Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ВИКОРИСТАННЯ НЕЙРО-КОМП'ЮТЕРНИХ ІНТЕРФЕЙСІВ ДЛЯ ЗБОРУ ТА НАКОПИЧЕННЯ ІНФОРМАЦІЇ ПРО КОРИСТУВАЧІВ ВЕБ РУСУРСІВ

Текстова частина

магістерської роботи

за спеціальністю „Інженерія програмного забезпечення" 121

Керівник магістерської
роботи
д.т.н., доц. А. М. Глибовець

_____
(підпис)

"____" _____ 2023 р.

Виконала студентка
С.М.Хмель
"____" _____ 2023 р.

Київ 2023
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики, к.ф.-м.н.
_____С. С. Гороховський
(підпис)
„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу

студенту 2 р.н. магістерської програми Комп’ютерні науки
Хмель Світлані Миколаївні
Розробити Альтернативній метод використання нейро-комп’ютерних інтерфейсів для збору та накопичення інформації про користувачів веб ресурсів

Зміст текстової частини до магістерської роботи:

Зміст
Анотація
Вступ
1 Аналіз існуючих нейро-комп'ютерних інтерфейсів на предмет виявлення основних показників, що можуть бути використання для аналізу інформації про відвідувані веб ресурси.

2 Розробити архітектуру системи та перевірити можливість інтеграції запропонованої архітектури на основі побудованого прототипу. Оцінити основні переваги та недоліки.

Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2023 р.

Керівник
А.М. Глибовець, доктор технічних наук, доцент

_____
(підпис)

Завдання отримала
С.М. Хмель _____
(підпис)

Тема: Розробити архітектуру системи збору та накопичення інформації про користувачів веб ресурсів з використанням нейро-комп'ютерних інтерфейсів.

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	01.11.2022	
2.	Огляд технічної літератури за темою роботи	15.11.2022	
3.	Виконання аналізу сучасних рішень	29.11.2022	
3.	Розробка архітектури системи збору та накопичення інформації	27.12.2022	
4.	Реалізація програмного застосунку на базі розробленої архітектури	17.01.2023	
5.	Інтеграція програмного застосунку з зовнішніми системами	14.02.2023	
6.	Написання пояснювальної записки	24.04.2023	
7.	Створення слайдів для доповіді та написання доповіді	27.04.2023	
8.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи	30.04.2023	
9.	Корегування роботи за результатами попереднього захисту	5.05.2023	
10.	Остаточне оформлення пояснювальної записки та слайдів	10.05.2023	

11.	Захист магістерської роботи (проекту)	18.06.2023	

Студент _____

Керівник _____

“____”_____

ЗМІСТ

Table of Contents

ВСТУП *6*

РОЗДІЛ 1: Загальні поняття та огляд BCI приладів *7*

1.1. Загальні поняття BCI **7**

1.2. Пристрої для отримання сигналу ЕЕГ **11**

РОЗДІЛ 2 Розробка архітектури системи взаємодії з браузером та збору інформації користувача *14*

2.1. Загальну архітектурне рішення системи **14**

2.2. Основні концепції розробки розширень Chrome **19**

2.3. Архітектура Cortex API **22**

2.3.1 Огляд робочого процесу API [17] 22

2.3.2 Підписка на дані 24

2.4. Розробка API з використанням minimal .NET API для зберігання та обміну даними .. **27**

2.5. Публікація та взаємодія з Azure **30**

2.6. Архітектура бази даних **31**

РОЗДІЛ 3 Реалізація прототипу збору та накопичення інформації *32*

3.1. Архітектура хром додатку **32**

3.2. Взаємодія з браузером **35**

3.3. Взаємодією з гарнітурою та збір даних користувача **36**

Висновки по роботі та рекомендації для подальших досліджень *39*

Список літератури *42*

Додаток А. Програмний код взаємодії з CortexAPI *44*

Додаток В. Програмний код ваємодії браурером *55*

ВСТУП

Актуальність. Використання інтернету та веб-ресурсів у наші дні набуває все більшої популярності, а отже, розуміння потреб користувачів, збільшення їх зацікавленості та покращення досвіду користувача та безпеки користування веб ресурсами є важливою задачею для розробників та бізнесу. Відстеження та аналіз цієї інформації може бути важливим інструментом для покращення якості та ефективності веб-ресурсів.

Одним з потенційних методів для збору та аналізу даних є використання нейрокомп'ютерних інтерфейсів, які здатні перетворювати сигнали мозку у корисну інформацію. Використання цих інтерфейсів може допомогти збирати дані про користувачів веб-ресурсів, такі як зацікавленість, ставлення до певних продуктів та послуг, а також емоційний стан.

Мета дослідження. Метою дослідження є аналіз можливостей використання нейрокомп'ютерних інтерфейсів для збору та накопичення інформації про користувачів веб-ресурсів.

Завдання дослідження.

- Проаналізувати стан сучасних нейрокомп'ютерних інтерфейсів та їх застосування;
- Визначити можливості застосування нейрокомп'ютерних інтерфейсів для збору та накопичення інформації про користувачів веб-ресурсів;
- Розробити та реалізувати методику збору даних за допомогою нейрокомп'ютерних інтерфейсів;
- Проаналізувати отримані дані та оцінити можливість їх використання для покращення якості та ефективності веб-ресурсів.

Об´єкт дослідження. Об’єктом дослідження є нейрокомп’ютерні інтерфейси та їх застосування для збору та накопичення інформації про користувачів веб-ресурсів.

Джерела дослідження. Документація SDK виробника приладу, веб ресурси, форуми, відео інструкції, література та статті в електронній версії.

Наукова новизна одержаних результатів дослідження полягає у демонстрації можливості аналізу та використання ментального стану користувача для покращення його взаємодії з веб ресурсами.

Практичне значення одержаних результатів. За рахунок використання розробленого прототипу розширюються можливості взаємодії користувача з веб ресурсами, додається можливість керування роботою браузера за допомогою ментальних команд та імпульсів, що передає BCI прилад. Інформація, що збирається на протязі користування веб ресурсами при наявності BCI приладу, дозволить краще зрозуміти поведінку та потреби користувача, та в подальшому покращити взаємодію з веб ресурсом та безпеку користування.

РОЗДІЛ 1: Загальні поняття та огляд BCI приладів

1.1. Загальні поняття BCI

BCI (brain–computer interface) це зв’язок між електричною активністю мозку з комп’ютером або іншими зовнішніми пристроями. Поняття BCI обмежується лише взаємодією з сигналом, що виробляє мозок, то ж сигнали, які виробляють нерви, м’язи або голос, не розглядаються.

Сигнали, отримані від мозку, можуть мати електрофізіологічний, магнітний або метаболічний характер. Перетворення цих сигналів на команди дає можливість керувати приладами та взаємодіяти з іншими програмними інтерфейсами. [2]

У BCI існують різні типи парадигм. Ці парадигми включають потенціали, пов’язані з подіями (ERP), повільні потенціали кори (SCP), сенсомоторні ритми, рухові образи, коливальну ЕЕГ-активність і зоровий викликаний потенціал (VEP).

BCI зазвичай складається з отримання, обробки сигналу, маніпулювання даними та зворотного зв’язку. [1]

Рисунок 1.1 – Загальна архітектура інтерфейсу мозок-комп’ютер (brain-computer). [1].

Інвазивний BCI.

Це отримання сигналів мозку за допомогою датчиків, що імплантовані безпосередньо в сіру речовину мозку шляхом нейрохірургії. Інвазивні BCI мають найбільш якісний сигнал, точніший та містить менше

шумів. Однак цей метод більш ризикований, тому дослідження інвазивного BCI проводилися тільки з тваринами. [2]

Частково інвазивний BCI.

При частково інвазивному BCI електроди імплантуються всередину черепа, але за межами мозку. Сила сигналу нижча але ризик для пацієнта теж менший.[2]

Неінвазивний BCI.

Неінвазивний BCI не вимагає внутрішньочерепної хірургії та імплантації пристрою в мозок. Це найбезпечніший спосіб BCI, оскільки вони не немає ризику інфекції чи кровотечі. Сила сигналу, при неінвазивному BCI нижча, ніж при інвазивному та частково інвазивному. Основним недоліком неінвазивного BCI є те, що сигнали ослаблюються в процесі проходження через череп, тверду мозкову оболонку та шкіру голови. Неінвазивний BCI використовує функціональну магнітно-резонансну томографію (МРТ), позитронно-електронну томографію (ПЕТ), функціональну спектроскопію ближнього інфрачервоного діапазону (fNIRS), магнітоенцефалографію (МЕГ), електроенцефалографію (ЕЕГ) та однофотонну емісійну комп’ютерну томографію (SPECT) для захоплення сигнали мозку.

Ми будемо розглядати ЕЕГ прилади, оскільки вони вважаються найбільш перспективними. Сигналів ЕЕГ легко фіксувати та аналізувати.

Прилади на основі ЕЕГ відносно дешевші, та їх використання не потребує хірургічного втручання, що є важливим фактором.

ЕЕГ реєструє коливання напруги внаслідок потоку іонного струму під час синаптичних збуджень у нейронах мозку. У цьому способі електроди прикріплюються до шкіри голови для отримання сигналів мозку. Номер електрода варіюється від 1 до 256 для різних ЕЕГ-гарнітур. Виміряний сигнал ЕЕГ — це різниця напруг між активним електродом і електродом порівняння протягом часу з амплітудою в мікровольтах (мкВ). Зазвичай амплітуда ЕЕГ коливається від -100 до +100 мікровольт. Сигнали ЕЕГ можна класифікувати відповідно до діапазонів частот, і кожен із цих діапазонів має певне біологічне значення.

	Частота (Hz)	*Амплітуда (mV)*	*Розміщення*	*Активність*
Delta	0.5-4 Hz	100-200	Фронтальний голови	Глибокий сон
Theta	4-8 Hz	5-10	Різні області	Сонливість, легкий сон Стан: Творчість, проникливість, мрії, знижена свідомість
Alpha	8-13 Hz	20-80	Задня область голови	Розслаблений Стан: фізично і психічно розслаблений
Beta	13-30 Hz	1-5	Ліва та права частина, симетричний розподіл, найбільш виражений фронтально	Активне мислення, пильність Стан: настороженість, нормальна активна свідомість, активне мислення <u>Наприклад:</u> Активна розмова Прийняття рішень Розв'язання проблеми Зосередження на завданні Вивчення нової концепції

				Бета-хвилі мозку найлегше виявити, коли ми зайняті активним мисленням.
Gamma	32 – 100 Hz	0.5-2	Соматосенсорна кора	Гіперактивність Стан: Підвищене сприйняття, навчання, вирішення проблемних завдань

Таблиця 1.1 – Смуги частот ЕЕГ з властивостями [1].

Структура отримання ЕЕГ для застосування BCI.

Головний мозок людини складається з двох основних частин: кори головного мозку і підкіркових відділів. Кора головного мозку, регулює сенсорну та моторну обробку, а також функції вищого рівня, наприклад, обробку мови, розпізнавання образів, міркування. Головний мозок розділений на дві півкулі, у яких кожна півкуля поділяється на чотири частки: тім’яна, потилична, лобова та скронева.

