МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Веб-сайт для парсингу даних cookies і http-headers на основі JavaScript фреймворку Angular"

Виконавець: Теплуха Сергій Олександрович

Керівник: д.т.н., проф. Воронін Альберт Миколайович

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології".

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Теплухи Сергія Олександровича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Веб-сайт для парсингу даних cookies і http-headers на основі JavaScript фреймворку Angular» затверджена наказом ректора від 12.10.2021 за № 2228/ст.

2. Термін виконання роботи: з 12.10.2021 по 31.12.2021.

3. Вихідні дані до роботи: теоретичні відомості та основи програмування сайтів, проектування та розробка сайту на основі веб фреймворку.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, аналіз можливих методів та засобів для розробки пропонованого продукту, план, структура та розробка продукту, висновок..

5. Перелік обов'язкового ілюстративного матеріалу: слайди, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт.	12.10.2021 – 15.10.2021	
2.	Аналіз існуючих методів та інструментів для створення застосунку.	16.10.2021 – 19.10.2021	
3.	Проаналізувати літературу та джерела за темою дипломного проекту.	20.10.2021 – 24.10.2021	
4.	Проектування та розробка дизайну веб додатку.	25.10.2021 – 31.10.2021	
5.	Написання Розділу 1 дипломної роботи.	01.11.2021 – 07.11.2021	
6.	Обрання всіх можливих інструментів та програм для виконання задач. Написання Розділу 2 дипломної роботи.	08.11.2021 – 17.11.2021	
7.	Написання Розділу 3 дипломної роботи. Створення програмної частини проекту. Завершення створення пояснювальної записки дипломної роботи.	18.11.2021 – 01.12.2021	
8.	Оформлення та друк пояснювальної записки.	02.12.2021 – 11.12.2021	
9.	Створення презентації, доповіді та підготовка до захисту дипломної роботи	12.12.2021 – 20.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи _____ Альберт ВОРОНІН
(підпис керівника)

Завдання прийняв до виконання _____ Сергій ТЕПЛУХА
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Веб-сайт для парсингу даних cookies і http-headers на основі JavaScript фреймворку Angular” складається із вступу, трьох розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел і містить 94 сторінки тексту, 40 рисунків та 1 таблицю. Список використаних джерел містить 16 найменувань.

Метою дипломної роботи є розгляд теоретичних та практичних засад створення веб сайту для полегшення роботи з cookies і http-headers даними.

Предметом дослідження є програмні та теоретичні інструменти, які використовуються при створення, основним предметом дослідження є розробка методів та функцій на базі даного фреймворку та його інструментів.

Об’єктом дослідження є фронтенд та структура сайту, його технології та інструменти.

Ключові слова: ФРОНТЕНД, ВЕБ-САЙТ, AOT, JIT, HTML, JAVASCRIPT, ANGULAR, ANGULARJS, CSS, FIGMA, БРАУЗЕР, ДОПОМІЖНІ БІБЛІОТЕКИ, COOKIE, LOCAL STORAGE, SESSION STORAGE, USER INTERFACE, SINGLE PAGE APLICATION, ПАРСИНГ САЙТУ, РОЗДІЛЕННЯ ДАНИХ.

.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Фронтенд – клієнтська сторона WEB-додатку.

JS – Java Script (мова програмування).

HTML – HyperText Markup Language (мова тегів).

CSS – Cascading Style Sheets (мова стилю сторінок).

WEB – додаток – додаток, в якому клієнтом є оглядач Інтернету.

RxJS – Reactive Extensions для JavaScript.

Cookie – фрагемент даних.

LS – local storage (локальне сховище).

SS – session storage.

UI – user interface.

AOT – ahead-of-time compilation.

JIT – just-in-time compilation.

ВСТУП

В наш час існує велика кількість різних додатків, систем, сайтів та бібліотек для полегшення роботи з мовами програмування, фреймворками, даними та інструментами для роботи в мережі. Більшість з яких є платні і не дають потрібного функціонала для веб-розробника.

Сайт — сукупність веб-сторінок, які є доступні у мережі Інтернет та використовують одне доменне ім'я.

Для полегшення роботи з cookie і http-headers є спеціальні інструменти які створюють необхідні умови щоб працювати з ними було не так складно. Прикладом можна вважати розподіл даних на окремі частини (парсинг).

Парсинг - це процес перетворення форматованого тексту в структуру даних. Тип структури даних може бути будь-яким відповідним поданням інформації, вигравіруваної у вихідному тексті.

Фреймворк - це реальна або концептуальна структура, призначена для підтримки чи керівництва для побудови чогось, що розширює структуру на щось корисне. У комп'ютерних системах фреймворк часто є багатошаровою структурою, яка вказує на те, які програми можна або потрібно будувати і як вони будуть взаємопов'язані. Деякі фреймворки комп’ютерної системи також містять актуальні програми, визначають інтерфейси програмування або пропонують інструменти програмування для використання фреймворків.

Angular - це платформа та фреймворк для створення односторінкових клієнтських програм за допомогою HTML та TypeScript. Angular написано в TypeScript. Компоненти визначають подання - це набори елементів екрану, які Angular може вибирати та змінювати відповідно до логіки програми та даних.

Заголовки HTTP дозволяють клієнту та серверу передавати додаткову інформацію із запитом або відповіддю HTTP.

Файли cookie є важливими для сучасного Інтернету, але є вразливими для вашої конфіденційності. Як необхідна частина веб-перегляду, файли cookie допомагають веб-розробникам надавати вам більш зручні відвідування веб-

сайтів . Cookie також використовується для відстеження дій користувачів на веб-сайті. Як правило, це робиться для збору статистики, а рекламні компанії на основі такої статистики формують анонімні профілі користувачів для більш точного націлювання реклами.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАДАЧІ

1.1. Поняття веб-сайту

Веб-сайт - це сукупність загальнодоступних, взаємопов’язаних веб - сторінок, які мають спільне доменне ім’я. Веб-сайти можуть створюватися та підтримуватися окремою особою, групою, бізнесом чи організацією для виконання різноманітних цілей.

Разом загально доступні веб-сайти складають Всесвітню павутину. Хоча це іноді називають «веб-сторінкою», це визначення є хибним, оскільки веб-сайт складається з кількох веб-сторінок. Веб-сайт також відомий як "веб-присутність" або просто "сайт".

Відкрити веб-сайт можна безпосередньо, ввівши його URL-адресу або здійснивши пошук у пошуковій системі.

Спочатку веб-сайти були класифіковані за їх доменами верхнього рівня. Деякі приклади включають:

- Веб-сайти державних установ = .gov;
- Веб-сайти навчальних закладів = .edu;
- Веб-сайти некомерційних організацій = .org;
- Комерційні веб-сайти = .com;
- Інформаційні сайти = .info.

Хоча ці розширення доменів верхнього рівня все ще існують, вони мало говорять про фактичний вміст веб-сайту.

У сучасному Інтернеті розширення ".com" є, безумовно, найпопулярнішим доменом разом з багатьма іншими розширеннями для окремих країн (.it, .de, .co.uk, .fr тощо)[1].

URL розшифровується як Uniform Resource Locator(рис.1.1). URL-адреса - це не що інше, як адреса даного унікального ресурсу в Інтернеті. Теоретично кожна дійсна URL-адреса вказує на унікальний ресурс. Такими ресурсами може бути сторінка HTML, документ CSS, зображення тощо. На практиці є деякі винятки, найпоширенішим є URL-адреса, що вказує на ресурс, якого більше немає або який перемістився. Оскільки ресурс, представлений URL-адресою та самою URL-адресою, обробляється веб-сервером, власнику веб-сервера належить ретельно керувати цим ресурсом та пов’язаною з ним URL-адресою.

Рис.1.1. Доменне ім’я і URL

Доменне ім’я - це рядок тексту, який відображається на цифрову IP-адресу, що використовується для надання доступу до веб сайту з клієнтського програмного забезпечення. Наприклад, доменне ім’я Google - google.com.

1.2. Поняття браузер

Браузер - це програмне забезпечення, яке здійснює доступ та відображає сторінки та файли в Інтернеті. Браузерам потрібно підключення до Інтернету (наприклад, за допомогою кабельного модему, прямого з'єднання Ethernet або Wi-Fi).

Список сучасних інтернет браузерів:

- Google Chrome;
- Microsoft Edge;
- Microsoft Internet Explorer;
- Mozilla Firefox;
- Opera;
- Apple Safari;
- Amazon Silk.

Коли ви вводите у веб-переглядачі адресу веб-сторінки, наприклад www.allaboutcookies.org, ця веб-сторінка повністю не зберігається на сервері, готовому до очікування доставки. Фактично кожна веб-сторінка, яку ви запитуєте, створюється індивідуально у відповідь на ваш запит.

Ви фактично викликаєте список запитів на отримання вмісту з різних каталогів ресурсів або серверів, на яких зберігається вміст цієї сторінки.

Зображення можуть надходити з одного сервера, текстовий вміст з іншого, сценарії, такі як сценарії дати з іншого, і оголошення з іншого. Як тільки ви переходите на іншу сторінку, сторінка, яку ви щойно переглянули, зникає. Це динамічний характер веб -сайтів.

1.3. Мови програмування для веб-додатків

В наш час з’явилась велика кількість різних мов програмування для сайтів, є можливість створювати сайти з використання тільки HTML і CSS, але для більшої динаміки використовують JavaScript чи TypeScript.

1.3.1. Мова програмування JavaScript

JavaScript - це текстова мова програмування, яка використовується як на стороні клієнта, так і на стороні сервера, що дозволяє зробити веб-сторінки інтерактивними. Якщо HTML та CSS - це мови, що надають структуру та стиль веб-сторінкам, JavaScript надає веб-сторінкам інтерактивні елементи, які

залучають користувачів. Поширеними прикладами JavaScript, якими ви можете користуватися щодня, є вікно пошуку на Amazon, відео з новинами, укладене в The New York Times, або оновлення вашої стрічки Twitter.

Включення JavaScript покращує зручність користування веб-сторінкою, перетворюючи її зі статичної сторінки в інтерактивну. Щоб підвести підсумок, JavaScript додає поведінку до веб-сторінок.

JavaScript в основному використовується для веб-програм та веб-браузерів. Але JavaScript також використовується поза Інтернетом у програмному забезпеченні, серверах та вбудованих апаратних засобах управління. Ось деякі основні речі, для яких використовується JavaScript:

1. Додавання інтерактивної поведінки до веб-сторінок

JavaScript дозволяє користувачам взаємодіяти з веб-сторінками. Майже немає обмежень у тому, що ви можете робити з JavaScript на веб-сторінці - це лише кілька прикладів:

- Показати або приховати додаткову інформацію одним натисканням кнопки;
- Змініть колір кнопки, коли миша наводить на неї;
- Прокрутіть карусель зображень на домашній сторінці;
- Збільшення або зменшення зображення;
- Відображення таймера або зворотного відліку на веб-сайті;
- Відтворення аудіо та відео на веб сторінці;
- Відображення анімації;
- Використання випадаючого меню гамбургера.

2. Створення веб та мобільних додатків

Розробники можуть використовувати різні фреймворки JavaScript для розробки та створення веб та мобільних додатків. Фреймворки JavaScript - це колекції бібліотек кодів JavaScript, які надають розробникам заздалегідь написаний код, який можна використовувати для рутинних функцій та завдань програмування-буквально фреймворк для створення веб-сайтів або веб-програм.

Популярні інтерфейсні фреймворки JavaScript включають React, React Native, Angular та Vue. Багато компаній використовують Node.js, середовище виконання JavaScript, побудоване на двигуні JavaScript V8 Google Chrome. Кілька відомих прикладів включають Paypal, LinkedIn, Netflix та Uber.

3. Створення веб-серверів та розробка серверних додатків

Крім веб-сайтів та програм, розробники також можуть використовувати JavaScript для створення простих веб-серверів та розвитку внутрішньої інфраструктури за допомогою Node.js.

4. Розробка ігор

Звичайно, ви також можете використовувати JavaScript для створення браузерних ігор. Це чудовий спосіб для початківців розробників відпрацювати свої навички роботи з JavaScript.

1.3.2. Мова програмування TypeScript

TypeScript - це супернабір JavaScript. TypeScript будується поверх JavaScript. Спочатку ви пишете код TypeScript. Потім ви компілюєте код TypeScript у звичайний код JavaScript за допомогою компілятора TypeScript (рис.1.2).

Отримавши простий код JavaScript, ви можете розгорнути його в будь якому середовищі, де працює JavaScript.

Файли TypeScript використовують розширення .ts, а не .js розширення файлів JavaScript.

Рис 1.2. Компіляція TypeScript

TypeScript використовує синтаксиси JavaScript і додає додаткові синтаксиси для підтримки типів.

Якщо у вас є програма JavaScript, у якій немає синтаксичних помилок, це також програма TypeScript. Це означає, що всі програми JavaScript є програмами TypeScript. Це дуже корисно, якщо ви мігруєте існуючу базу кодів JavaScript на TypeScript.

TypeScript переносить майбутній JavaScript на сьогодні. TypeScript підтримує майбутні функції, заплановані в ES Next для поточних механізмів JavaScript. Це означає, що ви можете використовувати нові функції JavaScript, перш ніж веб -браузери (або інші середовища) повністю їх підтримують. Щороку TC39 випускає кілька нових функцій для ECMAScript, що є стандартом JavaScript.

1.3.3. Різниця між JavaScript і TypeScript

Прив’язка даних.

TypeScript використовує такі поняття, як типи та інтерфейси для опису використовуваних даних. Такої концепції немає у JavaScript.

Екосистема.

У TypeScript екосистема досить потужна та інтуїтивно зрозуміла. Таким чином, він дозволяє статично вводити різні типи ідіоматичних функцій JavaScript, такі як типи об’єднання, перетин, дискримінаційне об’єднання.

Javascript пропонує можливість досліджувати та створювати код без кроку збірки.

Пакет Npm

За допомогою Typescript багато пакетів npm або мають статичні визначення типів, або мають зовнішній, який легко встановити. Javascript пропонує можливість досліджувати та створювати код без кроку збірки.

Прототипування

TypeScript має особливість прототипування. JavaScript не має цієї функції.

Особливості Javascript:

- Це міжплатформова мова;
- Він використовується на стороні клієнта та сервера;
- Легко навчитися і почати;
- Це динамічна мова: гнучка та потужна;
- У вас є "велика свобода" робити все, що завгодно, з будь -яким предметом;
- Сильний робочий процес тестування;
- Додані залежності;
- Framework не підтримується;

Особливості TypeScript:

- Ремонтопридатність;
- Запропонував велику продуктивність для розробників;
- Навігація по коду та запобігання помилкам;
- Код "відкриття" та рефакторинг;
- Додаткові анотації статичного типу / статичне введення;
- Додаткові функції для функцій;
- Підтримує ES6;
- Підтримує інтерфейси, підінтерфейси, класи та підкласи;
- Масштабована розробка на стороні клієнта HTML5;
- Доступна багата IDE з функціями автозаповнення та навігації по коду;

• Об’єктно-орієнтована на класах із успадкуванням приватних членів та інтерфейсів.

1.4. Поняття фреймворку

Фреймворк - це наборна структура, в якій завдання виконуються або завершуються. Як правило, фреймворк відноситься до часто пошарової структури, яка вказує на те, які програми можна або потрібно будувати і як вони будуть з'єднані між собою. В основному, фреймворк працює як своєрідна структура підтримки для того, що потрібно будувати поверх [1].

Фреймворк програмного забезпечення - це абстракція, в якій програмне забезпечення, що забезпечує загальні функціональні можливості, може бути вибірково змінено за допомогою додаткового коду, написаного користувачем.

Рамки інтерфейсу користувача допомагають створювати стилізовані та професійно виглядають веб -програми. Більшість включає в себе якусь систему сітки, щоб вирівнювати елементи, у них є колірні схеми, які обробляються за вас, і вони стилізують ваші HTML -компоненти за допомогою CSS, щоб вони виглядали чисто та професійно. Вони знаходяться в межах зовнішнього домену; однак, зазвичай, коли ми говоримо про фронтенд фреймворки, ми говоримо про фреймворки JavaScript. Деякі рамки інтерфейсу користувача включають:

Bootstrap:

- Дуже відомий і створений Twitter;
- Легко навчається і виглядає професійно;
- "Сайти завантаження" можна легко помітити;
- Налаштування компонентів може бути складним.

Materialize:

- Чистий вигляд;
- Трохи "веселіше", ніж Bootstrap;
- Багато варіантів стилю та кольору;
- Дотримується посібника Google щодо стилю Матеріал.

Вибір основи інтерфейсу користувача здебільшого зводиться до особистих переваг зовнішнього вигляду та цілей сайту. Різні стилі можуть сподобатися різним галузям або можуть передати різне повідомлення користувачам.

Frontend Frameworks.

Фронтенд - фреймворки в більшості випадків написані на JavaScript і призначені для організації функціональних можливостей та інтерактивності вашого веб-сайту. Деякі з них включають:

Vue:

- Легко вчитися;
- Дуже швидко;
- Усі інструменти, пов'язані з ним, добре упаковані;
- Бере частини з Angular та React та оптимізує їх;
- Гнучкий - ви можете використовувати його різними способами.

