективної розробки та швидкого розгортання.

Динамічні структури даних виконують ключову роль у розробці програмного продукту. Вони дозволяють ефективно зберігати та обробляти інформацію, змінюючи свій розмір та склад в залежності від потреб. У нашому веб-додатку Forkify ми використовуємо різноманітні динамічні структури даних, такі як об’єкти та масиви, для зберігання рецептів, інгредієнтів та інших корисних даних. Це дозволяє нам ефективно керувати інформацією та забезпечити зручний інтерфейс для користувачів.

Особливості програмування динамічних структур даних варіюються в залежності від вибраного підходу програмування. У нашому випадку ми використовуємо об'єктно-орієнтований підхід, де динамічні структури даних реалізовані у вигляді класів та об'єктів. Цей підхід дозволяє забезпечити чітку структуру програми та полегшує роботу з даними.

Основні етапи розробки веб-додатку з використанням об'єктно-орієнтованого підходу включають аналіз вимог, проектування архітектури програми, реалізацію коду, тестування та впровадження продукту. Після впровадження, програмний продукт може потребувати підтримки та покращень залежно від зворотного зв'язку користувачів. Важливо також грамотно оформити документацію на програмний продукт, що є складовою успішної розробки та підтримки веб-додатку.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО КЛАСИ ТА ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ В JAVASCRIPT

* 1. Теоретичні відомості

JavaScript підтримує об'єктно-орієнтовану парадигму програмування, яка базується на концепції об'єктів та класів.

* Класи. Класи в JavaScript визначають нові типи даних та їх структуру. Вони діють як шаблони для створення об'єктів.
* Об'єкти. Об'єкти є конкретними екземплярами класів. Вони містять дані та методи, описані в класі.
* Наслідування. Наслідування дозволяє створювати нові класи на основі вже існуючих. Підкласи успадковують властивості та методи батьківського класу і можуть додавати свої власні.
* Інкапсуляція, поліморфізм, абстракція. Концепції інкапсуляції, поліморфізму та абстракції дозволяють створювати код, який є більш модульним, гнучким та легко змінюється. Інкапсуляція дозволяє обмежити доступ до певних даних та методів класу. Поліморфізм дозволяє використовувати ті ж імена функцій чи методів в різних контекстах. Абстракція дозволяє приховувати деталі реалізації та пропонувати простий інтерфейс для взаємодії з об'єктами.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ ООП У ПРОЕКТІ

2.1 Програмна реалізація ООП

Ця програма реалізує систему керування інформацією про рецепти. Вона дозволяє користувачу додавати, шукати та виводити дані про різні рецепти з усього світу. Програма використовує MVC архітектуру(рис. 2.1) та класи для зміни даних про рецепти, їх зберігання, створення.

