МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

Факультет економіки і підприємництва

Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2023 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АДМІНІСТРУВАННЯ ДОСТУПУ ДО ВЕБ-САЙТІВ

Виконав – студент 2 курсу, групи 21м-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
Манзенко Володимир Сергійович

Науковий керівник – к.е.н., доцент
Концеба Сергій Михайлович

Умань – 2023

ЗМІСТ

ВСТУП

РОЗДІЛ 1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження актуальності

1.2 Огляд існуючих рішень

1.3 Постановка задачі

РОЗДІЛ 2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Основні положення

2.2 Вибір технологій для розробки

2.3 Концепція блокування

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Структурна схема додатку

3.2 Реалізація авторизації за допомогою сервісу Auth0

3.3 Реалізація сторінки налаштувань

3.4 Реалізація розширення для браузеру

3.5 Реалізація відображення повідомлення, що веб-ресурс заблокований

3.6 Тестування розширення

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

ВСТУП

У сучасному світі стрімко зростає кількість нових технологій, з'являється безліч веб-ресурсів, які полегшують наше життя, наприклад, соціальні мережі, відеохостинги, стрімінгові платформи та інші. З початком епідемії коронавірусу дуже багато людей почали працювати або навчатися з дому віддалено, а з початком війни ця кількість в Україні ще дужче збільшилась. Психологія людини так створена, що людині важно працювати або навчатися з дому, тому що дім асоціюється з відпочинком та сім'єю, а не з роботою або навчанням. Тому працюючи з дому людина має багато відволікаючих факторів, які заважають працювати. Людина починає займатися прокрастинацією. На даний момент не існує єдиного конкретного визначення поняття прокрастинації. У зв'язку з постійним оновленням досліджень феномену прокрастинації відбувається не спрощення чи доповнення поняття, а створюється розгалуження ідей стосовно даного феномену[1] Було визначено, що в сучасному тлумачному словнику термін «прокрастинація», який походить від латинських слів «pro» – попереду і «crastinus» – завтрашній, позначає «відкладання» або «зволікання», які часто використовуються як синоніми у науковій літературі, присвяченій даній проблемі [2]

Вільний доступ до будь-якого веб-ресурсу спокушає людину витрачати енергію на це, замість того, щоб сфокусуватися на роботі або навчанні та з максимальною ефективністю виконати завдання. Тому актуальною задачею є створення системи доступу до веб-ресурсів, яка допоможе людині контролювати свою інтернет активність, шляхом блокування веб-ресурсів, які вона додасть до чорного списку.

Шляхів розв'язання цієї проблеми може бути декілька: блокування веб-ресурсів у налаштуваннях маршрутизатора, блокування використовуючи функції, які пропонує компанія Google або створення власного розширення для браузера.

Найбільш доцільним методом блокування є створення розширення. Адже навіть та людина, яка погано розбирається у комп’ютерах зможе легко обмежити доступ до веб-ресурсів та почати використовувати час максимально ефективно.

Робота складається з трьох розділів: інформаційний огляд з аналізом існуючих рішень, вибір методу рішення та практична реалізація. У кінці наведені висновки, де описано наскільки дане рішення виявилось ефективним в порівнянні з існуючими.

РОЗДІЛ 1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження актуальності

Метою даної роботи є створення системи яка буде блокувати (обмежувати) доступ до обраних веб-ресурсів. Система представлятиме собою розширення для браузера, яке можна завантажити з Інтернет-магазину Chrome, вона, базуючись на параметрах, які вкаже користувач, буде здійснювати обмеження доступу до веб-сторінки.

На жаль, через війну та коронавірус люди змушені працювати та навчатися дома віддалено, а це приводить до того, що людина перестає контролювати свою Інтернет-активність. Замість того щоб навчатися або працювати, людина може переглядати соціальні мережі, дивитися відео на YouTube. Через це вона не виконує поставлені задачі, і згодом на фоні приходу дедлайнів, коли потрібно здавати роботу, а у людини нічого не готово, може виникнути стрес. Це дуже сильно впливає як на фізичне здоров’я так і на психологічне.

Завдяки цьому також може з’являтися прокрастинація. Прокрастинація все частіше стосується осіб студентського віку, адже сьогодні навчальний процес характеризується постійною психологічною напругою, інтенсивними розумовими навантаженнями, значною кількістю завдань з чіткою регламентованістю часових меж виконання, підготовкою до заліків, екзаменів.

Але прокрастинація може виникати і по іншим причинам, наприклад, низька мотивація, імпульсивність, низький життєвий тонус, підвищений рівень особистісної тривожності, нейротизм, втомленість, безпорадність, низький рівень комунікації, соціальні та сімейні труднощі, опір зовнішньому контролю, зайва самовпевненість, лінощі, дискомфорт від виконання певних завдань, ірраціональне бажання відкласти прийняття рішення чи виконання завдання до того моменту, поки не буде досягнуто «ідеальних» умов та стану, страх невідомого, нездатність до концентрації та зосередженості на завданні через негативний емоційний чи фізичний стан [10 - 12]

Дана система може допомогти людині правильно організувати тайм-менеджмент. Тайм-менеджмент – це практика раціонального використання власного часу та його організація. Тайм-менеджмент має базові складові, а саме: планування та постановка цілей, розстановка пріоритетів і дедлайнів по кожній задачі, аналіз витрат часових ресурсів, складання списків, самоорганізація і делегування та виключення зайвих завдань.

Але дана система буде корисна не тільки для людей, які хочуть контролювати свою Інтернет-активність та позбутися прокрастинації, а ще й для тих у кого є діти. На сьогоднішній день діти вже с маленького віку вміють користуватися комп'ютерами. Як ми знаємо Інтернет безмежний, тому деякий контент для дітей є неприпустимим. Тому батьки можуть використовувати цю систему для захисту їх дітей від небажаного контенту. Або можна використовувати з ціллю виховання, коли батьки блокують доступ до улюбленого ресурсу, наприклад YouTube, поки дитина не зробить домашнє завдання у школі.

1.2 Огляд існуючих рішень

Основною складовою Інтернет-мережі є сайт (веб-сторінка). Гура М. В. у своєму дисертаційному дослідженні наводить власне визначення Інтернет-сайту, розуміючи під ним відокремлений, логічно завершений елемент мережі Інтернет, який створений на основі технології гіперпосилань, розташований на сервері (host), має унікальну адресу (url), за якою до нього може отримати доступ будь-який користувач мережі Інтернет (інколи існують винятки з цього правила, зокрема, можливе блокування доступу до певного веб-сайту окремих користувачів або користувачів певної країни тощо), та у своїй основі містить Інтернет-сторінки, які мають графічний вигляд та можуть бути переглянуті за допомогою спеціальних комп'ютерних програм (браузерів) [3].

Існує декілька способів щоб обмежити доступ до веб-ресурсів:

- Блокування у налаштуваннях маршрутизатора

- Блокування використовуючи сервіси, які надає компанія Google

- Використання розширення для браузерів
- Використання десктопних програм
- Блокування шляхом внесення змін до hosts файлу
- Блокування сторінок провайдерами

Дані способи відрізняються доступністю, складністю у розумінні, ефективністю, а також цілями. Також способи можна класифікувати наступним чином (рис. 1.1).

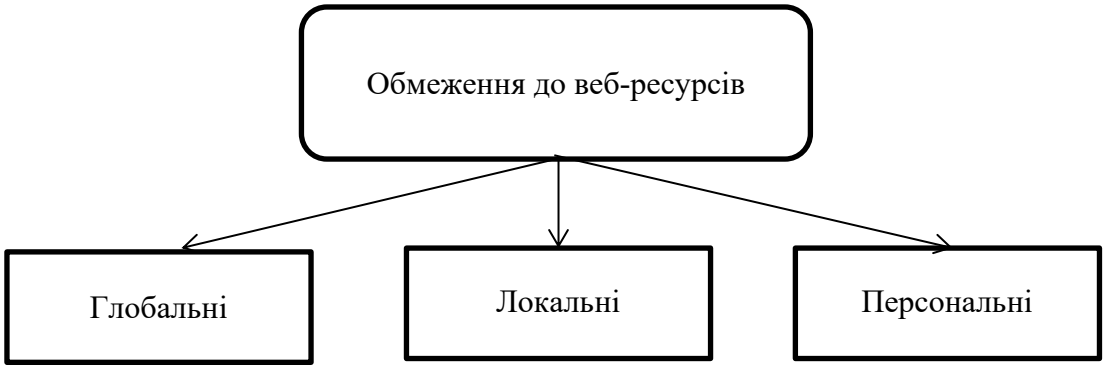


Рис. 1.1 Класифікація методів обмеження

До глобальних відносять метод блокування сторінок інтернет провайдерами. Даний спосіб використовується та регулюється державою через закон. Якщо виявляється, що якийсь ресурс якось шкодить державі, то через відповідну заяву та суд провайдери зобов'язуються обмежити доступ своїх абонентів до цих ресурсів. Яскравим прикладом може бути указ президента України Петра Порошенка від 28 квітня 2017, у якому йшлося про обмеження доступу до таких ресурсів як: «ВКонтакте», «Одноклассники», сервісів «Yandex» та «Mail.ru» [4].

Переваги:

1. Централізований контроль
2. Регуляція законом
3. Ефективність

Недоліки:

1. Досить просто обійти обмеження (наприклад використовувати VPN або proxy)

2. Ненадійність

3. Ізоляція великої кількості людей

Усі інші способи можна віднести як до локальних, так і до персональних. Це залежить від місця використання систем. Наприклад, якщо метод буде використаний у школах або університетах, то це локальні обмеження, тому що обмежується певна група людей. Якщо йде використання у себе вдома, то це персональне обмеження, тобто обмеження однієї або кількох людей.

Маршрутизатор - це пристрій, який працює на мережевому рівні моделі OSI, основною функцією якого є вибір шляху і пересилання пакетів. Маршрутизатор - це перша лінія захисту від вторгнення в мережу. Найвищий рівень безпеки на маршрутизаторі включає брандмауер, і є найкращим способом захисту комп'ютерної системи і інформації від атак. Маршрутизатори містять програмне забезпечення, назване прошивка, яке повинно оновлюватися в міру випуску виробником маршрутизатора.

Більшість маршрутизаторів підключаються до інших мережевих пристроїв тільки через мережеві кабелі і не вимагають драйверів для роботи в Windows або інших операційних системах. Однак маршрутизатори, які підключаються до комп'ютера через USB або FireWire, зазвичай вимагають, щоб драйверипрацювали належним чином. Маршрутизатор часто виступають в ролі DHCP-серверів в невеликих мережах, що видають унікальні IP-адреси. [5]

Більшість маршрутизаторів виробляються компаніями Asus, Xiaomi, TP-Link, D-Link, Microtik і Cisco.

Оскільки маршрутизатор є хабом запитів у мережу Інтернет, то у налаштуваннях ми можемо виділити URL адреси сторінок до яких ми хочемо обмежити доступ. Для цього потрібно спочатку зробити підготовчі етапи:

1. Створити резервну копію налаштувань маршрутизатора на випадок якщо щось зламається після налаштування обмежень.

2. Здійснити оновлення програмного забезпечення на пристроях. Дуже важливо встановити останні оновлення систем безпеки, щоб пристрої працювали максимально ефективно разом. Спочатку треба встановити останні оновлення прошивки для маршрутизатора. Далі вже потрібно оновити програмне забезпечення на інших пристроях.

3. На кожному пристрої, яке ви раніше підключали до мережі, може знадобитися забути мережу, щоб пристрій використав нові налаштування маршрутизатора при підключенні до мережі.

Після цього потрібно зайти у налаштування маршрутизатора. Для цього потрібно зайти у провідник, та натиснути на кнопку «Сеть» (рис. 1.2).

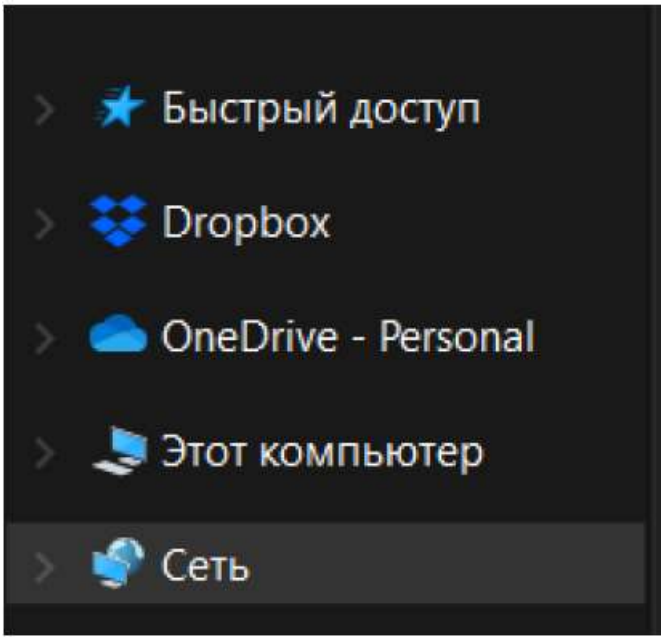


Рис. 1.2 Місцезнаходження кнопки «Сеть» у провіднику

Потім шукаємо назву нашого маршрутизатора та тиснемо правою кнопкою миші по ньому та обираємо пункт «Перегляд веб-сторінки пристрою» (рис. 1.3).

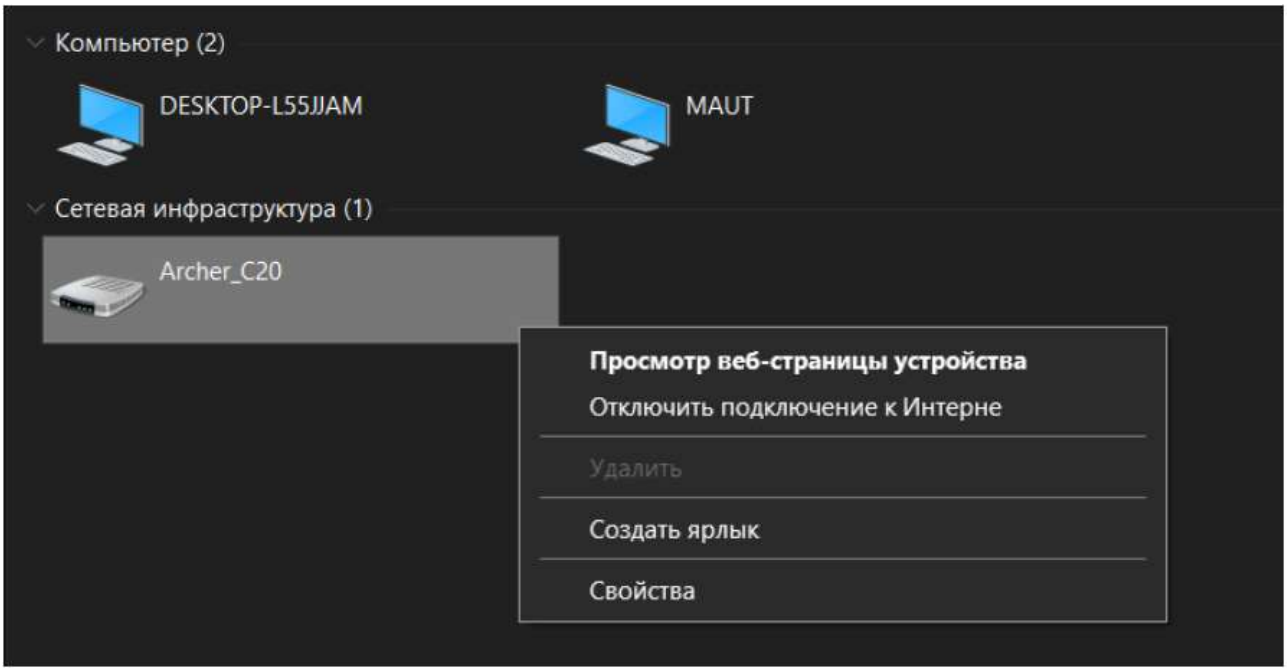


Рис. 1.3 Відкриття веб-сторінки налаштувань роутеру

Потрапляємо на сторінку налаштувань маршрутизатора та вводимо пароль від пристрою. Зазвичай стандартний пароль вказаний на нижній частині пристрою. (рис. 1.4).

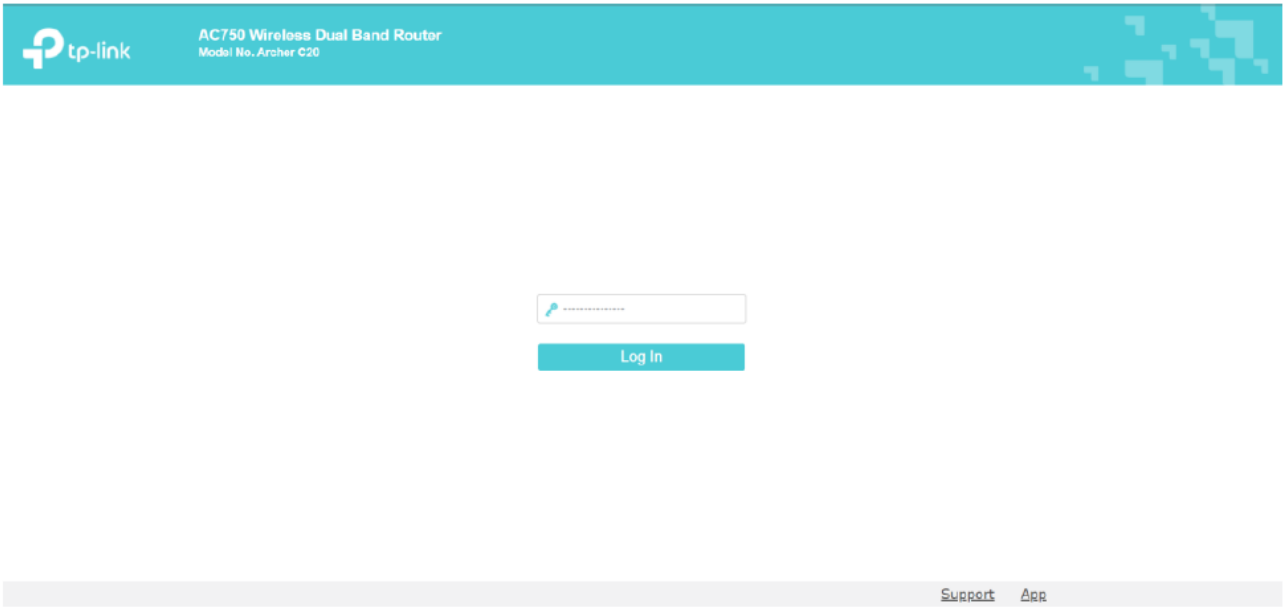


Рис. 1.4 Сторінка входу до налаштувань роутеру

Далі зліва у меню обираємо «Parental Controls» (рис. 1.5).



Рис. 1.5 Панель батьківського контролю

У першій секції перш за все потрібно увімкнути батьківський контроль. Далі у полі «MAC Address Of Parental PC» потрібно вписати MAC-адресу того пристрою, який буде вважатись адміністратором. Після чого натиснути «Save» (рис. 1.6).

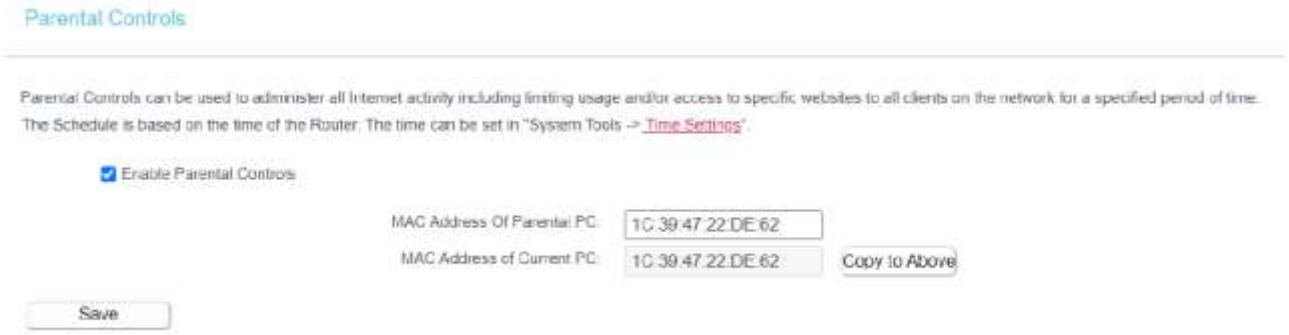


Рис. 1.6 Перша секція налаштувань

У наступній секції потрібно перерахувати MAC-адреси тих пристроїв, які ми хочемо обмежити (рис. 1.7)

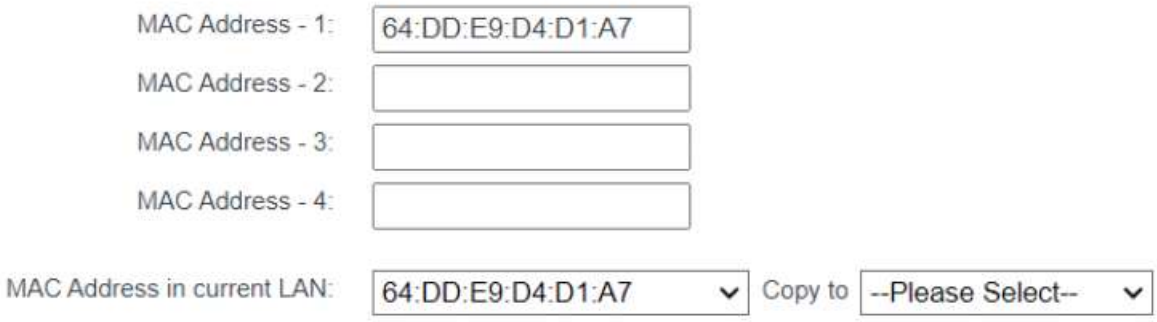


Рис.1.7 Друга секція налаштувань

Далі можемо налаштувати розклад у які дні та у які години буде обмежений доступ до ресурсу (рис. 1.8).