Тім'яна	орфографію, об'єкти, маніпуляції, сприйняття та просторове усвідомлення
Скронева	розпізнавання обличь, генерування емоцій
Лобова	навички, гнучке мислення, вирішення проблем, свідомі рухи, увагу, емоційний і поведінковий контроль
Потилична	інтерпретація зорових стимулів

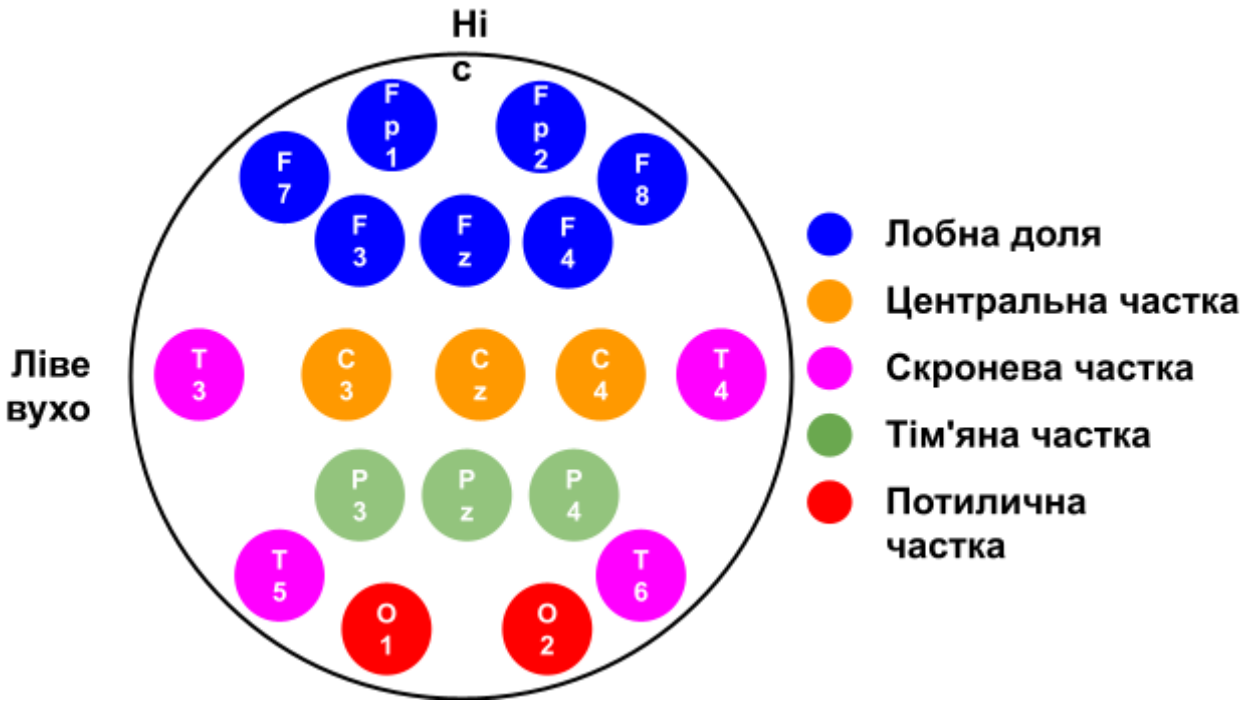


Рисунок 1.2 – Стандартизована схема розміщення електродів [1].

1.2. Пристрої для отримання сигналу ЕЕГ.

Існує два основних способи отримання сигналів ЕЕГ: бездротовий або дротовий. Вимірювання сигналу ЕЕГ виконуються за допомогою кількості електродів, що варіюються від 1 до приблизно 256. Контакт між електродами та шкірою зазвичай покращується за допомогою електропровідного гелю, що робить процедуру встановлення електродів загалом виснажливою та тривалою операцією. Тому було запропоновано та перевірено використання сухих електродів, які не потребують провідних гелів або паст. Але ефективність використання сухих електродів з точки зору швидкості інформації в середньому на 30% нижча, ніж отримання з BCI на основі вологих електродів.

Проводні BCI обмежують рух користувача та є більш складними, їх встановлення займає більше часу.

Нижче наведено найбільш відомі пристрої ЕЕГ, що є доступними на ринку:

Виробник	**Прилади/ Кількість каналів**	**Частота дискретизації**	**Тип зв'язку**
NeiroScan	SynAmps:64 Grael:32 NuAmps:40 Siesta:32	SynAmps: 20 kHz Grael: 4,096 Hz NuAmps: 1,000 Hz Siesta: 1,024 Hz	Wired
Brain Products	LiveAmp: 8/16/32	Між 250, 500 та 1000 Hz	Wireless
Bio Semi	16,32,64	2/4/8/16 Hz	Wired
Emotiv	**INSIGHT**:5 EPOC+:14 EPOC FLEX:32	128 Hz	Wireless
NeiroSky	1	512 Hz	Wireless
Advanced brain monitoring	ABM B-alert X-24:24	256 Hz	Wireless
g.tech.nautilus	64	500 Hz	Wireless
AntNeuro eego	64	2,048 Hz	Wireless
Neiroelectrics Enobio 32	32	500 Hz	Wireless
Muse	4	256 Hz	Wireless
Open BCI	Up to 16	256 Hz	Wireless
Cognionics Mobile	72	500 - 1000 Hz	Wireless
mBrainTrain	24	250 - 500 Hz	Wireless
MyndBand EEG headset	3	512 Hz	Wireless
Enobio	8,20 або 32	500 Hz	Wireless

Таблиця 1.2 – Зведена таблиця останніх пристроїв ЕЕГ [1].

При виконанні даної роботи використовувався бездротовий **Emotiv INSIGHT** на 5 каналів. Цей прилад використовує напівсухі полімерні датчики, та має датчики руху, що дозволяє виявляти рухи голови [8].

Рисунок 1.3 – Зовнішній вигляд приладу **Emotiv INSIGHT** [8].

Рисунок 1.4 – Напівсухі полімерні датчики **Emotiv INSIGHT** [8].

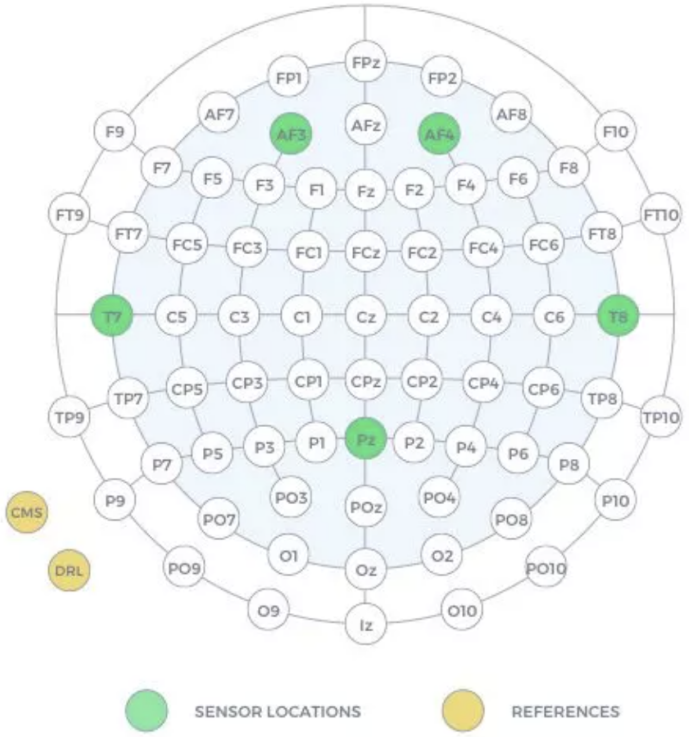


Рисунок 1.5 – Розташування датчиків приладу **Emotiv INSIGHT** [5].

РОЗДІЛ 2 Розробка архітектури системи взаємодії з браузером та збору інформації користувача

2.1. Загальну архітектурне рішення системи

Дана архітектура включає кілька компонентів, які працюють разом, щоб забезпечити збір, обробку і відображення даних, зібраних з EEG-пристрою:

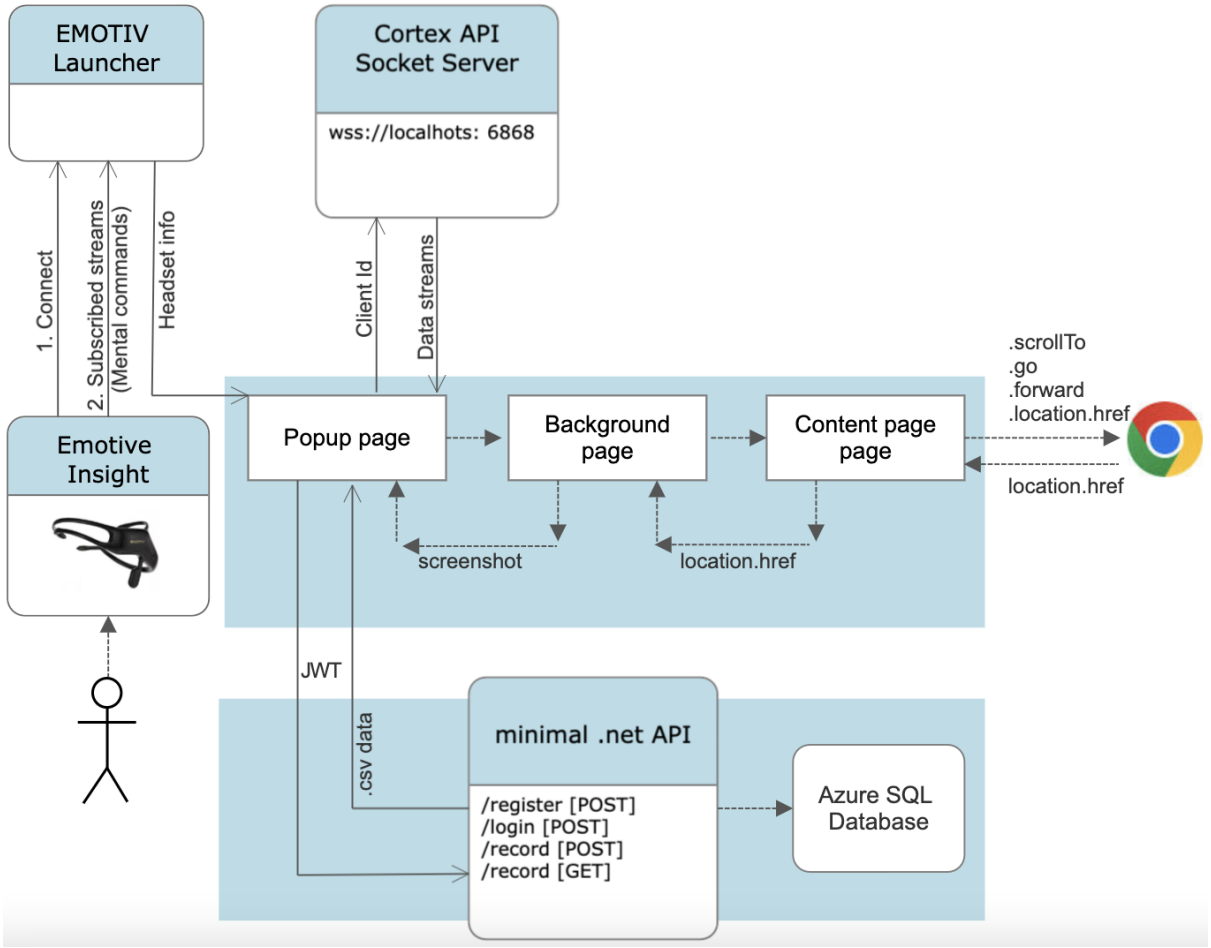


Рисунок 2.1 – Загальне архітектурне рішення.

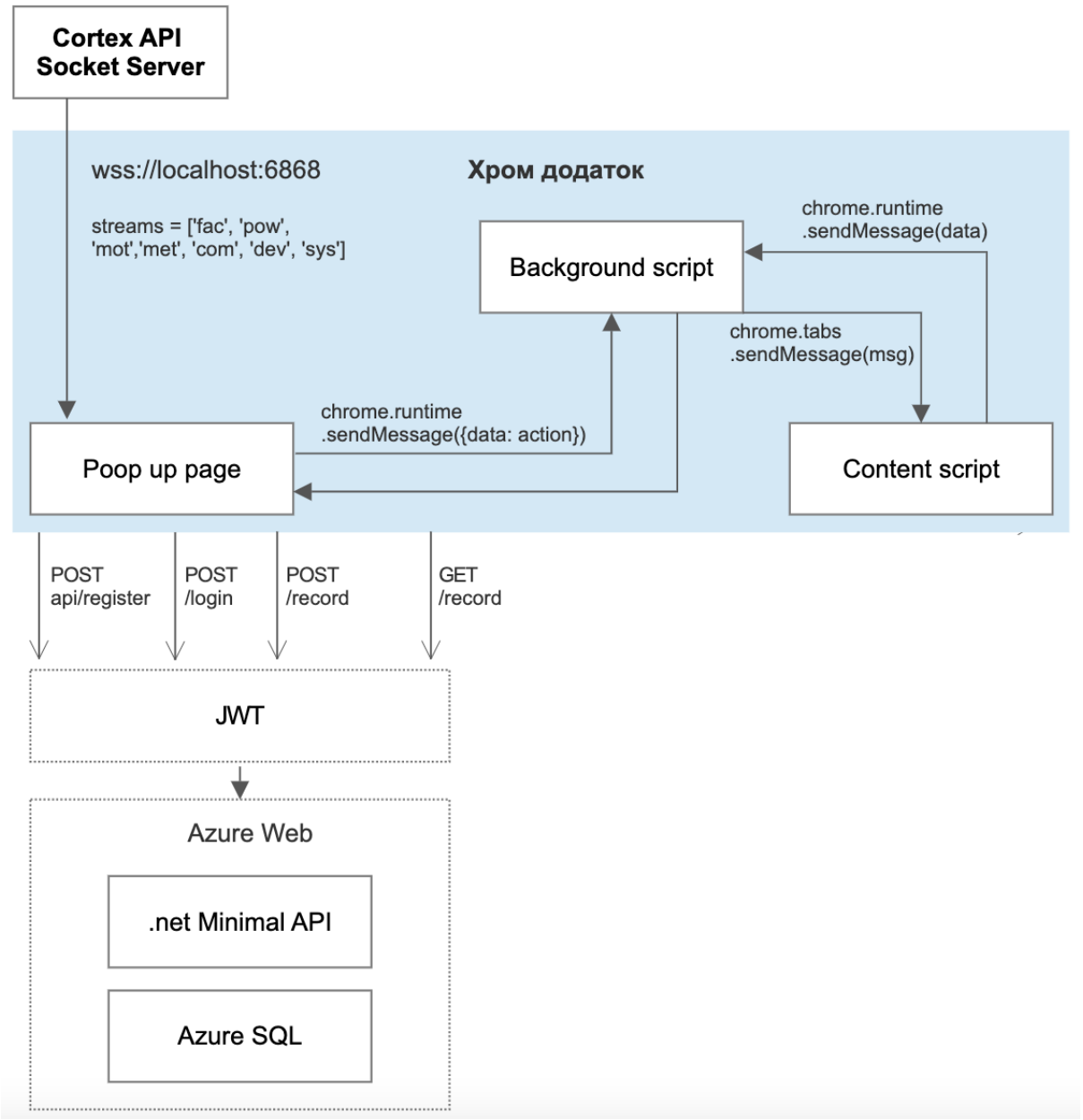


Рисунок 2.2 – Взаємодія між компонентами.