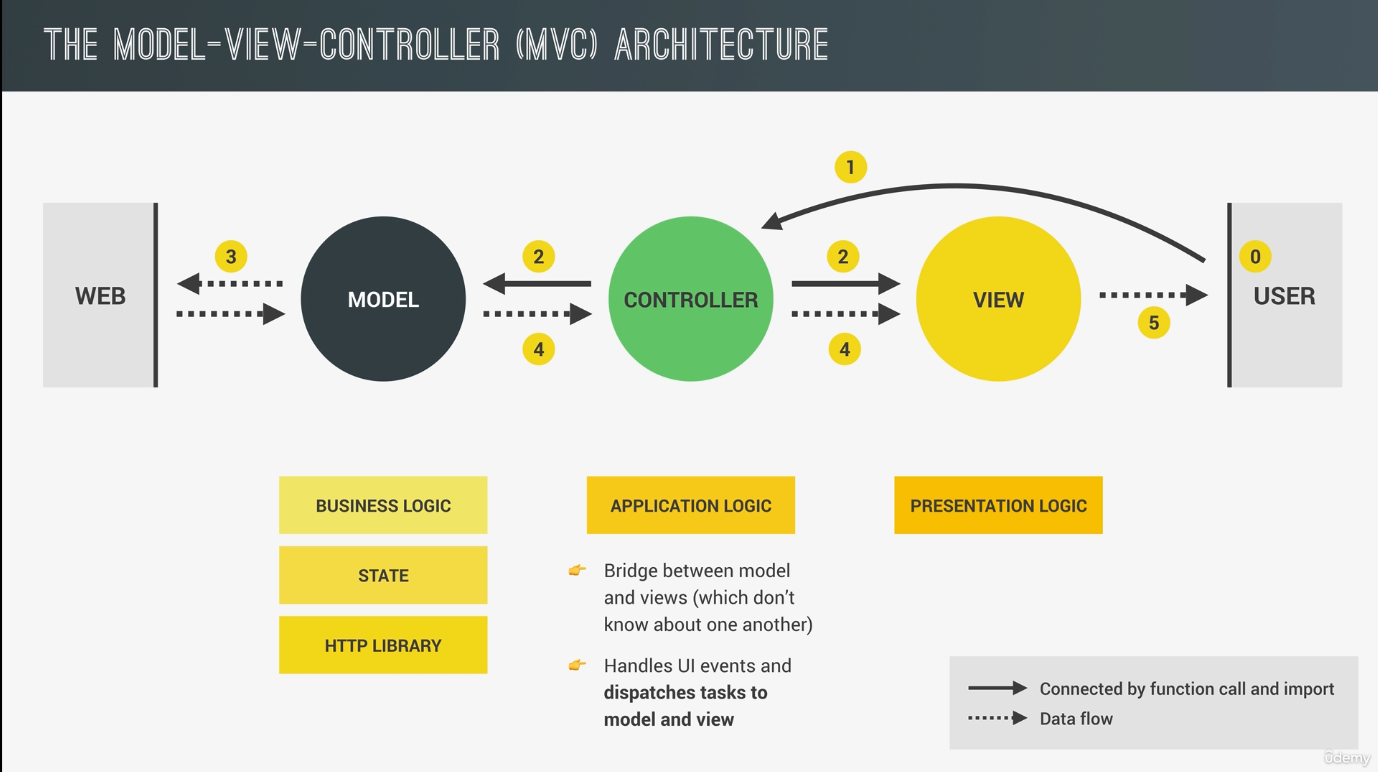


Рисунок 2.1 – MVC ахрітектура

Опис структури Forkify

JS файли

Controller.js: цей файл є головним модулем нашого проекту і відповідає за керування взаємодією користувача з додатком. Основні функції цього файлу включають:

* controlRecipes контролює відображення рецепту. Витягує ідентифікатор рецепту з URL, завантажує рецепт за вказаним ідентифікатором, а потім відображає його на сторінці.
* controlSearchResults контролює відображення результатів пошуку. Отримує запит пошуку від користувача, завантажує відповідні результати та відображає їх на сторінці.
* controlPagination контролює відображення сторінок результатів пошуку. Відображає нові результати пошуку на вказаній сторінці та оновлює пагінацію.
* controlServings контролює кількість порцій в рецепті. Оновлює дані про порції в стані програми та оновлює відповідне відображення рецепту.
* controlAddBookmark контролює додавання або видалення рецепту з закладок. Додає або видаляє рецепт зі списку закладок та оновлює відповідне відображення рецепту та списку закладок.
* controlBookmarks контролює відображення списку закладок. Відображає список закладок на сторінці.
* controlAddRecipe контролює додавання нового рецепту. Отримує дані нового рецепту від користувача, завантажує їх на сервер, оновлює відповідне відображення рецепту та списку закладок та закриває модальне вікно.

Файл також містить функцію `init`, яка ініціалізує додаток, додаючи обробники подій до різних елементів сторінки.

model.js: цей файл містить різноманітні функції, які керують станом додатка та взаємодію з API для отримання рецептів та виконання інших операцій. Основні функції включають:

* loadRecipe завантажує рецепт за заданим ідентифікатором з використанням API та створює об'єкт рецепту.
* loadSearchResults завантажує результати пошуку за заданим запитом з використанням API та зберігає їх у стані додатка.
* getSearchResultsPage повертає певну сторінку результатів пошуку відповідно до вказаного номера сторінки.
* updateServings оновлює кількість порцій у рецепті та відповідні кількості інгредієнтів.
* addBookmark додає рецепт до списку закладок та зберігає цей список.
* deleteBookmark видаляє рецепт зі списку закладок та зберігає зміни.
* uploadRecipe завантажує новий рецепт на сервер та зберігає його в стані додатка.

Крім того, файл також містить функцію init, яка ініціалізує додаток, завантажуючи список закладок з локального сховища браузера.

helpers.js: цей файл містить функцію AJAX, яка виконує асинхронний HTTP-запит до сервера з використанням методу fetch. Функція може приймати URL запиту та додаткові дані для завантаження.