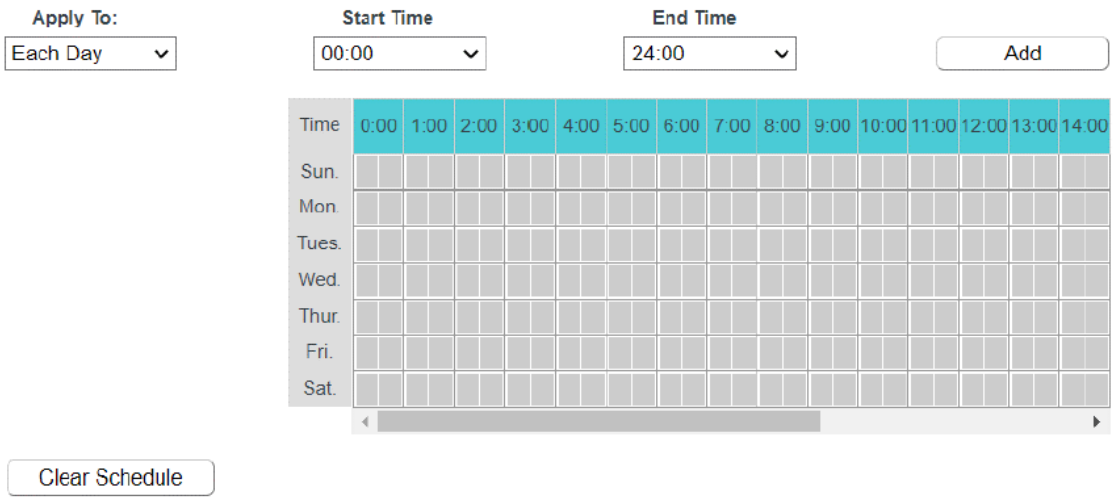


Рис. 1.8 Третя секція налаштувань

І у останній секції потрібно визначити URL-адреси веб-ресурсів до яких ми хочемо обмежити доступ (рис. 1.9).



Рис. 1.9 Секція з переліком заблокованих ресурсів

Після налаштувань зберігаємо зміни та перезавантажуємо маршрутизатор. Це потрібно, щоб оновились налаштування для підключених пристроїв. Перевіряємо чи спрацював даний метод (рис. 1.10).

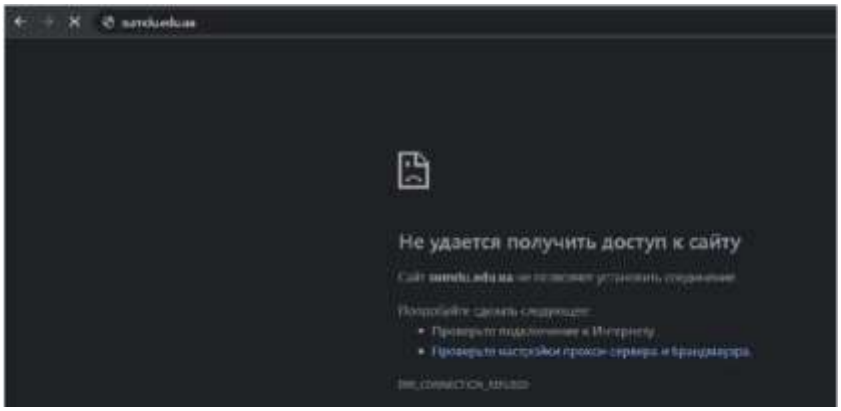


Рис. 1.10 Перевірка роботи методу

Даний метод спрацював добре. Серед переваг можна виділити наступні:

- Ефективність

- Гнучкість – можемо обмежити тільки ті пристрої, які нам потрібні, а також обрати час, коли ресурс буде недоступний

- Доступність – метод доступний усім, хто користується маршрутизатором

Недоліки:

- Складність – людина яка погано, або взагалі не розуміється у технологіях не зможе самостійно налаштувати обмеження

- Час налаштування – потрібно витратити достатньо багато часу, аби знайти потрібні MAC-адреси пристроїв та вписати їх до списку

- Забагато зайвих дій – після оновлення налаштувань потрібно кожен раз перезавантажувати маршрутизатор, щоб зміни вступили в силу.

- Незручність у додаванні нових адрес веб-сторінок – кожен раз, щоб додати нову адресу потрібно заходити у налаштування, а потім перезавантажувати його.

Отже даний метод є ефективним рішенням, якщо нам потрібно локально обмежити доступ певній групі людей, наприклад класу у школі, або студентам у аудиторії, і якщо не потрібно буде часто змінювати список заблокованих ресурсів. Для персонального користування краще розглянути інші варіанти обмежень.

Файл hosts – це локальний простий текстовий файл, який відображає ваш сервер, або імена хостів на адреси Інтернет-протоколу (IP). Щоразу, коли Windows підключається до мережі, використовуючи ім’я хоста, він звертається до фалу hosts. Якщо Windows знайде запис у цьому файлі, він зв’яжеться із зазначеним сервером. Якщо Windows не знайде відповідне ім’я хоста, тоді він вирішить ім’я хоста за допомогою служби доменних імен (DNS). Це процес, який використовується для отримання IP-адреси сервера, що стоїть за доменним ім’ям.

Кожен запис файлу hosts має власний рядок, в якому міститься числова IP-адреса, пробіл (або символ табуляції) та ім’я хоста або домену (рис. 1.11). Наприклад:

```
1:  127.0.0.1     example.com
```

Рис.1.11 Приклад запису адреси у hosts файл

У наведеному прикладі перша секція позначає IP-адресу, на яку буде перенаправлено даний запит (127.0.0.1). Друга секція позначає місце, з якого ми хочемо перенаправити запит (example.com). Після додавання інформації про домен ваша система перейде на вказану вами IP-адресу. Тобто у прикладі вище, ми пов’язуємо доменне ім’я example.com із IP-адресою 127.0.0.1

Варто зазначити, що деяке програмне забезпечення використовує власні методи для пошуку імен хостів. Це означає, що завжди існує ймовірність того, що воно може повністю проігнорувати файл host. Але у більшості випадків, браузери не ігноруюсь цей файл.

Насправді існує декілька причин, через які користувачу потрібно буде змінювати файл hosts. Найбільш поширеною причиною для цього є надання можливості людям переглядати або публікувати веб-вміст відразу після придбання нового доменного імені або передачі існуючого доменного імені іншому провайдеру послуг Інтернет. Нові та перенесені доменні імена мають період затримки, який може становити від декількох годин до декількох днів. Протягом цього періоду інформація про новий або перенесений домен поширюється по мережі Інтернет і, як правило, недоступна.

Якщо вам необхідно негайно оновити свій сайт і ви не можете чекати, поки інформація про домен поширюється в Інтернеті, ви можете відредагувати цей файл на своєму комп’ютері в якості тимчасового обхідного шляху. Але цей обхідний шлях діє тільки на тому комп’ютері/сервері, на якому були внесені зміни до hosts файлу. Це не робить веб-сайт доступним для будь-якого в Інтернеті.

Інша причина для зміни hosts файлу це власне блокування доступу до веб-ресурсів. Але це скоріш не блокування, а перенаправлення на потрібну нам IP-

адресу. На жаль, або на щастя, перенаправлювати ми зможемо лише на локальну IP-адресу поточного комп’ютера. Раніше можна було перенаправлювати на будь-яку IP-адресу, але це виявилось небезпечним, тому що зловмисники (хакери) могли отримати доступ до hosts файлу та внести свої зміни і таким чином перенаправлювати жертв на шкідливі ресурси.

Отже, щоб заблокувати доступ до веб-сторінки (сторінок), шляхом зміни hosts файлу потрібно:

1. У пошуку знайти Notepad та відкрити його від імені адміністратора (рис. 1.12)

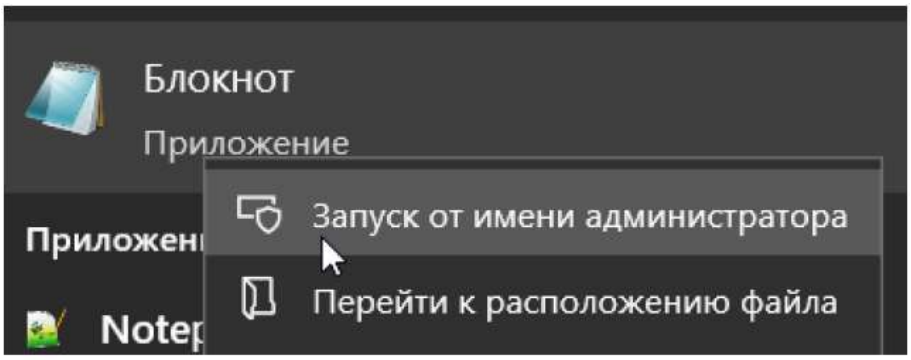


Рис.1.12 Відкриття блокноту

2. Натиснути Файл – Відкрити.

3. У відкритому вікні перейти за наступним шляхом: C – Windows - System32 – drivers – etc.

4. Обрати hosts та відкрити (рис. 1.13).

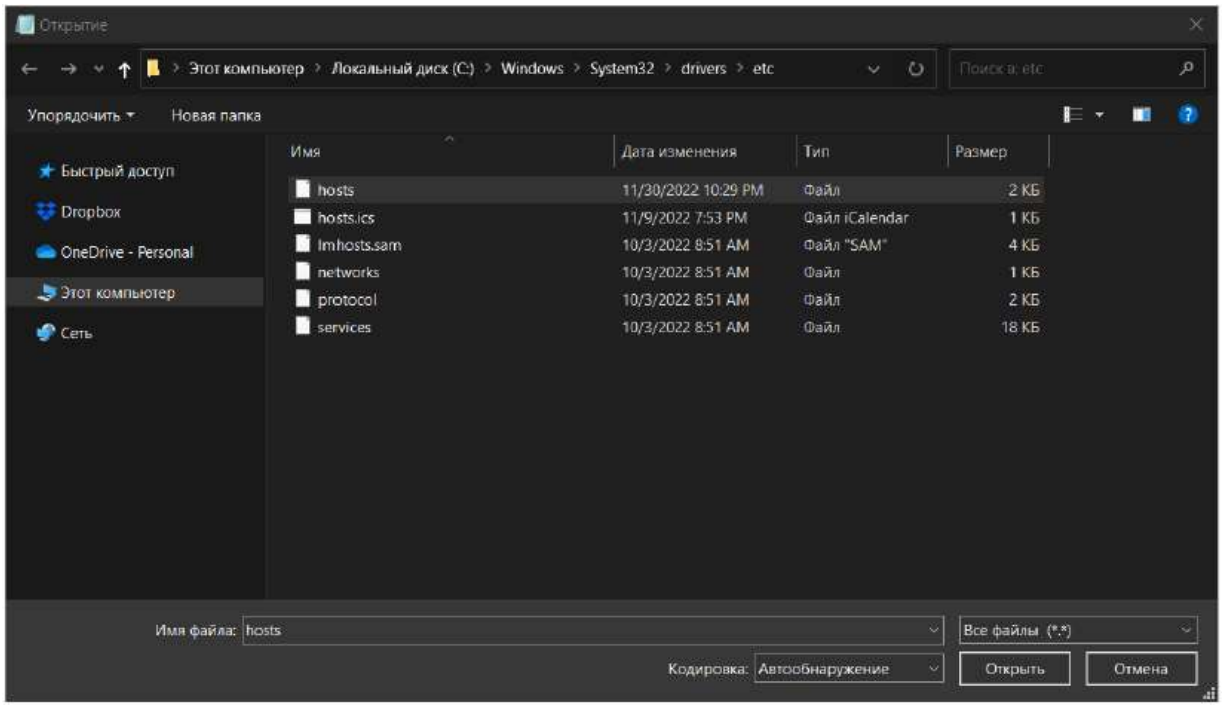


Рис.1.13 Пошук hosts файлу

Для прикладу обмежимо доступ до Facebook. Спочатку переконаємося, що зможемо зайти на сторінку логіну Facebook (рис. 1.14).

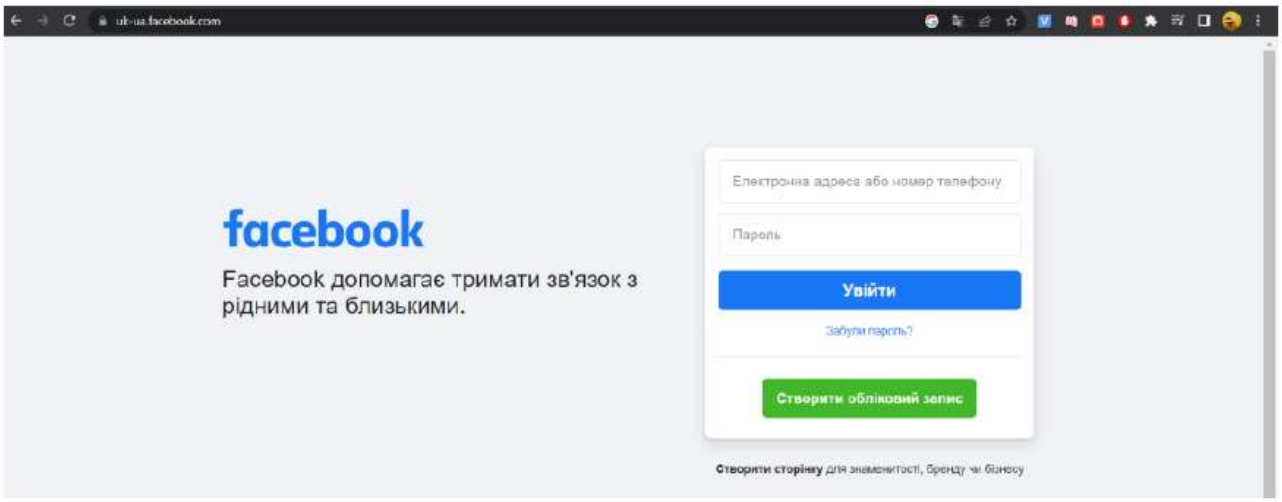


Рис.1.14 Сторінка до блоку

5. Тепер внесемо зміни до hosts файлу. Кожна URL-адреса повинна бути записана з нового рядка (рис. 1.15).

```
127.0.0.1       www.uk-ua.facebook.com
127.0.0.1       uk-ua.facebook.com
```

Рис. 1.15 Внесені зміни до hosts файлу

6. Збережемо файл. Для того, щоб зміни вступили у силу потрібно у браузері очистити кеш. Тому що браузери кешують IP-адреси сторінок, які ми

раніше відвідували, тому якщо у кеші є якась інформація, то першочергово браузер буде брати її звідти.

Тепер перевіримо, чи можемо ми відкрити сторінку. Як видно на (рис. 1.16) ми не можемо завантажити сторінку логіну Facebook.

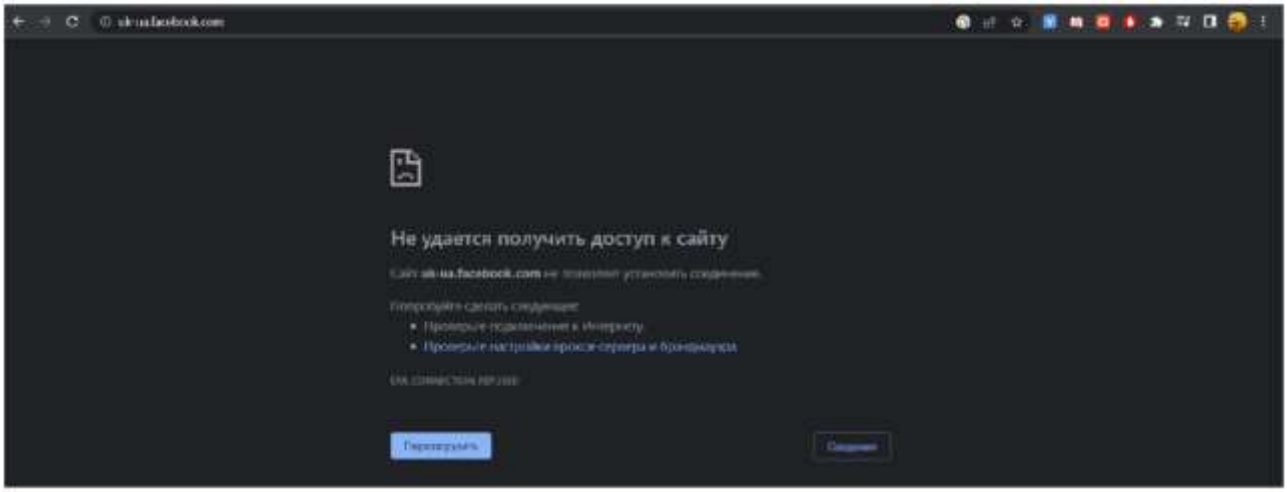


Рис. 1.16 Перевірка роботи методу

Отже, даний спосіб є досить ефективним методом для обмеження доступу до веб-ресурсів. Але він має досить багато недоліків, а саме:

- Користувачу, який погано розбирається з комп'ютерами буде досить складно налаштувати обмеження

- Після змін у hosts файлі потрібно кожен раз очищувати кеш браузера

- Не всі браузери можуть працювати з hosts файлом

- При заповненні hosts файлу потрібно враховувати усі можливі субдомени ресурсу

Одним із способів заблокувати сайт є використання десктопних програм. Це один із найзручніших методів блокування. Але він має також свої переваги та недоліки.

Існує безліч таких програм, але проаналізуємо лише декілька найбільш популярний з них, який називається Cold Turkey (рис. 1.17).

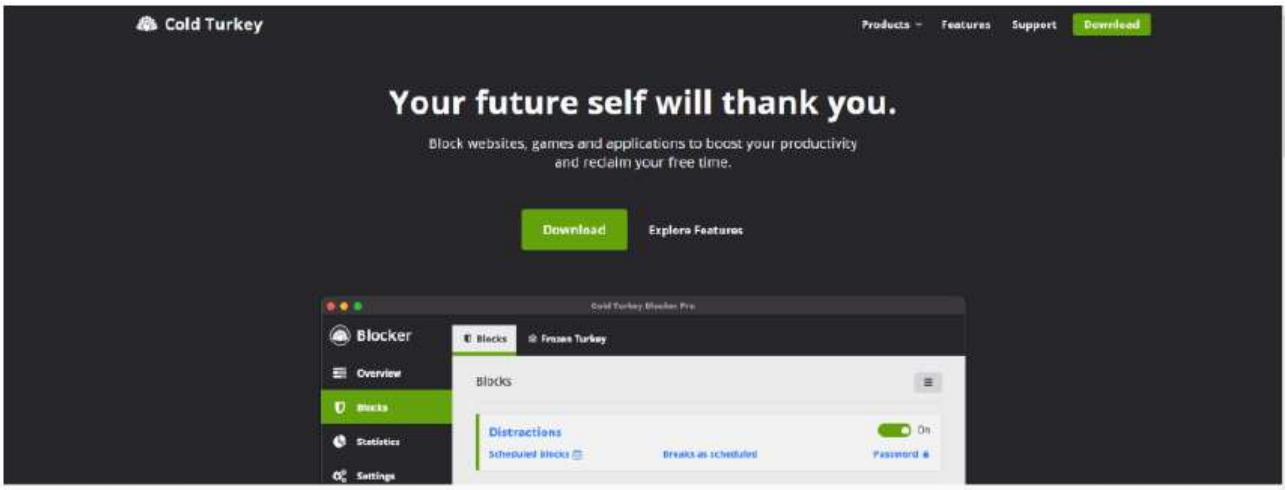


Рис. 1.17 Головна сторінка програми

Додаток встановлюється як звичайна програма. Після встановлення нам потрібно завантажити розширення від розробників додатку (рис 1.18).

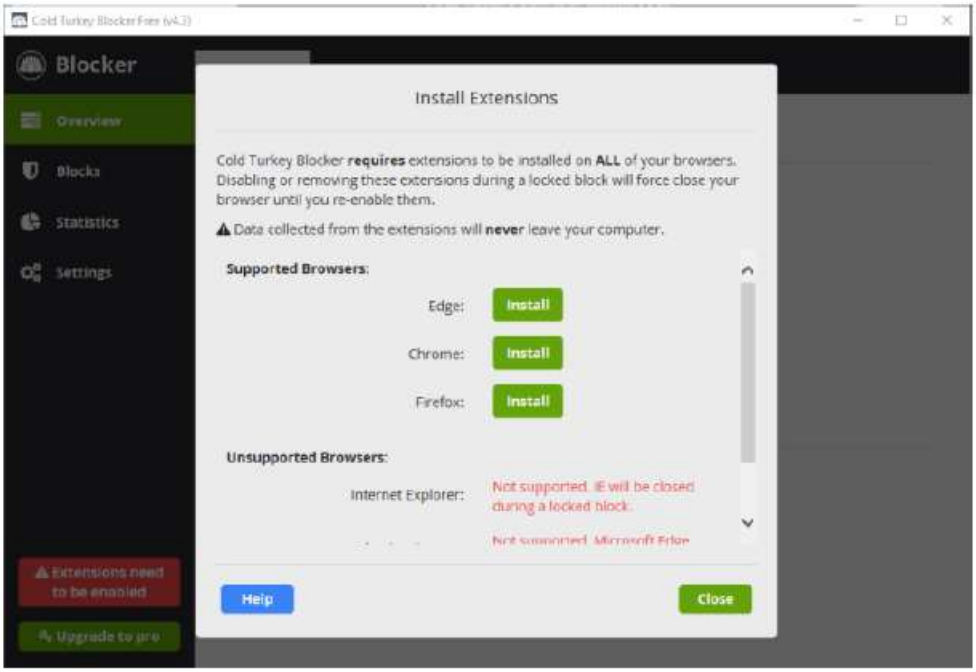


Рис. 1.18 Встановлення розширення

Далі у додатку перейдемо до секції “Blocks”, де можемо створити декілька сетів з веб-ресурсів, якими легко керувати (рис 1.19).