AngularJS:

- Створено компанією Google;
- Добре підтримується;
- Величезна кількість функцій;
- Покращує масштабованість додатків;
- Важко налагодити;
- Велика крива навчання;

Angular 2+:

- Створено компанією Google;
- Добре підтримується;
- Заохочує багаторазове використання;
- Покращує масштабованість додатків;
- Велика крива навчання.

React:

- Створено Facebook;
- Групує зовнішній код у компоненти;

- Організовує код та дані, щоб зробити код більш багаторазовим;
- Велика крива навчання.

1.5. Види збереження даних в браузері

Сучасні веб-браузери пропонують різні варіанти зберігання даних веб-сайтів у веб-переглядачах користувачів, що дозволяє отримувати ці дані залежно від потреб. Це дозволяє власникам веб-сайтів зберігати дані для тривалого зберігання, зберігати вміст веб-сайту або документи для використання в автономному режимі, зберігати налаштування користувачів, застосовувати стани тощо.

Різні типи сховищ браузера, доступні для зберігання даних веб-сайту:

- Використовує кейси для зберігання в браузері
- Персоналізація параметрів сайту
- Тривала діяльність сайту
- Збереження стану входу
- Зберігання даних та ресурсів локально, щоб сайт швидше завантажувався або використовувався без мережевого підключення
- Збереження документів, створених веб -програмами, локально для використання в автономному режимі
- Покращення продуктивності веб -сайту
- Скорочення запитів до серверних серверів

1.5.1. Cookie

Файли cookie - це, як правило, невеликі текстові файли з ідентифікаційними тегами, які зберігаються у каталозі браузера вашого комп’ютера або в підпапках даних програми.

Файли cookie створюються, коли ви використовуєте веб -переглядач для відвідування веб-сайту, який використовує файли cookie для відстеження ваших рухів на веб-сайті, допомагає вам відновити місце, де ви зупинилися, запам'ятати зареєстрований логін, вибір теми, налаштування та інші функції налаштування.

Веб-сайт зберігає відповідний файл (з тим самим ідентифікаційним тегом) до того, який вони встановили у вашому веб -переглядачі, і в цьому файлі вони можуть відстежувати та зберігати інформацію про ваші переміщення на веб -сайті та будь -яку інформацію, яку ви могли добровільно надати під час відвідування веб -сайту, наприклад як адресу електронної пошти.

Файли cookie часто незамінні для веб -сайтів, які мають величезні бази даних, потребують входу, мають настроювані теми та інші розширені функції.

Файли cookie зазвичай не містять багато інформації, окрім URL-адреси веб -сайту, який створив файл cookie, тривалості можливостей та ефектів cookie та випадкового числа. Через невелику кількість інформації, яку містить файл cookie, зазвичай її не можна використовувати для розкриття вашої особи або особистої інформації. Однак маркетинг стає дедалі складнішим і файли cookie в деяких випадках можна агресивно використовувати для створення профілю ваших звичок серфінгу.

1.5.2. Local storage і session storage

Об'єкти веб-сховища localStorage та sessionStorage дозволяють зберігати пари "ключ -значення" у браузері.

Цікаве в них те, що дані зберігаються після оновлення сторінки (для sessionStorage) і навіть повного перезавантаження браузера (для localStorage). Ми це побачимо дуже скоро.

На відміну від файлів cookie, об’єкти веб -сховища не надсилаються на сервер з кожним запитом. Завдяки цьому ми можемо зберігати набагато більше. Більшість браузерів дозволяють принаймні 2 мегабайти даних (або більше) і мають параметри для їх налаштування. На відміну від файлів cookie, сервер не може керувати об’єктами зберігання через заголовки HTTP. Все зроблено в JavaScript.

Зберігання пов'язане з початком (триплет домену/протоколу/порту). Тобто різні протоколи чи субдомени виводять різні об’єкти зберігання, вони не мають доступу один до одного до даних.

Обидва об’єкти зберігання надають однакові методи та властивості:

- setItem (ключ, значення) - зберігає пару ключ/значення.
- getItem (ключ) - отримати значення за ключем.
- removeItem (ключ) - видалити ключ з його значенням.
- clear () - видалити все.
- key (index) - отримати ключ на певну позицію.
- length - кількість збережених елементів.

1.6. Http заголовки

Заголовки HTTP дозволяють клієнту та серверу передавати додаткову інформацію із запитом або відповіддю HTTP. Заголовок HTTP складається з імені, нечутливого до регістру, після якого йде двокрапка (:), а потім її значення. Пробіли перед значенням ігноруються.

Користувацькі власні заголовки історично використовувалися з префіксом X, але цю угоду було припинено у червні 2012 року через незручності, які вона спричинила, коли нестандартні поля стали стандартними у RFC 6648; інші перелічені в реєстрі IANA, оригінальний вміст якого визначено в RFC 4229. IANA також веде реєстр запропонованих нових заголовків HTTP.

Заголовки можна згрупувати відповідно до їх контексту:

• Заголовки запитів містять додаткову інформацію про ресурс, який потрібно отримати, або про клієнта, який запитує ресурс.

• Заголовки відповідей містять додаткову інформацію про відповідь, наприклад її місцезнаходження або про сервер, який її надає.

• Заголовки репрезентацій містять інформацію про тіло ресурсу, наприклад, його тип MIME або застосоване кодування/стиснення.

• Заголовки корисного вантажу містять незалежну від представлення інформацію про дані корисного навантаження, включаючи довжину вмісту та кодування, що використовується для транспортування.

1.7. Bootstrap бібліотека

Bootstrap - набір інструментів для полегшення створення сайтів і веб-додатків. Включає в себе HTML і CSS-шаблони оформлення для форм, різних кнопок, блоків навігації та інших компонентів які включаються до веб-інтерфейсу, а також такі розширення, як - JavaScript-розширення.

Основні інструменти Bootstrap:

• Сітки – це уже задані розміри для колонок, їх можна відразу ж використовувати та змінювати при необхідності.

• Шаблон - гумовий або статичний шаблон документа.

• Типографіка – включає опис різних шрифтів, визначення деяких властивостей та класів для шрифтів.

• Медіа – дає можливість структуровано відображати медіа файли.

• Таблиці – включає велику кількість стилів для оформлення таблиць, включає навіть сортування.

• Форми - класи для різноманітного оформлення форм, яких немає в звичайних мовах кодування і подій, які можуть відбуватись з ними

• Навігація - класи оформлення для вкладок, переходу між сторінками, велика кількість різних навігаційних меню та панелі інструментів.

• Алерти - оформлення для спливаючих вікон та різних підсказок.

Даний фреймворк використовує систему сіток(табл.):

Таблиця

Система сіток

	 **Extra small** <576px	 **Small** ≥576px	 **Medium** ≥768px	 **Large** ≥992px	 **Extra large** ≥1200px
Максимальна ширина	None (auto)	540px	720px	960px	1140px
Префікс класа	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`
Число колонок	12	12	12	12	12

Переваги та недоліки Bootstrap фреймворку

Bootstrap переваги:

Швидкість розробки, дозволяє скоротити витрати кількості часу. Як правило, використання фреймворків і бібліотек значно полегшує роботу розробників і дозволяє швидше розробляти проекти. Bootstrap можливість використовувати готові функції та плагіни: з ними ви витрачаєте набагато менше часу на створення макета та верстки сайтів.

Адаптивність. Bootstrap дозволяє створювати гнучкі та адаптовані сайти. Дизайн сайту буде коректно відображатися на екранах різних пристроях будь-якого розміру, незалежно від їх діагоналі.

Крос-браузерність. Сайти, створені за допомогою фреймворка Bootstrap, виглядатимуть однаково в сучасних браузерах, але в старіших версіях вони можуть виходити з ладу по-різному.

Простота використання та швидкість розробки. Bootstrap простий у використанні в розробці, його легко зрозуміти. Початківці, які ще не знайомі з цим фреймворком, зможуть швидко та легко зрозуміти та навчитися працювати з цим інструментом не тільки тому, що він дуже простий, але й тому, що існує велика кількість уроків та інструкцій, доступних на офіційному та різноманітні приватні інтернет-портали.

Очистити код. Bootstrap дозволяє писати простий і якісний код, який легко читати навіть розробники. Це полегшує командну роботу.

Недоліки Bootstrap:

Візерунок. Більшість сайтів, створених за допомогою різних фреймворків або CMS, є шаблонними, а фреймворк Bootstrap є шаблоном. Сайти, які використовують цей шаблон, схожі один на одного: мають майже однакову структуру, меню на сайті (якщо воно було витягнуто з фреймворку), різну навігацію, більшість кнопок. Ця подібність дуже погана властивість, тому що в наш час кожен хоче бути унікальним, а унікальність нині запам’ятовується

більше. Рішення цієї проблеми полягає в тому, щоб використовувати менше матеріалів шаблону, ніж структуру, і максимально змінити ідеї шаблонів.

Відсутність гнучкості. Bootstrap – це інструмент, який має деякі обмеження, одним з яких є гнучкість. Ось чому цей фреймворк підходить не для всіх проектів, які можна створити.

Старі браузери. Bootstrap, як і всі сучасні програми, оновлюється, щоб підтримувати своїх клієнтів у формі, тому в старих браузерах сайти, створені на цьому фреймворку, можуть відображатися неправильно.

1.8. Програми для створення сайтів

На сьогоднішній день існує велика кількість програм для створення сайтів. Ви можете використовувати як прості, так і складні програми. Ви навіть можете використовувати простий блокнот для створення простих сайтів. Кожен веб-сайт складається з веб-сторінок, кожна з яких складається з html-коду, що відповідає за дизайн сторінки, і фактичного вмісту. Тому без html - програмного редактора створення сайту неможливо.

Прості програми для редагування HTML-коду:

• Блокнот - звичайна програма, вбудована в Windows. Для того, щоб записати html-код веб-сторінки в блокнот, потрібно запустити блокнот, написати в ньому html-код і зберегти документ з розширенням .html. Недоліком є велика кількість помилок, які ви можете навіть не помітити при написанні коду.

• Блокнот ++ - вже розширена версія Блокнота. У Блокноті ++ html-теги мають кольорове підсвічування шрифту, і при виникненні помилки розробник може відразу візуально знайти код помилки, а код сторінки відрізняється кольором від основного тексту сторінки, але це не допомагає програма.

• Atom – це html-редактор, розроблений компанією GitHub, також має велику кількість інструментів та різноманітних плагінів, має ряд розширень, які можна налаштувати, включаючи підсвічування кольором, автоматичне виявлення помилок тощо.

• Visual Studio Code - текстовий редактор, який має дуже велику кількість розширень і різноманітних функцій. Він допомагає розробникам знаходити помилки, виділяє текст коду, різні стилі, теги. Він добре відображає структуру файлу.

• JetBrains PhpStorm — це комерційне кросплатформне інтегроване середовище розробки PHP, розроблене JetBrains на основі платформи IntelliJ IDEA.

• WebStorm – це інтегроване середовище розробки для JavaScript та пов’язаних із ним технологій. Як і інші IDE JetBrains, він робить ваш досвід розробки більш приємним, автоматизує рутинну роботу та допомагає з легкістю справлятися зі складними завданнями.

1.9. SPA додаток

Односторінковий додаток (SPA) — це веб-додаток, який представляється користувачеві через одну сторінку HTML, щоб бути більш чуйним і точніше відтворювати настільну програму або рідну програму. SPA іноді називають односторінковим інтерфейсом (SPI).

Односторінковий додаток може отримати весь HTML, JavaScript і CSS-код програми під час початкового завантаження або може динамічно завантажувати ресурси для оновлення у відповідь на взаємодію користувача чи інші події. Інші веб-програми, навпаки, представляють користувачеві початкову сторінку, яка пов’язана з частинами програми на окремих сторінках HTML, що означає, що користувач повинен чекати завантаження нової сторінки щоразу, коли він робить новий запит.

SPA використовують HTML5 і Ajax (асинхронний JavaScript і XML), щоб забезпечити плавні та динамічні відповіді на запити користувачів, дозволяючи вмісту оновлюватися негайно, коли користувач виконує дію. Після завантаження сторінки взаємодія із сервером здійснюється через виклики Ajax, а дані повертаються, як правило, у форматі JSON (JavaScript Object Notation), щоб оновити сторінку без перезавантаження.

Окрім того, що кодування JavaScript стає трохи розумнішим для розробників, SPA має дві основні переваги для користувачів. Перша перевага полягає в тому, що ви видаляєте цю різку зміну сторінки, коли ви натискаєте посилання. У SPA елементи керування навігацією та основний інтерфейс зазвичай залишаються на сторінці, коли ви натискаєте посилання; насправді змінюється лише той вміст, який ви хочете змінити. Другою перевагою є базове збільшення швидкості, яке ви отримуєте завдяки зменшеному навантаженню відповіді сервера для частини вмісту порівняно з цілою сторінкою. Менше корисне навантаження передається через мережу швидше, і браузер може включити новий фрагмент вмісту швидше, ніж перемалювати абсолютно нову сторінку.

1.10. Постановка задачі

Існує велика кількість різних інструментів для спрощення роботи з даними, створення даних, редагування даних, розбиття даних по різним структурам. Для розбиття даних буде зроблено:

- Визначити потрібний стек технологій для створення сайту;
- Розробити план для створення сайту;
- Розробити макет сайту;
- Створити компоненти сайту;
- Створити сервіси для створення методів;
- Створити методи та функції для розробки;
- Створити тести для перевірки методів;
- Зв’язати сервіси з компонентами;
- Зробить тести;
- Зробити збереження до локального сховища;
- Протестувати методи для збереження в локальному сховищі.
- Виконати верстку сайту для комп’ютерів;
- Зробити верстку для мобільних пристроїв.

Висновки до розділу 1

На сьогоднішній день всі потребують технологій для спрощення роботи, адже це дозволяє прискорювати роботу та зменшує кількість роботи. До того ж такі технології і інструменти дають більш прості рішення. Через це використовуються різні бібліотеки, фреймворки та технології.

Сайт складається з комбінації HTML, JavaScript, каскадних таблиць стилів (CSS) і плагінів, які визначаються необхідним вмістом і функціональністю. Односторінковий додаток (SPA), також відомий як односторінковий інтерфейс — це веб-додаток або веб-сайт, який поміщається на одній веб-сторінці з метою забезпечення більш плавної роботи користувача.

Протягом багатьох років все більше і більше JavaScript було втиснуто у ваш веб-досвід. Загалом це добре – більше JavaScript означає більше динамічної функціональності для користувача. Інновації у використанні JavaScript та значне покращення продуктивності движків JavaScript перетворили мову з оформлення HTML у, можливо, основний інструмент для сучасної веб-розробки.

Серед великої кількості теххнологій слід обирати ті, які більше підходять під цей проект, якщо більш крупні системи, то краще використовувати різні фреймворки, якщо це односторінкові сайті, то можливе написання на чистому JavaScript.

Для даної постанови задач буде використано велику кількість допоміжних бібліотек (Bootstrap, Material, RxJS), різних інструментів та технологій.

РОЗДІЛ 2. АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОПОНОВАНОГО ПРОДУКТУ

2.1. Web storm

WebStorm — це сучасна екосистема JavaScript. Він включає в себе інтелектуальне завершення коду, виявлення помилок «на льоту», потужну навігацію та рефакторинг для JavaScript, TypeScript, мов таблиць стилів та всіх популярних фреймворків.

Функції WebStorm:

Інтелектуальна допомога з кодуванням – WebStorm надає вам розумну допомогу в кодуванні для JavaScript і мов, скомпільованих у JavaScript, Node.js, HTML і CSS. Насолоджуйтесь завершенням коду, потужними функціями навігації, виявленням помилок на льоту та рефакторингом для всіх цих мов.

- Сучасні фреймворки - WebStorm надає розширену допомогу в кодуванні для Angular, React, Vue.js і Meteor. Насолоджуйтесь підтримкою React Native, PhoneGap, Cordova та Ionic для мобільної розробки та розробляйте для сервера за допомогою Node.js.

- Розумний редактор – IDE аналізує ваш проект, щоб забезпечити найкращі результати завершення коду для всіх підтримуваних мов. Сотні вбудованих інспекцій повідомляють про будь-які можливі проблеми прямо під час введення тексту та пропонують варіанти швидкого вирішення.

Навігація та пошук - WebStorm допомагає вам ефективніше обійти код і заощадити час під час роботи з великими проектами. Перейдіть до визначення методу, функції чи змінної лише одним клацанням миші або знайдіть варіанти використання.

• Налагодження, трасування та тестування – WebStorm надає потужні вбудовані інструменти для налагодження, тестування та відстеження ваших програм на стороні клієнта та Node.js. Завдяки мінімальній конфігурації та продуманій інтеграції в IDE ці завдання значно полегшуються з WebStorm.

• Налагодження – WebStorm надає вбудований налагоджувач для вашого клієнтського коду (який працює з Chrome) і програм Node.js. Розмістіть точки зупинки, перегляньте код і оцініть вирази – і все це, не виходячи з IDE.

• Тестування. Виконуйте тестування з легкістю, оскільки WebStorm інтегрується з Karma test Runner, Mocha, Jest і Protractor. Виконуйте та налагоджуйте тести прямо в IDE, переглядайте результати в красивому візуальному форматі та перейдіть до коду тесту.

• Відстеження та профілювання – WebStorm містить spy-js, вбудований інструмент, який допомагає відстежувати код JavaScript. Дослідіть, як файли пов’язані з викликами функцій, і ефективно визначте будь-які можливі вузькі місця.