* timeout – допоміжна функція, яка повертає обіцянку, що відкидається після вказаного тайм-ауту. Ця функція використовується для обробки тайм-ауту під час виконання запиту.
* AJAX – основна функція, яка виконує HTTP-запит до сервера. Вона використовується для виконання запитів як методом GET, так і POST. У разі виконання успішного запиту, функція повертає дані в форматі JSON. У випадку помилки під час виконання запиту, генерується виключення з відповідним повідомленням.

Крім того, файл використовує константу TIMEOUT\_SEC з файлу конфігурації для встановлення тайм-ауту запиту.

config.js: у цьому файлі зберігаються константи, необхідні для налаштування та функціонування додатка. Ось їх короткий опис:

* API\_URL: URL-адреса API, до якої відбувається доступ для отримання рецептів та інших даних про їжу.
* TIMEOUT\_SEC: час у секундах, який використовується для встановлення тайм-ауту запиту до API. Якщо запит займає більше цього часу, він буде відкинуто.
* RES\_PER\_PAGE: кількість результатів пошуку, які повинні відображатись на одній сторінці.
* KEY: ключ API, необхідний для аутентифікації та взаємодії з API. Він ідентифікує користувача та дозволяє отримувати доступ до даних.
* MODAL\_CLOSE\_SEC: час у секундах, через який модальне вікно автоматично закриється після виконання певної дії або відображення повідомлення.

Ці константи використовуються в інших частинах додатка для встановлення параметрів, виконання запитів до API та визначення поведінки окремих елементів інтерфейсу користувача.

Views Folder

View.js: цей файл містить клас View, який є базовим класом для всіх представлень у додатку. Ось короткий опис його методів та їхнього призначення:

* render(data, render = true): Метод для рендерингу HTML-коду з даними у вказаному батьківському елементі. Приймає дані для рендерингу та параметр render, який вказує, чи потрібно виконати рендеринг. Повертає розмітку або нічого, якщо render встановлено в false.
* update(data): метод для оновлення існуючого HTML-коду з новими даними. Приймає нові дані та виконує оновлення елементів, що вже існують на сторінці.
* \_clear(): приватний метод для очищення вмісту батьківського елементу перед рендерингом нового вмісту.
* renderSpinner(): метод для рендерингу спіннера, який відображається під час завантаження даних.
* renderError(message = this.\_errorMessage): метод для рендерингу повідомлення про помилку. Приймає опціональний параметр message з текстом помилки.
* renderMessage(message = this.\_message): метод для рендерингу повідомлення для користувача. Приймає опціональний параметр message з текстом повідомлення.

Цей клас дозволяє легко взаємодіяти з DOM, оновлювати вміст сторінки та відображати різні повідомлення та стани за допомогою методів рендерингу

SearchView.js: у цьому файлі описаний клас SearchView, який відповідає за відображення та взаємодію зі сторінкою пошуку в додатку. Ось короткий опис його методів та їхнього призначення:

* \_parentEl: приватне поле, яке містить посилання на батьківський елемент, в якому розміщений елемент пошуку.
* getQuery(): метод, який отримує значення текстового поля пошуку, очищає його та повертає значення.
* \_clearInput(): приватний метод, який очищує значення текстового поля пошуку.
* addHandlerSearch(handler): метод, який додає обробник події для подання форми пошуку. При поданні форми відбувається виклик функції обробника, яка передається як аргумент.

Цей клас використовується для взаємодії зі сторінкою пошуку, отримання та обробки даних введеного користувачем запиту.

resultsView.js: у цьому файлі описано клас resultsView, який відповідає за відображення результатів пошуку в додатку. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке містить посилання на батьківський елемент, в якому розміщені результати пошуку.
* \_errorMessage: приватне поле з повідомленням про відсутність результатів пошуку.
* \_message: приватне поле з порожнім повідомленням.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для відображення результатів пошуку. Використовує метод render() з класу previewView для відображення окремих результатів пошуку. Повертає рядок HTML з результатами.