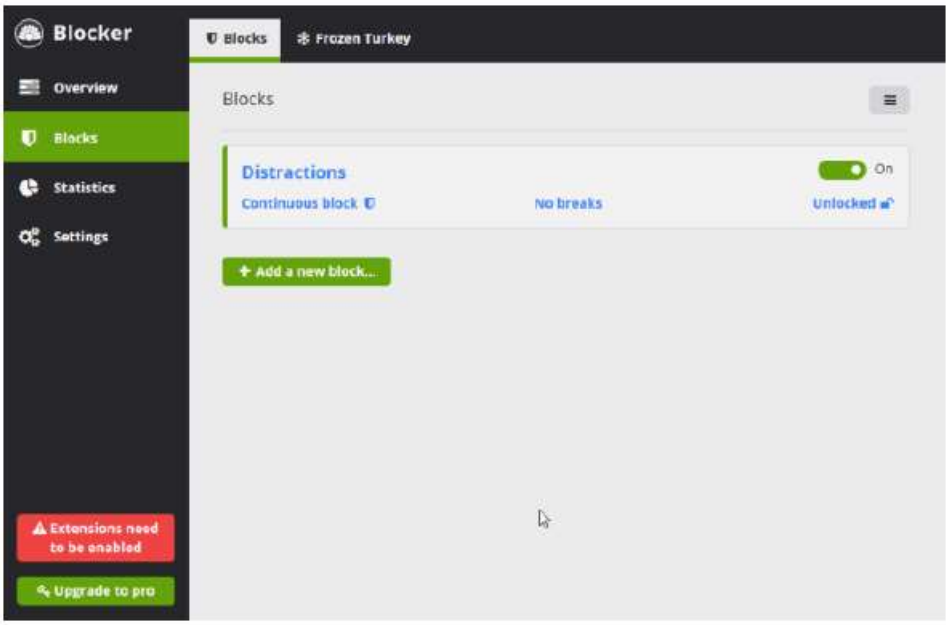


Рис. 1.19 Секція налаштувань Blocks

Додавати сайти до списку заблокованих можна як через додаток, так і через розширення. Даний додаток пропонує широкий спектр функцій, а саме: блокування доменів, блокування конкретних URL-адрес, блокування URL-адрес, використовуючи ключові слова, або регулярні вирази, або взагалі заблокувати будь-яку сторінку в Інтернеті. Окрім цього додаток також може заблокувати будь-яку програму, яка встановлена на комп'ютері, також може обмежити доступ до файлу, папки. Також можна заборонити відкривати додатки з Windows Store. За допомогою цього додатка навіть можна заблокувати доступ до усього комп'ютера, також є можливість налаштувати розклад блокування.

Цей додаток записує та зберігає усю активність, тому у секції Statistics відображена уся статистика активностей (рис 1.20).



Рис. 1.20 Статистика використання

Перевіримо роботу даного додатка. Відкриємо веб-сторінку УНУС (рис 1.21).

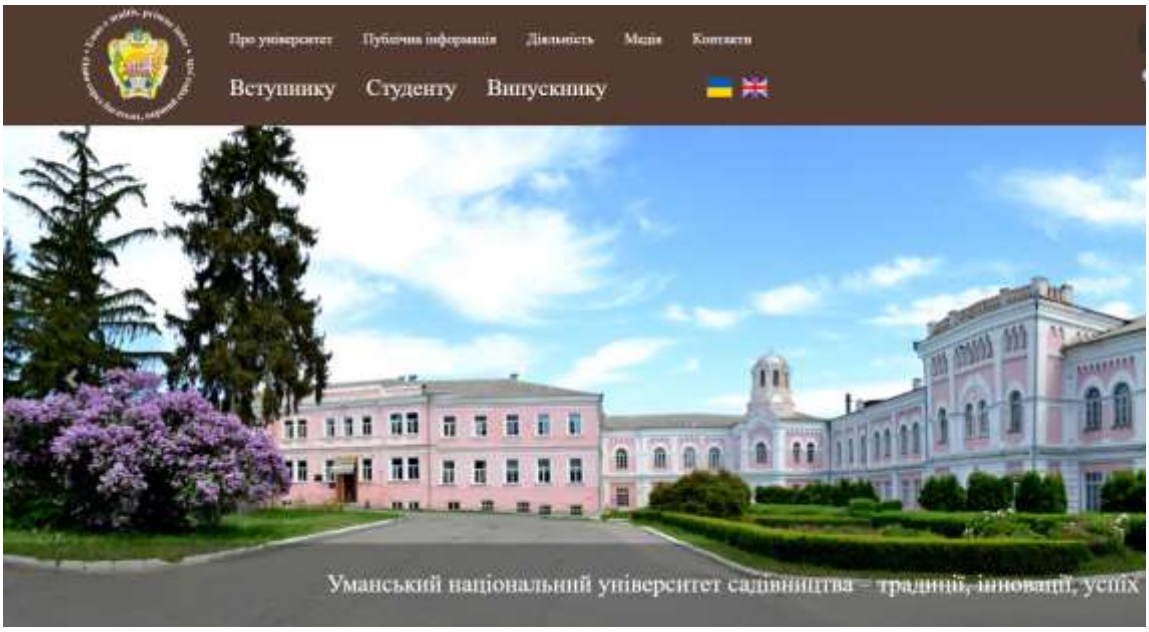


Рис. 1.21 Доступність ресурсу до блоку

Тепер використовуючи розширення додамо цю сторінку до чорного списку (рис 1.22).

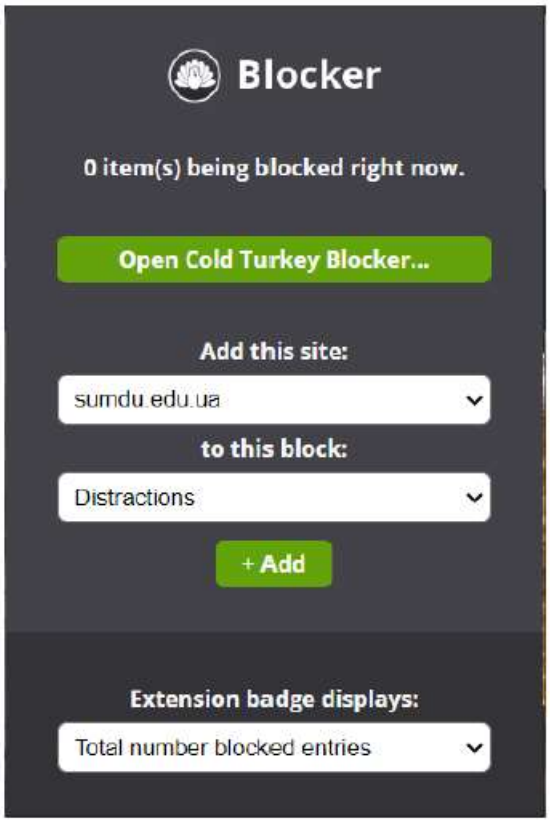


Рис. 1.22 Додавання сторінки до чорного списку

Як результат тепер ми не можемо відвідати цю сторінку (рис 1.23).

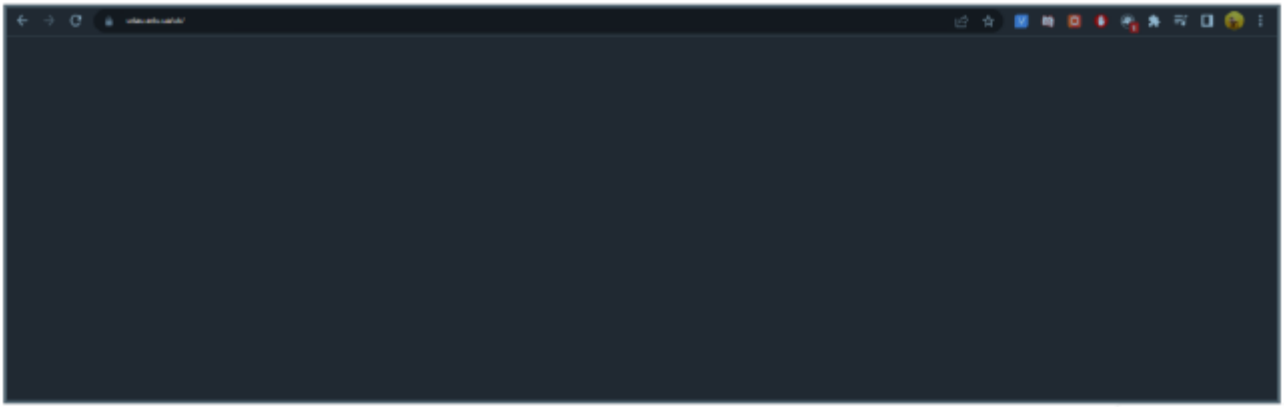


Рис. 1.23 Перевірка роботи методу

Отже даний додаток досить гарно виконує свої функції. Серед переваг я можу виділити наступні:

- Зручність у користуванні
- Наявність широкого вибору різноманітних функцій
- Можливість блокувати не тільки веб-сторінки
- Наявність досить обширної статистики використання
- Можливість налаштування розкладу
- Для використання не потрібно авторизуватись

Недоліки наступні:

- Багато функціоналу доступно лише за платною підпискою
- Необхідність встановлювати розширення
- Досить висока ціна за підписку
- Неможливість кастомізувати контент, який відображається після блоку веб-сторінки
- Відсутність блокування на мобільних пристроях
- Відсутність програми під систему Linux
- Невеликий список браузерів, які підтримують розширення

Отже, використання десктопних додатків є ефективним рішенням для контролю активності, але більшість із них розкриває свій потенціал лише після придбання платної версії продукту. Використання додатків не підійде для тих людей, котрі потребують лише обмеження доступу до веб-ресурсів, тому що основна перевага десктопних додатків це можливість блокування інших встановлених додатків, а тому для такої людини це використовувати буде не раціонально. Для таких людей більш раціонально буде використання розширень для браузера.

Найзручнішим та найпростішим способом заблокувати доступ до веб-ресурсе є використання розширення для браузера. Розширення – це програми, які можна встановити у браузер, що змінити його функціональність. Вони включають в себе додавання нових функцій або зміну існуючої поведінки, щоб зробити її більш комфортною для користування. Але, на жаль хоча більшість розширень корисні, деякі розробники створюють розширення, які негативно впливають на роботу браузера. Наприклад, розробники рекламного програмного забезпечення можуть без вашого відома встановлювати розширення, які будуть вставляти рекламу на сайтах, які ви переглядаєте.

Функціональність розширень не має меж, це може бути все що завгодно, наприклад:

- Блокування показу реклами на сайтах
- Оптимізація використання пам’яті для більш швидкої роботи браузера

- Додавання заміток
- Управління паролями
- Спрощення копіювання контенту з веб-сторінки
- Захист конфіденційної інформації

Існує також декілька різних видів роботи розширення. Деякі можуть працювати у фоновому режимі та автоматично виконувати задачі. Інші можуть працювати тільки після їх активації. Також є розширення, які просто додають додаткову функціональність, наприклад, у контекстне меню.

Усі розширення додають свої значки до панелі розширення, яка знаходиться справа від рядка пошуку. Це дає змогу легко відслідковувати, які розширення на даний момент встановлені та активовані, а також натиснувши на цей значок можна активувати розширення та використовувати його функції.

Щоб використовувати розширення потрібно їх спочатку встановити. Деякі розробники пропонують напряму завантажити їх розширення, але це не досить безпечно. Найкраще місце для завантаження розширень це Інтернет-магазин Chrome. Перед публікацією, розширення проходить верифікацію та перевірку, а це зменшує ризик отримати шкідливе програмне забезпечення. В основному більшість розширень є безкоштовними, але також є і платні розширення, які потрібно спочатку купити, перед використанням.

Хоча використання розширень покращує та спрощує перегляд веб-сайтів, проте надмірна кількість встановлених розширень може знизити швидкість роботи браузера. Тому що кожне розширення потребує певного об’єму пам’яті, тому якщо комп’ютер має недостатньо оперативної пам’яті, користувач може стикнутися з проблемами довгого завантаження сторінки або взагалі некоректної роботи браузера. Тому потрібно встановлювати лише ті розширення, якими ви будете користуватися, а якщо воно вже не потрібне, то краще видалити його.

Проаналізуємо найпопулярніше розширення для блокування веб-ресурсів, яке має найвищий рейтинг серед усіх - LeechBlock

LeechBlock – це простий інструмент для підвищення продуктивності, призначений для блокування сайтів, які заважають ефективно працювати. Розширення дозволяє створювати до 30 наборів сайтів для блокування, з різним часом та днями для кожного набору. Також можна блокувати в певні проміжки часу, після обмеження часу, або з комбінацією проміжків часу і обмеження часу. Цей інструмент має додаткові функції, а саме:

- Негайне блокування сайтів на певний час

- Контроль доступу. Можна встановити пароль, або випадковий код доступу для сторінки налаштувань.

- Затримка. Можливість встановити зворотній відлік, щоб затримати доступ до сайтів замість їх повного блокування

- Білий список. Список сайтів, які не будуть доступні для блокування

Встановимо дане розширення та перевіримо як воно працює (рис. 1.24).

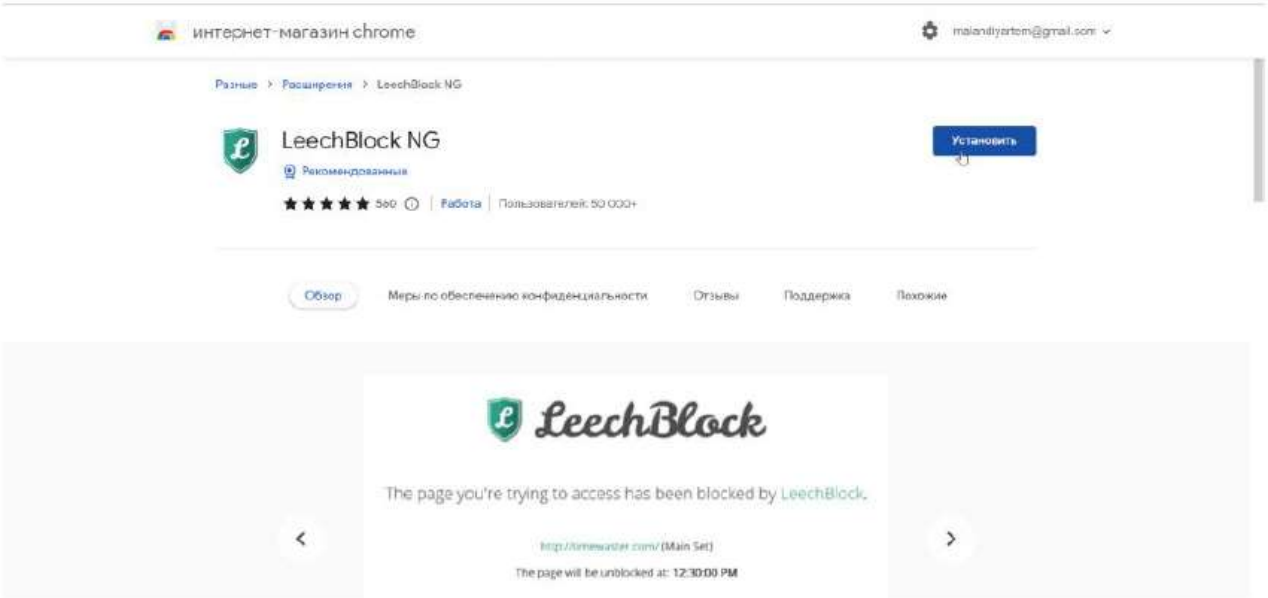


Рис. 1.24 Встановлення розширення

Активуємо розширення та перейдемо до пункту меню options (рис. 1.25).

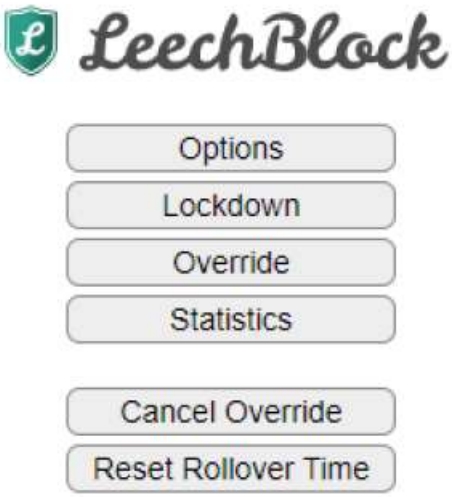


Рис. 1.25 Вікно після активації розширення

Перед початком блокування перевіримо, що ми можемо зайти на сайт, який хочемо заблокувати (рис. 1.26).

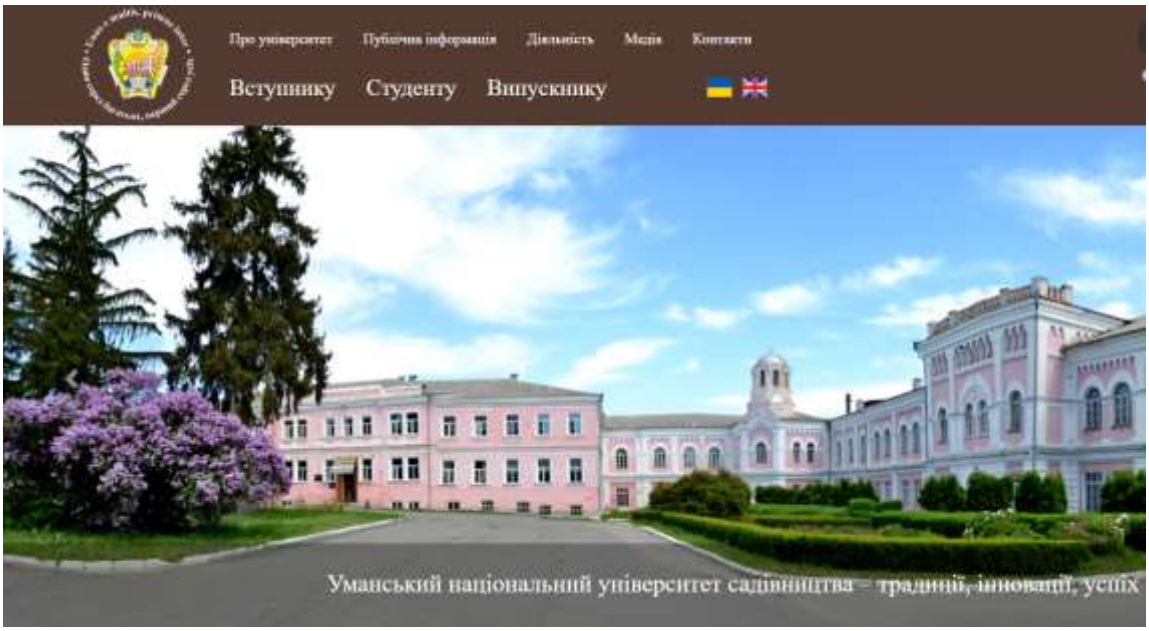


Рис. 1.26 Сторінка до блокування

Для того, щоб заблокувати веб-сторінку потрібно на сторінці options у розділі What to block ввести ім’я блоку, а потім перерахувати URL-адреси сторінок, які хочемо заблокувати. Кожна адреса з нового рядка (рис 1.27).

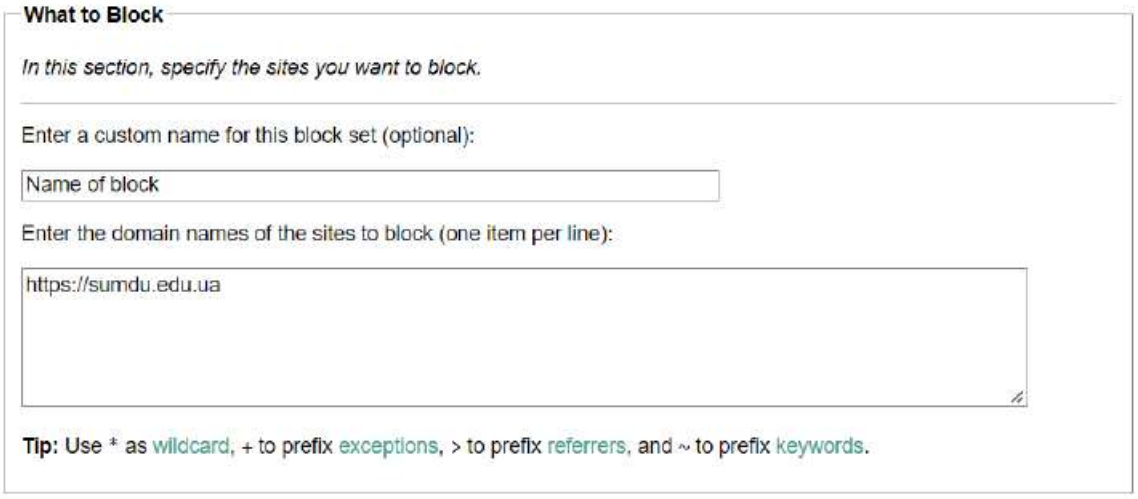


Рис. 1.27 Внесення змін до налаштувань, що заблокувати

Далі у секції When to block можемо налаштувати дні та години, коли буде недоступний ресурс (рис. 1.28).

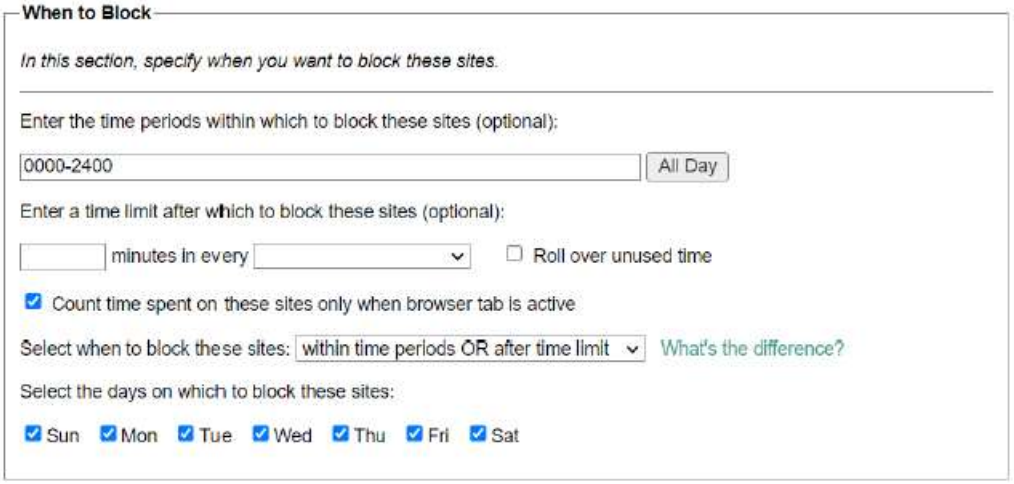


Рис. 1.28 Внесення змін до налаштувань, коли заблокувати сторінку

Наступна секція How to block. Тут потрібно описати як саме веб-сайт буде заблокований. Можна вказати URL-адресу на яку буде перенаправлено користувача, якщо він відвідає заблоковану сторінку (рис. 1.29).