• Повна інтеграція інструментів – WebStorm інтегрується з популярними інструментами командного рядка для веб-розробки, надаючи вам продуктивний, спрощений досвід розробки без використання командного рядка.

• Інструменти для створення – насолоджуйтесь простим уніфікованим інтерфейсом користувача для виконання завдань Grunt, Gulp або npm прямо з IDE. Усі завдання перераховані у спеціальному вікні інструментів, тому просто двічі клацніть назву завдання, щоб запустити його.

• Інструменти якості коду – на додаток до сотень власних перевірок WebStorm, він може запускати ESLint, JSCS, TSLint, Stylelint, JSHint або JSLint

для вашого коду та висвітлювати будь-які проблеми на льоту, прямо в редакторі.

• Шаблони проектів. Розпочинайте нові проекти з екрана привітання, використовуючи популярні шаблони проектів, як-от Express або Web starter kit, і отримайте доступ до ще більшої кількості генераторів проектів завдяки інтеграції з Yeoman.

• Функції IDE - WebStorm побудований на основі відкритої платформи IntelliJ. Насолоджуйтесь тонко налаштованим, але дуже настроюваним досвідом, який він надає, щоб відповідати вашому робочому процесу розробки.

• VCS – WebStorm надає уніфікований інтерфейс користувача для роботи з багатьма популярними системами контролю версій, забезпечуючи послідовну роботу користувачів у git, GitHub, SVN, Mercurial та Perforce.

• Локальна історія. Незалежно від того, використовуєте ви VCS чи ні, локальна історія може бути справжньою заощадженням коду. У будь-який момент ви можете переглянути історію певного файлу чи каталогу та повернутися до будь-якої з попередніх версій.

• Налаштування - WebStorm надзвичайно настроюється. Налаштуйте його так, щоб він ідеально відповідав вашому стилю кодування, від ярликів, шрифтів і візуальних тем до вікон інструментів і макета редактора.

2.2. Angular

Angular — це платформа розробки, побудована на TypeScript. Як платформа, Angular включає:

• Компонентний фреймворк для створення масштабованих веб-додатків

• Колекція добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, зв’язок клієнт-сервер тощо

• Набір інструментів для розробників, які допоможуть вам розробляти, створювати, тестувати та оновлювати код

• З Angular ви користуєтеся перевагами платформи, яка може масштабуватися від проектів окремого розробника до програм корпоративного рівня. Angular розроблено, щоб зробити оновлення максимально простим, тому скористайтеся перевагами останніх розробок з мінімумом зусиль. Найкраще те, що екосистема Angular складається з різноманітної групи з понад 1,7 мільйонів розробників, авторів бібліотек і творців контенту.

2.2.1. Ангуляр додатки

Компоненти

Компоненти — це будівельні блоки, з яких складається програма. Компонент включає в себе клас TypeScript з декоратором @Component(), шаблон HTML і стилі. Декоратор @Component() вказує таку специфічну для Angular інформацію:

• Селектор CSS, який визначає, як компонент використовується в шаблоні. Елементи HTML у вашому шаблоні, які відповідають цьому селектору, стають екземплярами компонента.

• Шаблон HTML, який вказує Angular, як відтворити компонент.

• Додатковий набір стилів CSS, які визначають зовнішній вигляд HTML-елементів шаблону.

Нижче наведено мінімальний компонент Angular(рис2.1).

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

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Рис.2.1. Мінімальний компонент проекту

Шаблони

Кожен компонент має шаблон HTML, який оголошує, як цей компонент відтворюється. Ви визначаєте цей шаблон або вбудованим, або шляхом до файлу.

Залежності

Залежності – це послуги або об’єкти, які потрібні класу для виконання своєї функції. Ін’єкція залежності, або DI, — це шаблон проектування, в якому клас запитує залежності із зовнішніх джерел, а не створює їх.

Фреймворк DI Angular надає залежності класу при створенні. Використовуйте Angular DI, щоб підвищити гнучкість та модульність у своїх програмах.

Ін’єкція залежностей дозволяє вам оголошувати залежності ваших класів TypeScript, не дбаючи про їх створення. Натомість Angular обробляє екземпляр

за вас. Цей шаблон проектування дозволяє писати більш тестований і гнучкий код. Незважаючи на те, що розуміння впровадження залежностей не є критичним для початку використання Angular, ми настійно рекомендуємо це як найкращу практику, і багато аспектів Angular певною мірою використовують це.

Angular CLI

Angular CLI — це найшвидший, простий і рекомендований спосіб розробки додатків Angular. Angular CLI робить ряд завдань безпроблемним. Ось кілька прикладів:

- ng build Компілює програму Angular у вихідний каталог.
- ng serve Створює та обслуговує вашу програму, перебудовуючи зміни у файлі.
- ng generate Генерує або змінює файли на основі схеми.
- ng test Виконує модульні тести для даного проекту.
- ng e2e Створює та обслуговує додаток Angular, а потім виконує наскрізні тести.

Бібліотеки Angular

Використовуйте платформу Angular, щоб включити одну з багатьох власних бібліотек, які надає Angular.

Деякі з доступних вам бібліотек включають:

- Angular Router Розширена навігація та маршрутизація на стороні клієнта на основі компонентів Angular. Підтримує відкладене завантаження, вкладені маршрути, спеціальне зіставлення шляхів тощо.
- Angular Forms Єдина система участі у формі та її перевірки.
- Angular HttpClient Надійний HTTP-клієнт, який може забезпечувати більш просунуту комунікацію клієнт-сервер.

- Angular Animations Багатіша система для керування анімацією на основі стану програми.
- Інструменти Angular PWA для створення прогресивних веб-додатків (PWA), включаючи сервісний працівник і маніфест веб-додатків.
- Angular Schematics Автоматизовані інструменти розробки, рефакторингу та оновлення, які спрощують розробку у великих масштабах.

Ці бібліотеки розширюють функціональність вашої програми, а також дозволяють вам більше зосередитися на функціях, які роблять вашу програму унікальною. Додайте ці бібліотеки, знаючи, що вони призначені для легкої інтеграції та оновлення одночасно з платформою Angular.

Ці бібліотеки потрібні лише тоді, коли вони можуть допомогти вам додати функціональність до ваших програм або вирішити певну проблему.

2.2.2. Angular observable

Observables забезпечують підтримку для передачі повідомлень між частинами вашої програми. Вони часто використовуються в Angular і є технікою для обробки подій, асинхронного програмування та обробки кількох значень.

Шаблон спостерігача — це шаблон розробки програмного забезпечення, в якому об’єкт, який називається суб’єктом, підтримує список своїх залежних, які називаються спостерігачами, і автоматично повідомляє їх про зміни стану. Цей шаблон подібний (але не ідентичний) шаблону оформлення публікації/підписки.

Об’єкти спостереження є декларативними, тобто ви визначаєте функцію для публікації значень, але вона не виконується, доки споживач не підписується на неї. Після цього споживач, на який підписався, отримує сповіщення, доки функція не завершиться або поки не скасує підписку.

Спостережуваний може передавати кілька значень будь-якого типу — літералів, повідомлень або подій, залежно від контексту. API для отримання

значень однаковий, незалежно від того, доставляють значення синхронно чи асинхронно.

Оскільки логіку налаштування та демонтажу обробляє спостережуваний, ваш код програми має турбуватися лише про підписку на споживання значень, а після завершення — про скасування підписки. Незалежно від того, чи був потік натискання клавіш, відповідь HTTP чи інтервальний таймер, інтерфейс для прослуховування значень і припинення прослуховування однаковий.

2.2.3. Promise

Об’єкт Promise представляє можливе завершення (або збій) асинхронної операції та її результуюче значення.

Promise — це проксі-сервер для значення, яке не обов’язково відомо під час створення обіцянки. Це дозволяє пов’язувати обробники з кінцевим значенням успіху асинхронної дії або причиною невдачі. Це дозволяє асинхронним методам повертати такі значення, як синхронні: замість того, щоб негайно повертати остаточне значення, асинхронний метод повертає обіцянку надати значення в якийсь момент у майбутньому.

Promise знаходиться в одному з таких станів:

- очікування: початковий стан, не виконано і не відхилено.
- виконано: це означає, що операція була успішно завершена.
- відхилено: це означає, що операція не вдалася.

Promise, що очікує на розгляд, може бути виконана зі значенням або відхилена з причиною (помилкою). Коли виконується будь-який з цих варіантів, викликаються пов’язані обробники, які стоять у черзі за допомогою методу then обіцянки. Якщо обіцянка вже була виконана або відхилена, коли відповідний обробник приєднано, буде викликано обробник, тому не існує умови змагання між завершенням асинхронної операції та підключенням її обробників.

2.2.4. Observable vs Promise

Основні відмінності:

Promises і Observables мають зовсім інший підхід до роботи з асинхронним кодом. Тут ми перевіримо чотири найбільш помітні відмінності.

Щоб наслідувати приклади, ви можете використовувати веб-консоль rxjs web dev Browser.

1. Одне значення проти кількох значень.

Найбільша відмінність полягає в тому, що обіцянки не змінять свою цінність після того, як вони будуть виконані. Вони можуть видавати (відхилити, розв'язувати) лише одне значення. З іншого боку, спостережувані можуть видавати кілька результатів.

2. Спостережувані підписки можна скасувати; проміс – ні.

Як тільки ви починаєте проміс, ви не можете його скасувати. Зворотній виклик, переданий конструктору Promise, відповідатиме за вирішення або відхилення обіцянки. Абонент пасивний; після запуску він може просто реагувати на результат.

Спостережувані менш пасивні. Після створення передплатника він може відмовитися від спостерігача в будь-який час. Це робить їх корисними в ситуаціях, коли ми більше не зацікавлені у відповіді. Наприклад, коли користувач залишає сторінку.

Існує багато способів скасувати/заповнити підписку. Давайте перевіримо три найпоширеніші:

- скасувати підписку: скасування підписки вручну з Observable
- take: оператор для взяття числа X елементів і скасування підписки
- takeUntil: оператор, який продовжує приймати значення, доки переданий Observable не видає будь-яке значення.

3. Нетерпляче проти лінивого виконання.

Існує різниця в тому, як виконуються Observables і Promises. Обіцянки виконуються з нетерпінням, тоді як Observables виконуються ліниво.

• Eagar: зворотний виклик Promise буде виконуватися відразу на рівні конструктора.

• Lazy: функція Producer запускається лише після створення підписки для цього Observable. Інакше він залишиться бездіяльним.

4. Runtime execution.

The ES Promises, once resolved, will queue the callback in the microtask queue. That means they will be executed after the current macro task has been completed.

За допомогою Observables ви можете точно налаштувати виконання під час виконання за допомогою планувальників. Планувальник контролює, коли починається підписка та коли надходять сповіщення.

• null: за замовчуванням сповіщення надходять синхронно та рекурсивно.

• queueScheduler: розкладає в черзі в поточному кадрі події.

• asapScheduler: розклади в черзі мікрозавдань. Така ж черга, що й обіцянки.

• asyncScheduler: схожий на планування завдання за допомогою setInterval. Тому воно буде заплановано в черзі завдань макросу.

• animationFrameScheduler: покладається на API requestAnimationFrame.

2.3. AngularJS

AngularJS — це структурна структура для динамічних веб-програм. Він дає змогу використовувати HTML як мову шаблонів і дозволяє розширити синтаксис HTML, щоб чітко та коротко виражати компоненти програми. Прив’язування даних і ін’єкція залежностей у AngularJS усувають більшу частину коду, який вам довелося б писати. І все це відбувається в браузері, що робить його ідеальним партнером для будь-якої серверної технології.

AngularJS – це те, чим був би HTML, якби він був розроблений для додатків. HTML — чудова декларативна мова для статичних документів.

Невідповідність між динамічними додатками та статичними документами часто вирішується за допомогою:

• бібліотека - набір функцій, корисних під час написання веб-програм. Ваш код відповідає, і він звертається до бібліотеки, коли вважає за потрібне. Наприклад, jQuery.

• фреймворки - конкретна реалізація веб-додатка, де ваш код заповнює деталі. Фреймворк відповідає, і він звертається до вашого коду, коли йому потрібно щось специфічне для програми. Наприклад, durandal, ember тощо.

AngularJS використовує інший підхід. Він намагається мінімізувати невідповідність імпедансу між HTML, орієнтованим на документ, і тим, що потрібно додатку, створюючи нові конструкції HTML. AngularJS навчає браузер новому синтаксису за допомогою конструкції, яку ми називаємо директивами. Приклади включають:

• Прив’язка даних, як у {{}}.

• Структури керування DOM для повторення, відображення та приховування фрагментів DOM.

• Підтримка форм і перевірки форми.

• Додавання нової поведінки до елементів DOM, як-от обробка подій DOM.

• Групування HTML у багаторазові компоненти.

• Повне рішення на стороні клієнта

AngularJS не є єдиною частиною загальної головоломки побудови клієнтської сторони веб-додатка. Він обробляє весь код клею DOM і AJAX, який ви колись написали вручну, і поміщає його в чітко визначену структуру. Це змушує AngularJS визначати, як має бути створена програма CRUD (Створення, читання, оновлення, видалення). Але, незважаючи на те, що воно є самовпевненим, воно також намагається переконатися, що його думка є лише відправною точкою, яку можна легко змінити. AngularJS поставляється з наступним стандартним:

Все, що вам потрібно для створення програми CRUD в єдиному набору: зв’язування даних, основні директиви шаблонів, перевірка форм, маршрутизація, глибоке посилання, багаторазові компоненти та ін’єкція залежностей.

Історія тестування: модульне тестування, наскрізне тестування, макети та тестові джгути.

Початковий додаток із макетом каталогу та тестовими сценаріями як відправною точкою.

AngularJS спрощує розробку додатків, представляючи розробнику більш високий рівень абстракції. Як і будь-яка абстракція, це коштує гнучкості. Іншими словами, не кожна програма добре підходить для AngularJS. AngularJS було створено з урахуванням програми CRUD. На щастя, програми CRUD представляють більшість веб-додатків.

Однак, щоб зрозуміти, чим добре AngularJS, це допомагає зрозуміти, коли програма не підходить для AngularJS.

Ігри та редактори графічного інтерфейсу є прикладами додатків з інтенсивним і складним маніпулюванням DOM. Такі програми відрізняються від додатків CRUD, і в результаті, ймовірно, не підходять для AngularJS. У цих випадках може бути краще використовувати бібліотеку з нижчим рівнем абстракції, наприклад jQuery.

AngularJS побудований на переконанні, що декларативний код краще, ніж імперативний, коли справа доходить до створення інтерфейсів інтерфейсу користувача та з'єднання програмних компонентів разом, тоді як імперативний код відмінно підходить для вираження бізнес-логіки.

Дуже гарна ідея відокремити маніпуляції DOM від логіки програми. Це значно покращує можливість тестування коду.

Це дійсно, дуже гарна ідея розглядати тестування додатків як однакову важливість з написанням додатків. На складність тестування суттєво впливає структура коду.

Відмінною ідеєю є відокремлення клієнтської сторони програми від сторони сервера. Це дозволяє паралельно виконувати роботу з розробки та повторно використовувати обидві сторони.

Справді, дуже корисно, якщо фреймворк веде розробників через весь шлях створення програми: від проектування інтерфейсу користувача, написання бізнес-логіки до тестування.

Завжди добре робити звичайні завдання тривіальними, а важкі – можливими.

AngularJS звільняє вас від таких проблем:

• Реєстрація зворотних викликів: реєстрація зворотних викликів захаращує ваш код, що ускладнює пошук лісу для дерев. Видалення стандартного коду, такого як зворотні виклики, — це добре. Це значно зменшує обсяг кодування JavaScript, яке вам потрібно зробити, і полегшує побачити, що робить ваша програма.

• Програмне маніпулювання HTML DOM: Маніпулювання HTML DOM є наріжним каменем додатків AJAX, але воно громіздке та схильне до помилок. Декларативно описуючи, як інтерфейс користувача повинен змінюватися в міру зміни стану вашої програми, ви звільняєтеся від низькорівневих завдань маніпулювання DOM. Більшість програм, написаних за допомогою AngularJS, ніколи не повинні програмно маніпулювати DOM, хоча ви можете, якщо хочете.

• Маршалування даних в інтерфейс користувача та з нього: операції CRUD складають більшість завдань додатків AJAX. Потік маршування даних від сервера до внутрішнього об’єкта до форми HTML, що дозволяє користувачам змінювати форму, перевіряти форму, відображати помилки перевірки, повертатися до внутрішньої моделі, а потім назад на сервер, створює багато шаблонів. код. AngularJS усуває майже весь цей шаблон, залишаючи код, який описує загальний потік програми, а не всі деталі реалізації.

• Написання тонни коду ініціалізації лише для початку: зазвичай вам потрібно написати багато сантехніки, щоб запрацювати базовий додаток AJAX «Hello World». За допомогою AngularJS ви можете легко завантажувати свою програму за допомогою служб, які автоматично вводяться у вашу програму у стилі Guice-подібного введення залежностей. Це дозволяє швидко розпочати

розробку функцій. Як бонус, ви отримуєте повний контроль над процесом ініціалізації в автоматизованих тестах.

2.4. Angular vs AngularJS

Angular — це заснована на Typescript альтернатива AngularJS, яка використовується для створення динамічних веб-додатків. Сьогодні найбільше використовується оновлена версія. Деякі з його ключових особливостей полягають у тому, що він простий у розробці, має вдосконалений модульний дизайн і значно швидше в порівнянні з AngularJS.