Цей клас наслідує від класу View та використовує методи та властивості цього класу для відображення вмісту сторінки. Експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри

recipeView.js: у цьому файлі описано клас `RecipeView`, який відповідає за відображення даних конкретного рецепту в додатку. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке містить посилання на батьківський елемент, в якому розміщується вміст рецепту.
* \_errorMessage: приватне поле з повідомленням про помилку при завантаженні рецепту.
* \_message: приватне поле з порожнім повідомленням.
* addHandlerRender(handler): метод, який додає обробник подій для відображення рецепту при завантаженні сторінки або зміні хеша URL.
* addHandlerUpdateServings(handler): метод, який додає обробник подій для оновлення кількості порцій рецепту.
* addHandlerAddBookmark(handler): метод, який додає обробник подій для додавання рецепту до закладок.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для відображення даних рецепту. Використовується для відображення зображення рецепту, його заголовку, часу приготування, кількості порцій, складників, інструкцій та кнопок для зміни кількості порцій та додавання до закладок.
* \_generateMarkupIngredients(ing): приватний метод, який генерує HTML-розмітку для відображення складників рецепту. Використовується в методі `\_generateMarkup()` для кожного інгредієнта рецепту.

Цей клас наслідує від класу `View` та використовує методи та властивості цього класу для відображення вмісту сторінки. Експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри.

previewView.js: у цьому файлі описано клас `previewView`, який відповідає за відображення попереднього перегляду рецепту. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке визначає, в якому елементі DOM відображатиметься попередній перегляд рецепту.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для попереднього перегляду рецепту. Використовується для відображення зображення рецепту, його заголовку, видавця та індикатора користувача-автора.

Цей клас також наслідує від класу `View` та використовує його методи та властивості для відображення вмісту сторінки. Експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри.

paginationView.js: у цьому файлі описано клас `paginationView`, який відповідає за відображення сторінок пагінації результатів пошуку. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке визначає, в якому елементі DOM відображатиметься пагінація.
* addHandlerClick(handler): метод, який додає обробник подій для кнопок пагінації. Він очікує клік на кнопці та викликає обробник, передаючи номер сторінки як параметр.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для кнопок пагінації. В залежності від поточної сторінки та кількості сторінок він створює кнопки "назад" і/або "вперед". Якщо поточна сторінка перша або остання, відповідні кнопки відсутні.

Цей клас також наслідує від класу `View` та використовує його методи та властивості для відображення вмісту сторінки. Експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри.

bookmarksView.js: у цьому файлі описано клас `bookmarksView`, який відповідає за відображення списку закладок користувача. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке визначає, в якому елементі DOM буде відображений список закладок.
* addHandlerRender(handler): метод, який додає обробник подій для рендерингу списку закладок. Він очікує подію завантаження сторінки та викликає обробник, щоб оновити список закладок.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для списку закладок. Він використовує метод `render` класу `previewView`, щоб створити HTML-розмітку для кожної закладки у списку. Потім ці розмітки об'єднуються в одну рядок за допомогою методу `join`.

Цей клас також наслідує від класу `View` та використовує його методи та властивості для відображення вмісту сторінки. Експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри.

addRecipeView.js: у цьому файлі описано клас `AddRecipeView`, який відповідає за відображення форми для додавання нового рецепту. Ось короткий опис його методів та їхнього призначення:

* \_parentElement: приватне поле, яке визначає, в якому елементі DOM буде відображена форма додавання рецепту.
* \_message: повідомлення, яке відображається користувачу після успішного завантаження рецепту.
* \_window: посилання на вікно форми додавання рецепту.
* \_overlay: посилання на оверлей, який використовується для закриття форми.
* \_btnOpen: кнопка, яка відкриває форму додавання рецепту.
* \_btnClose: кнопка, яка закриває форму додавання рецепту.
* constructor(): конструктор класу, який викликає конструктор батьківського класу та додає обробники подій для показу та закриття вікна форми.
* toggleWindow(): метод, який перемикає видимість вікна та оверлея.
* \_addHandlerShowWindow(): приватний метод, який додає обробник події для відкриття форми додавання рецепту.
* \_addHandlerHideWindow(): приватний метод, який додає обробники подій для закриття форми додавання рецепту.
* addHandlerUpload(handler): метод, який додає обробник події для відправки форми додавання рецепту. При натисканні на кнопку "Submit", дані з форми конвертуються в об'єкт та передаються у вигляді аргументу обробнику.
* \_generateMarkup(): приватний метод, який генерує HTML-розмітку для форми додавання рецепту. В даному випадку, цей метод залишений пустим, оскільки він не потребує вмісту для рендерингу.

Клас експортується як один екземпляр, що дозволяє використовувати його у інших частинах додатку без необхідності створювати нові екземпляри.