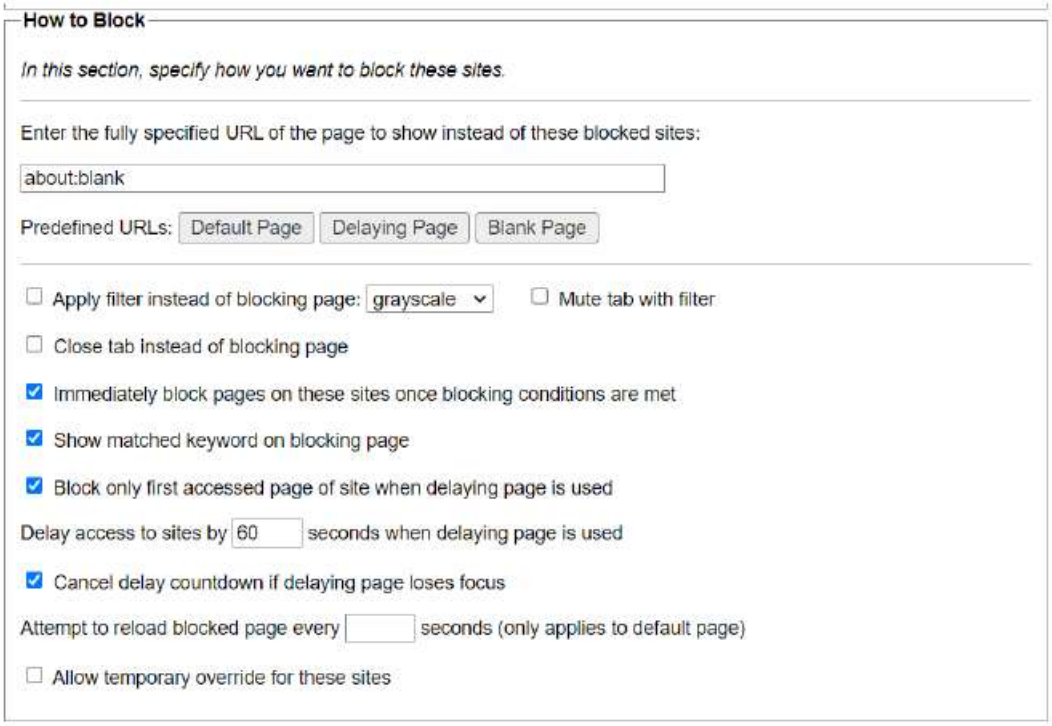


Рис. 1.29 Внесення змін до налаштувань, як заблокувати сторінку

На вкладці General можна більш детально налаштувати блокування. Наприклад можна збільшити кількість блоків (дане розширення дозволяє створити до 30 блоків), є можливість встановити пароль, відобразити таймер зворотного відліку, налаштувати попереджуване повідомлення, а також експортувати або імпортувати налаштування. Після цього збережемо зміни та перевіримо чи обмежили ми доступ до веб-сторінки. Відразу після збереження змін, заблокована сторінка була замінена на пусту (рис. 1.30).



Рис. 1.30 Перевірка роботи методу

Розширення LeechBlock спрацювало досить якісно. Хоча воно працює шляхом перенаправлення користувача на іншу сторінку, проте після переходу на заблоковану сторінку перенаправленя йде миттєво, що користувач не бачить

контенту сторінки, як це було у одному з десктопних додатків. Отже серед переваг я можу виділити наступні:

- Повністю безкоштовне розширення. Усі функції відразу доступні
- Широкий спектр налаштувань
- Якісна робота блокування
- Наявна можливість імпортувати або експортувати налаштування
- Відсутня необхідність авторизації

Серед недоліків є наступне:

- Відсутня локалізація на інші мови (доступна тільки англійська)
- Непривабливий інтерфейс

Отже, використання розширень для браузера є найзручнішим способом для обмеження доступу до веб-ресурсів для звичайного користувача комп'ютера, який хоче контролювати свою Інтернет-активність, або контролювати активність своїх дітей. Серед основних переваг у використанні розширень є: зручність у користуванні, швидкість оновлення списків заблокованих сторінок, перегляд статистики, можливість кастомізувати зовнішній вигляд заблокованого веб-ресурсу. Серед недоліків можна виділити наступні: кількість розширень для блокування доступних для завантаження небагата, деякі з них мають платну підписку, досить просто розблокувати сторінку, достатньо видалити розширення.

1.3 Постановка задачі

Аналіз існуючих методів обмеження доступу до веб-ресурсів та виявлені недоліки обґрунтовують рішення розробити власну систему обмеження доступу до веб-сторінок для здійснення ефективного та зручного блокування веб-сторінок. Оскільки під час аналізу виявилось, що найзручнішим методом блокування є використання розширень для браузера, то було прийняте рішення розробити власну систему на основі розширення для браузера.

Схема роботи розширення наступна:

1. Користувач завантажує розширення з Інтернет-магазину Chrome

2. Після завантаження він буде перенаправлений на основну сторінку розширення, де йому потрібно буде пройти авторизацію, використовуючи Google аккаунт. У якості авторизації буде використаний сервіс Auth0

3. Після успішної авторизації користувач потрапляє на сторінку dashboard, де буде зібрана уся інформація щодо заблокованих веб-ресурсів, а також його унікальний код. Там він зможе додати, видалити або змінити чорний список. Також тут він зможе обрати поведінку розблокування сторінок: таймер зворотного відліку, пароль або нічого

4. Також користувач зможе додати поточний сайт, на якому він знаходиться, до списку, відкривши віконце розширення. Але перед додаванням треба у спеціальне поле ввести унікальний код користувача, який можна отримати на сторінці dashboard.

Отже, уся система буде поділена на 3 частини. Перша частина це backend, яка керує базою даних та читає дані з неї. Друга частина це безпосередньо база даних. Третя частина це Frontend, який у свою чергу поділений на 2 під частини:

безпосередньо саме розширення та веб-сторінка для управління заблокованими ресурсами.

Віконце розширення виконує функції додавання поточної URL-адреси до списку заблокованих. А також перевіряє поточну URL-адресу, а саме чи знаходиться вона у списку заблокованих чи ні. Якщо так, то не відображатиме контент сторінки.

Frontend частина для управління заблокованими ресурсами містить 2 сторінки: сторінка логіну та сторінка управління списками заблокованих ресурсів. На ній знаходиться унікальний код користувача, список заблокованих ресурсів, налаштування поведінки блокування та статистика активності.

РОЗДІЛ 2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Основні положення

Сучасна веб-розробка складається з трьох основних частин:

- Бази даних
- Клієнтська частина
- Серверна частина

Клієнтська частина (frontend) - напрям у веб-розробці, який працює у браузері, з яким взаємодіє користувач. Це динамічні інтерфейси, меню, події по діям користувача, обмін даними з серверною частиною, одним словом клієнтська частина це та частина, яку бачить користувач.

В свою чергу клієнтська частина ділиться на 3 складові:

- HTML

- CSS

- JavaScript

HTML – це мова гіпертекстової розмітки. Вона відповідає за зміст та структуру сторінки. Вона складається з тегів, які в свою чергу складаються з імені всередині кутових дужках. Наприклад: <*video*>, <*section*>, <*h1*>. Також теги можуть мати атрибути. Теги у HTML не чутливі до регістру, проте прийнято писати теги маленькими літерами [6].

HTML являє собою скелет сторінки, він забезпечує її основу. Розширенням файлу є *.html, за допомогою нього браузери розуміють, що всередині файлу знаходиться розмітка веб-сторінки. Браузери аналізують його структуру, визначають розміщення елементів та відображають їх. З появою CSS, з HTML стало працювати простіше, тому що необхідність використовувати теги, які відповідають за візуальне оформлення, відпала.

CSS – це каскадна таблиця стилів, яка використовується для стилізації розмітки HTML. Наприклад: заокруглити кути відображення, змінити колір тексту, зробити відступи більше або менше, вирівняти блок по центру і тд.

Розробник також може робити так, щоб при наведенні на елемент він плавно змінював колір, або змінював свою форму, щоб елементі зникали або переміщувались, тобто робити просту анімацію[7].

Стилі можна писати всередині HTML – інлайново або підключити через зовнішній файл. Для цього існує спеціальний тег у HTML <*link*> у якого існують два атрибути: *«href»* - для вказання посилання на CSS файл та *«rel»* - для вказання, що це саме таблиця стилів. Хоча використання інлайнових стилів мають вищій пріоритет, проте гарною практикою вважається підключення зовнішнього файлу з розширенням *.css. Так Розмітка розділяється від стилізації, відповідно зменшується вірогідність помилки, зовнішній файл кешується та не завантажується повторно, що підвищує швидкість завантаження сторінки.

Сучасний frontend не може існувати без мови програмування JavaScript. Це найпопулярніша клієнтська інтерпретована мова сценаріїв. Багато найпопулярніших сайтів від Twitter і Gmail до YouTube та Facebook використовують JavaScript для створення інтерактивних веб-сторінок [8] Це не складна мова програмування, вода дозволяє сторінці включати сценарії, які будуть реагувати на якійсь події користувача, наприклад, клік мишкою, відправлення форми, прокручування сторінки та ін. Такі сценарії з відносно невеликим зусиллями можуть створити складну поведінку веб-сторінок.

JavaScript має ряд переваг:

- Незамінність для веб-розробки. У всіх браузерах є підтримка цієї мови, вона завжди ввімкнена за замовчуванням
- Швидкість роботи. JavaScript може частково обробляти сторінки на комп’ютерах користувача без запитів до серверу. Це економить час та інтернет-трафік.
- Потужна інфраструктура. За той час поки існує ця мова було створено безліч готових рішень у відкритому доступі, це робить мову більш гнучкою та простішою.

- Простота. Просту задачу можна вирішити за декілька хвилин. Для більш складних задач вже існують готові рішення, якими можна скористатися.
- Зручність інтерфейсів. Заповнення форм, вибір дії, активація кнопок, реагування на наведення миші и тд. Все це підвищує рівень юзабіліті.
- Постійна підтримка та регулярні оновлення

  Але також JavaScript має і ряд недоліків, а саме:
- Неможливість читати з файлу. Головна причина цього недоліку це безпека.
- Нестрога типізація. У мові має місце різна інтерпретація даних. Немає можливості попереднього виявлення помилки.
- Доступність для злочинців. У відкриту скриптову мову легше всього впровадити шкідливий код, який може нашкодити користувачу

Для зручності програмування на JavaScript можна використовувати бібліотеку jQuery. Це швидка, зручна та багатофункціональна бібліотека. Вона надає багато функції, за допомогою яких можна виконати різні завдання. Її використання значно полегшує виконання тих чи інших задач, також значно зменшує об’єм коду. API-інтерфейси можуть виконувати такі функції, як обробка подій, маніпулювання елементами HTML, додавання анімацій на сторінку. Також бібліотека дозволяє взаємодіяти з сервером без перезавантаження сторінки [9]. Але у сучасній веб-розробці jQuery поступово відходить на задній план, на його заміну з’явилося дуже багато фреймворків та бібліотек, які виявились ще швидшими, легкими у користуванні та надійними. Найпопулярніші з них це Angular, React та Vue. Усі вони створені на основі TypeScript, який у свою чергу є надбудовою над JavaScript.

У TypeScript виправлені більшість недоліків чистого JavaScript. Код написаний на TypeScript компілюється у код JavaScript, це потрібно тому що браузери розуміють лише мову JavaScript. Більшість проблем JavaScript виникають внаслідок динамічної типізації та в принципі дивної поведінки типів даних. У TypeScript же типізація статична, наприклад, є строковий тип, числовий логічний та ін. Також є можливість описувати свої типи даних. У TypeScript покращена підтримка об’єктно-орієнтованого програмування (ООП):

класси, об’єкти, наслідування, інтерфейси. Також одним із переваг TypeScript є наявність модифікаторів доступу до полів класу, а саме: public, private та protected. Також є і інші можливості: визначення полів у конструкторі, перетворення типів, абстрактні класи і тд.

Але у TypeScript є також і мінуси. Розробка веб-додатків на TypeScript коштує дорожче та займає більше часу. Особливо, якщо потрібно використовувати якусь бібліотеку чи фреймворк, які не перенесені на TypeScript. У цьому випадку розробникам доведеться самостійно визначати сигнатури усіх функцій та методів.

Як вже було сказано вище TypeScript є основою для більшості бібліотек та фреймворків. Наприклад, Angular – це фреймворк, розроблений компанією Google для створення клієнтських додатків. Перш за все він націлений на розробку Single Page Application (SPA), тобто одно сторінкових додатків. Angular надає можливість двостороннього зв’язування, яке дозволяє динамічно змінювати дані у одному місці інтерфейсу при зміні моделі у другому, шаблони, маршрутизація і тд.

Backend – це серверна частина веб-додатку, яка не помітна для користувача. Сюди відноситься: авторизація, зберігання та обробка даних, email розсилка и тд. Якщо у frontend тільки одна мова – JavaScript. То у backend мов більше, а саме:

1. C#
2. Asp net
3. Ruby
4. Python
5. Java
6. Node js

NodeJS – це платформа з відкритим вихідним кодом для побудована на движку Chrome V8. Вона дозволяє писати серверний код для веб-додатків та динамічних сторінок, а також програм командного рядка. В основі платформи – подійно-управляюча модель з неблокуючими операціями вводу-виводу, що

робить її ефективною та легкою. З NodeJS простіше масштабуватися. При підключенні до серверу тисяч користувачів NodeJS працює асинхронно, тобто ставить пріоритети та розподіляє ресурси. В той час як, наприклад, Java, виділяє на кожне підключення окремий потік. За допомогою NodeJS можна написати програму для вебу, Linux, MacOS та Windows. Також можна створювати кроссплатформенні додатки. NodeJS універсальний та гнучкий інструмент. Оскільки в основі нього лежить мова JavaScript, то розробники, які уже використовували JavaScript при написанні frontend частини, зможуть з легкістю написати і клієнтський і серверний код, не вивчаючи інструмент з нуля. В NodeJS можна швидко переходити на нові стандарти ECMAScript по мірі реалізації. Нові можливості мови стають доступними відразу після встановлення підтримуваної їх версій NodeJS.

Ще одним плюсом NodeJS є велика кількість модулів та бібліотек. Екосистема NodeJS с кожним роком все швидше і швидше розвивається завдяки менеджеру пакетів NPM. Він включає в себе більше ніж 500 000 модулів та open-source бібліотек, які знаходяться у відкритому доступі та полегшують розробку.

2.2 Вибір технологій для розробки

Враховуючі зазначені вище задачі, було обрано такі технології для розробки:

Клієнтська частина:

1. HTML5 (для розмітки)
2. CSS3 (а саме SCSS, для надання сторінці візуальної привабливості)
3. JavaScript (а саме надбудова TypeScript)
4. Angular
5. Material UI (бібліотека з готовими компонентами для ангуляр)

Серверна частина:

1. NodeJS (для організації запису та читання даних з бази даних)

2. Nodemon

3. Mongoose

4. Express

5. JWT (для ідентифікації користувача)

У якості бази даних буде використана хмарна версія MongoDB.

Оскільки продукт передбачає авторизацію перед використанням, то у якості авторизаційного сервісу буде використаний сервіс Auth0. Він спрощує процес авторизації та дозволяє авторизуватись через будь-які соціальні мережі без попередньої реєстрації. А система JWT токенів дозволить з легкістю ідентифікувати користувача.

2.3 Концепція блокування

Концепція блокування доступу до ресурсу буде досить проста. Взагалі, то як буде проводитись процес блокування користувач буде обирати самостійно. На вибір буде реалізовано 3 методи:

1. Перенаправлення користувача на конкретну URL-адресу, яку вкаже користувач у налаштуваннях

2. Перенаправлення на одну з заготовлених сторінок

3. Відкриття пустої вкладки

4. Заміна усього контенту веб-сторінки повідомленням про те, що доступ заблокований

Перед початком роботи користувачу необхідно буде перейти на головну сторінку розширення та авторизуватися через Google аккаунт, за допомогою сервісу Auth0. Auth0 після авторизації повертає JWT токен, який містить інформацію про користувача, але для більшої зручності буде створено власний JWT токен на основі JWT токену від Auth0, це дозволить більш тонко налаштувати інформацію про користувача та записати її у токен.

Після реєстрації користувача у базі даних та створення власного токену, на frontend частину буде відправлений сгенерований токен, який буде використовуватись для ідентифікації користувача при додаванні URL-адреси до чорного списку. Основна ідея для чого це потрібно полягає у тому, щоб навіть якщо користувач відкриє новий браузер, або буде використовувати інший комп'ютер, щоб він мав можливість увійти до свого аккаунту та застосувати ті налаштування, які він робив на першому комп'ютері.

Отже після успішного отримання токену, користувач матиме можливість його скопіювати. Далі потрібно активувати розширення та вставити скопійований токен у поле вводу, після чого функціонал додавання веб-сторінок до чорного списку буде доступний.

Щоб додати веб-сторінку до чорного списку у користувача буде 2 варіанти:

- Використовувати розширення;

- Використовувати сторінку управління розширенням.

Якщо використовувати розширення, то користувач матиме змогу додати до чорного списку тільки поточну відкриту сторінку. У випадку якщо користувач буде використовувати сторінку управління розширенням, то там він матиме змогу додати декілька URL-адрес до списку, а також переглянути їх.

Способів розблокування веб-сторінки також буде декілька. Який спосіб буде використаний залежить від налаштувань користувача. Всього їх буде декілька:

- Користувач власноруч видалив URL-адресу із списку;

- Користувач увів пароль (який він попередньо встановив);

- Час блокування сторінки вийшов (у ситуації, коли користувач вказав у налаштуваннях дату до якої буде заблокований ресурс).

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Структурна схема додатку

Структурна схема – схема, що відображає склад і взаємодію з управління частин продукту, що розробляються.

Діаграма використання (use case diagram) - це найбільш загальне уявлення функціонального призначення системи. На діаграмі використання застосовуються два типи основних сутностей: варіанти використання і дійові особи , між якими встановлюються такі основні типи відносин:

- асоціація між дійовою особою і варіантом використання;
- узагальнення між діючими особами;
- узагальнення між варіантами використання;
- залежності між варіантами використання.

Користувач має можливість авторизуватися та налаштувати конфігурацію для блокування. Коли люди авторизована, вона має можливість додавати та видаляти сторінки до списку, а також оновлювати конфігурацію. Щодо функціоналу додатку, коротко він представлений на рисунку (рис. 3.1).

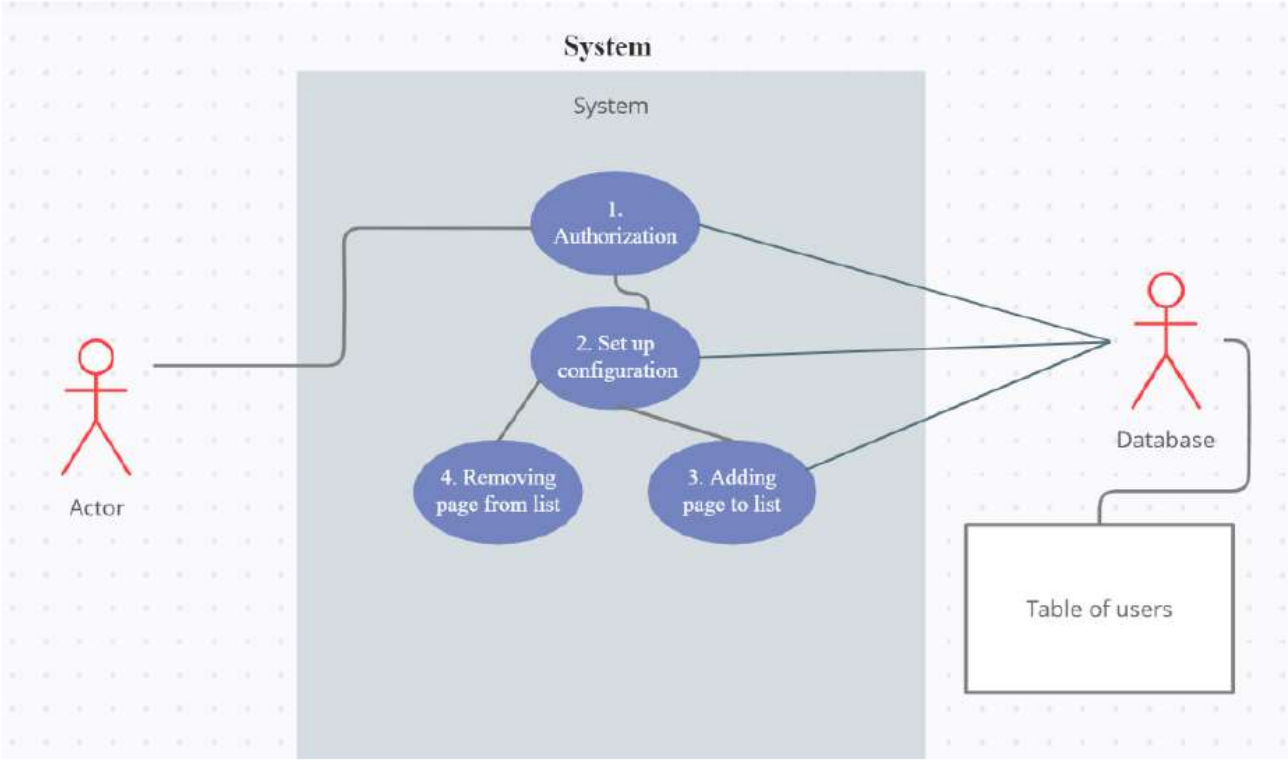


Рис. 3.1 Use-case діаграма

Одним з найбільш відомих і зрозумілих методів проектування реляційних баз даних є метод, в основу якого покладена модель «сутність-зв’язок».

Ключовими елементами моделі являються сутності, їх зв’язки та характеристики [13].