AngularJS, з іншого боку, написаний на Javascript і може використовуватися як термін для всіх версій Angular v1.x. Він був створений у 2009 році, і його також зазвичай називають Angular 1. Фреймворк розробки односторінкових додатків із відкритим кодом також має свої унікальні особливості, такі як можливість змінювати статичний HTML на динамічний. Він має ряд фільтрів і директив.

Відмінність:

Мова

Основна відмінність між двома фреймворками з відкритим вихідним кодом полягає в тому, що Angular заснований на Typescript (супернабір ES6), а AngularJs заснований на Javascript. Це, по суті, означає, що будуть відмінності в їх складових.

Використання компонентів і директив

Інша важлива відмінність полягає в тому, що робота Angular заснована на ієрархії компонентів, тоді як AngularJs має пакет директив, які дозволяють повторно використовувати код і писати окремі коди. Він також використовує область і контролери. Навіть у Angular є стандартні директиви, але різниця полягає в тому, як їх реалізує будь-який фреймворк.

Основа архітектури

Фреймворк Angular в основному використовує компоненти, включаючи структурні директиви та атрибутивні директиви, які постачаються з шаблонами. Хоча перший використовується для модифікації макета DOM, другий відповідає за зміну поведінки DOM. Структурні директиви замінюють елементи для досягнення своєї мети, а директива атрибутів змінюють зовнішній вигляд елементів. У підході Angular функціональні та логічні компоненти розділені відповідно до їх цілей і можуть краще служити додатку.

Angular JS, з іншого боку, працює на фреймворку модель-представлення-контролер (MVC), де, як контролер, він відіграє центральну роль в управлінні даними, правилами, отриманні вхідних даних та їх обробці в команди для надсилання до Модель і вид. Він також керує тим, як поводяться програми.

Модель відповідає за зберігання та керування отриманими даними, а View виводить дані після перегляду інформації, що міститься в моделі.

MVC досить простий, цілісний і допомагає прискорити розробку. Функція зв’язування також означає, що вам доведеться писати менше коду.

Ін'єкція залежності

Хоча AngularJS не використовує DI (він використовує директиви), Angular має ієрархічну систему DI для підвищення продуктивності.

Маршрутизація

AngularJS використовує @routeProvider.when для визначення інформації про маршрутизацію. Angular, з іншого боку, використовує URL-адресу, щоб імітувати директиву, щоб дістатися до клієнтського представлення. Тут конфігурація @Route використовується для інформації про маршрутизацію. Це дає Angular перевагу перед AngularJS.

Продуктивність і швидкість

AngularJs має окрему функцію двостороннього зв’язування, що означає менше часу та зусиль. Angular має кращу структуру, що стає вирішальним для підвищення швидкості та продуктивності його операцій.

Тестування та інструменти

AngularJs покладається на IDE та Webstorm, інструменти Javascript сторонніх розробників, для створення програм і тестування на помилки.

Angular знову оцінює використання інтерфейсу командного рядка (CLI) для створення проекту. Додаткова перевага полягає в тому, що це призводить до скорочення часу та більшої доступності, коли справа доходить до тестування.

Менеджмент

Проектами Angular легше керувати, ніж проектами AngularJS, оскільки вони структуровані. Це особливо плюс, якщо мова йде про великі програми.

2.5. Ahead-of-time

Додаток Angular складається в основному з компонентів та їх шаблонів HTML. Оскільки компоненти та шаблони, надані Angular, не можуть бути зрозумілі браузеру безпосередньо, додаткам Angular потрібен процес компіляції, перш ніж вони зможуть працювати у браузері.

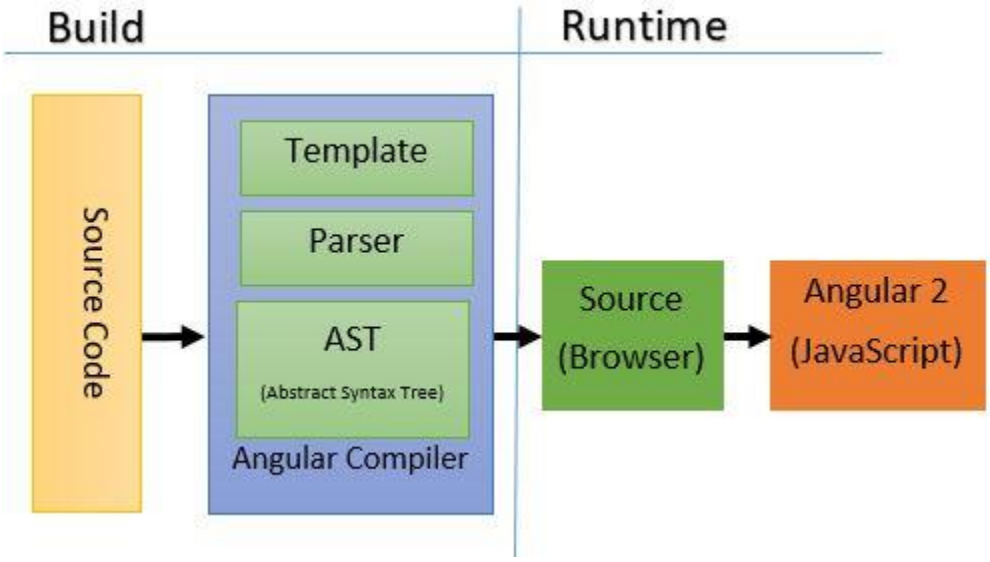


Рис.2.2. Ahead-of-time компіляція

Компілятор Angular Ahead-of-Time (AOT) перетворює ваш код Angular HTML і TypeScript в ефективний код JavaScript на етапі збірки, перш ніж

браузер завантажить і запустить цей код(рис2.2). Компіляція вашої програми під час процесу складання забезпечує швидший рендеринг у браузері.

Ось кілька причин, чому ви можете використовувати AOT.

Швидше відтворення За допомогою AOT браузер завантажує попередньо скомпільовану версію програми. Браузер завантажує виконуваний код, щоб він міг негайно відтворити програму, не чекаючи попередньої компіляції програми.

Менше асинхронних запитів Компілятор вбудовує зовнішні шаблони HTML і таблиці стилів CSS в JavaScript програми, усуваючи окремі запити ajax для цих вихідних файлів.

Менший розмір завантаження Angular Framework Немає необхідності завантажувати компілятор Angular, якщо програма вже скомпільована. Компілятор становить приблизно половину самого Angular, тому його пропуск різко зменшує корисне навантаження програми.

Виявити помилки шаблону раніше. Компілятор AOT виявляє та повідомляє про помилки прив’язки шаблону під час етапу збірки, перш ніж користувачі зможуть їх побачити.

Краща безпека AOT компілює HTML-шаблони та компоненти у файли JavaScript задовго до того, як вони подаються клієнту. Без шаблонів для читання та ризикованої оцінки HTML або JavaScript на стороні клієнта є менше можливостей для ін’єкційних атак.

Існує три етапи складання AOT.

• Аналіз коду. На цьому етапі компілятор TypeScript і колектор AOT створюють представлення джерела. Колектор не намагається інтерпретувати метадані, які він збирає. Він якнайкраще представляє метадані та записує помилки, коли виявляє порушення синтаксису метаданих.

• Генерація коду. На цьому етапі StaticReflector компілятора інтерпретує метадані, зібрані на етапі 1, виконує додаткову перевірку метаданих і видає помилку, якщо виявляє порушення обмеження метаданих.

• Перевірка типу шаблону. На цьому додатковому етапі компілятор шаблону Angular використовує компілятор TypeScript для перевірки виразів

зв’язування в шаблонах. Ви можете ввімкнути цю фазу явно, встановивши параметр конфігурації fullTemplateTypeCheck; див. параметри компілятора Angular.

2.6. Just-in-time

У обчисленнях компіляція «точно вчасно» (JIT) — це спосіб виконання комп’ютерного коду, який включає компіляцію під час виконання програми — під час виконання — а не перед виконанням.

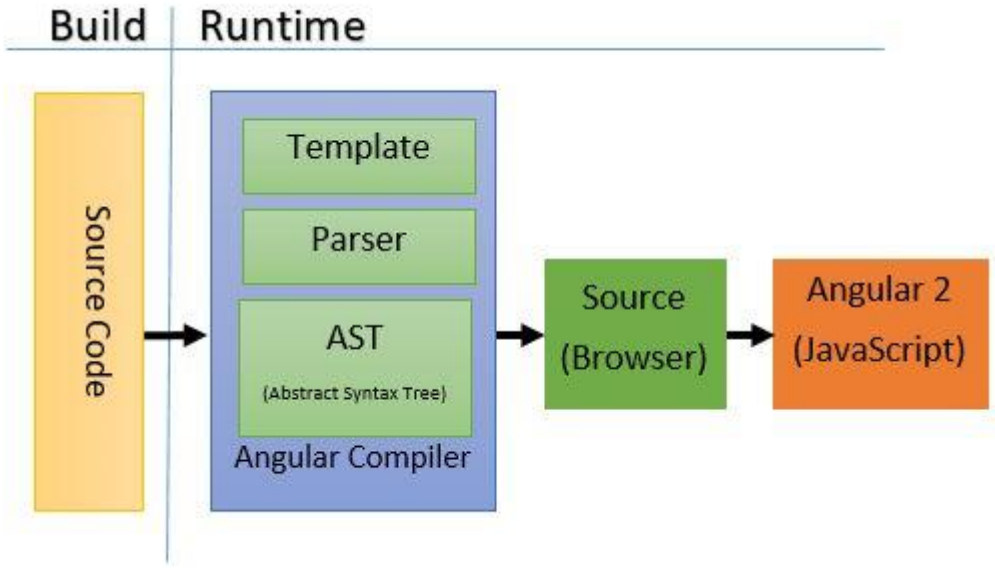


Рис.2.3. Just-in-time компіляція

Спочатку компілятор відповідав за перетворення мови високого рівня в об’єктний код (машинні інструкції), який потім був пов’язаний у виконуваний файл.

Компілятор Just-in-time (JIT) є особливістю інтерпретатора під час виконання, який замість того, щоб інтерпретувати байт-код кожного разу, коли викликається метод, компілює байт-код в інструкції машинного коду запущеної машини, а потім викликає цей натомість об'єктний код(рис2.3).

Потік в проекті Angular:

Використовуйте Typescript, HTML, CSS (SCSS або SASS) для розробки програми Angular.

Використовуйте ng build для побудови вихідного коду в пакети. Сюди входять активи, файли JS (модулі у випадку відкладеного завантаження та js map, постачальник і полізаповнення), index.html та CSS.

Потім ми вбудовуємо це у файл війни для розгортання jboss або розгортаємо безпосередньо за допомогою heroku або іншого хостингу, який підтримує Node. Потім ми зіставляємо цей хост із нашим доменом за допомогою CNAME.

Кінцевий користувач отримує доступ до нашого веб-додатка через домен. Браузер завантажить усі ресурси, включаючи HTML, CSS та JavaScript, які необхідні для перегляду за замовчуванням.

Angular завантажує програму Angular пройде процес компіляції JIT для кожного компонента програми. Потім програма відтворюється.

Примітка:

У JIT спочатку не весь код перетворюється на машинний. Тільки необхідний код (використовується негайно) буде перетворено в машинний код. Тоді, якщо метод або функціональність викликані і не містяться в машинному коді, то вони також будуть перетворені в машинний код. Це зменшує навантаження на центральний процесор і робить програму швидшою, оскільки використовує лише те, що потрібно.

Ви можете налагодити у браузері під час впровадження, оскільки код був скомпільований у режимі JIT з файлом карти. Цю довідку ви можете побачити та посилатися на вихідний код безпосередньо в інспекторі.

2.7. JIT та AOT переваги та недоліки

JIT завантажує компілятор і компілює код точно перед відображенням у браузері. AOT вже відповідав коду під час створення вашої програми, тому його не потрібно компілювати під час виконання.

Завантаження в JIT повільніше, ніж у AOT, оскільки він потребує компіляції вашої програми під час виконання. Завантаження в AOT набагато швидше, ніж у JIT, оскільки він уже скомпільував ваш код під час збірки.

JIT більше підходить для режиму розробки. AOT дуже підходить у випадку виробничого режиму.

Розмір пакета вищий порівняно з AOT. Розмір пакета оптимізовано в AOT, в результаті розмір пакета AOT становить половину розміру пакетів JIT.

Ви можете запустити свою програму в JIT за допомогою цієї команди:

- ng build OR ng service

Щоб запустити свою програму в AOT, ви повинні вказати –aot в кінці, наприклад:

- ng build --aot АБО ng serve --aot

Ви можете зловити помилку прив’язування шаблону під час відображення. Ви можете виявити помилку шаблону під час створення програми.

2.8. Constructor vs ngOninit

Як частина об’єктно-орієнтованого програмування на основі класів, конструктор відноситься до спеціального типу методу, який використовується для створення об’єкта. Він значною мірою використовується для ініціалізації класу та його підкласів. Клас визначається як шаблон, призначений для визначення імен і типів змінних, які існують в об'єкті. Будучи методом за замовчуванням класу, він створює новий екземпляр класу. Головне зауважити, що це функція класу (TypeScript або JavaScript ES2015/ES6), а не концепція Angular. Нижче перераховані його різні види.

Конструктор за замовчуванням

Конструктор відомий як конструктор за замовчуванням, якщо він не приймає жодного параметра. Компілятор Java вставляє конструктор за замовчуванням у код розробників, якщо вони не надають конструктор для класу, що реалізується. Його поведінка залежить від мови. Конструктори за замовчуванням можуть або ініціалізувати членів класу до нуля або інших однакових значень.

Параметризований конструктор

Коли розробник приймає хоча б один параметр, він розглядається як параметризований конструктор. Його можна використовувати для призначення значення приватним властивостям класу. Цікаво, що параметризовані конструктори автоматично викликають, коли програміст створює об’єкт певного класу.

Конструктори копіювання

Конструктор копіювання містить один формальний параметр – тип класу. Він підтримується як C++, так і Java. Основне використання цього типу конструктора — створення копій існуючих об’єктів класу. Разом з тим, загальне виконання залежить від дій компілятора.

Конструктори перетворення

Компіляторам легко створити належність одного класу на основі об’єкта до виділеного класу за допомогою конструкторів перетворення. Вони перетворюють аргументи або операнди в точний тип, неявно або явно.

Конструктори переміщення

Конструктори переміщення є незамінними частинами C++, які використовуються для перенесення тимчасових об'єктів в існуючі об'єкти. Вони переміщують ресурси в кучу і запобігають непотрібне копіювання даних у пам'яті.

Конструктори в Angular

Конструктор у Angular використовується для введення залежностей у клас компонента. Він створює новий екземпляр класу, коли компілятор

викликає «new MyClass ()». Викликаючи ‘new MyClass()’, дуже важливо, щоб точна відповідність параметра передавала конструктору Angular компонента класу.

Наприклад, у новому My Class (arg1, arg2, argN) arg1, arg2 та arg3 мають бути того самого типу, що й у конструкторі класу.

Angular конструктор, простими словами, призначений для написання речей у найкращій формі за допомогою ін’єкції залежності. Для людей, які тільки почали працювати з Angular, залежність відноситься до служби або об’єкта Angular, який клас тягне за собою для виконання функцій. І ін’єкція залежностей — це метод, у якому класи запитують залежності із зовнішніх ресурсів.

У конструкторі Angular аргумент конструктора з типом залежності вводить залежності. Необхідно, щоб конструктор компонента Angular був простим у всьому. Причина в тому, що проста логіка в конструкторі Angular допомагає легко працювати з модульним тестуванням.

ngOnInit() в Angular

Щоб використовувати ngOnInit в Angular, потрібно імпортувати в клас компонента таким чином – імпортувати {Component, OnInit} з ‘@angular/core’. Хоча впровадження ngOnInit для кожного компонента не є обов’язковим, це хороша практика, яка в подальшому спрямована на безперебійне функціонування платформи. Angular ngOnInit створюється самим Angular. Angular також відтворює та перевіряє зміни своїх обмежених властивостей даних. Цікаво відзначити, що Angular знищує компонент перед тим, як видалити його з DOM. ngOnInit в Angular можна підключити до компонентів і директив. Визначення ngOnInit Angular у класі допомагає під час виконання Angular знати, що саме час викликати метод. Основна перевага цього підходу полягає в тому, що він додає конкретну логіку ініціалізації біля життєвого циклу класу. Ось спосіб реалізації ngOnInit в Angular.

Крок 1. Додайте OnInit після ключових слів implements у компонент або директиву. Тут OnInit буде імпортовано з ядра Angular і створить контракт, який реалізує OnInit.

Крок 2: Після додавання OnInit з наступними елементами TypeScript підкреслить оголошення класу червоним кольором. Це свідчить про те, що ngOnInit не знайдено. У такому випадку допоможе створення власного методу ngOnInit. Ось тут у п’єсу виникає потреба пояснити компонент до і після.

І ngOnInit Angular, і Constructor в Angular містять концепцію об’єктно-орієнтованого програмування. Беручи до уваги порівняння між конструктором класу angular і ngOnInit, можна визначити використання обох компонентів життєвого циклу.