SCSS файли

\_main.scss: це імпорт зовнішніх стилів для різних частин веб-сайту. Він допомагає розділити стилі на логічні групи для кращого керування та організації коду. Наприклад, файли base, components, header, preview, searchResults, recipe та upload можуть містити стилі, пов'язані з основними структурними елементами, компонентами, заголовком, попереднім переглядом, результатами пошуку, рецептом та вікном завантаження відповідно. Такий підхід сприяє модульності та підтримці коду в майбутньому.

\_base.scss: цей CSS-файл визначає кольори, медіа-запити та стилі для веб-сторінки. Він використовує змінні для кольорів, медіа-запити для адаптивності і розміщує контент у контейнері з максимальною шириною 120 рем.

components.scss: цей SCSS-файл містить стилі для кнопок, тексту, індикаторів завантаження та повідомлень про помилки та успішність. Він використовує змінні для кольорів, розмірів кнопок та тексту. Класи задають різноманітні ефекти для кнопок, такі як зміна кольору та розміру при наведенні, обмеження курсору та анімації обертання для індикатора завантаження.

\_header.scss: цей SCSS-файл містить стилі для заголовка, поля пошуку, навігаційного меню та розділу з закладками. Він використовує змінні для кольорів та розмірів, задає ефекти при фокусуванні, наведенні та відкриванні розділу з закладками.

\_preview.scss: цей блок стилів визначає вигляд попереднього перегляду рецепту. Він встановлює стилі для посилання, яке містить зображення рецепту та інформацію про рецепт, таку як назва та видавець. Крім того, він встановлює стилі для випадків, коли рецепт є користувачем згенерованим, щоб позначити це іконкою. Використовуються змінні для кольорів, а також використовується ефект підсвічування при наведенні та анімація переміщення при натисканні.

\_recipe.scss: цей блок стилів визначає вигляд рецепту. Він встановлює стилі для зображення рецепту, назви, інформації про рецепт (такої як час приготування та порції), а також для списку інгредієнтів та інструкцій. Крім того, він встановлює стилі для користувачем згенерованого прапорця та застосовує градієнт для фону назви рецепту.

\_searchResults.scss: ці стилі визначають вигляд результата пошуку, розташування пагінації та нижнього колонтитула на сторінці. Вони встановлюють відступи, колір тексту, розміщення кнопок пагінації та посилання на Twitter у нижньому колонтитулі.

\_upload.scss: ці стилі визначають вигляд модального вікна для додавання рецепту. Вікно розташовується по центру сторінки, має білий фон, заокруглені кути та тінь. Закриття вікна можна здійснити за допомогою кнопки "X" у верхньому правому куті, або натисканням на затемнений фон. Стилі також визначають вигляд полів для введення даних, які розміщуються у дві колонки та мають певні властивості стилю, такі як розмір шрифту, колір тексту та фону, а також ефекти при фокусуванні на полі вводу.

Інші файли:

index.html: вихідний HTML файл, який відображає основну структуру нашого сайту та підключає JS та CSS файли.

package.json: файл конфігурації проекту, який містить інформацію про залежності та скрипти для запуску проекту.

node\_modules: каталог, де зберігаються всі залежності проекту, включаючи Parcel та core-js.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Інструкція користувача

Програма реалізує пошук та додовання рецептів, використовує localStorage для збереження помічених рецептів (рис. 3.1).