1) **Хром-розширення:**

Розширення збирає дані з EEG пристрою та надсилає їх на сервер. Для написання розширення використовується JavaScript, для взаємодії з приладом використовується **Cortex API**, що надала компанія - виробник приладу Emotiv, а також для відправки даних на сервер використовується REST.

Важливо, щоб розширення було легким та не уповільнювало роботу браузера, тому для прототипу не використовувались фреймворки, хоча в майбутньому можливо було б використовувати React для пришвидшення рендеренгу.

Для стилізації використовується Bootstrap, але вибір бібліотеки не є принциповою.

Для графічної візуалізації показників використовується Apexcharts але це також не є принциповим рішенням.

Для зборки розширення використовується Webpack.

Розширення побудовано за принципами **Crome Manifest v3** [11].

2) **Сервер:** Сервер приймає дані від розширення та зберігає їх у базі даних. Для обробки даних можна використовувати спеціалізовані бібліотеки для роботи з EEG-даними, а також для аналізу їх результатів. Для забезпечення безпеки та конфіденційності даних, сервер має відповідні заходи захисту, такі як шифрування та автентифікація.

3) **База даних:** Оскільки було важливо переконатися, що база даних може обробляти великі обсяги даних та забезпечувати швидкий доступ до них, було обрано Azure SQL.

База даних Azure SQL — це хмарна служба реляційної бази даних, яка надається корпорацією Майкрософт як частина платформи хмарних обчислень Azure. Це повністю керована служба бази даних, яка забезпечує високу доступність, автоматичне резервне копіювання та масштабоване сховище без необхідності керування обладнанням чи інфраструктурою [18].

База даних SQL Azure побудована на тому самому механізмі бази даних SQL Server і підтримує ті самі мови програмування та інструменти,

що й SQL Server, зокрема Transact-SQL (T-SQL), .NET і Java. Це спрощує міграцію наявних баз даних SQL Server до бази даних SQL Azure.

Деякі з ключових функцій бази даних Azure SQL включають:
Висока доступність і аварійне відновлення: База даних SQL Azure забезпечує автоматичну реплікацію та відновлення після відмови, щоб забезпечити високу доступність і аварійне відновлення у разі регіонального збою.

Масштабованість. База даних SQL Azure надає гнучкі параметри масштабування для задоволення мінливих вимог робочого навантаження, включаючи автоматичне та ручне масштабування.

Безпека. База даних SQL Azure забезпечує вбудовані функції безпеки, такі як захист брандмауером, шифрування під час передачі та виявлення загроз.

Також:

- Інтеграція зі службами Azure: база даних Azure SQL інтегрується з іншими службами Azure, такими як Azure Data Factory, Azure Data Lake Storage та Power BI, щоб забезпечити повне рішення аналітики даних

- Економічно ефективне ціноутворення: база даних Azure SQL надає гнучкі варіанти ціноутворення, включаючи оплату за використання та зарезервовану ємність, щоб оптимізувати витрати для різних робочих навантажень.

Загалом база даних Azure SQL надає високо доступне, безпечне та масштабоване хмарне рішення реляційної бази даних, яке дозволяє організаціям зосередитися на своїх основних бізнес-операціях, не турбуючись про керування інфраструктурою.

В цілому, дана архітектура включає кілька компонентів, які працюють разом, щоб забезпечити збір, обробку і відображення даних, зібраних з EEG-пристрою. Ця архітектура може бути додатково оптимізована для забезпечення швидкої обробки та зберігання даних, а також для забезпечення безпеки та конфіденційності даних користувача.

2.2. Основні концепції розробки розширень Chrome

Хром-розширення складається з декількох частин. Нижче приведений приклад файлової структури розширення [11]:

```
└── extension-sample/
    ├── manifest.json
    ├── service-worker.js
    ├── scripts/
    │   └── content-script.js
    ├── popup/
    │   ├── popup.css
    │   ├── popup.js
    │   └── popup.html
    ├── options/
    │   ├── options.css
    │   ├── options.js
    │   └── options.html
    └── icons/
        ├── 16.png
        ├── 32.png
        ├── 48.png
        └── 128.png
```

Рисунок 2.3 – Файлова структура хром-розширення

Компоненти хром-розширення:

Маніфест (manifest.json) — це файл конфігурації розширення Chrome. Це обов'язковий файл JSON, який має бути розташований у корені проекту. Він надає браузеру схему розширення з важливою інформацією, такою як:

Назва розширення, опис того, що воно робить, номер поточної версії та які значки використовувати.
Ключі API Chrome і дозволи, які потрібні розширенню.
Файли, призначені як робочі служби розширення, спливаючі файли HTML, сторінка параметрів, сценарії вмісту тощо.

Служба розширень (service-worker.js) — це сценарій на основі подій, який браузер запускає у фоновому режимі. Він часто використовується для обробки даних, координації завдань у різних частинах розширення та як менеджер подій розширення. Наприклад, сервіс-воркер може прослуховувати та реагувати на події, коли розширення встановлюється вперше, створюється нова вкладка, додається нова закладка, натискається піктограма панелі інструментів розширення тощо.

Service Worker може отримати доступ до всіх Extension API, але як тип Worker він не може використовувати DOM API, які надає глобальний об'єкт Window документа. Він також працює у власному середовищі, тому не може безпосередньо змінювати вміст веб-сторінки.

Розширення використовують **сценарії вмісту (content-script.js)** для введення коду на сторінки хосту. Вони дозволяють розширенню взаємодіяти зі сторінками в браузері та змінювати їх. Наприклад, вони

можуть вставити новий елемент на сторінку, змінити стиль веб-сайту, змінити елементи DOM тощо.

HTML-сторінки розширення

Дві найпоширеніші сторінки HTML:

Спливаюче вікно (popup). Багато розширень використовують спливаюче вікно (popup.html), щоб забезпечити такі функції, як відображення списку вкладок або додаткової інформації щодо поточної вкладки. Його можна знайти, клацнувши піктограму панелі інструментів розширення.

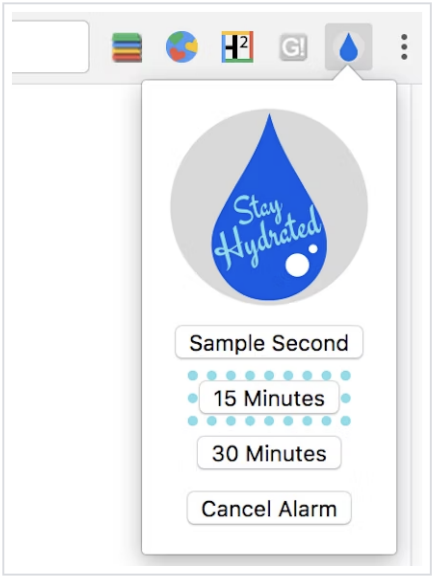


Рисунок 2.4 –Спливаюче вікно (popup) [11] .

Сторінка параметрів (options.html) надає користувачам можливість налаштувати розширення, наприклад вибрати, на яких сайтах воно працюватиме.

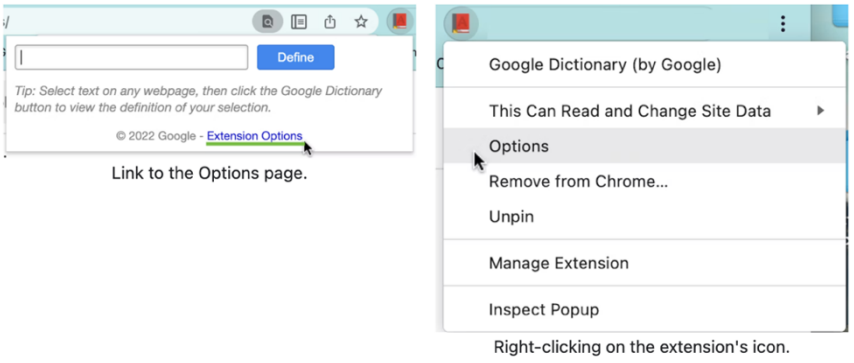


Рисунок 2.5 – Сторінка параметрів [11].

Обмін даними. Сценарії вмісту (content scripts) або інші сторінки розширення можуть надсилати або отримувати інформацію від service worker. У цих випадках будь-яка сторона може прослуховувати повідомлення, надіслані з іншого боку, і відповідати на тому самому каналі. Розширення можуть надсилати одноразовий запит або встановлювати довгострокове з'єднання для підтримки кількох повідомлень.

Зберігання даних. Chrome надає розширення зі спеціалізованим Storage API, доступним для всіх компонентів розширення. Він включає чотири окремі області зберігання для конкретних випадків використання та прослуховувач подій, який відстежує оновлення даних. Наприклад, коли ви зберігаєте зміни у спливаючому вікні, працівник служби розширення може відповісти за вказаною логікою.

2.3. Архітектура Cortex API

2.3.1 Огляд робочого процесу API [17]

1) Потрібно увійти в програму запуску EMOTIV за допомогою свого Emotive ID.

2) Потрібно попередньо згенерувати ***application ID***, ***client ID*** у Account dashboard на веб-сайті Emotiv.
3) Потрібно викликати ***requestAccess*** API, та прийняти його через EMOTIV Launcher.
4) Після отримання доступу підключіть гарнітуру EMOTIV через USB або Bluetooth.
5) Викличте ***queryHeadsets*** API, щоб отримати список доступних гарнітур.
6) Викличте ***controleDevice*** API для підключення до бажаної гарнітури.
7) Викличте ***authorize*** API, щоб отримати маркер Cortex для запитів підпослідовності.
8) Викличте ***createSession*** API, щоб відкрити новий сеанс і бути готовим до потокового передавання даних BCI.