Використання конструктора в Angular

Angular Constructor в основному використовується для введення залежностей у різні артефакти, такі як служби та компоненти. Задаючи параметр конструктора з типом залежності, програмісти можуть наказати Angular ввести залежності в конструктор компонента. Можна також передати необов'язкові залежності, сказавши Angular, що залежність необов'язкова. Це можна зробити, позначивши параметр за допомогою @Optional(). Якщо програмісту потрібно ввести додаткові параметри в сервіс Angular, декоратор @Inject допомагає передати параметри службі Angular. Якою б не була мета залежностей ін’єкції, важливою підказкою є те, щоб конструктор був простим у всьому. Це спосіб зробити модульне тестування простим.

Використання ngOnInit в Angular

ngOnInit Angular просто використовується для забезпечення виконання коду ініціалізації. Angular викликає ngOnInit після завершення компонента. Це гарантує, що компонент створено. Якщо програмісту потрібні додаткові завдання ініціалізації, важливо визначити метод ngOnInIt () для обробки. Найкраща практика знову викликати ngOnInit – реалізувати «логіку» в іншому методі та викликати цей метод із ngOnInit (). Навіть той самий метод можна викликати знову, але з іншої функції.

2.9. Lifecycle hooks

Екземпляр компонента має життєвий цикл, який починається, коли Angular створює екземпляр класу компонента і відтворює подання компонента разом з його дочірніми представленнями. Життєвий цикл продовжується з виявленням змін, оскільки Angular перевіряє, чи змінюються властивості, пов’язані з даними, і за потреби оновлює як подання, так і екземпляр компонента. Життєвий цикл закінчується, коли Angular знищує екземпляр компонента та видаляє його відтворений шаблон із DOM. Директиви мають схожий життєвий цикл, оскільки Angular створює, оновлює та знищує екземпляри під час виконання.

Ваша програма може використовувати методи перехоплення життєвого циклу, щоб використовувати ключові події в життєвому циклі компонента або директиви, щоб ініціалізувати нові екземпляри, ініціювати виявлення змін, коли це необхідно, реагувати на оновлення під час виявлення змін і очищати екземпляри перед видаленням.

Після того, як ваша програма створює екземпляр компонента або директиви, викликаючи його конструктор, Angular викликає методи перехоплення, які ви реалізували у відповідній точці життєвого циклу цього екземпляра.

Angular виконує гачкові методи в такій послідовності. Використовуйте їх для виконання наступних видів операцій.

ngOnChanges()

Відповідати, коли Angular встановлює або скидає властивості введення, пов’язані з даними. Метод отримує об’єкт SimpleChanges поточних і попередніх значень властивостей.

Зауважте, що це трапляється дуже часто, тому будь-яка операція, яку ви виконуєте тут, значно впливає на продуктивність. Подробиці див. у розділі Використання гачків виявлення змін у цьому документі.

Викликається перед ngOnInit() (якщо компонент має зв’язані вхідні дані) і щоразу, коли змінюються одна або кілька властивостей введення, прив’язаних до даних.

Зауважте, що якщо ваш компонент не має вхідних даних або ви використовуєте його без надання жодних вхідних даних, фреймворк не буде викликати ngOnChanges().

2.9.1. ngOnInit()

Ініціалізуйте директиву або компонент після того, як Angular вперше відобразить властивості, пов’язані з даними, і встановить вхідні властивості директиви або компонента. Подробиці див. у розділі Ініціалізація компонента або директиви в цьому документі.

Викликається один раз, після першого ngOnChanges(). ngOnInit() все ще викликається, навіть якщо ngOnChanges() ні (якщо немає введених даних, прив’язаних до шаблону).

2.9.2. ngDoCheck()

Виявляти зміни, які Angular не може або не може виявити самостійно, і діяти відповідно до них. Подробиці та приклад див. у розділі Визначення виявлення користувацьких змін у цьому документі.

Викликається відразу після ngOnChanges() під час кожного запуску виявлення змін і відразу після ngOnInit() під час першого запуску.

2.9.3. ngAfterContentInit()

Відповідь після того, як Angular проектує зовнішній вміст у подання компонента або в подання, в якому знаходиться директива.

Подробиці та приклад див. у розділі Відповідь на зміни вмісту в цьому документі.

Викликається один раз після першого ngDoCheck().

2.9.4. ngAfterContentChecked()

Відповідь після того, як Angular перевірить вміст, спроектований в директиву або компонент.

Подробиці та приклад див. у розділі Відповідь на прогнозовані зміни вмісту в цьому документі.

Викликається після ngAfterContentInit() і кожного наступного ngDoCheck().

2.9.5. ngAfterViewInit()

Відповідь після того, як Angular ініціалізує подання компонента та дочірні подання або подання, що містить директиву.

Щоб переглянути зміни в цьому документі, перегляньте деталі та приклад у розділі Відповідь.

Викликається один раз після першого ngAfterContentChecked().

2.9.6. ngAfterViewChecked()

Відповідайте після того, як Angular перевірить подання компонента та дочірні подання або подання, що містить директиву.

Викликається після ngAfterViewInit() і кожного наступного ngAfterContentChecked().

2.9.7. ngOnDestroy()

Очищення безпосередньо перед тим, як Angular знищить директиву або компонент. Скасуйте підписку на Observables та від'єднайте обробники подій, щоб уникнути витоку пам'яті. Подробиці див. у розділі Очищення при знищенні екземпляра в цьому документі.

Викликається безпосередньо перед тим, як Angular знищить директиву або компонент.

2.10. Angular Material

Angular Material — це компонент бібліотеки інтерфейсу користувача, розроблений Google у 2014 році. Він спеціально розроблений для розробників AngularJS. Це допомагає структурувати програму. Його компоненти допомагають створювати привабливі, послідовні та функціональні веб-сторінки та веб-додатки. Він використовується для створення адаптивного та швидшого веб-сайту.

Ці компоненти роблять додаток або веб-сайт більш послідовним і адаптивним до дизайну. Він поєднує в собі класичні принципи успішного дизайну з інноваціями та технологіями.

Google розробив Angular Material у 2014 році. У той час він був позначений як AngularJS, щоб зробити програми привабливішими та швидшими. Google знову написав код з нуля і видалив JS, а потім назвав його Angular Material у вересні 2016 року. Компоненти UI/UX відомі як Angular Materials.

Особливості Angular матеріалу:

- Це вбудований адаптивний дизайн.
- Angular Material має стандартний CSS.
- Нова версія компонентів інтерфейсу користувача, які представляють собою кнопки, прапорці та текстові поля, використовується для дотримання концепцій Angular Material Design.

• Він має спеціальні функції, такі як картки, панель інструментів, швидкий набір, бічна навігація, гортання, бічна навігація тощо.

• Це кросплатформний браузер, який використовується для створення веб-компонентів.

• Адаптивний дизайн

• Angular Material має адаптивний дизайн, тому його веб-сайт може вмістити будь-який розмір.

• Веб-сайти, створені Material, сумісні з Android, iPhone, планшетами та ноутбуками.

• Розширюваний

• Легко додати нові правила CSS, щоб замінити існуючий CSS

• Він підтримує тіні та кольори.

• Кольори та відтінки однакові в Angular Material.

• Angular Material можна використовувати безкоштовно.

• Висока якість

• Компоненти Angular Material доступні кожному. Він протестований з багатьма параметрами, що гарантує продуктивність і надійність програми.

• Він має прості API з послідовною міжплатформною поведінкою.

• Універсальний

• Він надає інструменти, які допомагають розробникам створювати власні компоненти із загальними шаблонами.

• Його можна налаштувати в межах.

• Він був створений командою Angular для ідеальної інтеграції.

Ми можемо використовувати Angular Material для розробки нового додатка з нуля або в існуючій програмі.

2.11. RxJS бібліотека

RxJS — це бібліотека для складання асинхронних програм і програм на основі подій за допомогою спостережуваних послідовностей. Він надає один основний тип, Observable, типи супутників (Observer, Schedulers, Subjects) та оператори, створені за допомогою Array#extras (map, filter, reduce, every тощо), щоб дозволити обробляти асинхронні події як колекції.

ReactiveX поєднує шаблон Observer з шаблоном Iterator і функціональне програмування з колекціями, щоб заповнити потребу в ідеальному способі керування послідовністю подій.

Основні концепції в RxJS, які вирішують асинхронне керування подіями:

• Observable: представляє ідею викликаної колекції майбутніх цінностей або подій.

• Observer: це набір зворотних викликів, який знає, як прослухати значення, що надаються Observable.

• Subscription: представляє виконання Observable, в першу чергу корисна для скасування виконання.

• Оператори: це чисті функції, які забезпечують функціональний стиль програмування для роботи з колекціями за допомогою таких операцій, як map, filter, concat, reduce тощо.

• Subject: еквівалентний EventEmitter і єдиний спосіб багатоадресної передачі значення або події кільком спостерігачам.

• Schedulers: це централізовані диспетчери для контролю паралельності, що дозволяє нам координувати, коли обчислення відбуваються, наприклад, setTimeout або requestAnimationFrame або інші.

Pull і Push — це два різних протоколи, які описують, як виробник даних може спілкуватися зі споживачем даних.

Що таке Pull? У системах Pull Споживач визначає, коли він отримує дані від Виробника даних. Сам Виробник не знає, коли дані будуть передані Споживачеві.

Кожна функція JavaScript є системою Pull. Функція є виробником даних, і код, який викликає функцію, споживає її, «витягуючи» одне значення, що повертається з її виклику.

Pull Passive: створює дані за запитом. Активний: визначає, коли запитуються дані.

Push Active: виробляє дані у власному темпі. Пасивний: реагує на отримані дані.

Що таке Push? У системах Push Виробник визначає, коли надсилати дані Споживачеві. Споживач не знає, коли отримає ці дані.

Обіцянки є найпоширенішим типом Push-системи в JavaScript сьогодні. Обіцянка (Виробник) доставляє вирішене значення зареєстрованим зворотним викликам (Споживачам), але, на відміну від функцій, саме Promise відповідає за точне визначення того, коли це значення «виштовхується» до зворотних викликів.

RxJS представляє Observables, нову систему Push для JavaScript.

2.11.1. Оператори RxJS

Оператори - це функції. Є два види операторів:

Оператори Pipeable – це тип, який можна передати до Observables за допомогою синтаксису observableInstance.pipe(operator()). До них належать filter(...) і mergeMap(...). При виклику вони не змінюють наявний екземпляр Observable. Замість цього вони повертають новий Observable, логіка підписки якого заснована на першому Observable.

Оператор Pipeable, по суті, є чистою функцією, яка приймає один Observable як вхід і генерує інший Observable як вихід. Підписка на вихідний Observable також підпишеться на вхідний Observable.

Оператори створення — це інший тип операторів, які можна викликати як окремі функції для створення нового Observable. Наприклад: of(1, 2, 3) створює спостережуваний, який видаватиме 1, 2 і 3 один за одним. Більш детально оператори створення будуть розглянуті в наступному розділі.

2.12. Lazy-loading Angular

За замовчуванням NgModules швидко завантажуються, а це означає, що як тільки програма завантажується, так і всі NgModules, незалежно від того, чи потрібні вони негайно. Для великих додатків з великою кількістю маршрутів розгляньте відкладене завантаження — шаблон проектування, який завантажує NgModules за потреби. Відкладене завантаження допомагає зменшити початкові розміри пакетів, що, у свою чергу, допомагає скоротити час завантаження.

Відкладене завантаження – це процес завантаження компонентів, модулів або інших активів веб-сайту, якщо вони потрібні.

Оскільки Angular створює SPA (односторінковий додаток), усі його компоненти завантажуються одночасно. Це означає, що також може бути завантажено багато непотрібних бібліотек або модулів.

Для невеликої програми це буде добре. Але в міру зростання програми час завантаження буде збільшуватися, якщо все завантажувати відразу. Відкладене завантаження дозволяє Angular завантажувати компоненти та модулі, коли вони потрібні.

Бібліотеки Angular, такі як RouterModule, BrowserModule і FormsModule, є NgModules. Angular Material, який є стороннім інструментом, також є типом NgModule.

NgModule складаються з файлів і коду, пов’язаних з певним доменом або які мають подібний набір функціональних можливостей.Типовий файл NgModule оголошує компоненти, директиви, канали та служби.Він також може імпортувати інші модулі, які потрібні в поточному модулі.

2.13. Figma

Figma — це програма для розробки інтерфейсу, яка працює у браузері. Figma надає вам усі інструменти, необхідні для фази проектування проекту, включаючи векторні інструменти, які здатні повноцінно ілюструвати, а також можливості створення прототипів і генерування коду для передачі.

Незважаючи на те, що Figma базується на браузері, існує настільна версія для Windows і Mac OS. Імовірно, це програми обгортання, які запускають браузер всередині них, але незалежно від того, так це чи ні, Figma завжди працює онлайн в цих програмах.

Примітка: якщо ви втратите з'єднання, ви все одно можете продовжувати працювати над будь-яким документом, який ви вже відкривали.

Природа Figma «завжди онлайн» насправді забезпечує деякі з найбільших переваг інструменту. Однією з цих переваг є те, що Figma дозволяє співпрацювати в реальному часі. Ви та члени вашої команди можете одночасно ввійти в дизайн, одночасно вносячи в нього зміни. А той факт, що всі ці проекти зберігаються в Інтернеті, означає, що вам ніколи не доведеться турбуватися про те, що члени команди не синхронізуються з проектом. Останні зміни завжди зберігаються безпосередньо у файлі, і вам не потрібно турбуватися про передачу файлів між членами команди або передавання файлів на будь-яку сторонню платформу зберігання та з неї.

Це також означає, що ви можете перейти до дизайну одночасно з клієнтом. Тож навіть якщо ви перебуваєте окремо від клієнта, ви все одно можете провести сидячу нараду, на якій ви обидва дивитеся на одне й те саме. Клієнт може робити пропозиції, а ви можете реалізувати їх тут же.

Бібліотеки компонентів

Figma дозволяє створювати бібліотеки компонентів для повторного використання, до яких має доступ вся команда. Компоненти дають дизайнерам фору для будь-яких існуючих систем проектування, і коли компонент оновлюється в центральній бібліотеці, ці зміни вносяться в усі проекти для кожного.

Прототипування та виведення коду

Для створення прототипу ви можете створювати підключення та точки доступу у вашому дизайні, щоб ви могли імітувати, як користувач буде проходити через цей інтерфейс. На етапі кодування Figma може генерувати код SVG, CSS, а також код iOS і Android.

Це охопило аспекти співпраці Figma, але кращу частину Figma насправді складніше визначити. Це та кульмінація всіх дрібниць, які Figma робить дуже добре, і є їх довгий список. Ми зібрали цілу купу з них у серію під назвою Figma Tips & Tricks. Їх надто багато, щоб згадати тут, але деякі з моїх улюблених включають:

У Figma вузол на векторі може мати кілька сегментів, приєднаних до нього (більше, ніж просто два стандартних). Більш складні фігури, які ви можете створити за допомогою них, називаються «векторними мережами».

Figma дозволяє швидко імпортувати кілька зображень одночасно, розміщуючи їх саме там, де ви хочете.

Коли ви переміщуєте об’єкт по полотну у Figma, ви побачите його висоту та ширину, відображені на лінійках.

Скопіюйте код SVG безпосередньо з об’єкта в Figma, і навпаки, ви можете скопіювати код SVG з редактора коду та вставити його як графіку на полотно Figma.

2.14. GIT

Git — це безкоштовна розподілена система контролю версій з відкритим вихідним кодом, призначена для швидкої та ефективної роботи з усім, від малих до дуже великих проектів.

Git простий у освоєнні та має крихітний слід із блискавичною продуктивністю. Він перевершує інструменти SCM, такі як Subversion, CVS, Perforce і ClearCase, завдяки таким функціям, як дешеве локальне розгалуження, зручні області переходу та кілька робочих процесів.

Розгалуження та злиття

Функція Git, яка дійсно виділяє його серед майже всіх інших SCM, — це його модель розгалуження.

Git дозволяє і заохочує вас мати кілька локальних гілок, які можуть бути повністю незалежними одна від одної. Створення, об’єднання та видалення цих ліній розробки займають секунди.

Це означає, що ви можете робити такі дії, як:

Перемикання контексту без тертя. Створіть гілку, щоб випробувати ідею, зафіксуйте кілька разів, перейдіть до того місця, звідки ви розгалужувалися, застосуйте виправлення, перейдіть до місця, де ви експериментуєте, і об’єднайте його.

Кодові рядки на основі ролей. Майте гілку, яка завжди містить лише те, що йде на виробництво, іншу, в яку ви об’єднуєте роботу для тестування, і кілька менших для щоденної роботи.

Робочий процес на основі функцій. Створюйте нові гілки для кожної нової функції, над якою ви працюєте, щоб ви могли легко перемикатися між ними, а потім видаляйте кожну гілку, коли ця функція буде об’єднана у ваш основний рядок.

Одноразове експериментування. Створіть гілку, щоб поекспериментувати, зрозумійте, що вона не спрацює, і просто видаліть її, покинувши роботу, і ніхто більше її не побачить (навіть якщо ви тим часом натиснули інші гілки).

Відділення

Примітно, що коли ви натискаєте до віддаленого сховища, вам не потрібно натискати всі свої гілки. Ви можете поділитися лише однією зі своїх філій, кількома з них або всіма. Це, як правило, звільняє людей пробувати нові ідеї, не турбуючись про необхідність планувати, як і коли вони збиратимуться об’єднати їх або поділитися ними з іншими.