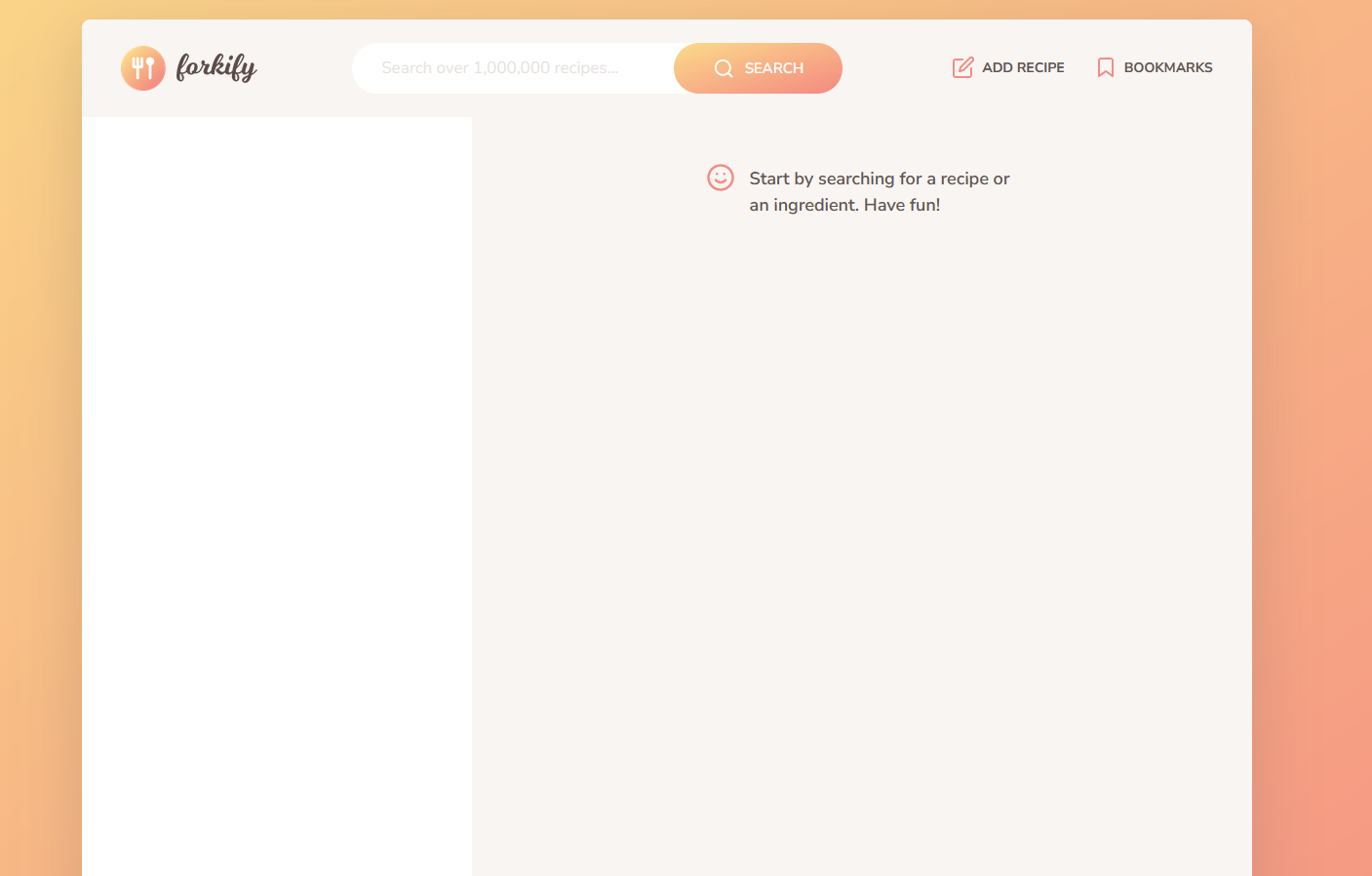


Рисунок 3.1 – Меню програми

Далі можемо вбити у пошук страву чи інгрідієнт з яким ми бажаємо щось подивитись та побачити результат (рис. 3.2-3.3).