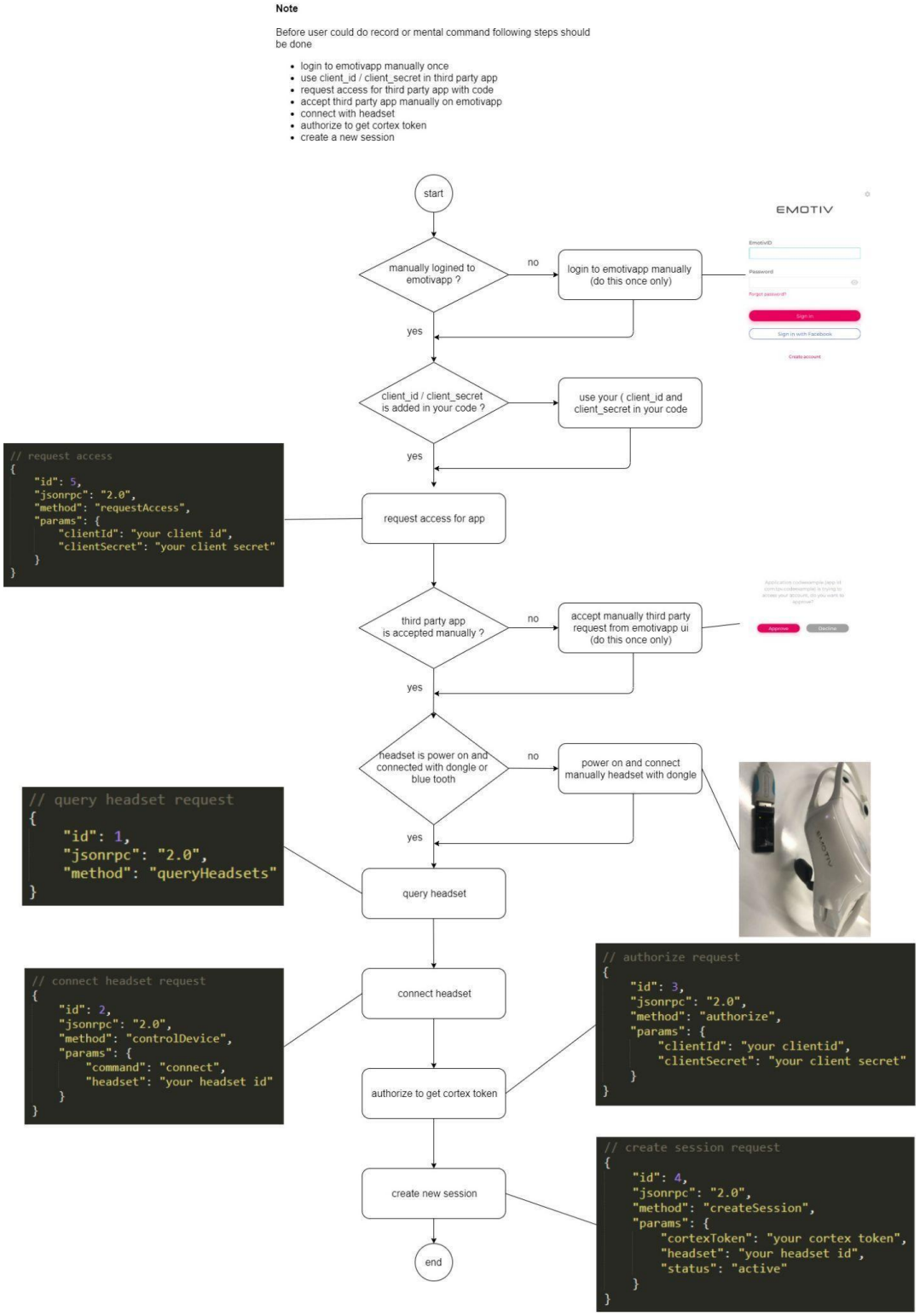


Рисунок 2.6 – Структура робочого процесу Cortex API [16]

2.3.2 Підписка на дані

Відкривши ***session*** за допомогою гарнітури, ***subscribe*** до одного або кількох потоків даних.

Кожна пара даних дає вам доступ у реальному часі до даних із гарнітури (ЕЕГ, рух...) або даних, обчислених Cortex (потужності діапазону, розумові команди...)

Після успішної підписки на потік даних Cortex продовжить надсилати вам ***data sample object***.

Підписка пов’язана з ***session***. Усі підписки на сеанс автоматично скасовуються, коли сеанс закривається.

Доступні потоки даних залежать від користувача, моделі та гарнітури.

У данній роботі встановлена підписка на такі дані: "mot", "dev", "pow", "met", "com", "fac", "sys".[17]

mot	Дані про рух голови. **Сенсори:** Accelerometer: 3-axis +/-8g Gyroscope: 3-axis +/-2000 dps (converted to 4 normalised quaternions) Magnetometer: 3-axis +/- 12 gauss *Приклад даних:* {'streamName': 'mot', 'labels': ['COUNTER_MEMS', 'INTERPOLATED_MEMS', 'Q0', 'Q1', 'Q2', 'Q3', 'ACCX', 'ACCY', 'ACCZ', 'MAGX', 'MAGY', 'MAGZ']} {'mot': [33, 0, 0.493859, 0.40625, 0.46875, -0.609375, 0.968765, 0.187503, -0.250004, -76.563667, -19.584995, 38.281834], 'time': 1627457508.2588}

dev	{'signal': 1.0, 'dev': [4, 4, 4, 4, 4, 100], 'batteryPercent': 80, 'time': 1627459265.4463}
met	Показники продуктивності: - Engagement (ENG) - Excitement (EXC) - Stress (FRU) - Relaxation (MED) - Interest (VAL) - Focus (FOC) (показуються на 0.1Hz) Формат: {'streamName': 'met', 'labels': ['eng.isActive', 'eng', 'exc.isActive', 'exc', 'lex', 'str.isActive', 'str', 'rel.isActive', 'rel', 'int.isActive', 'int', 'foc.isActive', 'foc']} *Приклад даних:* {"met": [true, 0.26539, true, 0.288098, 0.0, true, 0.305797, true, 0.238838, true, 0.423677, true, 0.154464], "time": 1677440323.0809} {"met": [false, null, false, null, null, false, null, false, null, false, null, false, null], "time": 1677436167.8629}
com	Ментальні команди: нейтральні + до 4 попередньо підготовлених елементів на тренувальний профіль ('neutral', 'push', 'left', 'pull')
fac	Вирази обличча: Blink, Wink L/R, Surprise, Frown, Smile, Clench *Приклад даних:* {'id': 6, 'jsonrpc': '2.0', 'result': {'failure': [], 'success': [{'cols': ['eyeAct', 'uAct', 'uPow', 'lAct', 'lPow'], 'sid': '28d91712-07ec-4d98-86ed-1b0f044248eb', 'streamName': 'fac'}
eeg	Теоретично, eeg дані можуть буди отримані, але потрібна окрема ліцензія від постачальника (PRO license). В цій роботі eeg включені в підписку, але не отримуються. [3] Emotiv

	Приклад даних: {'eeg': [99, 0, 4291.795, 4371.795, 4078.461, 4036.41, 4231.795, 0.0, 0], 'time': 1627457774.5166}

На чому базуються виявлення? Як створювалися алгоритми?

Виявлення базуються на кількох різних експериментальних наборах даних, зібраних від добровольців під час власних експериментів Emotive [6].

Наприклад, виявлення фрустрації було проведено з більш ніж 30 суб'єктами змішаної статі, які носили гарнітури Emotiv, ЕКГ, респіратор, GSR і сфігмоманометр під час певної діяльності. Їх супроводжував психолог, вони знімали на відео та повідомляли про себе до та після події, а базові показники всього обладнання були зроблені до та після.

Експерименти складалися з серії онлайн-ігор, які були розроблені, щоб викликати розчарування, наприклад, гра в стилі PacMan, де елементи керування випадково виходили з ладу або змінювалися все частіше, і гра FPS, де суб'єкт поступово перевантажувався, а його обладнання було ненадійним.

Крім того, деякі з виявлень не покладаються на попередні дані, вони збирають дані з тегами та тренують нові сигнатури на основі даних ЕЕГ кожної особи (набір розумових команд працює таким чином).

2.4. Розробка API з використанням minimal .NET API для зберігання та обміну даними

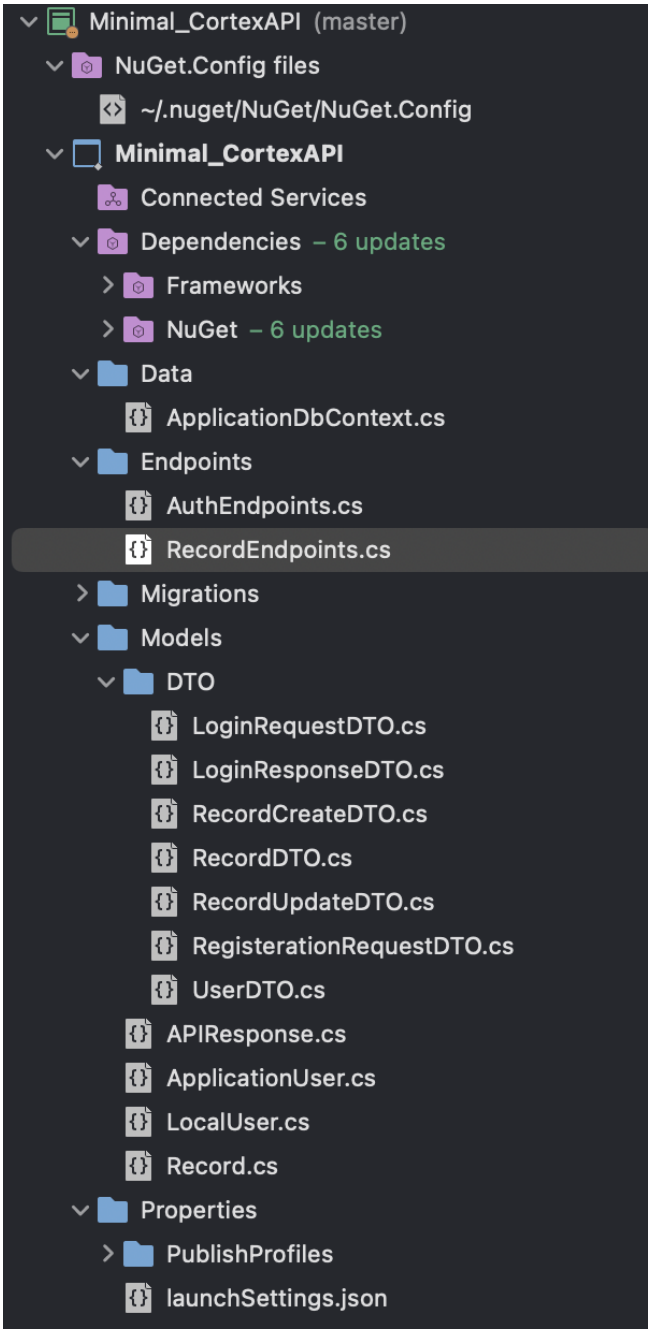
Для зберігання отриманий даних був розроблений API з використанням minimal .NET API [12]. Для обміну даними використовується CRUD, для авторизації - JWT:

Рівень API: цей рівень обробляє вхідні HTTPS-запити та надсилає HTTPS-відповіді. Він складається з кінцевих точок для різних методів HTTP, таких як GET, POST, PUT і DELETE. Ці контролери взаємодіють із сервісним рівнем для виконання операцій CRUD над даними.

Сервісний рівень: цей рівень містить бізнес-логіку програми. Він відповідає за реалізацію шаблону репозиторію для взаємодії з базою даних і виконання операцій CRUD. Він також реалізує авторизацію JWT для перевірки запитів користувачів.

Рівень сховища: цей рівень взаємодіє з базою даних для виконання операцій CRUD. Він містить реалізацію DTO для передачі даних між різними рівнями програми.

Рівень бази даних: цей рівень містить базу даних SQL, яка зберігає дані програми.


Minimal_CortexAPI (master)
NuGet.Config files
~/.nuget/NuGet/NuGet.Config
Minimal_CortexAPI
Connected Services
Dependencies – 6 updates
Frameworks
NuGet – 6 updates
Data
ApplicationDbContext.cs
Endpoints
AuthEndpoints.cs
RecordEndpoints.cs
Migrations
Models
DTO
LoginRequestDTO.cs
LoginResponseDTO.cs
RecordCreateDTO.cs
RecordDTO.cs
RecordUpdateDTO.cs
RegisterationRequestDTO.cs
UserDTO.cs
APIResponse.cs
ApplicationUser.cs
LocalUser.cs
Record.cs
Properties
PublishProfiles
launchSettings.json

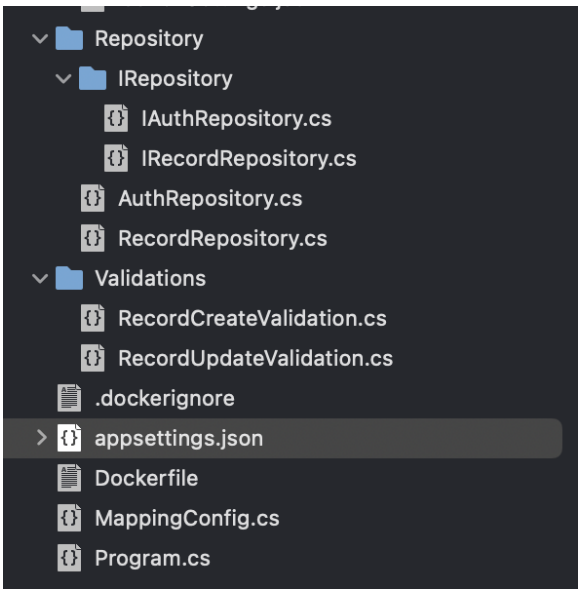


Рисунок 2.7 – Структура проекту

У цій архітектурі елементи AuthEndpoints і RecordEndpoints визначають кінцеві точки для різних методів HTTP.

Клас RecordEndpoints викликає інтерфейс IRecordRepository, який реалізований класом RecordRepository.

Клас AuthEndpoints викликає інтерфейс IAuthRepository, який реалізований класом AuthRepository.