Модель має 4 сутності:

- «Гість»;

- «Профіль»;

- «Веб-ресурси»

Кожен гість має можливість створити лише 1 профіль. Користувач може заблокувати декілька веб-ресурсів.(рис. 3.2).

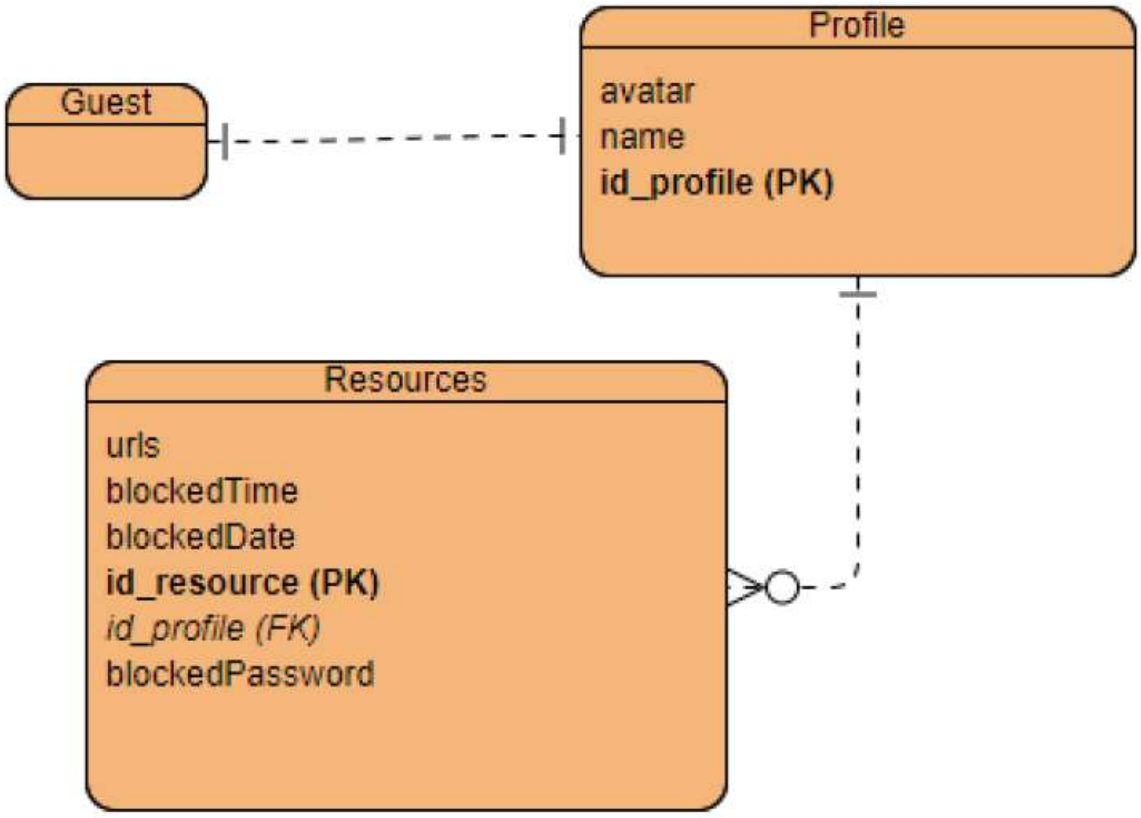


Рис.3.26 ER-діаграма

3.2 Реалізація авторизації за допомогою сервісу Auth0

Авторизація є ключовою частиною даного розширення, оскільки зберігання параметрів та налаштувань кожного користувача буде зберігатись у базі даних, тому авторизація допоможе ідентифікувати користувача та полегшить їх менеджмент.

Існує багато способів як ідентифікувати користувача. Наприклад розробити реєстрацію, де користувачу потрібно буде ввести електронну пошту та пароль. Проте у такому випадку потрібно також розробити алгоритм підтвердження електронної пошти, щоб уникнути спаму реєстрацій. Але є більш елегантний спосіб ідентифікувати користувача, це використати сервіс Auth0.

Auth0 – це платформа автентифікації та авторизації для додатку. Він надає усі інструменти необхідні для створення та запуску безпечної інфраструктури ідентифікації, включаючи автентифікацію, захист даних та управління паролями. Auth0 можна використовувати для реалізації єдиного логіну, безпарольної, багатофакторної автентифікації та багато іншого. Він надає API, тому розробники можуть використовувати Auth0 у своїх додатках для реалізації автентифікації своїх користувачів. Auth0 має ряд переваг, а саме:

- Автентифікація на основі JWT токену
- Автентифікація через соціальні мережі
- Багатофакторна автентифікація
- Можливість реалізації функціоналу зміни та нагадування паролю

Auth0 допомагає підготувати найбільш захищений додаток. Він має багато функцій, які роблять його чудовим варіантом для розробників, а саме:

- Універсальний логін
- Єдина точка входу
- Багатофакторна автентифікація
- Логін без паролю

Для реалізації авторизації спочатку створимо сторінку з якої користувач буде виконувати вхід. Напишемо просту верстку, де буде відображена кнопка авторизації (рис. 3.3).

```
1:  <div class="container">
2:    <h2 class="authHeader">Please authorize!</h2>
3:    <button class="authBtn" mat-stroked-button color="primary"
    (click)="registerViaAuth0()">Authorize</button>
4:  </div>
```

Рис. 3.3 Код верстки сторінки входу

Вона буде мати такий зовнішній вигляд (рис. 3.4).

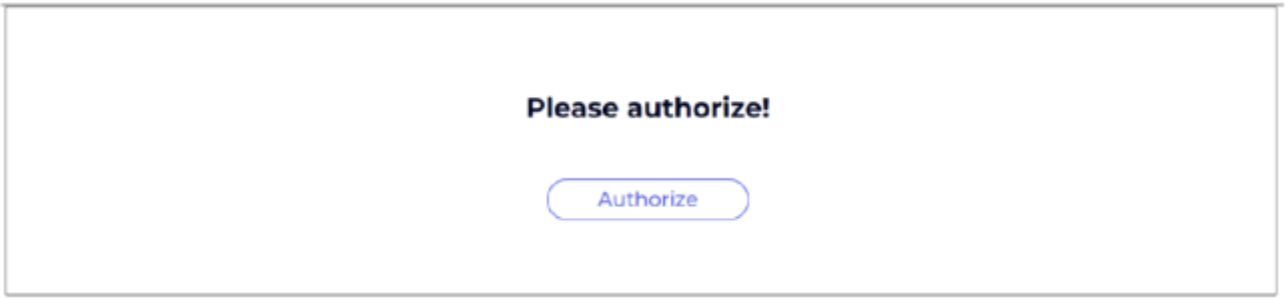


Рис. 3.4 Візуальний вигляд сторінки логіну

Далі підключимо сервіс Auth0 для реалізації авторизації. Для цього перейдемо на офіційний сайт Auth0, розділ Applications та створимо новий додаток (рис 3.5).



Рис.3.5 Підключення сервісу Auth0

У вікні оберемо Single Page Web Application, оскільки при розробці використовується Angular та натиснемо Create. Далі потрібно налаштувати створений додаток. У секції Application URLs, а саме у розділі Allowed Callback URLs, потрібно записати URL-адресу сторінки нашого розширення на яку буде перенаправлено користувача після авторизації. Також у розділі Allowed Logout

URLs запишемо URL-адресу сторінки на яку буде перенаправлено користувача після того, як він вийде з аккаунту (рис 3.6).

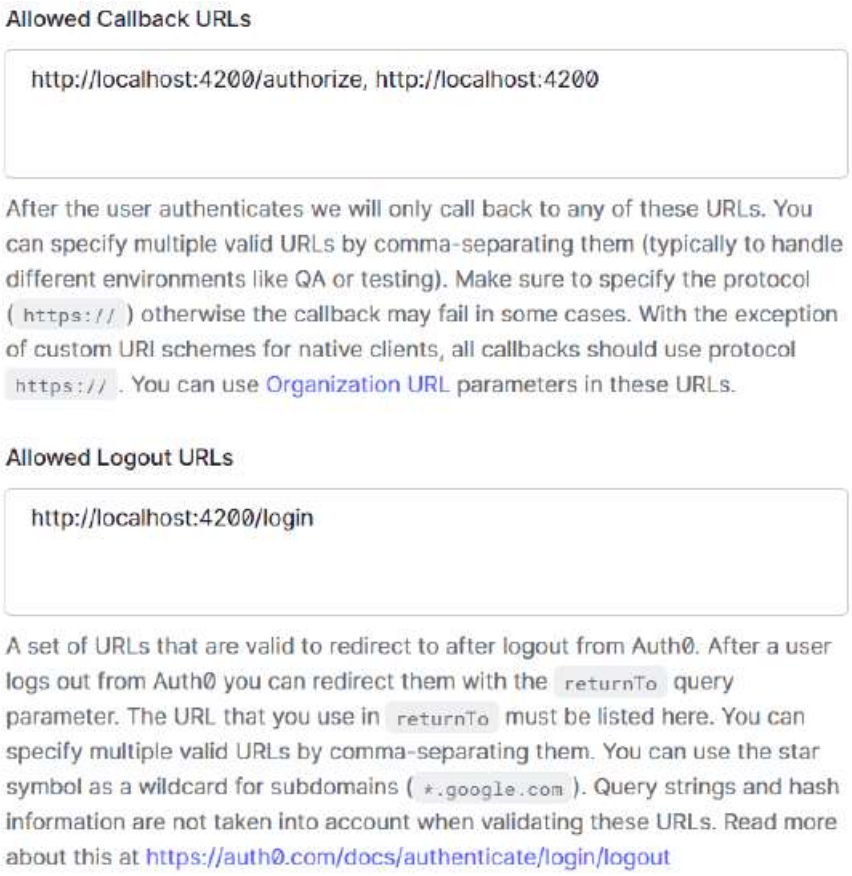


Рис. 3.6 Налаштування Auth0

Збережемо зміни та напишемо логіку авторизації на сторінці логіну. Для використання класів та функцій Auth0 потрібно встановити відповідний npm пакет командою *npm i @auth0/auth0-angular* . Створимо метод, який буде викликатися після натиснення на кнопку Authorize. Даний метод буде перенаправляти користувача у сервіс Auth0 для подальшого логіну (рис. 3.7).

```
1:   registerViaAuth0() {
2:       this.auth.loginWithRedirect({
3:         appState: { target: '/authorize' },
4:       });
5:     }
```

Рис. 3.7 Реалізація перенаправлення на сервіс Auth0

Після авторизації користувач отримає JWT токен, який містить інформацію про нього, але оскільки було прийняте рішення про те, щоб

генерувати власний JWT токен, на основі існуючого, то потрібно на backend частині проекту отримувати JWT токен від Auth0 та генерувати власний. Також функцією backend частини буде запис нового користувача до бази даних, якщо до цього його не було там. Тож на backend частини створимо ендпоінт, який буде приймати JWT токен від Auth0, парсити його, та на основі інформації з нього створювати нового користувача та записувати до бази даних. Повний код backend частини (додаток А).

На клієнтській частині створимо сторінку Home, на яку користувач буде перенапрвлений після отримання нашого сгенерованого JWT токену.

Протестуємо чи дійсно працює авторизація. На сторінці login натиснемо на кнопку Authorize, після чого користувач буде перенаправлений у сервіс авторизації Auth0, де матиме змогу авторизуватися через Google акаунт (рис 3.8).

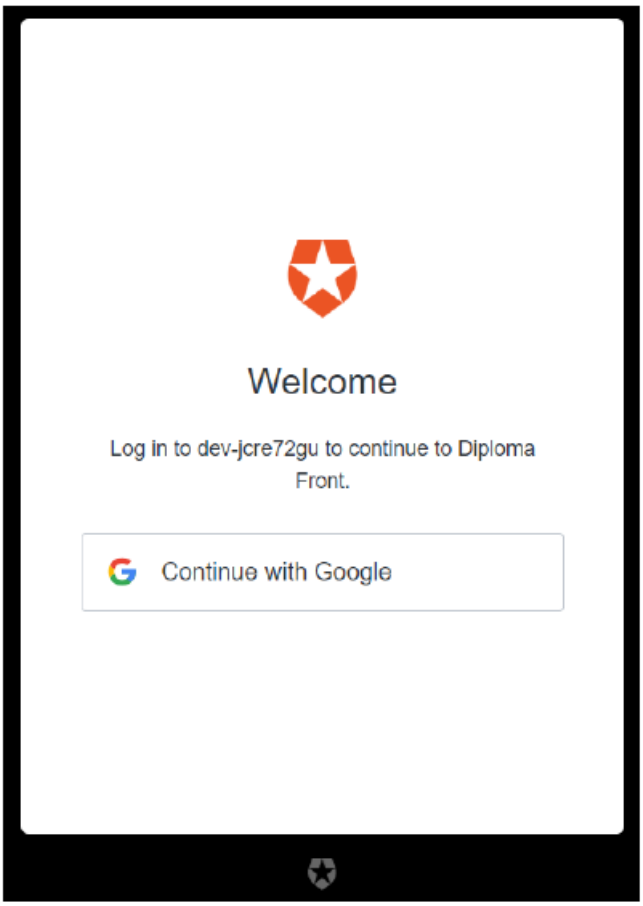


Рис. 3.8 Авторизація за допомогою Auth0

Після вибору Google акаунта, на backend частині у логах можна побачити, що спрацював ендпоінт на авторизацію користувача на основі JWT токену з Auth0. Також користувач потрапив на сторінку Home, де буде реалізоване управління заблокованими ресурсами (рис 3.9).

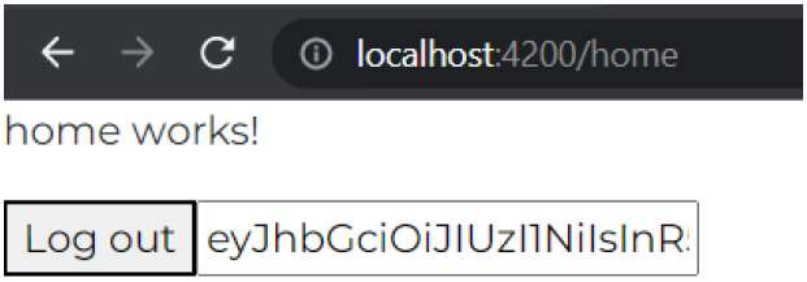


Рис. 3.9 Сторінка після аторизації

3.3 Реалізація сторінки налаштувань

Після встановлення розширення користувачу потрібно буде налаштувати його. А саме вказати сторінки, які він хоче заблокувати, обрати час блокування та пароль для розблокування. Також на цій сторінці користувач буде бачити власний токен, який він буде використовувати при використанні розширення.

Для початку потрібно створити макет для майбутньої сторінки. У якості сервісу був обраний онлайн-сервіс Figma. На мій погляд Figma буде кращим сервісом, ніж Photoshop, тому що цей ресурс має такі плюси: figma працює як на Windows, Mac так і на телефонах, це безкоштовний продукт, уся робота ведеться на сервері, тому не потрібно турбуватися за передачу макету між дизайнером та верстальником, figma дає можливість працювати над макетом декільком людям одночасно.

Всього буде одна сторінка, де користувач може знайти власний токен, а також управляти заблокованими сторінками (рис 3.10).

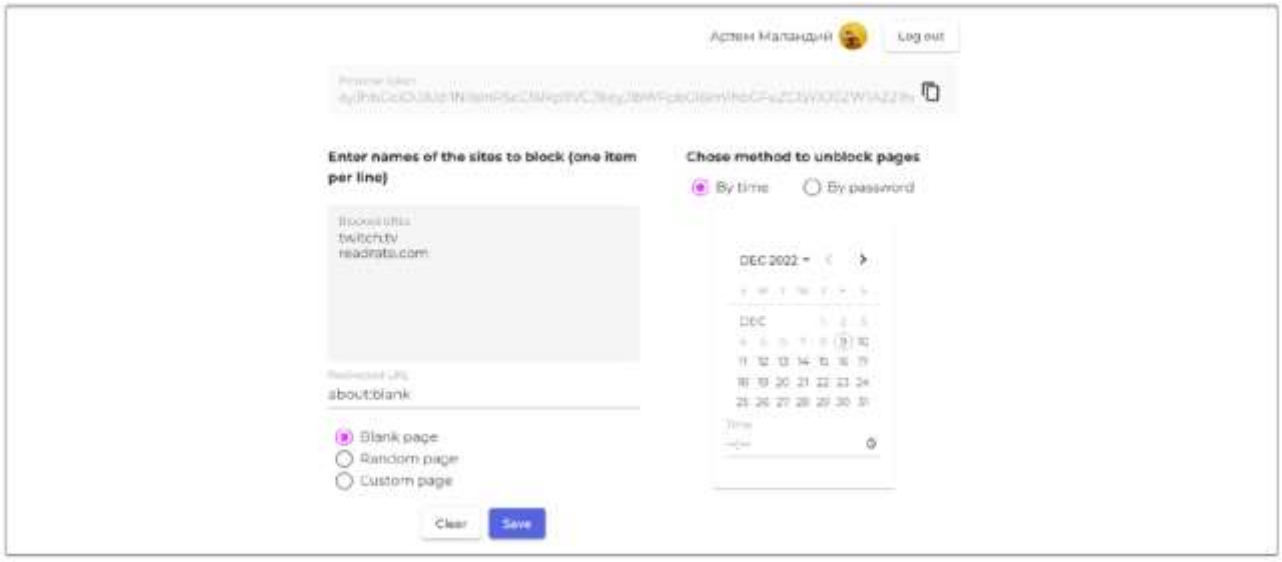


Рис. 3.10 Сторінка управління налаштуваннями

Для стилізації простих елементів таких як: кнопки, поля вводу, каледнару, радіобатонів и тд. буде використана бібліотека Angular Material UI. Вона містить у собі готові рішення, які не потребують багатьох зусиль для їх використання. Повний код frontend частини (додаток Б).

Після створення візуальної частини сторінки, на backend частині потрібно створити потрібні ендпоінти для запису та читання даних з бази даних. Перший ендпоінт був створений для авторизації та генерації власного JWT токену.

Другий ендпоінт буде відповідати за читання даних конкретного користувача (рис. 3.11). Повний код backend частини (додаток А).

```
exports.getUserInfo = (req, res, next) => {
    try {
        User.find({_id: req.params.userId})
            .exec()
            .then(user => {
                return res.status(200).json({
                    userInfo: user
                });
            })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Рис. 3.11 Ендпоінт на отримання повної інформації про користувача

Наступний ендпоінт буде відповідати за оновлення інформації користувача, а також за зміну налаштувань (рис. 3.12).

```
exports.updateUserInfo = (req, res, next) => {
    try {
        const blockedPages = req.body.blockedPages[0] === '' ? [] : req.body.blockedPages;
        const blockedDate = req.body.blockedDate;
        const blockedPassword = req.body.blockedPassword;
        const redirectUrl = req.body.redirectUrl;
        const blockedTime = req.body.blockedTime;
        User.updateOne({_id: req.params.userId}, {
            settings: {
                blockedPages: blockedPages,
                blockedDate: blockedDate,
                blockedPassword: blockedPassword,
                redirectUrl: redirectUrl,
                blockedTime: blockedTime
            }
        }).then(() => {
            return res.status(200).json({
                status: 'List has been updated'
            });
        });
    } catch (err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Рис. 3.12 Ендпоінт на оновлення інформації про користувача

Четвертий ендпоінт відповідний за додавання URL-адреси до списку заблокованих. У якості параметрів він приймає URL-адресу, та якщо у базі її ще немає, то записує, в інакшому випадку ігнорує (рис. 3.13).

```
exports.addPage = (req, res, next) => {
    const blockedPage = req.body.blockedPage;
    User.find({_id: req.params.userId})
        .exec()
        .then(user => {
            const blockedPages = user[0].settings.blockedPages;
             if (!user[0].settings.blockedPages.includes(blockedPage)) {
                blockedPages.push(blockedPage);
                User.updateOne({_id: user[0]._id}, {
                    settings: {
                        blockedPages: blockedPages,
                        blockedDate: user[0].settings.blockedDate,
                        blockedPassword: user[0].settings.blockedPassword,
                        redirectUrl: user[0].settings.redirectUrl,
                        blockedTime: user[0].settings.blockedTime
                    }
                }).then(() => {
                    return res.status(200).json({
                        status: 'List has been updated'
                    });
                });
            }})
}
```

Рис. 3.13 Ендпоінт на додавання сторінки до чорного списку

І останній ендпоінт відповідний за видалення URL-адреси, якщо кристувач ввів пароль або пройшов часовий строк блоку (рис. 3.14).

```
exports.removePage = (req, res, next) => {
    const removedPage = req.body.removedPage;
    User.find({_id: req.params.userId})
        .exec()
        .then(user => {
            const blockedPages = user[0].settings.blockedPages;
            const indexOfBlocked = blockedPages.indexOf(removedPage);
            const newBlockedPages = [...blockedPages.slice(0,
indexOfBlocked),...blockedPages.slice(indexOfBlocked + 1)];
            User.updateOne({_id: user[0]._id}, {
                settings: {
                    blockedPages: newBlockedPages,
                    blockedDate: user[0].settings.blockedDate,
                    blockedPassword: user[0].settings.blockedPassword,
                    redirectUrl: user[0].settings.redirectUrl,
                    blockedTime: user[0].settings.blockedTime
                }
            }).then(() => {
                return res.status(200).json({
                    status: 'List has been updated'
                });
            });
        })
}
```

Рис. 3.14 Ендпоінт на видалення сторінки з чорного списку

Останнє, що потрібно зробити на backend частині це створити middleware, який буде перевіряти валідність користувача, шляхом читання токену із заголовку та його парсингу. Middleware це така функція, яка буде спрацьовувати кожен раз, коли на сервер приходе будь-який запит. Тобто перед початком роботи з базою даних, запит пройде через middleware, і якщо перевірка пройдена успішно, то запит йде далі до ендпоінту. Це досить простий, але надійний захист від намагання хакерів підробити запит від імені іншого користувача. Отже, middleware на перевірку авторизації буде виглядати наступним чином (рис .3.15).

```
module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decoded = jwt.verify(token, "secret");
        req.userData = decoded;
        next();
    } catch (error) {
        return res.status(401).json({
            message: 'Auth failed'
        });
    }
};
```

Рис. 3.15 Middleware перевірки авторизації

Після створення візуальної частини сторінки та написання ендпоінтів приступимо до написання логіки зміни налаштувань на frontend частині. У компонент Home імпортуємо наступні сервіси для подальшої роботи: Router, AuthService, GetUserInfoService, FormBuilder. GetUserInfoService це наш власний сервіс, який відповідає на створення запитів на backend частину.