Існують способи зробити це за допомогою інших систем, але ця робота є набагато складнішою та схильною до помилок. Git робить цей процес

неймовірно простим і змінює спосіб роботи більшості розробників, коли вони його вивчають.

Висновок до розділу 2

Angular і Angular JS мають певні переваги, які можуть запропонувати розробникам, Angular займає перше місце в кількох категоріях. Крім того, динамічні веб та односторінкові програми Angular надзвичайно зручні для мобільних пристроїв. Код AngularJS, з іншого боку, не підтримує мобільні програми, і це, можливо, найбільша перевага Angular перед AngularJs.

Відкладене завантаження — це техніка в Angular, яка дозволяє асинхронно завантажувати компоненти JavaScript, коли активовано певний маршрут. Це покращує швидкість завантаження програми, розділяючи програму на кілька пакетів. Коли користувач переміщається по додатку, пакети завантажуються відповідно до потреб.

Основна роль ngOnInit полягає в тому, щоб надати сигнал про те, що Angular ініціалізував компонент і що користувачі можуть продовжувати роботу. З іншого боку, конструктор значно використовується для ініціалізації членів класу, але він не може виконати всю роботу. Це вигідно лише у випадку ін’єкції залежності та ініціалізації поля класу. При цьому компілятор повинен уникати написання роботи на конструкторі. ngOnInit є кращим місцем для на написання робочого коду, який необхідний під час створення екземпляра класу.

РОЗДІЛ 3. РОЗРОБКА ПРОДУКТУ

3.1. Розробка плану проекту

Для початку роботи, перш за все потрібно створити план розвитку проекту, а саме:

1) Створення дизайнерського макету:
   - Вибрати цвітову гаму;
   - Обрати шрифт та його розміри;
   - Створення іконок;
   - Створення зображень, відповідних до теми продукту;
   - Створення структури дизайну сайту.
2) Створення верстки сайту:
   - Розділити застосунок на умовні компоненти;
   - Визначити статичну частину застосунку;
   - Визначити динамічну частину застосунку;
   - Поетапне заповнення кожного з компонентів відповідним контентом за допомогою HTML і CSS:
     - — Створення верстки меню сайту;
     - — Створення блоку з відображення основної частини сайту(головне зображення та підпис;
     - —Створення блоку з історією парсингу;
     - —Створення блоку, який приймає дані для парсингу;
     - —Створення блоку з результатами парсингу;

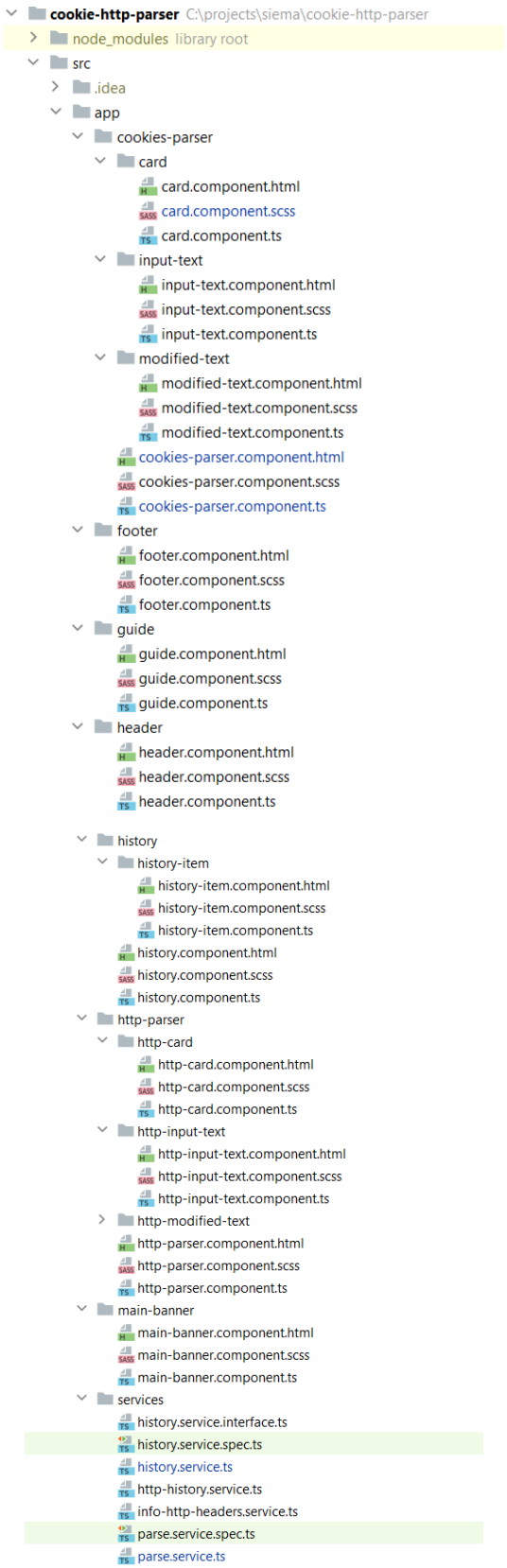
— Створення блоку з інструкцією;
— Створення блоку з інформацією `про нас` та контактну інформацію.

- Створення мобільної версії додатку.

3) Створення функціонала/логіки сайту:
   - Створення роутінгу;
   - Створення сервісів;
   - Створення функцій та методів;
   - Створення інтерфесів;
   - Створення юніт тестів;
   - Перевірка логіки на тести;
   - Прив’язування логіки до компонентів;
   - Прив’язування компонентів до верстки сайту;
   - Створення історії парсингу сайту;
   - Перевірка на юніт тести.
4) Тестування готового сайту та пошук помилок.

3.2. Структура проекту

Структура даного проекту виглядає як показано на рисункі(рис.3.1).


cookie-http-parser C:\projects\siema\cookie-http-parser
node_modules library root
src
.idea
app
cookies-parser
card
card.component.html
card.component.scss
card.component.ts
input-text
input-text.component.html
input-text.component.scss
input-text.component.ts
modified-text
modified-text.component.html
modified-text.component.scss
modified-text.component.ts
cookies-parser.component.html
cookies-parser.component.scss
cookies-parser.component.ts
footer
footer.component.html
footer.component.scss
footer.component.ts
guide
guide.component.html
guide.component.scss
guide.component.ts
header
header.component.html
header.component.scss
header.component.ts
history
history-item
history-item.component.html
history-item.component.scss
history-item.component.ts
history.component.html
history.component.scss
history.component.ts
http-parser
http-card
http-card.component.html
http-card.component.scss
http-card.component.ts
http-input-text
http-input-text.component.html
http-input-text.component.scss
http-input-text.component.ts
http-modified-text
http-parser.component.html
http-parser.component.scss
http-parser.component.ts
main-banner
main-banner.component.html
main-banner.component.scss
main-banner.component.ts
services
history.service.interface.ts
history.service.spec.ts
history.service.ts
http-history.service.ts
info-http-headers.service.ts
parse.service.spec.ts
parse.service.ts

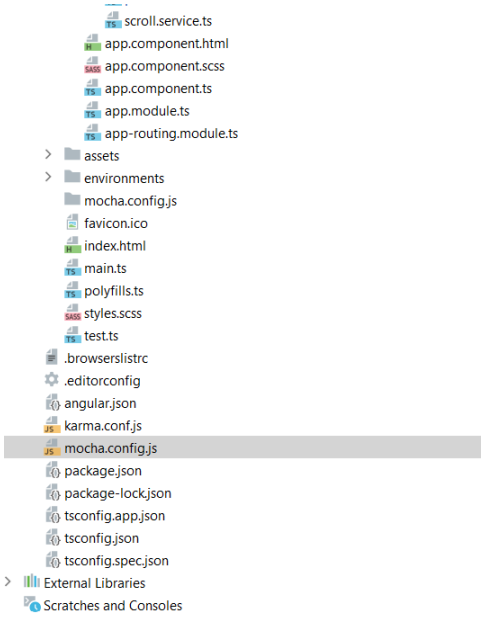


Рис.3.1. Структура проекту

- Cooke-parser – основна папка, де находиться весь проект;
- Node_modules – папка з різними модулями та настройками ангуляр проекту;
- Src – папка з компонентами та сервісами:
  - Styles.scss – файл з основними стилями, які приміняються на всьому проекті.
  - Main.ts – файл звідки починається запуск програми;
  - Index.html – основна інформація про файл(рис3.2), в ньому вказані шрифти, які були підключені до сайту, основні мета теги, назва сайту і вказаний app-root, який вміщує в себе більше компонентів.

```html
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/html">
  <head>
    <meta charset="utf-8" />
    <title>Dev Helpers</title>
    <base href="/" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" type="image/x-icon" href="favicon.ico" />
    <link rel="preconnect" href="https://fonts.gstatic.com" />
    <link
      href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap"
      rel="stylesheet"
    />
    <link
      href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet"
    />
    <link rel="preconnect" href="https://fonts.gstatic.com" />
    <link
      href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap"
      rel="stylesheet"
    />
    <link
      href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet"
    />
  </head>

  <body class="mat-typography">
    <app-root></app-root>
  </body>
</html>
```

Рис.3.2. index.html

— Assets – папка з файлами зображень;

— App – папка з компонентами та сервісами:

- App.modules.ts – файл з основними імпортами, з модулями, де ми вказуємо з якого файлу слід починати зчитувати інформацію(bootstrap), показуємо компоненти, які включені в наш модуль, показуємо підключені модулі та сервіси(рис.3.3).

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { MatButtonModule } from '@angular/material/button';
import { FormsModule } from '@angular/forms';
import { MatExpansionModule } from '@angular/material/expansion';
import { AppComponent } from './app.component';
import { CardComponent } from './cookies-parser/card/card.component';
import { InputTextComponent } from './cookies-parser/input-text/input-text.component';
import { CookiesParserComponent } from './cookies-parser/cookies-parser.component';
import { HttpParserComponent } from './http-parser/http-parser.component';
import { HeaderComponent } from './header/header.component';
import { GuideComponent } from './guide/guide.component';
import { FooterComponent } from './footer/footer.component';
import { MainBannerComponent } from './main-banner/main-banner.component';
import { HistoryComponent } from './history/history.component';
import { HistoryItemComponent } from './history/history-item/history-item.component';
import { ModifiedTextComponent } from './cookies-parser/modified-text/modified-text.component';
import { HttpCardComponent } from './http-parser/http-card/http-card.component';
import { HttpInputTextComponent } from './http-parser/http-input-text/http-input-text.component';
import { HttpModifiedTextComponent } from './http-parser/http-modified-text/http-modified-text.component';

import { HistoryService } from './services/history.service';
import { ParseService } from './services/parse.service';
import { scrollService } from './services/scroll.service';

@NgModule({
  declarations: [
    AppComponent,
    CardComponent,
    InputTextComponent,
    CookiesParserComponent,
    HttpParserComponent,
    HeaderComponent,
    GuideComponent,
    FooterComponent,
    MainBannerComponent,
    HistoryComponent,
    HistoryItemComponent,
    ModifiedTextComponent,
    HttpCardComponent,
    HttpInputTextComponent,
    HttpModifiedTextComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatButtonModule,
    FormsModule,
    MatExpansionModule,
  ],

  providers: [HistoryService, ParseService, scrollService],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Рис.3.3. Module.ts

- App.components(ts, css, html)
- App-routing.ts – вказує на переміщення по нашій сторінці, куди потрібно перейти при клікі на посилання.

- Всі інші папки зображують окремі компоненти, які входять до нашого проекту, крім сервісів, сервіси зберігають в собі логіку та потрібні дані.

- Karma and mocha – тести;
- Angular.json – файл, в якому зображено основну залежності, які були підключені по типу стилів.
- Package.json – файл, де вказано версії файлів та технологій, вказує основні команді для CLI.

3.3. Створення дизайнерського макету

Спочатку було обрано графічний редактор, існує велика кількість графічних редакторів, але я обрав Figma. Після обрання графічного редактору я вирішив обрати кольорову схему, де були такі кольори:

- #FFF3EE(Для заднього фону):

- #FFF (Білий)(для заднього фону)
- #000 (Чорний)(для шрифту)
- #FF4D00(для кнопок):

Після створення цвітової схеми було вирішено обрати потрібні розміри для шрифту, а саме:

- H1 - 48px;
- H2 – 34px;
- H3 – 20px;
- P - 12px;
- A - 18px.

Обравши потрібний розмір шрифту, було створено різні маленькі іконки та великі зображення:

1) Зображення печива(рис.3.4):

Рис.3.4. Зображення печива

2) Зображення HTTP(рис.3.5):

Рис.3.5. Зображення HTTP

3) Іконки(рис.3.6):

Рис.3.6. Іконки

3.4. Створення верстки сайту

Для початку, слід визначити на які компоненти буде поділено ваш додаток(рис.3.7):

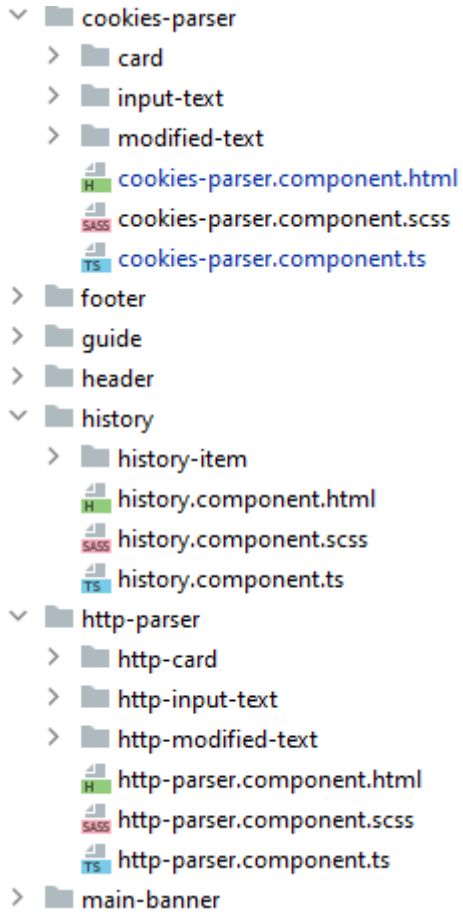


Рис.3.7. Компоненти сайту

Компоненти сайту:

- Header – меню сайту(рис.3.8), створений за допомогою display: flex, вміщує в себе роутинг по веб сторінці.

Рис.3.8. Меню сайту

- Main-banner – головний блок сайту(рис3.9) – показує користувачу основну інформацію про сайт

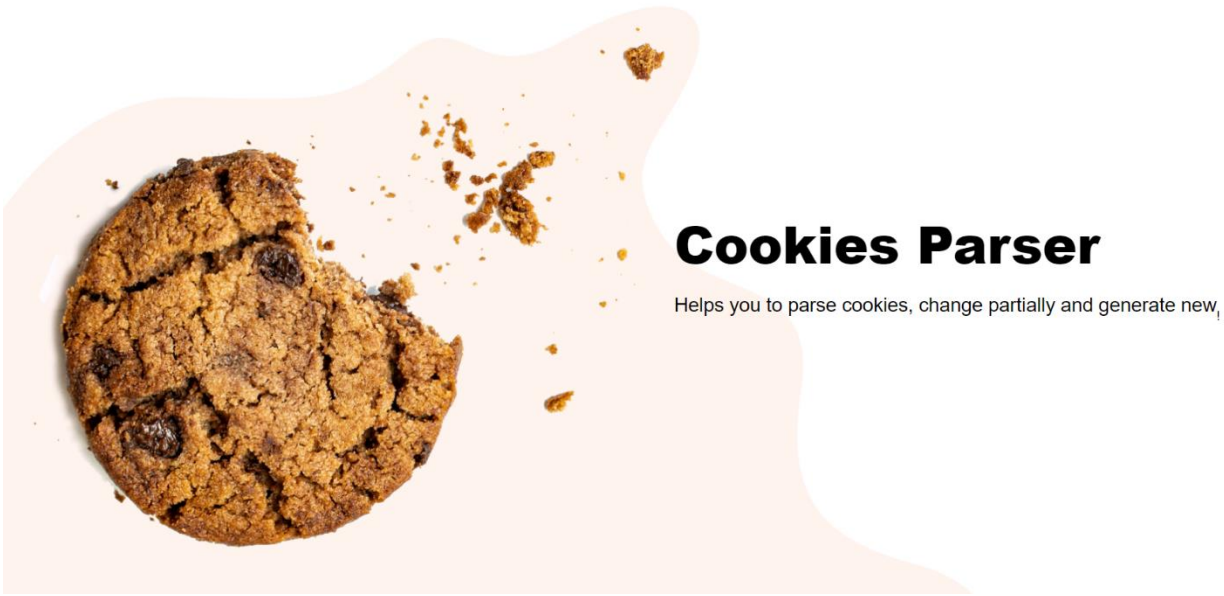


Рис.3.9. Головний блок сайту(cookie)

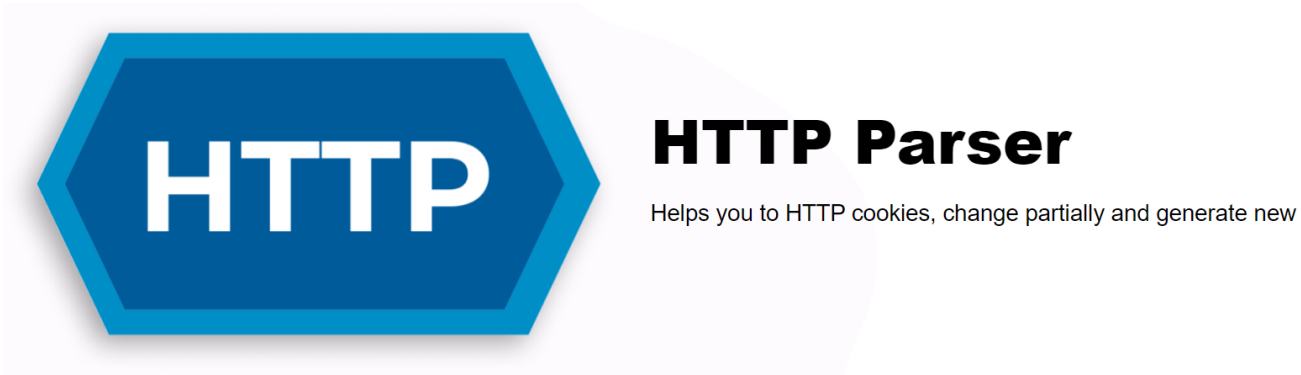


Рис.3.10. Головний блок сайту(Http)

- History – блок з історією(рис.3.11), при наведені на історію вона міняє колір, якщо історія обрана, то вона має другий колір, існує полоса прокрутки для ПК, а для мобільної версії немає, щоб користувачам було простіше користуватись даною сторією.