Класи RecordRepository та AuthRepository взаємодіє з базою даних SQL для виконання операцій CRUD над моделями Record та LocalUser відповідно.

Для аутетнікації використовується Microsoft.AspNetCore Authentication.AuthenticationMiddleware

Для авторизації використовується Microsoft. AspNetCore. Authentication.AuthorizationMiddleware

UseRouting налаштовує маршрутизацію для обробки вхідних запитів. Потім UseAuthorization авторизує запит на основі схем автентифікації та політик, налаштованих у програмі. UseEndpoints відображає кінцеві точки на відповідні запити.

Для JWT для автентифікації користувача використовується Microsoft.AspNetCore.Authentication.JwtBearer. Це пакет у структурі ASP.NET Core, який забезпечує перевірку веб-токенів JSON (JWT) у вхідних запитах.

Для керування автентифікацією та авторизацією користувачів використовується Microsoft.AspNetCore.Identity.

Файл appsettings.json містить налаштування конфігурації програми, наприклад рядок підключення до бази даних SQL. Файл Program.cs відповідає за налаштування та запуск програми .NET minimal API.

2.5. Публікація та взаємодія з Azure

Публікація наразі відбувається використовуючи публікацію Visual Studio, але в подальшому розвитку проекту можливо було б налаштувати CI/CD іншим чином.

Рисунок 2.8 – Публікація застосунку

2.6. Архітектура бази даних

В якості бази даних було обрано SQL Azure database. База даних SQL Azure — це повністю керована платформа як служба (PaaS) механізм баз даних, який обробляє більшість функцій керування базою даних, таких як оновлення, виправлення, резервне копіювання та моніторинг без участі користувача. Доступність - 99,99%.

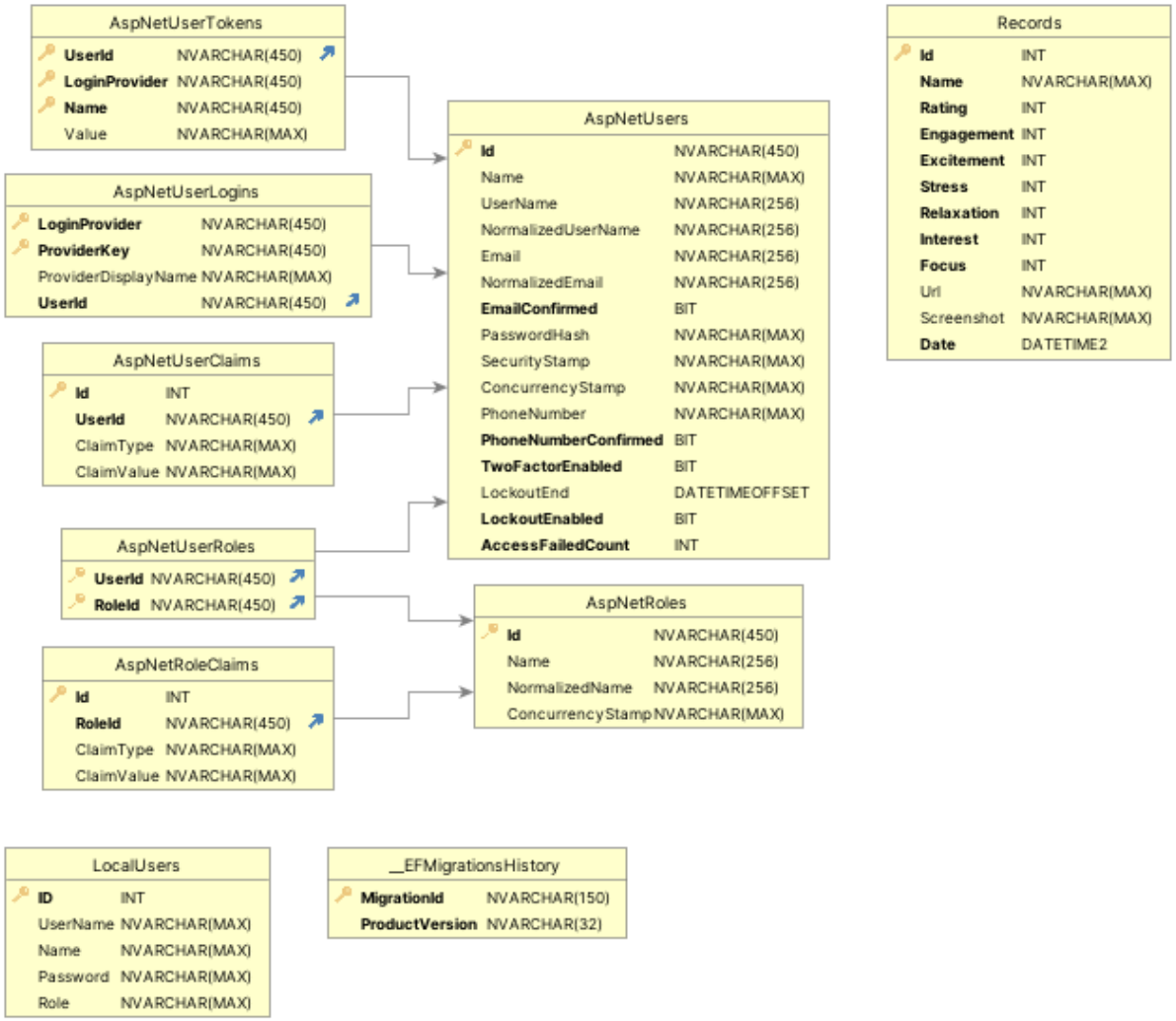


Рисунок 2.9 – Структура бази даних

Ця таблиця представляє зв’язок між користувачем і створеними ним записами.

З цією структурою бази даних можливо використовувати Entity Framework або іншу ORM, щоб зіставляти моделі з таблицями бази даних і взаємодіяти з ними у своїй програмі.

РОЗДІЛ 3 Реалізація прототипу збору та накопичення інформації

3.1. Архітектура хром додатку

Для збору даних користувача був розроблений хром додаток. Додаток використовує Cortex API для взаємодією з гарнітурою, то ж:

1) Реалізований процес, зображений на малюнку *2.3*, та описаний у розділі 2.3.1 *(Огляд робочого процесу API)*
2) Створена підписка на отримання даних, що описується у розділі 2.3.1 *(Підписка на дані)*
3) Створений графічний інтерфейс для відображення отриманих даних.
4) Створена інтерфейс для регістрації та входу користувача.
5) Дані з гарнітури отримуються за допомогою Websocket.
6) Дані відправляються на сервер за допомогою REST API.

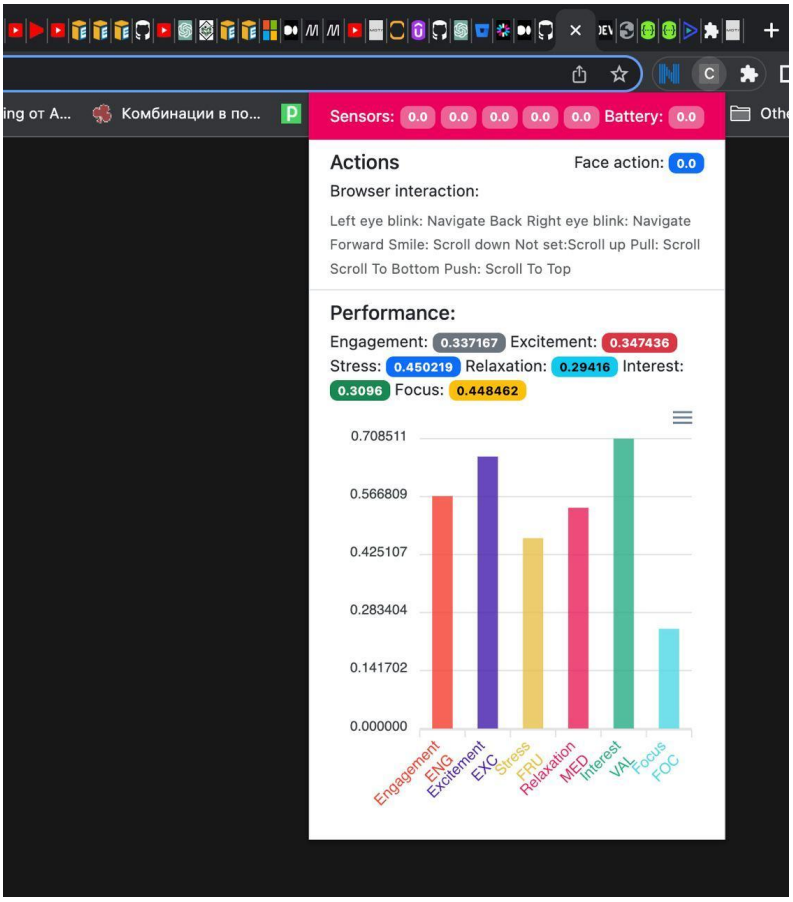


Рисунок 3.1 – Хром-додаток

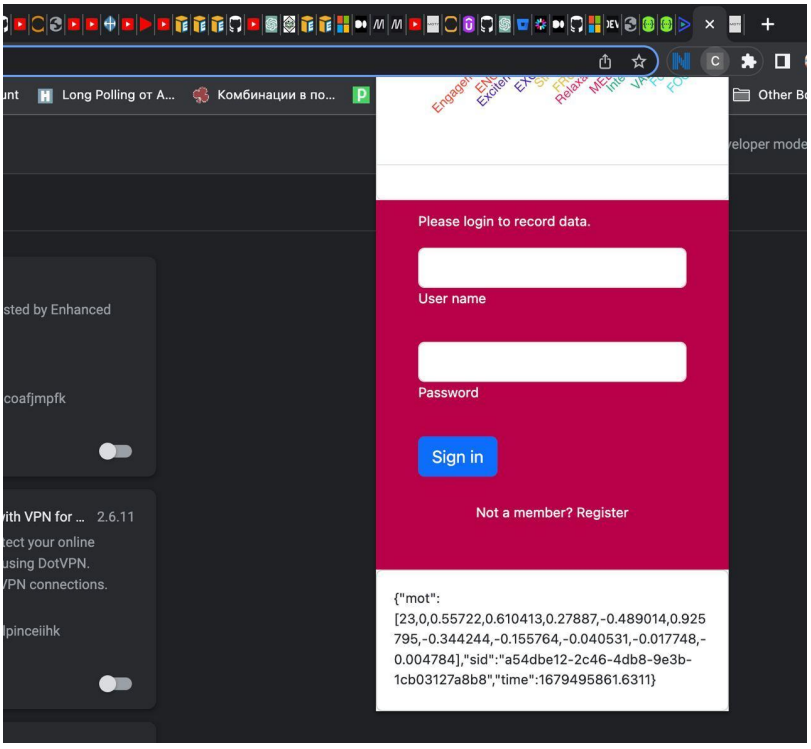


Рисунок 3. 2 – Хром-додаток

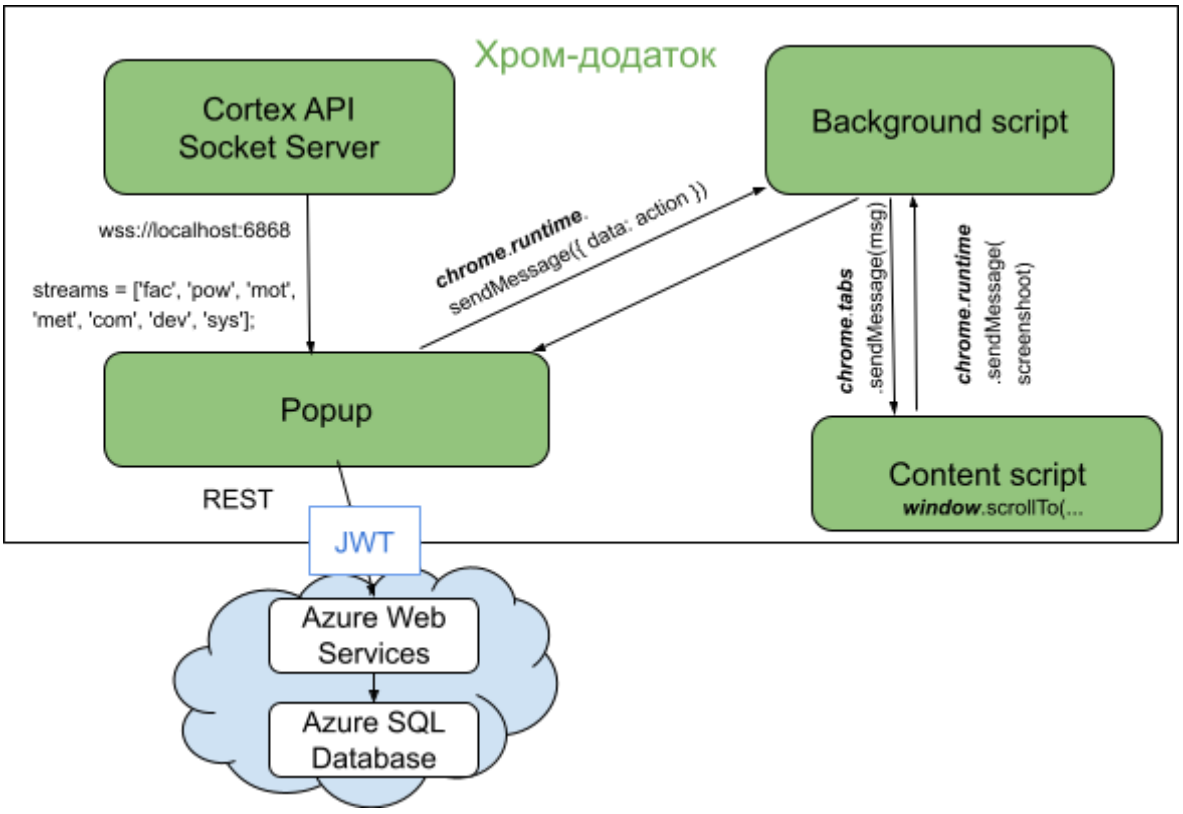


Рисунок 3.3 – Архітектура хром-додатку.