У життєвому циклі onInit запишемо логіку, яка буде спрацьовувати, коли компонент буде проініціалізований (рис. 3.16).

```
ngOnInit(): void {
  this.minDate = new Date();
  this.createForm();
  this.personalToken = localStorage.getItem('accessToken');
  this.getUserInfoService.getUserInfo().subscribe((data: UserInfo) => {
    this.fillForm(data.userInfo[0].settings);
    this.userEmail = data.userInfo[0].name;
    this.imgSrc = data.userInfo[0].avatar;
  });
}
```

Рис.3.16 Логіки при створенні компонента

Перш за все встановлюємо поточну дату для календаря, щоб неможливо було обрати ранішу дату. Далі створюємо форму, яка буде містити усі введені налаштування. Зберігаємо персональний токен для відображення на сторінці.

Відправляємо запит до серверу, щоб отримати актуальні дані користувача та автоматично заповнити сторінку налаштувань.

Далі створимо метод, який буде спрацьовувати після відправки форми.

Цей метод буде генерувати json-об’єкт та відправляти його на сервер для того, щоб зміни були записані у базі даних (рис. 3.17).

```
1:     onSubmit(valueFromForm) {
2:       this.settings.blockedPassword = valueFromForm.formPassword;
3:       this.settings.blockedPages = this.stringToArray(valueFromForm.formBlockedPages);
4:       this.settings.blockedDate = valueFromForm?.formDate;
5:       this.settings.blockedTime = this.timeInMilisec(valueFromForm?.formTime);
6:       this.settings.redirectUrl = valueFromForm.formRedirect;
7:       this.getUserInfoService.setUserInfo(this.settings);
8:     }
```

Рис.3.17 Логіка відправки налаштувань на сервер

3.4 Реалізація розширення для браузеру

Розширення для браузеру грає ключову роль у даному проекті, тому що саме воно буде відслідковувати поточку URL-адресу та виконувати код для блокування її. Найголовнішим файлом розширення є manifest.json, у ньому зібрана уся інформація про розширення. За допомогою цього файлу браузер зможе ідентифікувати код, як код розширення. Manifest виглядає наступним чином (3.18).

```
{
  "name": "My app",
  "version": "0.0.1",
  "description": "Description of my app",
  "manifest_version": 3,
  "action": {
    "default_popup": "index.html"
  },
  "background": {
    "service_worker": "background.js",
    "type": "module"
  },
  "host_permissions": [
    "http://*/*",
    "https://*/*",
    "*://*/*"
  ],
  "permissions": [
    "tabs",
    "unlimitedStorage",
    "activeTab",
    "declarativeContent",
    "debugger",
    "alarms",
    "webNavigation",
    "storage"
  ]
}
```

Рис. 3.18 Файл маніфесту

У цьому файлі описується загальна інформація про розширення це ім’я, версія, опис, версія маніфесту, посилання на файл, який відображає вікно при активації розширення, посилання на скрипт, який виконується у фоновому режимі, навіть тоді, коли розширення деактивоване, і останніми йдуть дозволи, які є у даного розширення.

Перш за все спочатку потрібно створити вікно, яке буде відображатися при активації розширення. Оскільки у реалізації розширення в основі лежить Angular, то створимо компонент Home, який буде виступати у вигляді основної сторінки. Повний код розширення (додаток В). Дане вікно буде мати 2 стани: коли користувач ввів персональний токен та коли не вводив. Вікно у другому стані буде виглядати наступним чином (рис. 3.19).

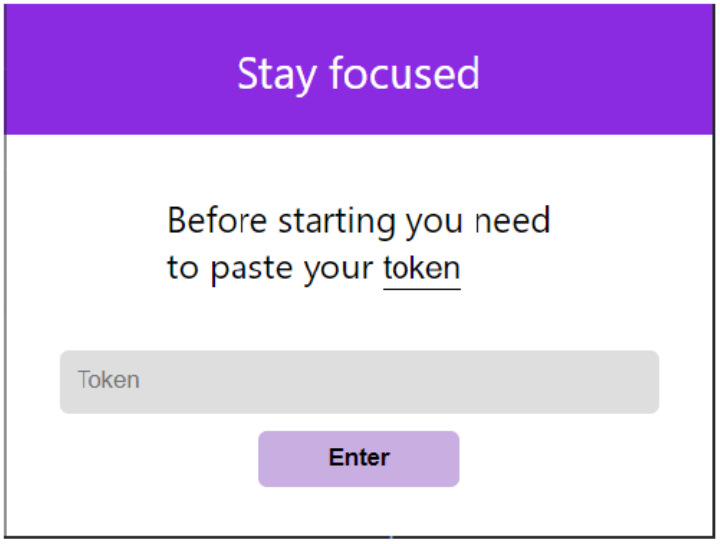


Рис. 3.19 Другий стан розширення

Вікно під час першого стану (рис. 3.20).

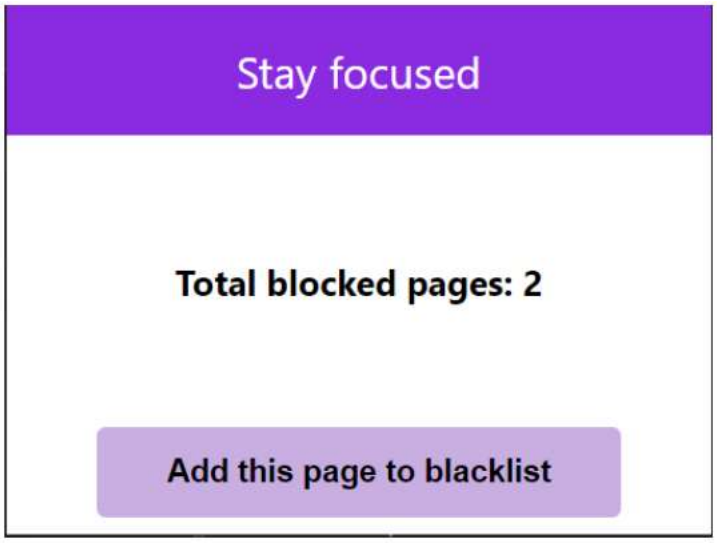


Рис. 3.20 Перший стан розширення

Окрім компоненту Home, також створимо сервіс, який буде відповідати за відправку запитів на сервер, а саме: отримання даних про користувача та додавання поточної сторінки до списку заблокованих. Метод отримання даних користувача (рис. 3.21).

```
1:    getUserInfo(userId: string) {
2:      return this.http.get<UserInfo>(`http://localhost:3000/user/getUserInfo/${userId}`, {
3:        headers: {
4:          'Authorization': `Bearer ${localStorage.getItem('accessToken')}`
5:        }
6:      });
7:    }
```

Рис. 3.21 Отримання інформації про користувача

Та метод додавання сторінки до списку (рис. 3.22).

```
addPage(pageUrl: string) {
    const userId = localStorage.getItem('userId');
    return this.http.post<StatusOfAdding>(`http://localhost:3000/user/addPage/${userId}`,
{
      blockedPage: pageUrl
    },{
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('accessToken')}`
      }
    }).subscribe(data => {
      if (data.status === 'List has been updated') {
        this.listUpdated$.next(true);
      }
    });
  }
```

Рис.3.22 Додавання сторінки до чорного списку

Наступним етапом буде реалізація скрипта, який буде виконуватись у фоновому режимі, відслідковувати поточну URL-адресу, перевіряти чи знаходиться вона у списку заблокованих та проводити перенаправлення користувача на задану URL-адресу.

Відслідковування поточної URL-адреси буде виконуватись за допомогою івенту, який зареєстрований на вкладках браузера onUpdated. Даний івент спрацьовує кожний раз коли оновлюється вкладка бразуера, тому при його спрацюванні буде проводитись порівняння зі списком заблокованих сторінок.

Оскільки даний івент спрацьовує ще тоді, коли вкладка не завантажилась, то можна буде уникнути ситуації, коли перехід на заблоковану сторінку дозволяв на декілька секунд бачити контент сторінки.

Фоновий скрипт повинен мати можливість відправляти запити за серверну частину, тому йому потрібно зберігати унікальний токен та id користувача. Для зберігання цих даних було використано локальне сховище, де можна зберігати невеликі дані. Отже реалізація івенту onUpdated виглядає наступним чином (рис. 3.23).

```
chrome.tabs.onUpdated.addListener(async (tabId, changeInfo, tab) => {
try{
  if (changeInfo.status === 'loading' && changeInfo.url) {
    const userId = await LS.get('userId');
    const accessToken = await LS.get('accessToken');

    getBlockedList(userId.userId, accessToken.accessToken, tabId, changeInfo.url);
  }
} catch (e) {
  console.log(e);
}
});
```

Рис. 3.23 Івент onUpdated

Метод GetBlockedList безпосередньо виконує пошук співпадінь поточної URL-адреси з URL-адресами, які знаходяться у базі даних. Та у випадку співпадіння перенаправляє користувача на задану сторінку. Повний код розширення знаходиться у додатку В.

3.5 Реалізація відображення повідомлення, що веб-ресурс заблокований

Оскільки у налаштуваннях блокування було реалізовано, що користувач може обрати сторінку на яку його буде автоматично перенаправляти при відвіданні заблокованої сторінки. Потрібно розробити власну сторінку, на якій буде відображатися повідомлення, що веб-ресурс заблокований, та показувати дату після якої він буде автоматично розблокований, а також пароль, при введені якого сторінка автоматично буде розблокована, за умови, що користувач у налаштуваннях задав дату та пароль для блокування. Візуальний вигляд сторінка буде мати наступний (рис. 3.24). За умови, що користувач додав дату розблокування та пароль, якщо ні, то буде відображено лише повідомлення.

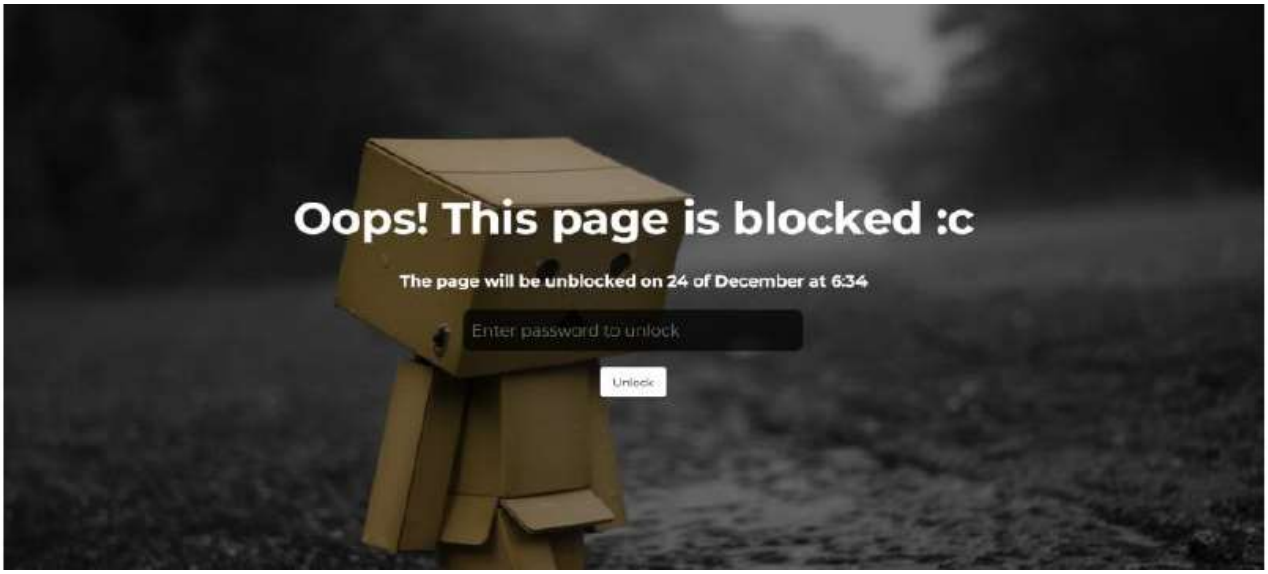


Рис. 3.24 Повідомлення, що сторінка заблокована

Дана сторінка відноситься до frontend частини тому повний код знаходиться у додатку Б.

3.6 Тестування розширення

Завантажимо розширення, та активуємо. Після чого перейдемо до сторінки налаштувань та скопіюємо унікальний токен для подальшої роботи з розширенням. Також налаштуємо поведінку роботи розширення. У якості посилання для перенаправлення оберемо ту сторінку, яка була створена нами, також оберемо час після якого сторінка буде автоматично розблокована (рис. 3.25)

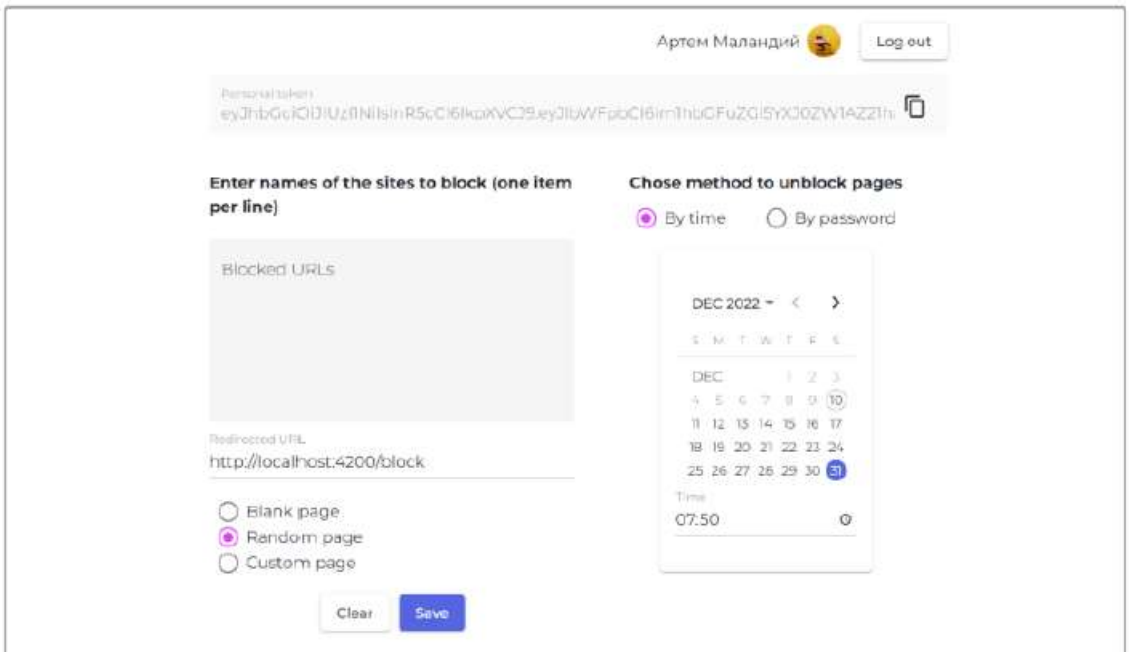


Рис. 3.25 Налаштування для блокування

Збережемо зміни та перейдемо на сторінку, яку хочемо заблокувати (рис. 3.26)

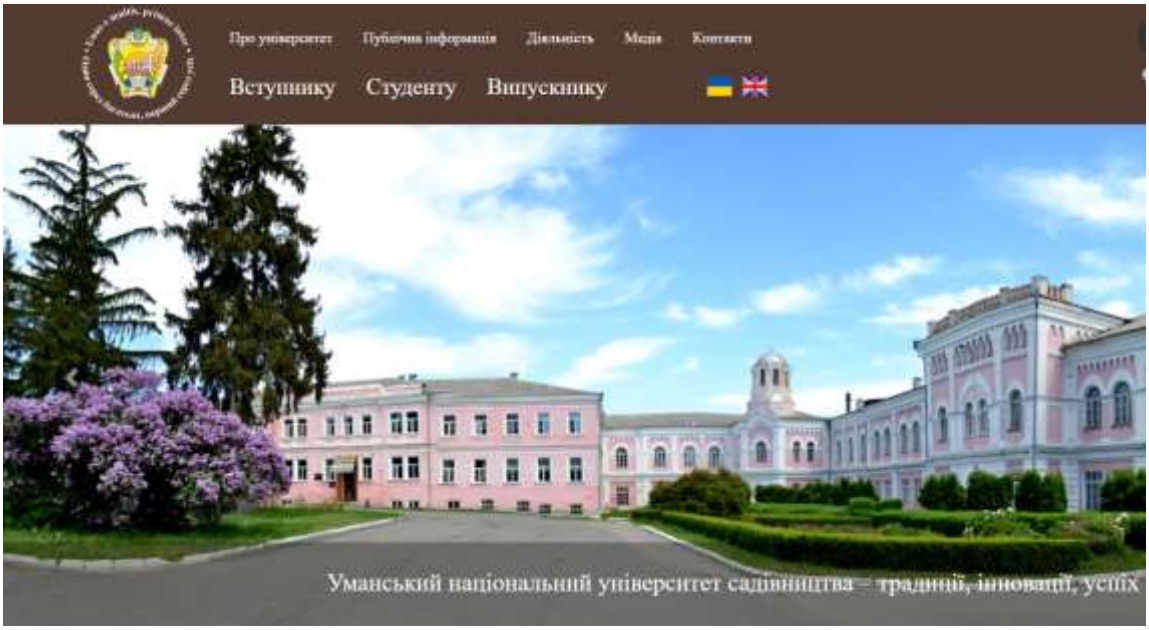


Рис. 3.26 Веб-ресурс до блоку

Знаходячись на цій сторінці активуємо розширення, введемо унікальний токен та додамо поточку сторінку до списку заблокованих (рис. 3.27).

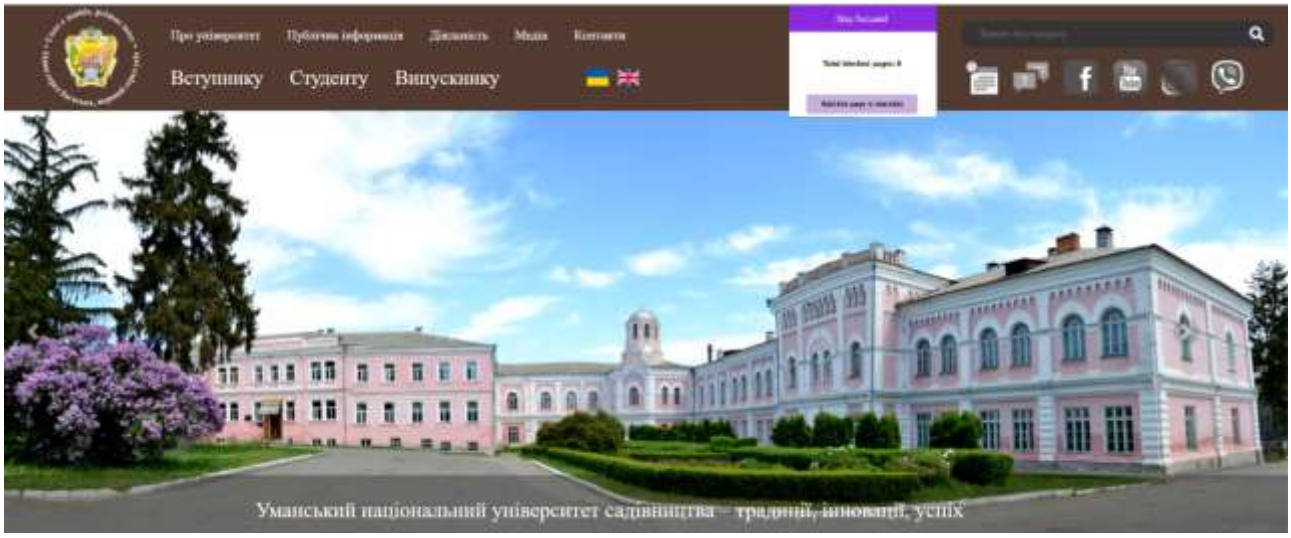


Рис.3.27 Додавання веб-ресурсу до чорного списку

Після додавання сторінки до списку заблокованих, користувач був перенаправлений на сторінку, вказану у налаштуваннях. Як можемо бачити тут також відображається дата, після якої сторінка буде автоматично розблокована (рис. 3.28).

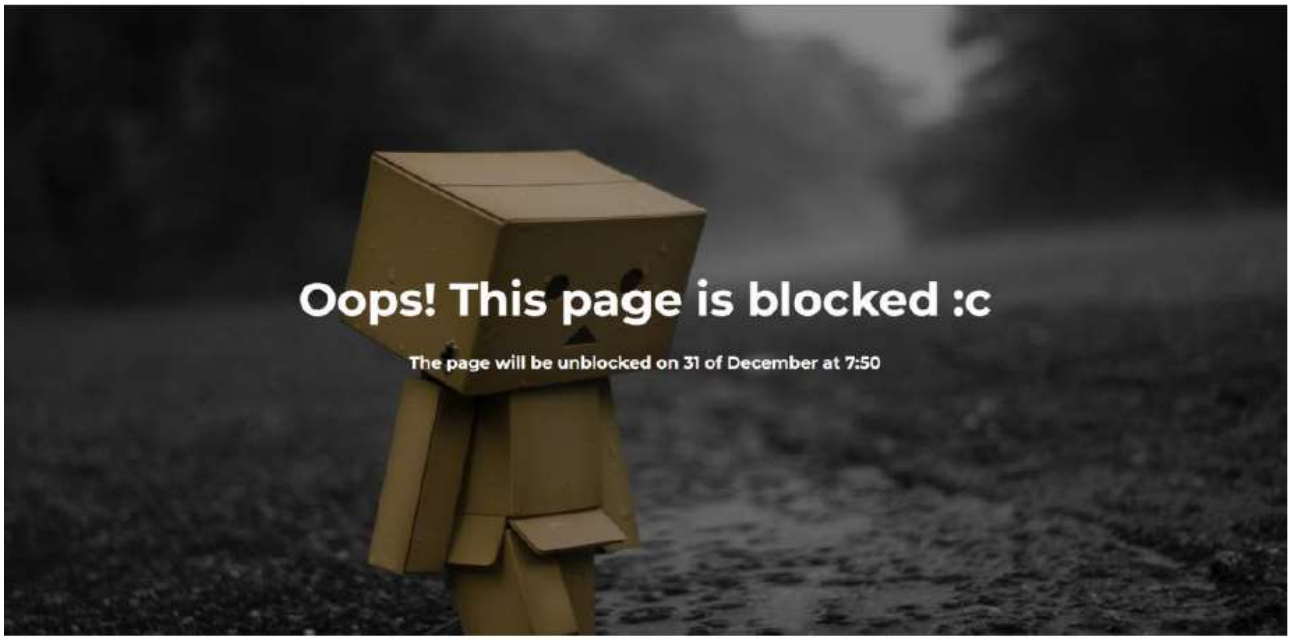


Рис. 3.28 Перевірка роботи розширення

У налаштуваннях тепер відображається домен тієї сторінки, яку ми додали до списку. Тобто блокування проходить не по конкретній URL-адресі, а по домену, що є біль ефективним (рис. 3.29).