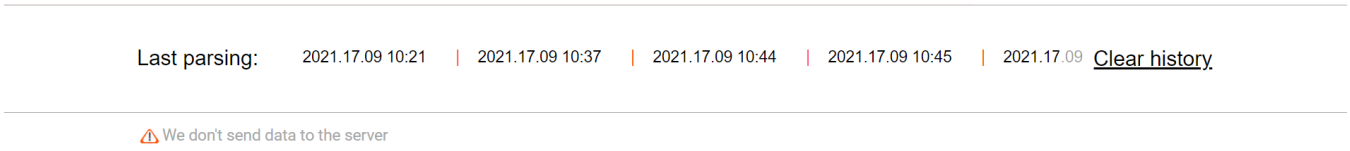


Рис.3.11. Історія

- Cookie-parser/http-parser - блок з вводом даних для подальшого парсування даних(рис.3.12).

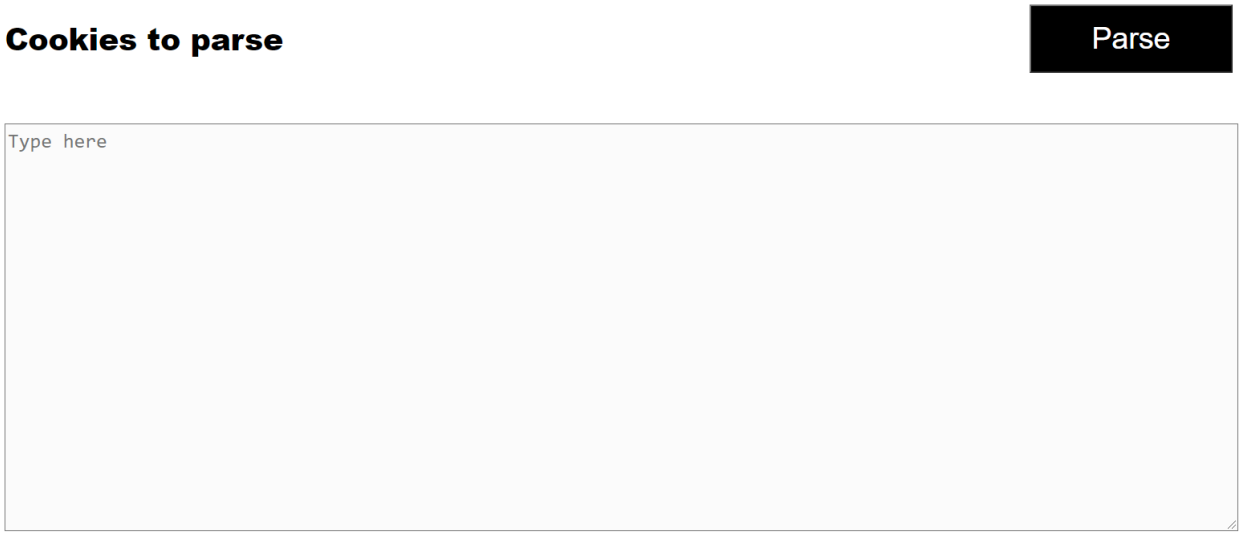


Рис.3.12. Блок з вводом даних

- Guide – блок з інструкцією(рис) щодо користування парсером, на ньому зображений хедер та вставлене відео, яке вкладене до ютубу, добавлено до сайту за допомогою спеціального тега iframe(рис.3.13).

Guide

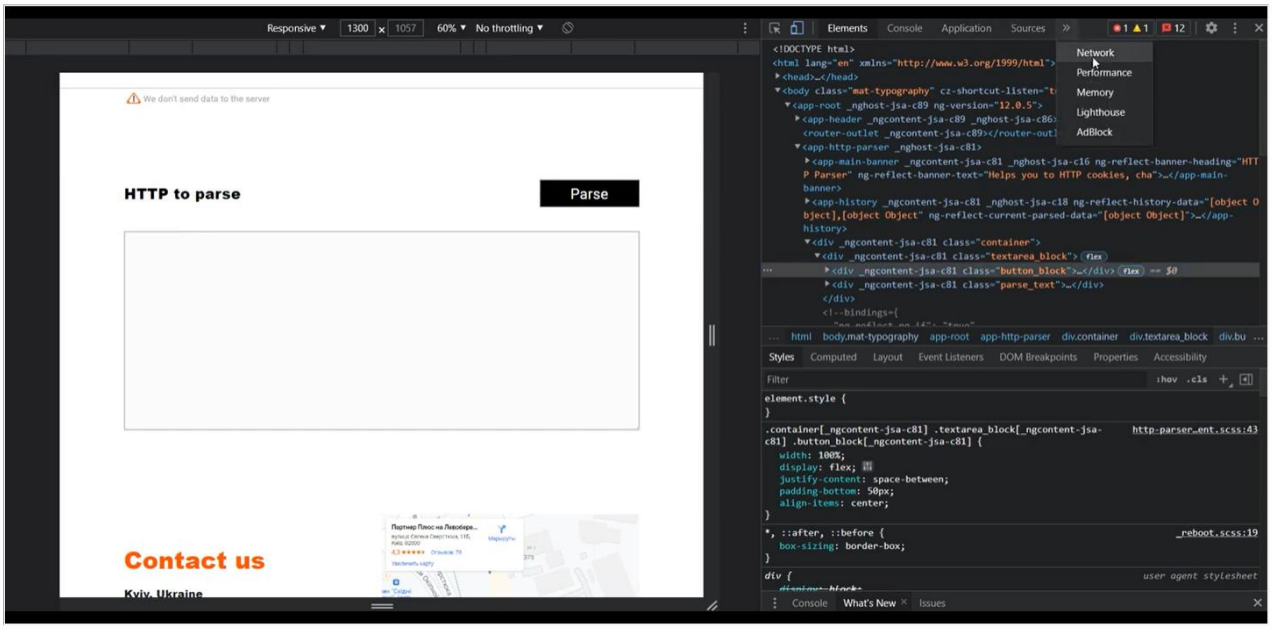


Рис.3.13. Блок з інструкцією

```
<div class="container">
  <div class="all-guids">
    <h4>Guide</h4>
    <iframe
      src="https://www.youtube.com/embed/F7cZFa8cRtc?fs=1&autoplay=1&loop=1"
      playlist="VIDEO_ID"
      allow=" autoplay; accelerometer ; clipboard-write; encrypted-media; gyroscope; picture-in-picture"
      allowfullscreen
    ></iframe>
  </div>
</div>
```

Рис.3.14. Тег Iframe

- Footer – блок з інформацією про розробника сайту і про сайт(рис.3.15), карта добавлена за допомогою тегу iframe.

About

This parser was created for developers who need to parse data about HTTP headers and cookies. Here you can change your data by keys and values, also in the http parser you can see what the different headers mean and try to change your value in the cookie from base64 to string.

Contact us

Kyiv, Ukraine

Yevher

contact-de@company.de

063 00 00 000

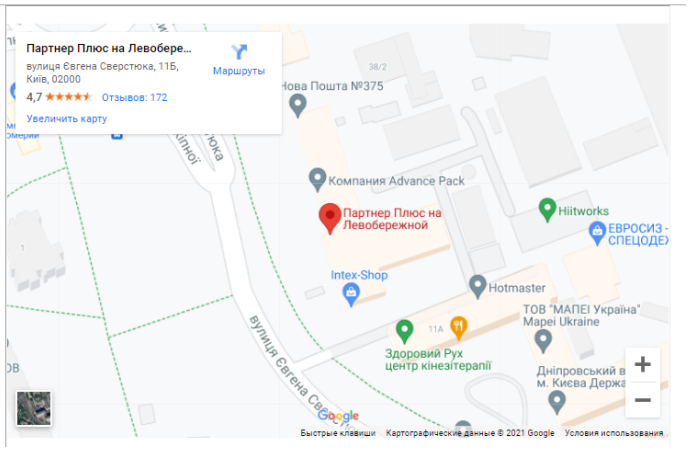


Рис.3.15. Тег з інформацією про сайт

- Додаткові компоненти(динамічні) – компоненти, які при натискані на кнопку приходять на зміну ввода даних(рис.3.16)

Cookies to parse

New parsing

__Secure-3PSIDCC=AJi4QfGwjDqlVo6xdNSR2rrCp4zk7ux0QL4migBZuqDXlW1DW2DR4xfsUfkFRdD9eYXh5lhWHtGS; expires=Sat, 17-Sep-2022 07:21:12 GMT; path=/; domain=.google.com; Secure; HttpOnly; priority=high; SameSite=none strict-transport-security: max-age=31536000

Modified Cookies

Parsed cookie data has been changed

Cancel modification

__Secjbkjbkbkfgdgure-3PSIDCC=AJwegsgdfdyкрkysfsdsegqrqwrqwrqwresfsssssssssi4QfGwjDqlVo6xdNSR2rrCp4zk7ux0QL4migBZuqDXlW1DW2DR4xfsUfkFRdD9eYXh5lhWHtGS; expires=Sat, 17-Sep-2022 07:21:12 GMT; path=/; domain=.google.com; Secure; HttpOnly; priority=high;;

Рис.3.16. Змінені дані

Рис.3.17. Результат парсингу

Після створення декстопної версії було створено мобільну версію сайту(рис):

Last parsing: 2021.17.09 10:21 | 2021.17.09 10:37 | Clear history

We don't send data to the server

Cookies to parse New parsing

__Secure-3PSIDCC=AJi4QfGwjDqlVo6xdNSR2rrCp4zk7ux0QL4migBZuqDXlW1DW2DR4xfsUfkFRdD9eYXh5lhWHtGS; expires=Sat, 17-Sep-2022 07:21:12 GMT; path=/; domain=.google.com; Secure; HttpOnly; priority=high; SameSite=none strict-transport-security: max-age=31536000

Results:

__Secjbkjbkbkfgdgure-3PSIDCC=AJwegsgdfdyкрkysfsdsegqrqwrqwrqwresfsssssssssi4QfGwjDqlVo6xdNSR2rrCp4zk7ux0QL4migBZuqDXlW1DW2DR4xfsUfkFRdD9eYXh5lhWHtGS

expires=Sat, 17-Sep-2022 07:21:12 GMT

path=/

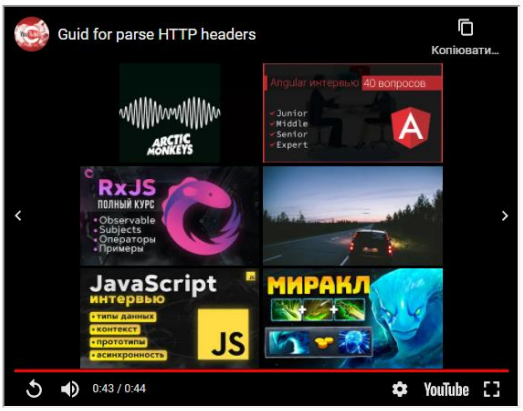
Key identified

path

Value identified

/

Guide



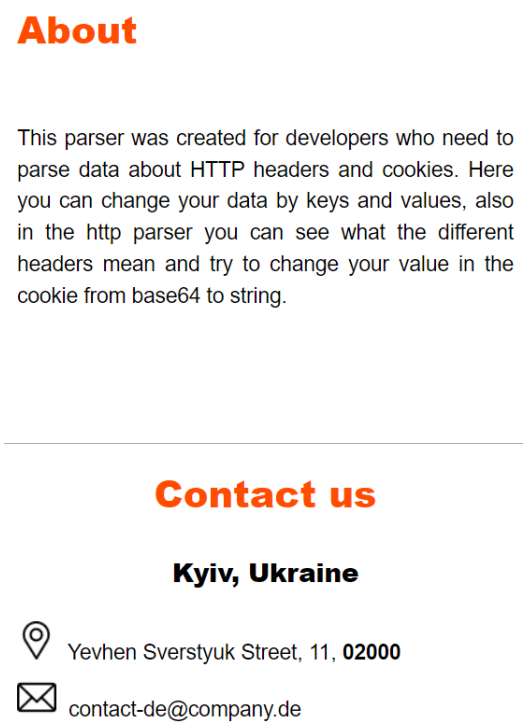


Рис.3.18. Мобільна версія сайту

3.5. Створення функціоналу

Для початку було створено роутинг для переходу між компонентами(рис.3.20), перехід відбувається між сторінками cookie і http парсера та між елементами історії, для цього був створений масив об’єктів який вміщує в себе шлях до компонента та сам компонент. При цьому шлях потрібно добавляти в посилання, які написані в верстці файлу(рис.3.21)

```
let routes: Routes;
routes = [
  { path: '', redirectTo: 'cookies-parser', pathMatch: 'full' },
  {
    path: 'cookies-parser',
    component: CookiesParserComponent
  },
  {
    path: 'cookies-parser/:id',
    component: CookiesParserComponent,
    pathMatch: 'full'
  },
  {
    path: 'http-parser',
    component: HttpParserComponent
  },
];
```

Рис.3.20. Роутінг

```
<nav class="menu-link">
  <ul>
    <li *ngFor="let item of menuRoutes">
      <a [routerLink]="item.path" routerLinkActive="active">
        {{ item.label }}
      </a>
    </li>
  </ul>
</nav>
```

Рис.3.21. Додавання шляху до посилань

Після створення роутингу для переходу між сторінками було створено сервіси, які доповнюють сайт динамікою та функціоналом. Створення логіки для кукі парсера за допомогою різних функцій. По-перше приходять дані, які розподілені по знаку “;”(рис.3.22), тому потрібно зробити розподіл даних по цьому знаку за допомогою стандартної функції для розбиття split(). Після розбиття даних отримуємо масив рядків, які вміщують в себе два слова які розподілені знаком “=”, тому ми робимо розбиття по цьому ж знаку і вертаємо в результаті масив об’єктів. Об’єкти вміщують в себе два ключі: ключ і значення, які мають свої дані, такі дії проводяться і з http парсером, але там різниця в тому, що розбиття проходе через знак перенесення рядку та через знак “:”(рис.3.23). Наприклад:

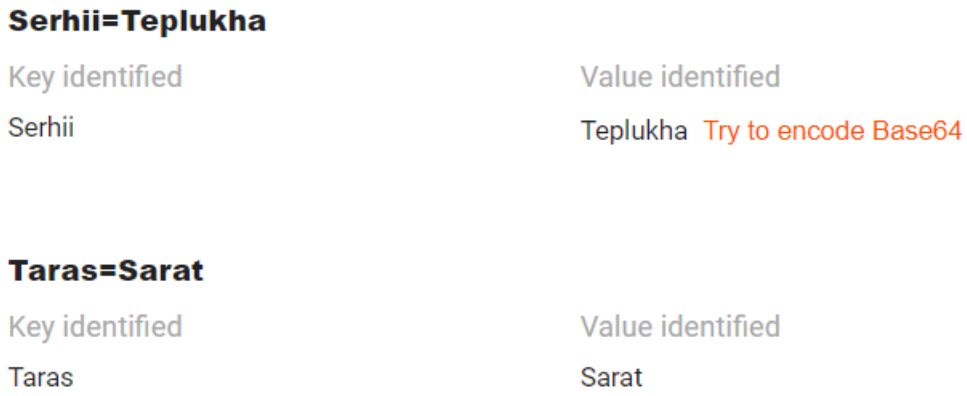


Рис.3.22. Розбиття кук

Рис.3.23. Розбиття http заголовку

Проблема створення даної логіки була в тому, що існує багато тестів, які потрібно завершити успішно, тому було створено велику кількість тестів. Щоб всі тести пройшли перевірку потрібно було виправляти та добавляти логіку для парсингу. Наприклад були можливі помилки при розбитю даних або в виходу даних чи фільтрувати дані(рис3.24), щоб вхідні дані не булі пусті.

```
.filter<ICookiesNode | any>((item: any): item is ICookiesNode => {
  return item !== null;
});
```