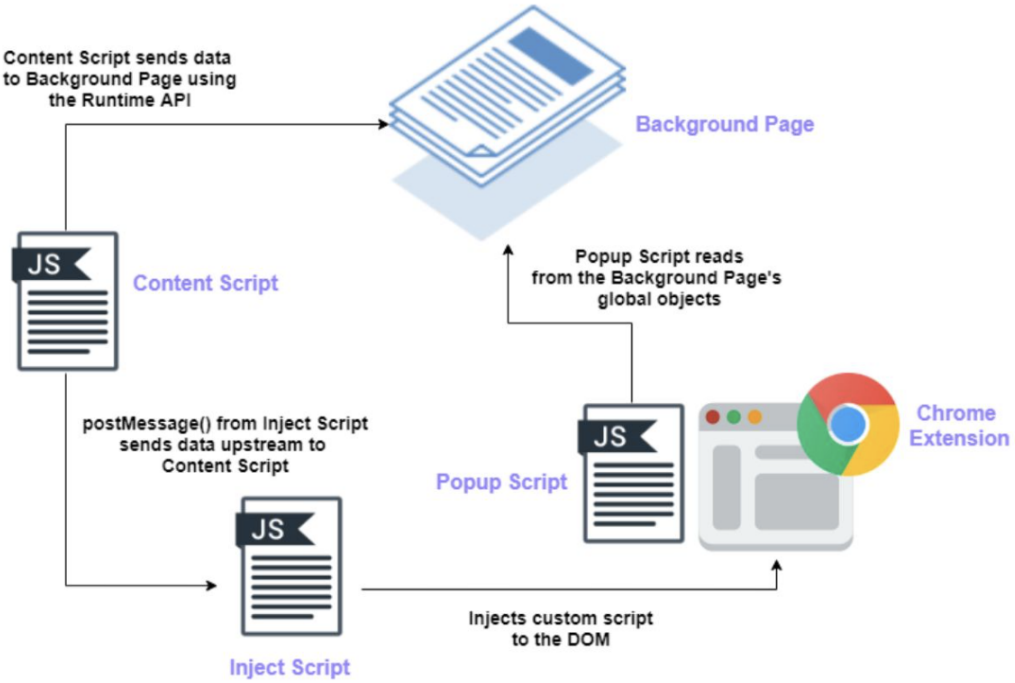


Рисунок 3.4 – Передача повідомлень у хром-додатку.

3.2. Взаємодія з браузером

Взаємодія з браузером відбувається шляхом передачі команди у content-script, якій має безпосередній доступ до сторінки браузера. Тобто відбувається JavaScirpt inject, з можливостями, що надає JavaScript.

У прототипі були використані наступні команди (але в подальшому потрібно провести аналіз найбільш потрібних користувачу дій):

```
chrome.runtime.onMessage.addListener((msg, sender, sendResponse) => {
   console.log("content got message: ", msg);
   if (msg.data === 'scrollUp') {
       window.scrollTo({ top: document.documentElement.scrollTop -100,
behavior: 'smooth' })
   }
   if (msg.data === 'scrollDown') {
       window.scrollTo({ top: document.documentElement.scrollTop +100,
behavior: 'smooth' })
   }
   if (msg.data === 'scrollToTop') {
       window.scrollTo({ top: 0, behavior: 'smooth' })
   }
   if (msg.data === 'scrollToBottom') {
       window.scrollTo(0, document.body.scrollHeight);
   }
   if (msg.data === 'goBack') {
       window.history.go(-1);
   }
   if (msg.data === 'goForward') {
       window.history.forward();
   }
});
```

3.3. Взаємодією з гарнітурою та збір даних користувача

Взаємодієя з гарнітурою та збір даних користувача відбувається у popup сторінці. Таке рішення було обсновано частою втратою сигналу з гарнітури, тож у popup сторінці цю проблему простіше відстежити і усунути.

Для взаємодією з гарнітурою використовується Cortex API, то ж при надходженні сигналу з гарнітури, з popup відправляється запит до service worker, і якщо в момент запиту один з Chrome tab є активним, то service worker надсилає повідомлення до Popup page з такими даними:

- Screenshot сторінки
- URL сторінки

Дані було обрано, щоб продемонструвати можливості взаємодії додатку з сторінкою браузера. В подальшому розвитку проекту можливо передавати інші дані, за переліком – всі які можна отримати з сторінки браузера за допомогою JavaScript.

Зібрані дані надсилаються до серверу для збереження у базі даних.

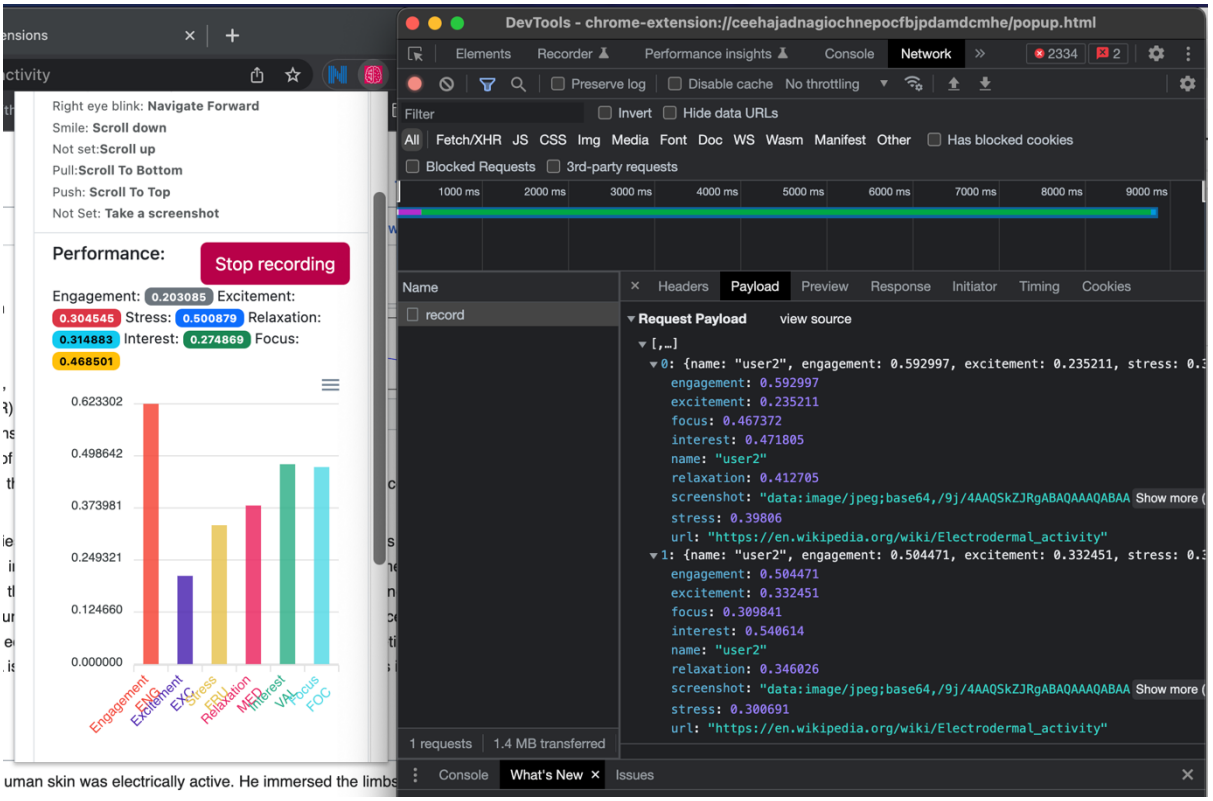


Рисунок 3.5 – Приклад даних.

Зібрані дані можливо завантажити у вигляді .csv файлу.

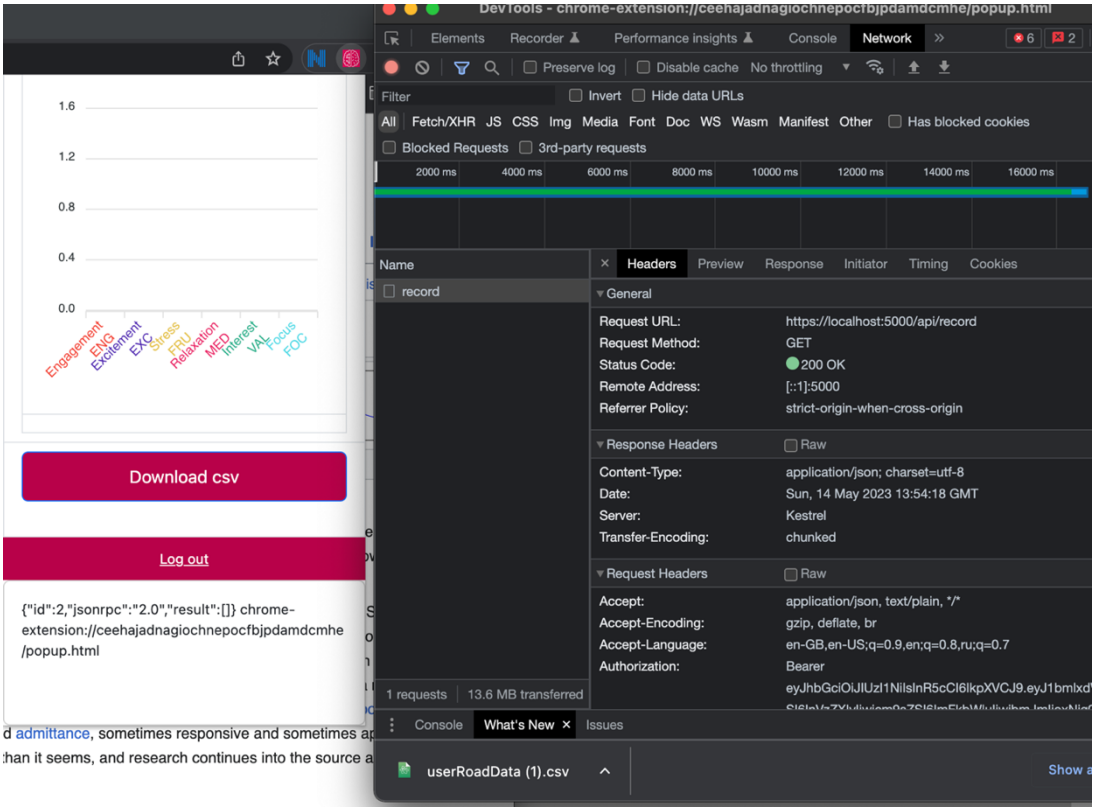


Рисунок 3.6 – Приклад запиту даних.