Рисунок 3.2 – Вводимо у пошук бажаний рецепт

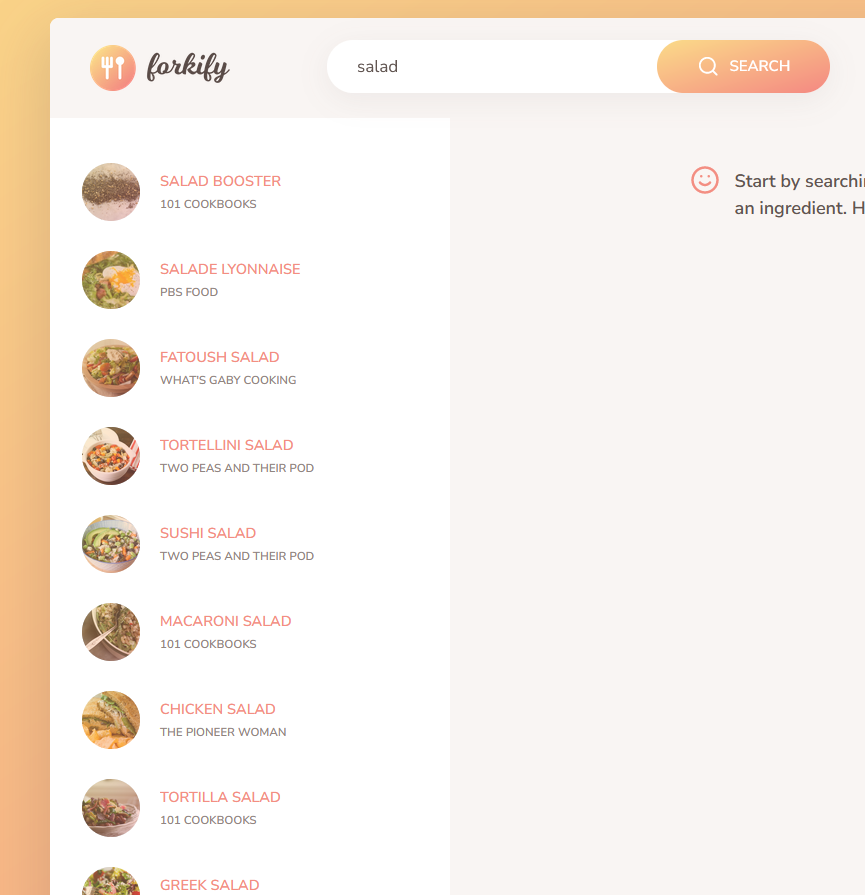
****

Рисунок 3.3 – Результати пошуку

Користувач може переходити між рецептами та дивитись усі можливі рецепти для цієї страви (рис. 3.4).

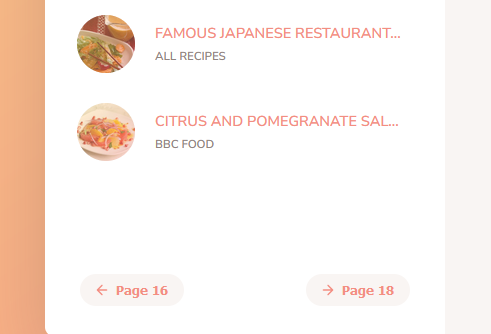
****

Рисунок 3.4 – Можливість переходити між рецептами

Обравши страву бачимо назву, час приготування, кількість осіб на яку розрахований рецепт,інгрідієнти та кнопка, яка посилається на відео рецепту (рис. 3.5).

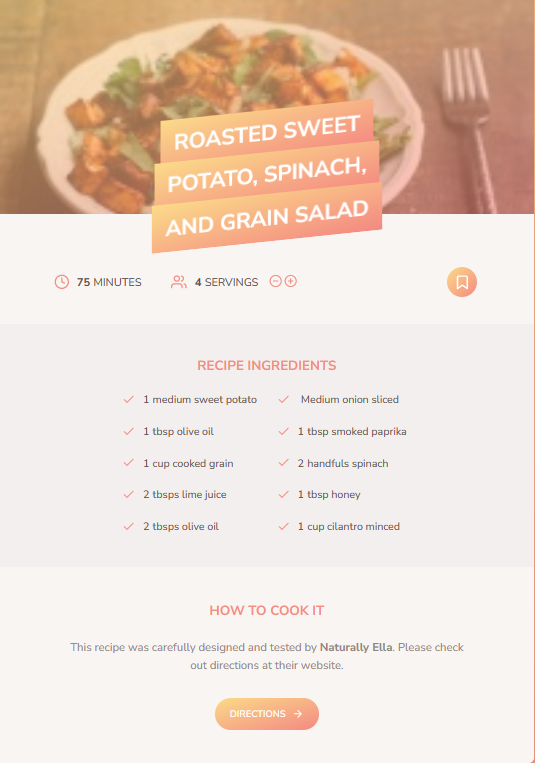
****

Рисунок 3.5 – Сторінка з детальною інформацією про страву та її приготування

При бажані користувач може змінити кількість осіб, на яких розрахований рецепт (рис. 3.6).