Рис.3.24. Фільтрування даних

Написання тестів(рис):

```
describe( description: 'Parse cookies', specDefinitions: () => {
  // the tests container
  it( expectation: 'should be length < 10"', assertion: () => {
    // the single test
    const test = new ParseService(); // this will be your class
    const res = test.parseCookies(
      cookieToParse: '22-Jul-2022 10:08:51 GMT; path=/; domain=.google.com; priority=high'
    );
    expect(res).to.not.have.lengthOf.above( value: 10);
  });
  it( expectation: 'shoulde be array when cookieToParse= array', assertion: () => {
    // the single test
    const test = new ParseService(); // this will be your class
    const res = test.parseCookies(
      cookieToParse: '22-Jul-2022 10:08:51 GMT; path=/; domain=.google.com; priority=high'
    );
    expect(res).to.deep.equal( value: [
      { identifiedKey: '22-Jul-2022 10:08:51 GMT', identifiedValue: undefined },
      { identifiedKey: ' path', identifiedValue: '/' },
      { identifiedKey: ' domain', identifiedValue: '.google.com' },
      { identifiedKey: ' priority', identifiedValue: 'high' },
    ]);
  });

        it( expectation: 'shoulde be trimed ', assertion: () => {
          // the single test
          const test = new ParseService(); // this will be your class
          const res = test.parseCookies( cookieToParse: 'name=fdf;sfdfs=ff; ');
          expect(res.length).to.equal( value: 2);
        });
        it( expectation: 'null should NOT be removed ', assertion: () => {
          // the single test
          const test = new ParseService(); // this will be your class
          const res = test.parseCookies( cookieToParse: 'null;sfdfs=ff; ');
          expect(res.length).to.equal( value: 2);
        });
        it( expectation: 'should be [] when cookies ""', assertion: () => {
          // the single test
          const test = new ParseService(); // this will be your class
          const res = test.parseCookies( cookieToParse: '');
          expect(res).to.deep.equal( value: []);
        });
```

Рис.3.25. Написання тестів

Після написання та перевірки сервісів проходить зв’язуванні логіки з шаблоном. Дані передаються за допомогою {{дані}}. Це динамічна передача даних(рис.3.26).

```
<div class="container">
  <div class="backround-img">
    <div class="main-banner">
      <div class="clear-place"></div>
      <div class="banner-text">
        <h3>{{ bannerHeading }}</h3>
        <p>{{ bannerText }}</p>
        <div class="image-cookie">
          <img src="assets/cookie.png" />
        </div>
      </div>!
    </div>
  </div>
</div>
```

Рис.3.26. Динамічна передача даних

За допомогою ngIf ми можемо скривати дані та відображати. За допомогою ngFor ми можемо перебирати дані з нашого масиву(рис.3.27).

```
<ng-container *ngIf="this.origNode !== this.modifiedNode">
  <div class="container" >
    <div class="modified-block" >
      <span class="cookies-parser"> Modified Cookies </span>
      <button (click)="onClickCancel()">Cancel modification</button>
    </div>
    <p class="changed-text">Parsed cookie data has been changed</p>
  </div>
  <div class="container" >
    <div class="modified" >
      <div class="modified-text" >
        <app-modified-text
          *ngFor="let node of currentParsedData.modifiedNodes"
          [node]="node"
        >
        </app-modified-text>
      </div>
    </div>
  </div>
</ng-container>
```

Рис.3.27. Передача різних типів даних

Під час прив’язки даних було створено різні інтерфейси для типізації даних(рис3.28).

```
export interface IHistoryItem {
  id: number | null;
  dateTime: string;
  originalNodes: ICookiesNode[];
  cookiesToParse: string;
  modifiedNodes: ICookiesNode[];
}

export interface ICookiesNode {
  identifiedKey: string;
  identifiedValue: string | undefined;
}
export interface IHTTPHeaders {
  key: string;
  descriptionKey: string;
}
```

Рис.3.28. Створення інтерфейсів

Зв’язування логіки та компонентів(рис.3.29).

```
bannerHeading: string = 'Cookies Parser';
bannerText: string =
  'Helps you to parse cookies, change partially and generate new';
currentParsedData: IHistoryItem;
historyData: IHistoryItem[] = [];
modifiedNode: string;
origNode: string;
id: number;

private onDestroy$ = new Subject();

@Input() item: IHistoryItem;

constructor(
  private parseService: ParseService,
  private historyService: HistoryService,
  private activatedRoute: ActivatedRoute
) {}

dataParseHidden(): boolean {
  return this.currentParsedData.id === null;
}
```

```
ngOnInit(): void {
  this.historyData = this.historyService.getCookiesHistory();

  this.activatedRoute.params
    .pipe(takeUntil(this.onDestroy$))
    .subscribe( next: (e : Params ) => {
      const id = parseInt(e.id);
      if (isNaN(id)) {
        this.currentParsedData = this.historyService.getInitialHistoryItem();
        return;
      }
      const historyItem = this.historyService.getHistoryById(id);
      if (historyItem) {
        this.currentParsedData = historyItem;
      } else {
        this.currentParsedData = this.historyService.getInitialHistoryItem();
      }
    });
}
```

Рис.3.29. Зв’язування логіки та компонентів

Прив’язування логіки до кнопок(рис.3.30).

```
onClickCancel() {
  this.currentParsedData.modifiedNodes = this.currentParsedData.originalNodes;
  this.currentParsedData.originalNodes = this.parseService.parseCookies(
    this.currentParsedData.cookiesToParse
  );
  if (this.currentParsedData.id !== null) {
    const itemFromHistory = this.historyService.getHistoryById(
      this.currentParsedData.id
    );
    if (itemFromHistory)
      this.historyService.updateHistoryItem(itemFromHistory);
  }
}

ngOnDestroy() {
  this.onDestroy$.next();
  this.onDestroy$.complete();
}

onClickNewParse(item: IHistoryItem) {
  this.currentParsedData = this.historyService.getInitialHistoryItem();
}

onSelectHistoryItem(item: IHistoryItem): void {
  this.currentParsedData = item;
  this.modifiedNode = JSON.stringify(item.modifiedNodes);
  this.origNode = JSON.stringify(item.originalNodes);
}
```

```
onClickClearHistory() {
  this.historyService.clearCookiesHistory();
  this.historyData = this.historyService.getCookiesHistory();
}

onClickParseCookies(): void {
  if (this.currentParsedData.cookiesToParse.trim() !== '') {
    const parseData = this.parseService.parseCookies(
      this.currentParsedData.cookiesToParse
    );
    const parseDataMod = this.parseService.parseCookies(
      this.currentParsedData.cookiesToParse
    );
    this.currentParsedData = this.historyService.createHistoryItem(
      parseData,
      this.currentParsedData.cookiesToParse,
      parseDataMod
    );
    this.historyService.addNewCookiesHistory(this.currentParsedData);
    this.historyData = this.historyService.getCookiesHistory();
  }
}
```

Рис.3.30. Зв’язування логіки та кнопок сайту

Після цього було створено історію сайту, яка вміщує в себе елементи історії, кожна історія – це новий елемент масиву, тому вона була винесена в окремиий компонент(рис.3.31-3.35). Під час створення історії було два вибору збереження даних:

1) Зберігати дані де кожен ключ відповідає одному значенню(проблема заключається в видалені всіх елементів)
2) Зберігати дані де один ключ дорівнює всім значенням.

   Було обрано другий варіант, де одному ключу – всі значення.

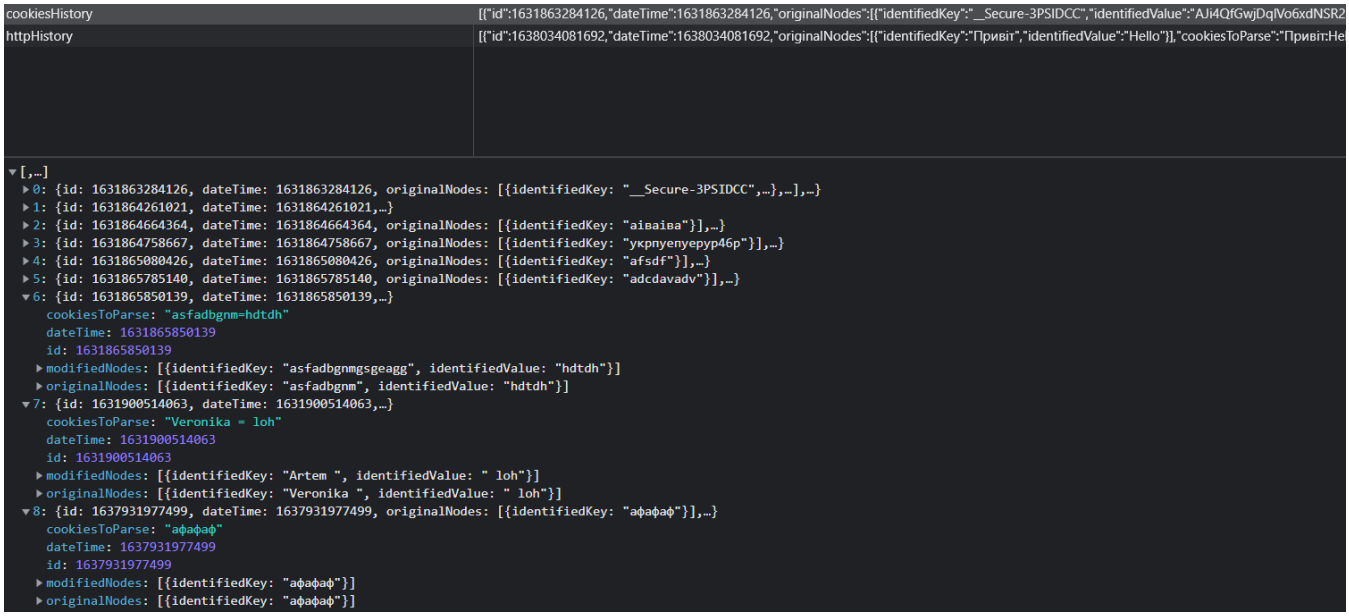


Рис.3.31. Локальне сховище

```
<div class="container" *ngIf="historyData.length !== 0">
  <div class="history-elements">
    <ng-container>
      <div class="history-block">
        <span>Last parsing: </span>
      </div>
      <div class="history-items">
        <app-history-item
          (selectHistoryItem)="onClickSelectHistory($event)"
          *ngFor="let item of historyData"
          [currentParsedData]="currentParsedData"
          [item]="item"
        >
        </app-history-item>
      </div>
      <button (click)="onClickClearHistory()">Clear history</button>
      <p *ngIf="lengthItems">Too much, pls clear your history</p>
    </ng-container>
  </div>
</div>
<div class="container-data" *ngIf="historyData.length !== 0">
  <div class="data-send">
    <img src="assets/warning.png" /><span>We don't send data to the server</span>
  </div>
</div>
```

Рис.3.32. Основний блок історії

```
<span class="description-info">
  <span class="description-info-hidden">
    <button (click)="deleteThisItemInHistory()">&#10006;</button>
  </span>
  <span (click)="onClickItemHistory()">
    <a [routerLink]="['/', currentRouteSegment, item.id]" routerLinkActive="active">
      {{ item.dateTime | date: format: "yyyy.d.MM h:mm" }}</a
    >
  </span>
</span>
```

Рис.3.33. Кожний елемент історії

```
    private httphistoryService: HttpHistoryService,
    private router: Router,
    private activatedRoute: ActivatedRoute
  ) {
    this.activatedRoute.url.pipe(
      takeUntil(this.onDestroy$)
    ).subscribe( next: (u :UrlSegment[] ) => {
      if (u[0]) {
        this.currentRouteSegment = u[0].path;
      }
    })
  }

  deleteThisItemInHistory(): void {
    this.historyService.removeHistoryItem(this.item);

    this.httphistoryService.removeHTTPHistoryItem(this.it
  }

  onClickItemHistory(): void {
    this.selectHistoryItem.emit(this.item);
  }

  ngOnDestroy() {
    this.onDestroy$.next();
    this.onDestroy$.complete();
  }
```

Рис.3.34. Зв’язування компонента та логіки історії

Прописання тестів для історії(рис.3.35)

```
describe( description: 'History service',  specDefinitions: () => {
  beforeEach( action: () => {
    historyService.clearCookiesHistory();
  });

  it( expectation: 'should be empty history',  assertion: function () {
    expect(historyService.getCookiesHistory()).to.be.empty;
  });
  it( expectation: 'should be one historyItem after addCookiesHistoryItem',  assertion: function () {
    historyService.addNewCookiesHistory(historyItem);
    expect(historyService.getCookiesHistory()).to.be.deep.equal( value: [
      {
        ...historyItem,
      },
    ]);
  });

  it( expectation: 'should be undefined when we dont have history',  assertion: function () {
    historyService.addNewCookiesHistory(historyItem);
    expect(historyService.getHistoryById( id: 1235)).to.be.an( type: 'undefined');
  });
  it( expectation: 'should be [] when we choose clear history',  assertion: function () {
    historyService.addNewCookiesHistory(historyItem);
    historyService.addNewCookiesHistory(historyItemCopy);
    expect(historyService.clearCookiesHistory()).to.be.an( type: 'undefined');
    const clearCookies = historyService.clearCookiesHistory();
  });
```

Рис.3.35. Тести для історії

Висновок до розділу 3

Одним з перших завдань при розробці веб сайту є створення його плану. При створені плану обов’язково потрібно додавати тестування, адже основним напрямком в створені сайту є тестування.

Модульне тестування — це техніка тестування окремого модуля чи компонента чи простої функції, щоб дізнатися, чи зможемо ми досягти всіх необхідних варіантів використання та сценаріїв (позитивних чи негативних).

Модульне тестування допомагає нам зменшити кількість помилок/проблем під час тестування як для нещодавно впровадженого коду, так і для розширених функцій.

Існує багато інструментів для написання та виконання модульного тестування. Ми можемо реалізувати модульне тестування за допомогою Jasmine, Cypress, Jest та багатьох інших.

Тільки після позитивно пройденого тестування можна зв’язувати компоненти та логіку сайту.

ВИСНОВКИ

Розглянувши різні технології, методи та інструменти для розробки веб-сайту не можна сказати, що одні технології мають велику перевагу над іншими. Всі технології мають свою переваги та недоліки, деякі створені для великого навантаження, для складних веб-сайтів, а деякі навпаки – для малих, односторінкових.

Однак, в наш всі технології намагються покращити свої показники в швидкодії та в простоті вивчення.

Під час створення проекту даного типу, краще використовувати такі інструменти та технології: WebStorm/Visual Studio, HTML, CSS, JavaScript/TypeScript, GIT, Angular/React/Vue, Figma/Adobe photoshop, RxJS, Bootstrap/Material/PrimeNG.

При створені сайту слід притримуватись плану:

1. Розробити структури сайту.
2. Створити дизайн сайту.
3. Розроботи верстку сайту.
4. Розробити основний функціонал та логіку сайту.
5. Створити тестів для логіки.
6. Прив’язати логіку до верстки.
7. Тестувати сайт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WebSite [Електронний ресурс]. Режим доступу: https://www.techopedia.com/definition/5411/website (дата звернення 02.11.2021) – Назва з екрана.
2. Browser [Електронний ресурс]. Режим доступу: https://www.computerhope.com/jargon/b/browser.htm (дата звернення 02.11.2021) – Назва з екрана.
3. JavaScript [Електронний ресурс]. Режим доступу: https://www.hackreactor.com/blog/what-is-javascript-used-for (дата звернення 02.11.2021) – Назва з екрана.
4. TypeScript [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/ (дата звернення 02.11.2021) – Назва з екрана.
5. JS vs TS [Електронний ресурс]. Режим доступу: https://www.guru99.com/typescript-vs-javascript.html (дата звернення 02.11.2021) – Назва з екрана.
6. Framework [Електронний ресурс]. Режим доступу: https://techmonitor.ai/what-is/what-is-a-framework-4945801 (дата звернення 02.11.2021) – Назва з екрана.
7. Cookie [Електронний ресурс]. Режим доступу: https://www.allaboutcookies.org/cookies/ (дата звернення 02.11.2021) – Назва з екрана.
8. Cookie [Електронний ресурс]. Режим доступу: https://searchsoftwarequality.techtarget.com/definition/cookie (дата звернення 02.11.2021) – Назва з екрана.
9. LocalStorage [Електронний ресурс]. Режим доступу: https://blog.logrocket.com/localstorage-javascript-complete-guide/ (дата звернення 02.11.2021) – Назва з екрана.

10. SessionStorage [Електронний ресурс]. Режим доступу: https://learn.javascript.ru/localstorage (дата звернення 02.11.2021) – Назва з екрана.
11. HTTP headers [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Headers (дата звернення 02.11.2021) – Назва з екрана.
12. Bootstrap [Електронний ресурс]. Режим доступу: https://getbootstrap.com/ (дата звернення 02.11.2021) – Назва з екрана.
13. WebStorm [Електронний ресурс]. Режим доступу: https://www.jetbrains.com/ru-ru/webstorm/ (дата звернення 02.11.2021) – Назва з екрана.
14. All of Angular [Електронний ресурс]. Режим доступу: https://angular.io/ (дата звернення 02.11.2021) – Назва з екрана.
15. Promise [Електронний ресурс]. Режим доступу https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261 (дата звернення 02.11.2021) – Назва з екрана.
16. AngularJs [Електронний ресурс]. Режим доступу https://angularjs.org/ (дата звернення 02.11.2021) – Назва з екрана.