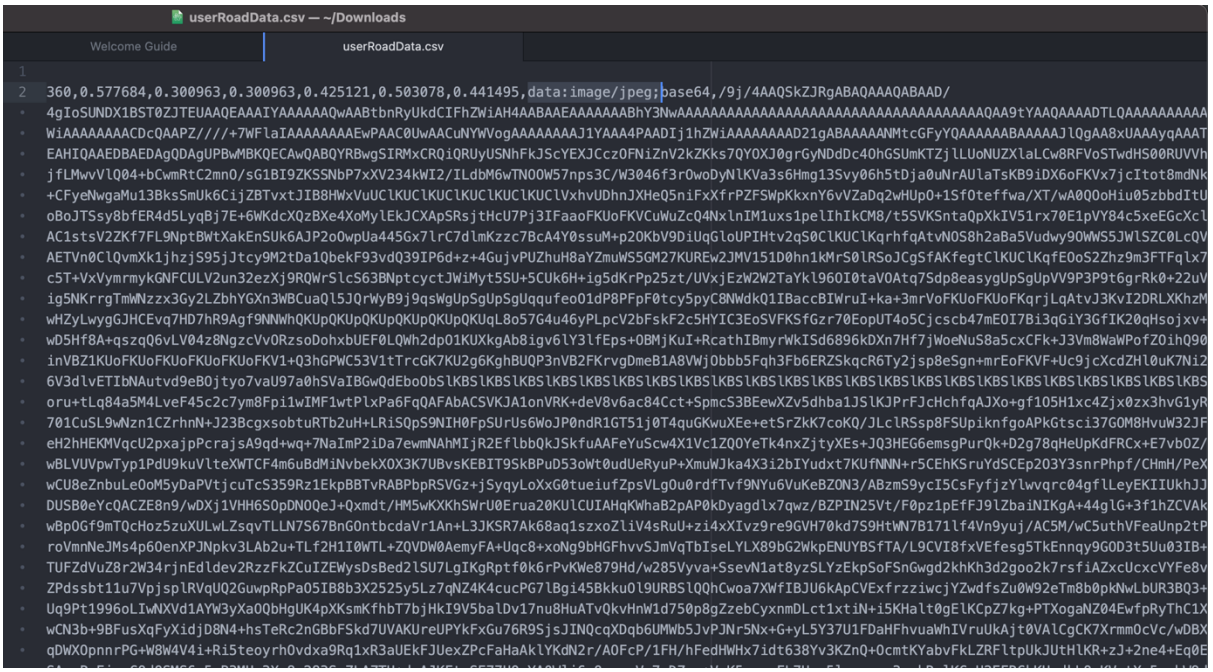


Рисунок 3.7 – Приклад .csv файлу.

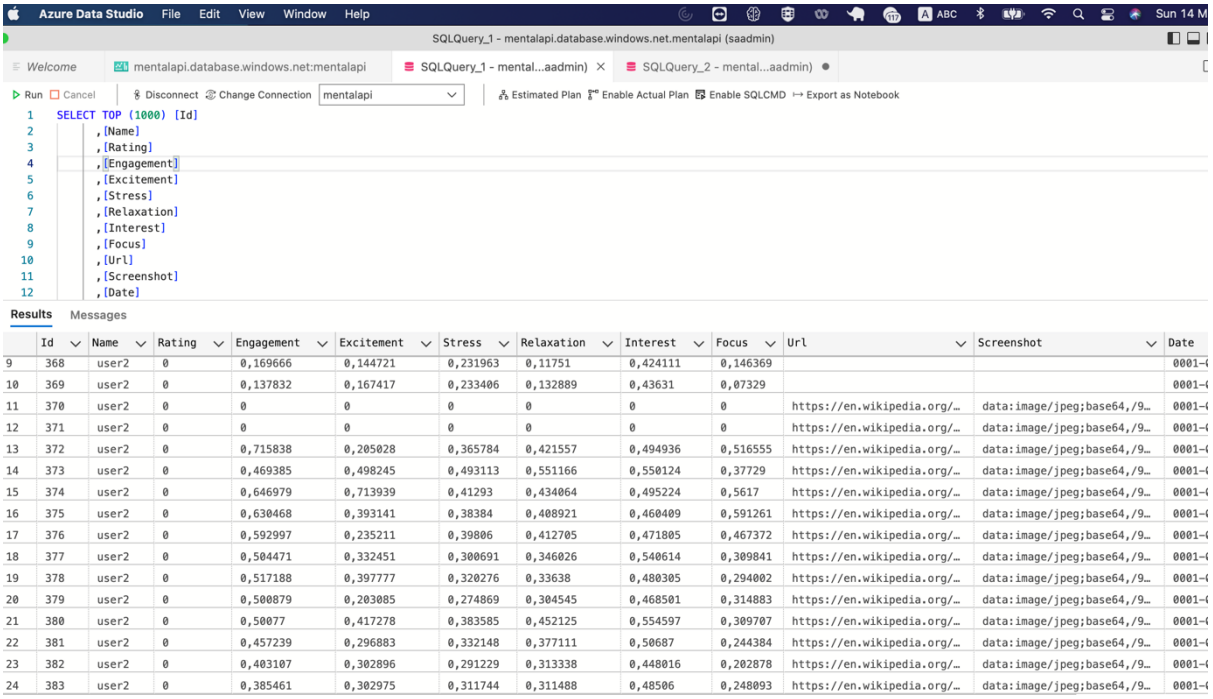


Рисунок 3.8 – Приклад зібраних даних.

Висновки по роботі та рекомендації для подальших досліджень

В ході роботи було проаналізовано можливості взаємодії між BCI приладом та Chrome браузером. Був розроблений прототип, що дозволяє керувати сторінками Chrome за допомогою ментальних команд та збирати дані про досвід користування веб ресурсами, що доповнені даними про емоційний та ментальний стан користувача.

Розроблений прототип може використовуватись для покращення навчального процесу або маркетингових досліджень, тобто він передбачає модель B2B (business-to-business). Подальший напрямок розвитку додатку передбачає підтримку найбільш розповсюджених гарнітур BCI.

Зроблені наступні висновки:

По перше, вибір приладу можливо переглянути, з причин, що описані нижче.
Для збору даних був використаний EEG Emotiv Insight. Вибір був оснований на тому що, по опису виробника:

1) Наявність EEG сенсорів, можливість отримання як не оброблених EEG сигналів, так і оброблених (дані продуктивності, ментальні команди, оброблені сигнали лицевих м’язів - певні вирази обличчя)
2) Зручність використання Emotiv Insight - це бездротових прилад, з терміном роботи від акумулятора - до 20 годин.
3) Ціновий діапазон. Вартість приладу - 500$, що є середньо доступною ціною на ринку подібних приладів

4) API - є можливість використовувати різні мови програмування - Python, C++, JavaScript

В ході досліджень виявлено декілька недоліків, які можуть бути перешкодою для користування розробленим додатком, навіть якщо не брати до уваги досить високу ціну приладу і технічної підтримки.

1) Погана якість сигналу. Прилад має 5 напівсухих полімерних датчиків, то ж встановлення датчиків у вірну позицію займає багато часу. По інструкції, датчики потрібно змочувати розчином, але при тестуванні виявилось, що для повного контакту потрібно, щоб волосся було вологим, або голова має бути поголена. При тестуванні не вдалося отримати 100% з‘єднання, максимальна отримана якість була 80%. Відповідно користуватися ментальними командами було не можливо.
2) Під час користування додатком, датчики постійно потрібно змочувати, бо інакше втрачається сигнал або втрачається зв’язок. Це **вплинуло на розробку архітектури додатку**, бо в наданій реалізації з’єднання з API (websocket з’єднання) та отримання даних відбувається у popup сторінці, хоча хороша практика - використовувати service-worker для отримання даних, а у popup дані потрібно передавати за допомогою повідомлень, наприклад, якщо потрібна візуалізація даних.
3) Для отримання EEG даних потрібна окрема ліцензія.
4) Для отримання ментальних команд потрібен окремий запит в службу підтримки.
5) Слабка підтримка с боку комюніті, мало прикладів роботи з API.
6) Для користування додатком потрібен clientId.

По-друге, збір даних, які поєднують в собі інформацію про досвід користувача разом з даними про ментальні стани та управління за

допомогою ментальних команд, відкриває широкі перспективи і можливості. Для розкриття цих можливостей необхідно провести додаткові дослідження та вдосконалити прототип.

В цілому, використання BCI є цікавим і перспективним напрямком, але не дивлячись на шалених розвиток, який відбувся за останні 10 років у цьому напрямку, існує ще багато питань, які можуть бути перешкодою для щоденного використання BCI користувачами, що не спеціалізуються в даній області.

Список літератури

1. Rashid, M., Sulaiman, N., Abdul Majeed, A. P. P., Musa, R. M., Ab. Nasir, A. F., Bari, B. S., & Khatun, S. (2020). Current Status, Challenges, and Possible Solutions of EEG-Based Brain-Computer Interface: A Comprehensive Review. Frontiers in Neurorobotics. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2020.00025/full (дата звернення: 20.02.2023)
2. Wireless Sensors for Brain Activity—A Survey. Mahyar TajDini, Volodymyr Sokolov, Ievgeniia Kuzminykh, Stavros Shiaeles, and Bogdan Ghita URL: https://www.researchgate.net/publication/346714810_Wireless_Sensors_for_Brain_Activity-A_Survey (дата звернення: 20.02.2023)
3. A Beginner's Guide to Brain-Computer Interface and Convolutional Neural Networks by Alexandre Gonfalonieri URL: https://towardsdatascience.com/a-beginners-guide-to-brain-computer-interface-and-convolutional-neural-networks-9f35bd4af948 (дата звернення: 5.05.2023)
4. Wikipedia contributors. Brain–computer interface. URL: https://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface (дата звернення: 5.05.2023)
5. Emotiv. Insight 2 User Manual: Insight Sensor Coverage. URL: https://emotiv.gitbook.io/insight-2-user-manual/introduction/insight-sensor-coverage (дата звернення: 20.02.2023)
6. Emotiv. "What are the Detections Based On? How Were the Algorithms Created?" URL: https://www.emotiv.com/knowledge-base/what-are-the-detections-based-on-how-were-the-algorithms-created/ (дата звернення: 20.02.2023)

7. Emotiv. "Where Can I Get Access to Raw EEG API?" URL: https://www.emotiv.com/knowledge-base/where-can-i-get-access-to-raw-eeg-api/ (дата звернення: 20.02.2023)
8. Emotiv. “Insight” URL: https://www.emotiv.com/insight/ (дата звернення: 21.02.2023)
9. Emotiv. "What are the Performance Metrics? Detection Suite." URL: https://www.emotiv.com/knowledge-base/what-are-the-performance-metrics-detection-suite/ (дата звернення: 21.02.2023)
10. Emotiv. "Are Emotiv Products Medical Devices?" URL: https://www.emotiv.com/knowledge-base/are-emotiv-products-medical-devices/ (дата звернення: 21.02.2023)
11. Google. (2023). "Extensions 101 - Chrome Developers." Google Chrome Developer Documentation. URL: https://developer.chrome.com/docs/extensions/mv3/getstarted/extensions-101/ (дата звернення: 20.02.2023)
12. Microsoft. (2023). "Create a web API with ASP.NET Core." Microsoft Docs. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&tabs=visual-studio (дата звернення: 21.02.2023)
13. Emotiv. "Basics of Neural Oscillations." URL: https://www.emotiv.com/tutorials/basics-of-neural-oscillations/ (дата звернення: 21.02.2023)
14. Emotiv. "Neuroscience Guide." URL: https://www.emotiv.com/neuroscience-guide/ (дата звернення: 21.02.2023)
15. Emotiv. "BCI Guide." URL: https://www.emotiv.com/bci-guide/ (дата звернення: 23.02.2023)
16. Emotiv. "Where Can I Get Access to Raw EEG API?" Отримано з https://www.emotiv.com/knowledge-base/where-can-i-get-access-to-raw-eeg-api/ (дата звернення: 22.02.2023)

17. Emotiv Cortex API Documentation. URL: https://emotiv.gitbook.io/cortex-api/ (дата звернення: 14.03.2023)
18. Microsoft Azure SQL Database: Platform-as-a-Service (PaaS) Overview. URL: https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql (дата звернення: 21.04.2023)