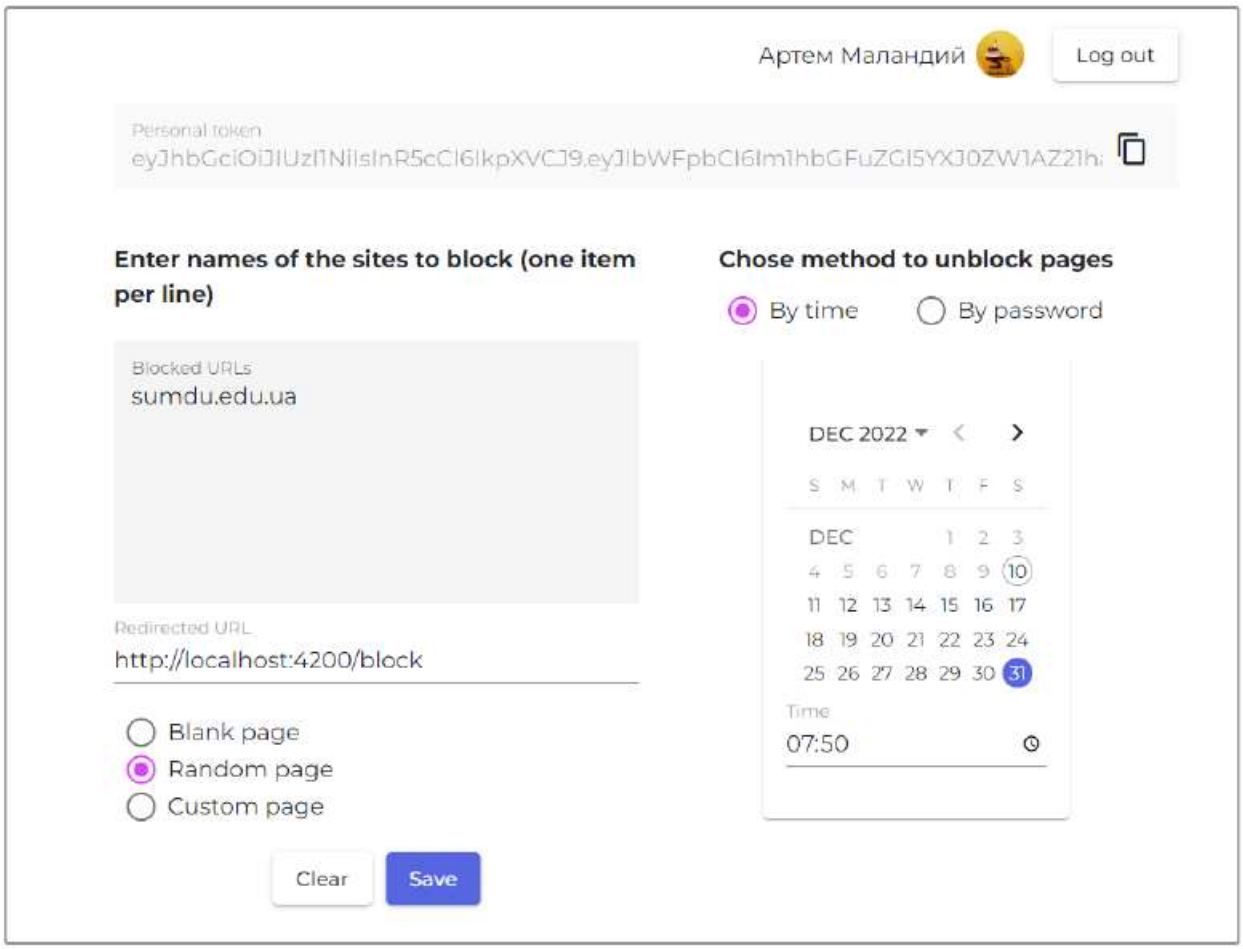


Рис.3.29 Відображення домену заблокованої сторінки

Після видалення домену з цього списку та збереження налаштувань, заблокований веб-ресурс знову стане доступний для відвідання.

ВИСНОВКИ

Метою роботи було створення системи, яка б дозволяла обмежувати доступ до певного ресурсу. Для отримання максимально ефективного результату було виконано наступні задачі: проаналізовані сучасні методи блокування, виділені їх переваги та недоліки, було обрано засоби реалізації системи, які найбільш підходять під постановку задачі, було реалізовано авторизацію, через сервіс Auth0, який є зручним та безпечним способом ідентифікації користувача, були розроблені 3 частини проекту: backend, що відповідає за роботу з базою даних, frontend, що відповідає за сторінку налаштувань для користувача, та extension, що безпосередньо є розширенням, яке аналізує поточну URL-адресу та проводить блокування. При розробці системи були враховані переваги та недоліки існуючих рішень, тому система виявилась ефективнішою, ніж деякі аналоги.

Процес блокування базується на аналізі поточної URL-адреси та у разі находження цієї адреси у списку заблокованих, перенаправляє користувача на інший веб-ресурс, який він вказав у налаштуваннях. Розблокування сторінки доступне декількома способами: на сторінці налаштувань із списку видалити потрібний веб-ресурс, або якщо користувач вказав час для розблокування або пароль. Серед переваг системи можна виділити наступні:

- Висока швидкодія за рахунок низького використання оперативної пам’яті;

- Гнучка система налаштувань для заблокованих веб-ресурсів;

- Авторизація через Auth0, завдяки якій навіть при зміні комп’ютера користувач матиме можливість продовжити користуватися розширенням

Тому спираючись на сукупність вищеперерахованих чинників, можна сказати, що система відповідає усім вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Куріцина А.В., Оверчук В.А. Теоретичні підходи до визначення психологічного змісту феномену прокрастинації. *Актуальні проблеми психології.* Випуск 19. Житомир. 2019.

2. Steel P. The Nature of Procrastination: A Meta-Analytic and Theoretical Review of Quintessential Self-Regulatory Failure. Psychological Bulletin. 2007. № 133. P. 65-94.

3. Гура В. С. Цивільно-правова охорона Інтернет-сайту: автореф. дис. … канд. юрид.наук / В. С. Гура. К., 2006. 20 с.

4. Про застосування персональних спеціальних економічних та інших обмежувальних заходів (санкцій): Указ Президента України від 28.04.2017 р. № 133/2017. Урядовий кур'єр. 2017. 29 квіт. (№ 35). С. 10.

5. Технічні канали витоку інформації. Порядок створення технічних комплексів захисту інформації. С. О.Іваненко, О. В. Гавриленко, О. А. Липський, А. С. Шевцов. Київ: ІССЗІ НТУ "КПІ", 2016. 104 с

6. Brooks, David R. - Programming in HTML and PHP. 2017. с. 1-5.

7. Keith J. Grant - CSS in depth 2019. с. 24-36.

8. Marijn Haverbeke - Eloquent JavaScript, 2018. с. 18 – 19

9. Джон Резиг, Беэр Бибо, Иосип Марас Секрети JavaScript ніндзя. Друге видання. 2017. с 30-33.

10. Бабатіна С.І. Синдром академічної прокрастинації і соціальна відповідальність у студентському віці. *Вісник Національного технічного університету України «Київський політехнічний інститут»*. Філософія. Психологія. Педагогіка. 2013. №2. С. 24-25

11. Базика Є.Л. Стан прокрастинації як чинник, який пригнічує формування успішної особистості студента: результати дослідження. *Практична психологія та соціальна робота*. 2014. №5. С. 23-30.

12. Вайда Т.С. Прокрастинація як компонент поведінки працівників ОВС та її профілактика під час професійної підготовки курсантів у ВНЗ МВС України. *Юридичний бюлетень*. 2016. №2. С. 197-211.

13. Saied M.M. Tahaghoghi, Hugh E. Williams Learning MySQL, 2007 С. 48.

ДОДАТОК А

Налаштування серверу

```
const http = require('http');
const app = require('./app');

const port = process.env.PORT || 3000;

const server = http.createServer(app);

server.listen(port);
```

Лістинг головного файлу серверу

```
const express = require('express');
const app = express();
const morgan = require('morgan');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

const userRoutes = require('./api/routes/user');

mongoose.connect('mongodb+srv://admin:J5W1m4VI9QF7Sx6y@artem-
diploma.l8qlcbl.mongodb.net/?retryWrites=true&w=majority');
mongoose.Promise = global.Promise;

app.use(morgan('dev'));
app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());

app.use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', '*');

    if (req.method === 'OPTIONS') {
        res.header('Access-Control-Allow-Methods', 'PUT, POST, PATH, DELETE, GET');
        return res.status(200).json({});
    }
    next();
});

app.use('/user', userRoutes);

app.use((req, res, next) => {
   const error = new Error('Not found');
   error.status = 404;
   next(error);
});

app.use((error,req, res, next) => {
    res.status(error.status || 500);
    res.json({
        error: {
            message: error.message
        }
    })
});

module.exports = app;
```

Лістинг middleware на перевірку авторизації

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decoded = jwt.verify(token, "secret");
        req.userData = decoded;
        next();
    } catch (error) {
        return res.status(401).json({
            message: 'Auth failed'
        });
    }
};
```

Лістинг створення моделі користувача

```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,
    email: {
        type: String,
        required: true,
        unique: true
    },
    settings: {
        blockedPages: [
            { type: String, }
        ],
        blockedDate: {
            type: Number | null,
        },
        blockedTime: {
            type: Number,
        },
        blockedPassword: {
            type: String | null,
        },
        redirectUrl: {
            type: String | null,
        }
    },
    name: {
        type: String
    },
    avatar: {
        type: String
    }
});

module.exports = mongoose.model('User', userSchema);
```

Лістинг реєстрації усіх ендпоінтів для роботи системи

```
const express = require("express");
const router = express.Router();
const checkAuth = require('../middleware/check-auth');
const UserController = require('../controllers/user');

router.get('/', checkAuth, UserController.user_get_all);
router.post('/authorize-via-aut0', UserController.authorize_via_aut0);
router.get('/getUserInfo/:userId', checkAuth, UserController.getUserInfo);
router.post('/updateUserInfo/:userId', checkAuth, UserController.updateUserInfo);
router.post('/addPage/:userId', checkAuth, UserController.addPage);
router.post('/removePage/:userId', checkAuth, UserController.removePage);
router.delete('/:userId', checkAuth, UserController.user_delete);

module.exports = router;
```

Лістинг контролерів: Контролер авторизації

```
exports.authorize_via_aut0 = (req, res, next) => {
    try {
        const decoded = jwt.decode(req.body.token);
        User.find({email: decoded.email})
            .exec()
            .then(user => {
                if (user.length === 0) {
                    // Write user in database
                    const newUser = new User({
                        _id: new mongoose.Types.ObjectId(),
                        email: decoded.email,
                        name: decoded.name,
                        avatar: decoded.picture,
                        pages: []
                    });
                    newUser.save()
                        .then(result => {
                            const token = jwt.sign({
                                    email: newUser.email,
                                    userId: newUser._id,
                                    name: newUser.name,
                                    avatar: newUser.avatar,
                                },
                                'secret',
                                {
                                    expiresIn: "24h"
                                });
                            return res.status(200).json({
                                message: 'Auth successful',
                                accessToken: token
                            });
                        })
                        .catch(err => {
                            console.log(err);
                            res.status(500).json({
                                error: err
                            });
                        });
                } else {
                    // Create our access token
                    const token = jwt.sign({
                            email: user[0].email,
                            userId: user[0]._id,
                            name: user[0].name,
                            avatar: user[0].avatar,
                        },
                        'secret',
```

```
                                'secret',
                                {
                                    expiresIn: "24h"
                                });
                            return res.status(200).json({
                                message: 'Auth successful',
                                accessToken: token
                            });
                        })
                        .catch(err => {
                            console.log(err);
                            res.status(500).json({
                                error: err
                            });
                        });
                } else {
                    // Create our access token
                    const token = jwt.sign({
                            email: user[0].email,
                            userId: user[0]._id,
                            name: user[0].name,
                            avatar: user[0].avatar,
                        },
                        'secret',
                        {
                            expiresIn: "24h"
                        });
                    return res.status(200).json({
                        message: 'Auth successful',
                        accessToken: token
                    });
                }
            })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Контролер отримання інформації про користувача

```
exports.getUserInfo = (req, res, next) => {
    try {
        User.find({_id: req.params.userId})
            .exec()
            .then(user => {
                return res.status(200).json({
                    userInfo: user
                });
            })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Контролер оновлення інформації про користувача

```
exports.updateUserInfo = (req, res, next) => {
    try {
        const blockedPages = req.body.blockedPages[0] === '' ? [] : req.body.blockedPages;
        const blockedDate = req.body.blockedDate;
        const blockedPassword = req.body.blockedPassword;
        const redirectUrl = req.body.redirectUrl;
        const blockedTime = req.body.blockedTime;
        User.updateOne({_id: req.params.userId}, {
            settings: {
                blockedPages: blockedPages,
                blockedDate: blockedDate,
                blockedPassword: blockedPassword,
                redirectUrl: redirectUrl,
                blockedTime: blockedTime
            }
        }).then(() => {
            return res.status(200).json({
                status: 'List has been updated'
            });
        });
    } catch (err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Контролер додавання сторінки до списку

```
exports.addPage = (req, res, next) => {
    try {
        const blockedPage = req.body.blockedPage;
        User.find({_id: req.params.userId})
            .exec()
            .then(user => {
                const blockedPages = user[0].settings.blockedPages;
                if (!user[0].settings.blockedPages.includes(blockedPage)) {
                    blockedPages.push(blockedPage);
                    User.updateOne({_id: user[0]._id}, {
                        settings: {
                            blockedPages: blockedPages,
                            blockedDate: user[0].settings.blockedDate,
                            blockedPassword: user[0].settings.blockedPassword,
                            redirectUrl: user[0].settings.redirectUrl,
                            blockedTime: user[0].settings.blockedTime
                        }
                    }).then(() => {
                        return res.status(200).json({
                            status: 'List has been updated'
                        });
                    });
                } else {
                    return res.status(200).json({
                        status: 'This page is already exist'
                    });
                }
            })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Контролер видалення сторінки із списку

```
exports.removePage = (req, res, next) => {
    try {
        const removedPage = req.body.removedPage;
        User.find({_id: req.params.userId})
            .exec()
            .then(user => {
                const blockedPages = user[0].settings.blockedPages;
                const indexOfBlocked = blockedPages.indexOf(removedPage);
                const newBlockedPages = [...blockedPages.slice(0, indexOfBlocked),
...blockedPages.slice(indexOfBlocked + 1)];
                User.updateOne({_id: user[0]._id}, {
                    settings: {
                        blockedPages: newBlockedPages,
                        blockedDate: user[0].settings.blockedDate,
                        blockedPassword: user[0].settings.blockedPassword,
                        redirectUrl: user[0].settings.redirectUrl,
                        blockedTime: user[0].settings.blockedTime
                    }
                }).then(() => {
                    return res.status(200).json({
                        status: 'List has been updated'
                    });
                });
            })
    } catch(err) {
        console.log(err);
        res.status(500).json({
            error: err
        });
    }
}
```

Контролер видалення користувача

```
exports.user_delete = (req,res, next) => {
    User.remove({ _id: req.params.userId })
        .exec()
        .then(result => {
            res.status(200).json({
                message: 'User deleted'
            });
        })
        .catch(err => {
            console.log(err);
            res.status(500).json({
                error: err
            });
        });
};
```

Контролер отримання усіх користувачів

```
exports.user_get_all = (req,res, next) => {
    User.find()
        .select("email _id name avatar")
        .exec()
        .then(users => {
            const response = {
                count: users.length,
                users: users.map(user => {
                    return {
                        email: user.email,
                        name: user.name,
                        avatar: user.avatar,
                        _id: user._id,
                    };
                })
            };
            res.status(200).json(response);
        })
        .catch(err => {
            console.log(err);
            res.status(500).json({
                error: err
            });
        });
};
```

ДОДАТОК Б

Компонент Login сторінки:

```
<div class="container">
  <h2 class="authHeader">Please authorize!</h2>
  <button class="authBtn" mat-stroked-button color="primary"
(click)="registerViaAuth0()">Authorize</button>
</div>
```

```
@import './src/app/shared/variables/variables.scss';

.container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding-top: 100px;
}

.authHeader {
  font-size: $authHeaderSize;
  margin-bottom: 64px;
  font-weight: bold;
  color: $first_main_color;
}

.authBtn {
  font-size: $authBtnSize;
  width: 238px;
  height: 49px;
  border: 2px solid;
  border-radius: 25px;

  @media screen and (min-width: 320px) and (max-width: 767px) {
    font-size: $authBtnSizeSmall;
    width: 170px;
    height: 39px;
    border: 1px solid;
  }
}
```

```
import { Component } from '@angular/core';
import { AuthService } from '@auth0/auth0-angular';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
})
export class LoginComponent {

  constructor(public auth: AuthService) {}

  registerViaAuth0() {
    this.auth.loginWithRedirect({
      appState: { target: '/authorize' },
    });
  }
}
```

Компонент сторінки авторизації

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '@auth0/auth0-angular';
import { Auth0Service } from '../../services/github-service/auth0';
import { Router } from '@angular/router';
import { JwtService } from '../../../../../shared/shared-module/services/jwt-service/jwt.service';
import { CONSTANTS } from '../../../../../shared/constants/constants';
import { IAuth0AccessToken } from '../../../../../shared/interfaces/auth0AccessToken';
import { IJwtTokenDecode } from '../../../../../shared/interfaces/jwtTokenDecode';

@Component({
  selector: 'app-auth',
  templateUrl: './auth.component.html',
  styleUrls: ['./auth.component.scss'],
})
export class AuthComponent implements OnInit {
  canClose = false;

  constructor(
    private router: Router,
    private auth0: AuthService,
    private auth0Service: Auth0Service,
    private jwtService: JwtService
  ) {}

  ngOnInit() {
    this.auth0
      .getAccessTokenSilently({
        detailedResponse: true,
      })
      .subscribe((data: IAuth0AccessToken) => this.auth0Service.sendAuth0AccessToken(data.id_token));

    this.auth0Service.getAccessTokenOur().subscribe((token: string) => {
      this.jwtService
        .decodeToken(token)
        .then((tokenInfo: IJwtTokenDecode) => {
          localStorage.setItem(CONSTANTS.LOCAL_STORAGE_KEYS.ACCESS_TOKEN, token);
          localStorage.setItem(CONSTANTS.LOCAL_STORAGE_KEYS.USER_ID, tokenInfo.userId);
          localStorage.setItem(CONSTANTS.LOCAL_STORAGE_KEYS.USER_NAME, tokenInfo.name);
          localStorage.setItem(CONSTANTS.LOCAL_STORAGE_KEYS.USER_AVATAR, tokenInfo.avatar);
          this.router.navigate(['/home']);
        })
        .catch(() => {
          this.router.navigate(['/not-found']).then(() => this.auth0.logout());
        });
    });
  }
}
```

```
<div class="container">
  <h2 *ngIf="!canClose; else closeWindow" class="authHeader">Authorization in progress...</h2>
</div>

<ng-template #closeWindow>
  <h2 class="authHeader">You can close this window</h2>
</ng-template>
```

```
@import './src/app/shared/variables/variables.scss';

.container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding-top: 100px;
}

.authHeader {
  font-size: $authHeaderSize;
  margin-bottom: 64px;
  font-weight: bold;
  color: $first_main_color;
}
```

Сервіс роботи з Auth0 токеном

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Subject } from 'rxjs';
import {IAccessTokenOur} from "../../../../../shared/interfaces/accessTokenOur";

@Injectable({ providedIn: 'root' })
export class Auth0Service {
  private accessTokenOur$ = new Subject<string>();

  constructor(private http: HttpClient) {}

  sendAuth0AccessToken(auth0Token: string) {
    this.http.post(`http://localhost:3000/user/authorize-via-aut0`,
      {
        token: auth0Token
      }
    ).subscribe((token: IAccessTokenOur) => {
      console.log(token);
      this.accessTokenOur$.next(token.accessToken);
    });
  }

  getAccessTokenOur() {
    return this.accessTokenOur$.asObservable();
  }
}
```

Компонент сторінки налаштувань

```
import {Component, OnInit} from '@angular/core';
import {FormBuilder, FormControl, FormGroup} from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '@auth0/auth0-angular';
import {GetUserInfoService} from "../../services/get-user-info.service";
import {MatRadioChange} from "@angular/material/radio";

export interface UserInfo {
  userInfo: [{
    avatar: string,
    email: string,
    name: string,
    settings: FormInfo
  }]
}

export interface FormInfo {
  blockedPages: string[],
  blockedDate: number | null,
  blockedPassword: string | null,
  redirectUrl: string,
  blockedTime: number | null
}

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})

export class HomeComponent implements OnInit {
  userEmail: string;
  imgSrc: string;
  personalToken: string;
  methodRadioValue = 'time';
  redirectRadioValue = 'blank';
  selectedDate: Date | null;
  minDate: Date;
  redirectUrl = 'about:blank';
  settings: FormInfo = {
    blockedDate: null,
    blockedTime: 0,
    blockedPassword: null,
    blockedPages: [],
    redirectUrl: null
  };

  formGroup: FormGroup;

  constructor(
    private router: Router,
    private auth0: AuthService,
    private getUserInfoService: GetUserInfoService,
```