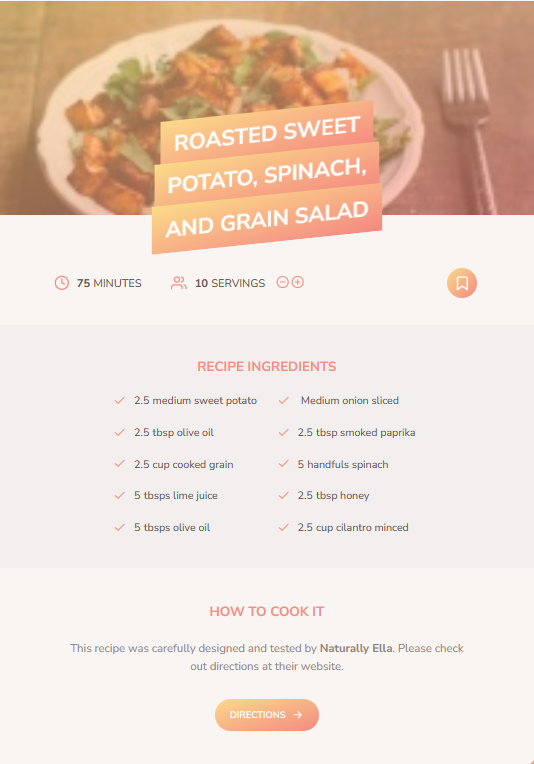
****

Рисунок 3.6 – Зміна кількості осіб на 10

Також можна позначити рецепт та він збережеться у нас в закладках, при повторному відвідувані сторінці усі помічені рецепти залишаються (рис. 3.7-3.8).



Рисунок 3.7 – Кнопка для збереження рецептів

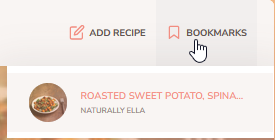


Рисунок 3.8 – Збережені рецепти у закладках

Користувач може додати свій рецепт (рис. 3.9).

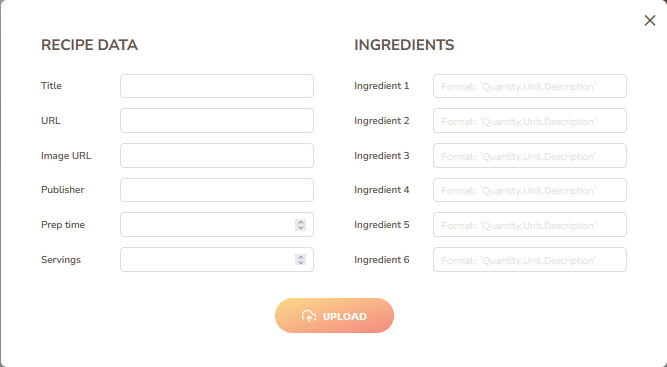


Рисунок 3.9 – Можливість додавання рецепту

ВИСНОВКИ

У ході роботи було розроблено програму для роботи з рецептами, з можливістю додавання, пошуку, виділення та перегляду рецептів. Завдання виконано в повному обсязі, всі функції було реалізовано і протестовано.

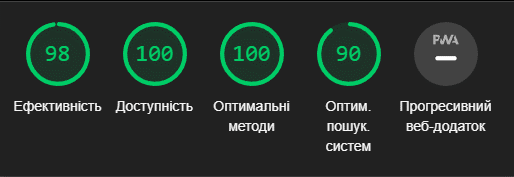
Веб-сайт використовує підхід до структурування CSS, розділяючи його на логічні частини за допомогою імпорту різних файлів стилів. Це полегшує управління та розробку, оскільки кожен файл може містити стилі, які відповідають за певний функціонал або частину веб-сайту.

CSS використовує змінні для стандартизації кольорів, розмірів і інших стилів, що полегшує їх зміну та підтримку.Є використання медіа-запитів для реагування на різні розміри екрану та забезпечення адаптивного дизайну.

Веб-сайт має інтерактивні компоненти, оскільки в коді згадується обробник подій та функції, які додаються до різних елементів. Частини коду, такі як `addRecipeView`, `bookmarksView`, `paginationView` та `previewView`, можуть вказувати на те, що веб-сайт використовує динамічне завантаження контенту або взаємодію з бекендом для отримання та відображення даних. Присутній код для модального вікна, що використовується для додавання нового рецепту, що свідчить про можливість користувачам взаємодіяти з сайтом та додавати власний контент. Наявні компоненти для розбиття контенту на сторінки та навігації між ними, що полегшує навігацію користувачам, особливо при наявності великої кількості даних.

В цілому, веб-сайт має високий рівень функціональності та структури, що дозволяє забезпечити зручну та ефективну взаємодію з користувачем

Швидкодія вебсайту:



ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

* 1. Vite, URL: <https://vitejs.dev/>
  2. OOP Javascript, URL: <https://www.freecodecamp.org/news/object-oriented-programming-javascript/>
  3. Parcel, URL: <https://parceljs.org/>
  4. Core-js, URL: <https://www.npmjs.com/package/core-js>

ДОДАТОК А

Увесь код програми можна подивитись за посиланням:

<https://github.com/Marve10s/Forkify>