Додаток А. Програмний код взаємодії з CortexAPI

```
let socketUrl = 'wss://localhost:6868';

function initWebSocket() {
    socket = new WebSocket(socketUrl);

    let headsetId = '';
    let ctResult = '';
    let authToken = '';
    let cortexToken = '';
    let sessionId = '';

    const streams = ['fac', 'pow', 'mot', 'met', 'com', 'dev', 'sys'];
    const profileName = 'Test_2';

    function requestAccess() {
        console.log('requestAccess 1')
        return new Promise(function(resolve, reject) {
            console.log('requestAccess 2')
            const REQUEST_ACCESS_ID = 1
            let requestAccessRequest = {
                "jsonrpc": "2.0",
                "method": "requestAccess",
                "params": {
                    "clientId": user.clientId,
                    "clientSecret": user.clientSecret
                },
                "id": REQUEST_ACCESS_ID
            }
```

```
            socket.send(JSON.stringify(requestAccessRequest));
            socket.addEventListener('message', (data) => {
                console.log("requestAccess data: ", data);

                try {
                    if (JSON.parse(data)['id'] == REQUEST_ACCESS_ID) {
                        resolve(data);
                    }
                } catch (error) {}
            })
        })
    }

    function queryHeadsetId(socket) {
        console.log('queryHeadsetId 1');
        const QUERY_HEADSET_ID = 2
        let queryHeadsetRequest =  {
            "jsonrpc": "2.0",
            "id": QUERY_HEADSET_ID,
            "method": "queryHeadsets",
            "params": {}
        }

        return new Promise(function(resolve, reject) {
            console.log('queryHeadsetId : sending request: ',
queryHeadsetRequest);
            socket.send(JSON.stringify(queryHeadsetRequest));
            socket.addEventListener('message', (resp) => {
                try {

                    if (JSON.parse(resp.data)['id'] == QUERY_HEADSET_ID) {

                        if (JSON.parse(resp.data)['result'].length !== 0) { //
Connected headset
                            headsetId =
JSON.parse(resp.data)['result'][0]['id']
                            resolve(headsetId.toString());
                        }
                        else {
                            console.log('No have any headset, please connect
headset with your pc.')
```

```
                    }
                }
            } catch (error) {}
        })
    })
}

/**
 * - query headset infor
 * - connect to headset with control device request
 * - authentication and get back auth token
 * - create session and get back session id
 */
async function querySessionInfo() {
    await queryHeadsetId(socket).then((headset) => {
        console.log('Got data from queryHeadsetId: ', headset);
        headsetId = headset
    })

    await controlDevice(headsetId).then((result) => {
         ctResult = result
    })

    await authorize().then((auth) => {
        authToken = auth
    })

    await createSession(authToken, headsetId).then((result)=>{
        sessionId=result
    })

    console.log('HEADSET ID ----------------------------------')
    console.log(headsetId)
    console.log('\r\n')
    console.log('CONNECT STATUS -------------------------------')
    console.log(ctResult)
    console.log('\r\n')
    console.log('AUTH TOKEN ----------------------------------')
    console.log(authToken)
    console.log('\r\n')
    console.log('SESSION ID ----------------------------------')
```

```
    console.log(sessionId)
    console.log('\r\n')
}

async function controlDevice (headsetId) {
    const CONTROL_DEVICE_ID = 1
    let controlDeviceRequest = {
        "jsonrpc": "2.0",
        "id": CONTROL_DEVICE_ID,
        "method": "controlDevice",
        "params": {
            "command": "connect",
            "headset": headsetId
        }
    }
    return new Promise(function(resolve, reject){
        socket.send(JSON.stringify(controlDeviceRequest));
        socket.addEventListener('message', (resp) => {

            try {
                if(JSON.parse(resp.data)['id'] == CONTROL_DEVICE_ID){
                    resolve(resp.data)
                }
            } catch (error) {}
        })
    })
}

async function authorize() {
    return new Promise(function(resolve, reject){
        const AUTHORIZE_ID = 1
        let authorizeRequest = {
            "jsonrpc": "2.0", "method": "authorize",
            "params": {
                "clientId": user.clientId,
                "clientSecret": user.clientSecret
            },
            "id": AUTHORIZE_ID
        }
        socket.send(JSON.stringify(authorizeRequest))
        socket.addEventListener('message', (resp) => {
```

```
                try {
                    if(JSON.parse(resp.data)['id']==AUTHORIZE_ID){
                        cortexToken =
JSON.parse(resp.data)['result']['cortexToken']
                        console.log('authorize ::: cortexToken',
cortexToken);
                        resolve(cortexToken)
                    }
                } catch (error) {}
            })
        })
    }

    async function createSession(authToken, headsetId){
        const CREATE_SESSION_ID = 1
        let createSessionRequest = {
            "jsonrpc": "2.0",
            "id": CREATE_SESSION_ID,
            "method": "createSession",
            "params": {
                "cortexToken": authToken,
                "headset": headsetId,
                "status": "active"
            }
        }
        return new Promise(function(resolve, reject){
            socket.send(JSON.stringify(createSessionRequest));
            socket.addEventListener('message', (resp)=>{
                // console.log(data)
                try {
                    if(JSON.parse(resp.data)['id']==CREATE_SESSION_ID){
                        let sessionId = JSON.parse(resp.data)['result']['id']
                        resolve(sessionId)
                    }
                } catch (error) {}
            })
        })
    }

    /**
     * - check if user logined
```

```
    * - check if app is granted for access
    * - query session info to prepare for sub and train
    */
   async function checkGrantAccessAndQuerySessionInfo() {
       let requestAccessResult = ""
       await requestAccess().then((result)=>{requestAccessResult=result})

       let accessGranted = JSON.parse(requestAccessResult)

       // check if user is logged in CortexUI
       if ("error" in accessGranted) {
           console.log('You must login on CortexUI before request for grant
access then rerun')
           throw new Error('You must login on CortexUI before request for
grant access')
       } else {
           if(accessGranted['result']['accessGranted']){
               console.log('before querySessionInfo')
               await querySessionInfo()
               console.log('after querySessionInfo')
           }
           else{
               console.log('You must accept access request from this app on
CortexUI then rerun')
               throw new Error('You must accept access request from this app
on CortexUI')
           }
       }
   }

   function subRequest(stream) {
       const SUB_REQUEST_ID = 6
       let subRequest = {
           "jsonrpc": "2.0",
           "method": "subscribe",
           "params": {
               "cortexToken": authToken,
               "session": sessionId,
               "streams": stream
           },
           "id": SUB_REQUEST_ID
```

```
    }
    console.log('sub eeg request: ', subRequest)
    socket.send(JSON.stringify(subRequest))
}
/**
 *
 * - check login and grant access
 * - subcribe for stream
 * - logout data stream to console or file
 */
async function sub(streams) {
    console.log("Start sub flow");
    subRequest(streams)
}

/**
 * - check login and grant access
 * - create profile if not yet exist
 * - load profile
 * - sub stream 'sys' for training
 * - train for actions, each action in number of time
 *
 */

async function train (profileName, trainingActions, numberOfTrain) {
    console.log("start training flow");

    // to training need subcribe 'sys' stream
    subRequest(['sys']);

    // create profile
    let status = "create";
    let createProfileResult = ""
    await setupProfile(authToken,
        headsetId,
        profileName, status).then((result)=>{createProfileResult=result})

    // load profile
    status = "load"
    let loadProfileResult = ""
    await setupProfile(authToken,
```

```
            headsetId,
            profileName, status).then((result)=>{loadProfileResult=result})

        // training all actions
        let self = this

        for (let trainingAction of trainingActions){
            for (let numTrain=0; numTrain<numberOfTrain; numTrain++){
                // start training for 'neutral' action
                console.log(`START TRAINING "${trainingAction}" TIME
${numTrain+1} ---------------`)
                console.log('\r\n')
                await trainRequest(authToken,
                    sessionId,
                    trainingAction,
                    'start');
                //
                // FROM HERE USER HAVE 8 SECONDS TO TRAIN SPECIFIC ACTION
                //
                // accept 'neutral' result
                console.log(`ACCEPT "${trainingAction}" TIME ${numTrain+1}
--------------------`)
                console.log('\r\n')
                await trainRequest(authToken,
                    sessionId,
                    trainingAction,
                    'accept')
            }

            let status = "save"
            let saveProfileResult = ""

            // save profile after train
            await self.setupProfile(self.authToken,
                self.headsetId,
                profileName, status)
                .then((result)=>{
                    saveProfileResult=result
                    console.log(`COMPLETED SAVE ${trainingAction} FOR
${profileName}`);
                    return true;
```

```
            });
    }
}


function setupProfile(authToken, headsetId, profileName, status) {
    const SETUP_PROFILE_ID = 2
    let setupProfileRequest = {
        "jsonrpc": "2.0",
        "method": "setupProfile",
        "params": {
            "cortexToken": authToken,
            "headset": headsetId,
            "profile": profileName,
            "status": status
        },
        "id": SETUP_PROFILE_ID
    }
    // console.log(setupProfileRequest)

    return new Promise(function(resolve, reject){
        socket.send(JSON.stringify(setupProfileRequest));
        socket.addEventListener('message', (data)=>{
            if(status=='create'){
                resolve(data)
            }

            try {
                // console.log('inside setup profile', data)
                if(JSON.parse(data)['id']==SETUP_PROFILE_ID){
                    if(JSON.parse(data)['result']['action']==status){
                        resolve(data)
                    }
                }

            } catch (error) {

            }
        })
    });
}
```

```
async function live(profileName) {
    // load profile
    let loadProfileResult=""
    let status = "load"
    await setupProfile(authToken,
        headsetId,
        profileName,
        status).then((result)=>{loadProfileResult=result})
    console.log(loadProfileResult)

    //sub 'com' stream and view live mode
    subRequest(['com'], authToken, sessionId)

    socket.addEventListener('message', (data)=>{
        console.log('Live:: ' , data);
    });
}

/*************************************************/
socket.addEventListener('open', async () => {
    console.log('WebSocket connection established.');
    console.log("send data...");
    console.log("querySessionInfo..");
    await querySessionInfo();
    await sub(streams);
    await live(profileName);
});

socket.addEventListener('message', async (event) => {
    const message = event.data;

    addData(message);
    let data = JSON.parse(message);
    if (Object.keys(data)[0] === "met") {
        data = data.met;
        emotion_1.innerText = data[1];
        emotion_2.innerText = data[3];
        emotion_3.innerText = data[6];
        emotion_4.innerText = data[8];
        emotion_5.innerText = data[10];
```

```
            emotion_6.innerText = data[12];

            const time = data.time;
            await sendActionToContentScript('getUrl');
            const resp = await sendActionToContentScript('takeScreenshot');
            if (resp?.imgSrc) {
                CURRENT_LOCATION = resp?.imgSrc;
            }

            const prevPerformanceState =
JSON.parse(localStorage.getItem('prevPerformanceState'));

            const performanceState = {
                engagement: data[1],
                excitement: data[3],
                stress: data[6],
                relaxation: data[8],
                interest: data[10],
                focus: data[12],
            };

            let recordData =
                [
                    performanceState['engagement'],
                    performanceState['excitement'],
                    performanceState['stress'],
                    performanceState['relaxation'],
                    performanceState['interest'],
                    performanceState['focus'],
                    CURRENT_LOCATION
                ];
            localStorage.setItem('_prevPerformanceData',
JSON.stringify(performanceState));
            if (START_RECORDING) {
                console.log("Recording..prevPerformanceState.",
prevPerformanceState);
                if (isCadenceChanged(prevPerformanceState, performanceState,
CADENCE_THRESHOLD)) {
                    const screenshotData = await
sendActionToContentScript('takeScreenshot');
                    recordData.push(screenshotData);
```

```
            }
            saveUserPerformanceData(recordData);
        }
        await renderChart(data)
    }

    if (data?.dev) {
        battery_level.innerText = data.dev[3] + '%';
        sensor_1.innerText =  data.dev[2][0];
        sensor_2.innerText =  data.dev[2][1];
        sensor_3.innerText =  data.dev[2][2];
        sensor_4.innerText =  data.dev[2][3];
        sensor_5.innerText =  data.dev[2][4];
    }
    if (Object.keys(data)[0] === "action") {
        current_action.innerText =  `Current action: ${data.com[0]}`;
    }

    if (data?.fac) {
        if (data.fac[0] === 'blink') {
            face_action.innerText = data.fac[0];
            face_action_num.innerText = ''
            await sendActionToContentScript('scrollUp');
        }

        if (data.fac[4] > 0.5) {
            console.log("smile ", data.fac[3])
            face_action.innerText = data.fac[3];
            face_action_num.innerText = data.fac[4]
            await sendActionToContentScript('scrollDown');
        }
    }
});
```

Додаток В. Програмний код ваємодії браурером

```
chrome.runtime.onMessage.addListener((msg, sender, sendResponse) => {
   console.log("content got message: ", msg);
   if (msg.data === 'scrollUp') {
       window.scrollTo({ top: document.documentElement.scrollTop -100,
behavior: 'smooth' })
```

```
    }
    if (msg.data === 'scrollDown') {
        window.scrollTo({ top: document.documentElement.scrollTop +100,
behavior: 'smooth' })
    }
    if (msg.data === 'scrollToTop') {
        window.scrollTo({ top: 0, behavior: 'smooth' })
    }
    if (msg.data === 'scrollToBottom') {
        window.scrollTo(0, document.body.scrollHeight);
    }
    if (msg.data === 'goBack') {
        window.history.go(-1);
    }
    if (msg.data === 'goForward') {
        window.history.forward();
    }
```