```
07:     fillForm(userSettings: FormInfo) {
08:       this.formGroup.get('formBlockedPages').setValue(userSettings.blockedPages.join('\n'));
09:       this.formGroup.get('formRedirect').setValue(userSettings.redirectUrl);
10:       this.selectedDate = new Date(userSettings.blockedDate);
11:       this.formGroup.get('formTime').setValue(this.milisecToTime(userSettings.blockedTime));
12:       this.formGroup.get('formPassword').setValue(userSettings.blockedPassword);
13:     }
14:
15:     updateFormDate(value) {
16:       this.formGroup.get('formDate').setValue(Date.parse(value));
17:     }
18:
19:     onSubmit(valueFromForm) {
20:       this.settings.blockedPassword = valueFromForm.formPassword;
21:       this.settings.blockedPages = this.stringToArray(valueFromForm.formBlockedPages);
22:       this.settings.blockedDate = valueFromForm?.formDate;
23:       this.settings.blockedTime = this.timeInMilisec(valueFromForm?.formTime);
24:       this.settings.redirectUrl = valueFromForm.formRedirect;
25:       this.getUserInfoService.setUserInfo(this.settings);
26:     }
27:
28:     resetForm() {
29:       this.formGroup.reset();
30:     }
31:
32:     private stringToArray(s: string | null) {
33:       return s ? s.split('\n') : [];
34:     }
35:
36:     private timeInMilisec(time: string | null) {
37:       if (!time) {
38:         return 0;
39:       }
40:       const hoursInMilisec = Number(time.split(':')[0]) * 3600000;
41:       const minInMilisec = Number(time.split(':')[1]) * 60000;
42:       return hoursInMilisec + minInMilisec;
43:     }
44:
45:     private milisecToTime(time: number) {
46:       if (time === 0){
47:         return;
48:       }
49:
50:       let seconds = Math.floor(time / 1000);
51:       let minutes = Math.floor(seconds / 60);
52:       let hours = Math.floor(minutes / 60);
53:
54:       seconds = seconds % 60;
55:       minutes = seconds >= 30 ? minutes + 1 : minutes;
56:
57:       minutes = minutes % 60;
58:
59:       hours = hours % 24;
```

```
60:
61:       return `${this.padTo2Digits(hours)}:${this.padTo2Digits(minutes)}`;
62:     }
63:
64:     private padTo2Digits(num) {
65:       return num.toString().padStart(2, '0');
66:     }
67:   }
68:
```

```
 1:   .page-wrapper {
 2:     width: 100%;
 3:     height: 100vh;
 4:   }
 5:
 6:   .home-page {
 7:     width: 100%;
 8:     height: 100%;
 9:     padding: 20px 400px;
10:     display: flex;
11:     gap: 15px;
12:     flex-direction: column;
13:     justify-content: flex-start;
14:   }
15:
16:   .header {
17:     width: 100%;
18:     display: flex;
19:     justify-content: flex-end;
20:   }
21:
22:   .header-content {
23:     display: flex;
24:     flex-direction: row;
25:     align-items: center;
26:     gap: 20px;
27:
28:     .header-userData {
29:       display: flex;
30:       flex-direction: row;
31:       align-items: center;
32:       gap: 7px;
33:
34:       img {
35:         width: 33px;
36:         border-radius: 25px;
37:       }
38:     }
39:   }
40:
41:   .form-field-token {
42:     width: 100%;
43:   }
44:
45:   .blocked-sites ::ng-deep.mat-grid-tile-content {
46:     align-items: center;
47:     flex-direction: column;
48:     justify-content: flex-start;
49:
50:     .blocked-sites-title {
51:       font-weight: bold;
52:     }
53:
```

```
54:     .blocked-list {
55:       margin-top: 20px;
56:       width: 100%;
57:
58:       textarea {
59:         resize: none;
60:         &::-webkit-scrollbar {
61:           display: none;
62:         }
63:       }
64:     }
65:   }
66:
67:   .method-tile ::ng-deep.mat-grid-tile-content {
68:     align-items: center;
69:     justify-content: flex-start;
70:     flex-direction: column;
71:
72:     .method-title {
73:       font-weight: bold;
74:     }
75:
76:     mat-radio-group {
77:       margin-top: 10px;
78:     }
79:
80:     mat-radio-button {
81:       margin: 0 20px;
82:     }
83:
84:     .inline-calendar-card {
85:       margin-top: 20px;
86:     }
87:
88:     .password-wrapper {
89:       width: 75%;
90:       margin-top: 20px;
91:     }
92:   }
93:
94:   .redirect-url ::ng-deep.mat-grid-tile-content {
95:     align-items: center;
96:     flex-direction: column;
97:     justify-content: flex-start;
98:
99:     mat-form-field {
00:       width: 100%;
01:     }
02:
03:     mat-radio-group {
04:       width: 95%;
05:       display: flex;
06:       flex-direction: column;
```

```
      justify-content: flex-start;
      align-items: flex-start;
    }
  }

  .btns {
    margin-top: 20px;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    gap: 10px;
  }
```

```
<div class="page-wrapper">
  <div class="home-page">
    <div class="header">
      <div class="header-content">
        <div class="header-userData">
          <span>{{ userEmail }}</span>
          <img [src]="imgSrc" alt="avatar">
        </div>
        <button mat-raised-button (click)="logout()">Log out</button>
      </div>
    </div>
    <div class="accessToken">
      <mat-form-field class="form-field-token" appearance="fill">
        <mat-label>Personal token</mat-label>
        <input disabled matInput type="text" [(ngModel)]="personalToken">
        <button matSuffix mat-icon-button [cdkCopyToClipboard]="personalToken">
          <mat-icon>content_copy</mat-icon>
        </button>
      </mat-form-field>
    </div>
    <div class="settings">
      <form [formGroup]="formGroup" (ngSubmit)="onSubmit(formGroup.value)" class="form">
        <mat-grid-list cols="4" rowHeight="250" gutterSize="10px">
          <mat-grid-tile
            class="blocked-sites"
            [colspan]="2"
          >
            <span class="blocked-sites-title">Enter names of the sites to block (one item per line)</span>
            <mat-form-field class="blocked-list" appearance="fill">
              <mat-label>Blocked URLs</mat-label>
              <textarea
                rows="8"
                matInput
                formControlName="formBlockedPages"
              ></textarea>
            </mat-form-field>
          </mat-grid-tile>
          <mat-grid-tile
            class="method-tile"
            [colspan]="2"
            rowspan="2"
          >
            <span class="method-title">Chose method to unblock pages</span>
            <mat-radio-group (change)="methodChange($event)">
              <mat-radio-button [checked]="true" value="time">By time</mat-radio-button>
              <mat-radio-button value="pass">By password</mat-radio-button>
            </mat-radio-group>
            <mat-card *ngIf="methodRadioValue === 'time'; else password" class="inline-calendar-card">
              <mat-calendar
                [minDate]="minDate"
                [(selected)]="selectedDate"
```

```
                  (selectedChange)="updateFormDate($event)"
                ></mat-calendar>
                <mat-form-field>
                  <input
                    matInput
                    type="time"
                    placeholder="Time"
                    formControlName="formTime"
                  >
                </mat-form-field>
              </mat-card>

            </mat-grid-tile>
            <mat-grid-tile
              class="redirect-url"
              [colspan]="2"
            >
              <mat-form-field>
                <input
                  matInput
                  [disabled]="redirectRadioValue === 'blank' || redirectRadioValue === 'random'"
                  type="text" placeholder="Redirected URL"
                  formControlName="formRedirect"
                  [(ngModel)]="redirectUrl"
                >
              </mat-form-field>
              <mat-radio-group (change)="redirectChange($event)">
                <mat-radio-button [checked]="true" value="blank">Blank page</mat-radio-button>
                <mat-radio-button value="random">Random page</mat-radio-button>
                <mat-radio-button value="custom">Custom page</mat-radio-button>
              </mat-radio-group>
              <div class="btns">
                <button class="clear" (click)="resetForm()" type="button" mat-raised-button>Clear</button>
                <button class="save" color="primary" type="submit" mat-raised-button>Save</button>
              </div>
            </mat-grid-tile>
          </mat-grid-list>

          <ng-template #password>
            <mat-form-field class="password-wrapper">
              <input
                matInput
                type="password"
                placeholder="Password"
                formControlName="formPassword"
              >
            </mat-form-field>
          </ng-template>
        </form>
      </div>
    </div>
  </div>
```

Сервіс для відправлення запитів на сервер

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {FormInfo} from "../components/home/home.component";

interface UnlockPageResponse {
  status: string;
}

@Injectable({
  providedIn: 'root'
})
export class GetUserInfoService {

  constructor(private http: HttpClient) { }

  getUserInfo() {
    const userId = localStorage.getItem('userId');
    const userToken = localStorage.getItem('accessToken');
    return this.http.get(`http://localhost:3000/user/getUserInfo/${userId}`,
      {
        headers: {
          'Authorization': `Bearer ${userToken}`
        }
      }
    );
  }

  setUserInfo(info: FormInfo) {
    const userId = localStorage.getItem('userId');
    const userToken = localStorage.getItem('accessToken');
    return this.http.post(`http://localhost:3000/user/updateUserInfo/${userId}`,
      {...info},
      {
        headers: {
          'Authorization': `Bearer ${userToken}`
        }
      }
    ).subscribe();
  }

  unlockPage(url: string) {
    const userId = localStorage.getItem('userId');
    const userToken = localStorage.getItem('accessToken');
    return
this.http.post<UnlockPageResponse>(`http://localhost:3000/user/removePage/${userId}`,
      {
        removedPage: new URL(url || '').hostname.replace('www.', '')
      },
      {
        headers: {
          'Authorization': `Bearer ${userToken}`
        }
      }
```

```
      ).subscribe(data => {
        if (data.status === 'List has been updated') {
          window.location.href = url;
        }
      });
    }
  }
```

Компонент заблокованої сторінки

```
import { Component, OnInit } from '@angular/core';
import {GetUserInfoService} from "../../services/get-user-info.service";
import {UserInfo} from "../home/home.component";
import {ActivatedRoute} from "@angular/router";

@Component({
  selector: 'app-block',
  templateUrl: './block.component.html',
  styleUrls: ['./block.component.scss']
})
export class BlockComponent implements OnInit {
  isTimeExist: boolean;
  isPasswordExist: boolean;
  timeUnblocked: string;
  password: string;
  definedPassword: string;
  monthNames = ["January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
  ];

  constructor(
    private getUserInfoService: GetUserInfoService,
    private route: ActivatedRoute) { }

  ngOnInit(): void {
    this.getUserInfoService.getUserInfo().subscribe((data: UserInfo) => {
      console.log(data);
      if (data.userInfo[0].settings.blockedTime !== 0 &&
data.userInfo[0].settings.blockedDate) {
        this.isTimeExist = true;
        const time = new Date(data.userInfo[0].settings.blockedTime +
data.userInfo[0].settings.blockedDate);
        this.timeUnblocked = `${time.getDate()} of ${this.monthNames[time.getMonth()]} at
${time.getHours()}:${time.getMinutes()}`
      }
      if (data.userInfo[0].settings.blockedPassword) {
        this.definedPassword = data.userInfo[0].settings.blockedPassword;
        this.isPasswordExist = true;
      }
    });
  }

  unlockPage() {
    this.route.queryParams
      .subscribe(params => {
        if (this.definedPassword === this.password) {
          this.getUserInfoService.unlockPage(params.prev);
        }
        }
      );
  }

}
```

```
.block-wrapper {
  width: 100%;
  height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-size: cover;
  background-blend-mode: darken;
  background-color: rgba(0, 0, 0, .4);
}

.block-content {
  position: absolute;
  font-size: 55px;
  font-weight: bold;
  color: #fff;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  gap: 20px;

  .unblocked-time {
    font-size: 20px;
  }

  .unlock-password {
    width: 50%;
    font-size: 20px;
    padding: 10px;
    background: rgba(0, 0, 0, 0.7);
    outline: none;
    border: none;
    border-radius: 9px;
    color: white;
  }
}
```

```
<div class="block-wrapper" style="background-image: url('./assets/img/1.png');">
  <div class="block-content">
    <span>Oops! This page is blocked :c</span>
    <span *ngIf="isTimeExist" class="unblocked-time">The page will be unblocked on {{ timeUnblocked }}</span>
    <input
      class="unlock-password"
      type="password"
      *ngIf="isPasswordExist"
      placeholder="Enter password to unlock"
      [(ngModel)]="password"
    />
    <button *ngIf="isPasswordExist" (click)="unlockPage()" mat-raised-button>Unlock</button>
  </div>
</div>
```

ДОДАТОК В

Файл маніфесту

```
{
  "name": "My app",
  "version": "0.0.1",
  "description": "Description of my app",
  "manifest_version": 3,
  "action": {
    "default_popup": "index.html"
  },
  "background": {
    "service_worker": "background.js",
    "type": "module"
  },
  "host_permissions": [
    "http://*/*",
    "https://*/*",
    "*://*/*"
  ],
  "permissions": [
    "tabs",
    "unlimitedStorage",
    "activeTab",
    "declarativeContent",
    "debugger",
    "alarms",
    "webNavigation",
    "storage"
  ]
}
```

Компонент віконця при активації розширення

```
<ng-container *ngIf="!isTokenExist; else addSites">
  <div class="wrapper">
    <div class="title">
      <span>Stay focused</span>
    </div>

    <div class="descr">
      <span>Before starting you need <br> to paste your </span><button
(click)="goTosite()">token</button>
    </div>

    <div class="token">
      <input type="text" placeholder="Token" [(ngModel)]="accessToken">
      <button (click)="enterToken()">Enter</button>
    </div>
  </div>
</ng-container>

<ng-template #addSites>
  <div class="add-wrapper">
    <div class="title">
      <span>Stay focused</span>
    </div>

    <div class="descr">
      <span>Total blocked pages: {{ totalBlockedPages }}</span>
    </div>

    <div class="addNew">
      <button (click)="addPage()">Add this page to blacklist</button>
    </div>
  </div>
</ng-template>
```

```
.wrapper {
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
  gap: 20px;

  .title {
    width: 100%;
    text-align: center;
    padding: 20px;
    font-size: 25px;
    color: #fff;
    background-color: blueviolet;
  }

  .descr {
    font-size: 20px;
    color: #000;

    button {
      cursor: pointer;
      background: transparent;
      font-size: 18px;
      outline: none;
      border: 0;
      border-bottom: 1px solid #000000;
    }
  }
}

.token {
  margin-bottom: 27px;
  width: 100%;
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;
  gap: 10px;

  input {
    width: 80%;
    padding: 10px;
    border-radius: 5px;
    outline: none;
    border: none;
    background-color: #DEDEDE;
  }

  button {
    cursor: pointer;
```

```
54:       padding: 8px 40px;
55:       background-color: #C9AEE2;
56:       font-weight: bold;
57:       outline: none;
58:       border: 0;
59:       border-radius: 5px;
60:     }
61:   }
62:
63:   // Add sites
64:   .add-wrapper {
65:     width: 100%;
66:     height: 100%;
67:     display: flex;
68:     flex-direction: column;
69:     justify-content: space-between;
70:     align-items: center;
71:     gap: 20px;
72:
73:     .title {
74:       width: 100%;
75:       text-align: center;
76:       padding: 20px;
77:       font-size: 25px;
78:       color: #fff;
79:       background-color: blueviolet;
80:     }
81:
82:     .descr {
83:       font-size: 20px;
84:       color: #000;
85:       font-weight: bold;
86:     }
87:
88:     .addNew {
89:       margin-bottom: 10px;
90:       button {
91:         cursor: pointer;
92:         padding: 15px 40px;
93:         background-color: #C9AEE2;
94:         font-weight: bold;
95:         outline: none;
96:         border: 0;
97:         border-radius: 5px;
98:         font-size: 18px;
99:       }
00:     }
01:   }
```

```
import {CookieService} from "ngx-cookie-service";
import * as jwt_decode from "jwt-decode";
import {BackendQueryService} from "../../services/backend-query.service";

interface IToken {
  avatar: string,
  email: string,
  exp: number,
  iat: number,
  name: string,
  userId: string,
}

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})

export class HomeComponent implements OnInit {
  title = 'ext';
  accessToken = '';
  cookieValue: string | undefined = '';
  currentUrl: string | undefined = '';

  isTokenExist = false;
  totalBlockedPages = 0;

  constructor(
    private cookieService: CookieService,
    private backendQueryService: BackendQueryService,
    private cdr: ChangeDetectorRef) {
  }

  addPage() {
    chrome.tabs.query({active: true, currentWindow: true}, (tabs) => {
      const activeTab = tabs[0];
      const activeTabUrl = activeTab.url;
      const activeTabId = activeTab.id;
      const blockedPage = new URL(activeTabUrl || '').hostname.replace('www.', '');
      this.backendQueryService.addPage(blockedPage);
      chrome.runtime.sendMessage({ blockCurrent: true });
      chrome.tabs.reload(activeTabId || 0);
    });
  }

  ngOnInit(): void {
    this.backendQueryService.listUpdated$.subscribe(isUpdated => {
      if (isUpdated) {
        this.totalBlockedPages += 1;
        this.cdr.detectChanges();
      }
    });
```

```
import {CookieService} from "ngx-cookie-service";
import * as jwt_decode from "jwt-decode";
import {BackendQueryService} from "../../services/backend-query.service";

interface IToken {
  avatar: string,
  email: string,
  exp: number,
  iat: number,
  name: string,
  userId: string,
}

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})

export class HomeComponent implements OnInit {
  title = 'ext';
  accessToken = '';
  cookieValue: string | undefined = '';
  currentUrl: string | undefined = '';

  isTokenExist = false;
  totalBlockedPages = 0;

  constructor(
    private cookieService: CookieService,
    private backendQueryService: BackendQueryService,
    private cdr: ChangeDetectorRef) {
  }

  addPage() {
    chrome.tabs.query({active: true, currentWindow: true}, (tabs) => {
      const activeTab = tabs[0];
      const activeTabUrl = activeTab.url;
      const activeTabId = activeTab.id;
      const blockedPage = new URL(activeTabUrl || '').hostname.replace('www.', '');
      this.backendQueryService.addPage(blockedPage);
      chrome.runtime.sendMessage({ blockCurrent: true });
      chrome.tabs.reload(activeTabId || 0);
    });
  }

  ngOnInit(): void {
    this.backendQueryService.listUpdated$.subscribe(isUpdated => {
      if (isUpdated) {
        this.totalBlockedPages += 1;
        this.cdr.detectChanges();
      }
    });
```

Сервіс для відправлення запитів на сервер

```
import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {BehaviorSubject} from "rxjs";
interface UserInfo {
  userInfo: [{
    avatar: string,
    email: string,
    name: string,
    settings: FormInfo
  }]
}
interface FormInfo {
  blockedPages: string[],
  blockedDate: number | null,
  blockedPassword: string | null,
  redirectUrl: string,
  blockedTime: number | null
}
interface StatusOfAdding {
  status: string
}
@Injectable({
  providedIn: 'root'
})
export class BackendQueryService {
  listUpdated$ = new BehaviorSubject(false);
  constructor(private http: HttpClient) { }
  getUserInfo(userId: string) {
    return this.http.get<UserInfo>(`http://localhost:3000/user/getUserInfo/${userId}`, {
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('accessToken')}`
      }
    });
  }

  addPage(pageUrl: string) {
    const userId = localStorage.getItem('userId');
    return this.http.post<StatusOfAdding>(`http://localhost:3000/user/addPage/${userId}`,
{
      blockedPage: pageUrl
    },{
      headers: {
        'Authorization': `Bearer ${localStorage.getItem('accessToken')}`
      }
    }).subscribe(data => {
      if (data.status === 'List has been updated') {
        this.listUpdated$.next(true);
      }
    });
  }
}
```

Скрипт, який виконується у фоновому режимі

```
const LS = chrome.storage.local;

chrome.runtime.onInstalled.addListener(() => {
  console.log('onInstalled...');
});

chrome.tabs.onUpdated.addListener(async (tabId, changeInfo, tab) => {
try{
  if (changeInfo.status === 'loading' && changeInfo.url) {
    const userId = await LS.get('userId');
    const accessToken = await LS.get('accessToken');

    getBlockedList(userId.userId, accessToken.accessToken, tabId, changeInfo.url);
  }
} catch (e) {
  console.log(e);
}
});

chrome.runtime.onMessage.addListener(async (request, sender, reply) => {
  if (request.token) {
    await LS.set({ accessToken: request.token });
  }
  if (request.userId) {
    await LS.set({ userId: request.userId });
  }
  if (request.blockCurrent) {
    chrome.tabs.query({active: true, currentWindow: true}, async (tabs) => {
      const activeTab = tabs[0];
      const activeTabUrl = activeTab.url;
      const activeTabId = activeTab.id;
      const userId = await LS.get('userId');
      const accessToken = await LS.get('accessToken');
      getBlockedList(userId.userId, accessToken.accessToken, activeTabId, activeTabUrl);
    });
  }
  return true;
});

function getBlockedList(userId, accessToken, tabId, url) {
  try {
    // make request to backend
    fetch(`http://localhost:3000/user/getUserInfo/${userId}`, {
      method: "GET",
      headers: {
        "Authorization": `Bearer ${accessToken}`,
      }})
      .then((response) => response.json())
      .then((data) => {
        const currentUrl = new URL(url).hostname.replace('www.', '');
        if (data.userInfo[0].settings.blockedPages.includes(currentUrl)) {
          let urlToRedirect = '';
```

```
53:             if (data.userInfo[0].settings.redirectUrl.includes('localhost')) {
54:               urlToRedirect = `${data.userInfo[0].settings.redirectUrl}?prev=${url}`
55:             } else {
56:               urlToRedirect = data.userInfo[0].settings.redirectUrl;
57:             }
58:             chrome.tabs.update(tabId, { url: urlToRedirect });
59:           }
60:         });
61:     } catch (e) {
62:       console.log(e);
63:     }
64:   }
```