НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій, СТІРЕНКО

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Метод інтеграції та обробки мультимодальних даних»

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-64
Гордієнко Нікіта Юрійович _____

Керівник:
Доцент, кандидат технічних наук,
Роковий Олександр Петрович _____

Консультант з нормконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
Доцент, кандидат технічних наук
Писаренко Андрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 6404. 00.001 ВП	Відомість дипломного проєкту	1	
3	А4	ДП 6404. 00.002 ТЗ	Технічне завдання	3	
4	А4	ДП 6404. 00.003 ПЗ	Пояснювальна записка	94	
5	А4	ДП 6404. 00.004 Д1	Схема інтерфейсу мобільного додатку.	1	
6	А4	ДП 6404. 00.005 Д2	Діаграма класів сутностей використаних зі сторони серверного ПЗ.	1	
7	А4	ДП 6404. 00.006 Д3	Діаграма таблиць у створеній базі даних.	1	
8	А4	ДП 6404. 00.007 Д4	Принципова схема реалізації методу.	1	
9	А4	ДП 6404. 00.008 Д5	Структурна схема серверного програмного забезпечення.	1	

				ДП 6404 00.001		
	ПІБ	Підп.	Дата			
Розробн.	Гордієнко Н.Ю.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Роковий О.П.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІП-64	
Н/контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

Національний технічний університет України
«Київський політехнічний інститут»

Інститут (факультет) інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»
(повна назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Сергій, СТІРЕНКО
"___" _____ 2020 року

ЗАВДАННЯ
на дипломний проєкт студента

_____Нікіти ГОРДІЄНКА_____
(ім'я, прізвище)

1. Тема проєкту *«Метод інтеграції та обробки мультимодальних даних»*

керівник проєкту _____*к.т.н, доцент. Олександр РОКОВИЙ*_____,
(ім'я, прізвище, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___" _____ 20__ року N ___

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту __технічна документація, теоретичні дані, інтернет-публікації за темою роботи_____

4. Зміст пояснювальної записки

− Провести огляд методів та засобів для збору даних ЕЕГ;
− Провести аналіз засобів для розробки програмного забезпечення;
− Розробити програмну реалізацію методу адаптивної реконфігурації;
− Провести моделювання та аналіз розробленого методу;

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	*к.т.н, доцент Роковий О. П.*	*10.03.2020*	*31.05.2020*
2	*к.т.н, доцент Роковий О. П.*	*10.03.2020*	*31.05.2020*
3	*к.т.н, доцент Роковий О. П.*	*10.03.2020*	*31.05.2020*
4	*к.т.н, доцент Роковий О. П.*	*10.03.2020*	*31.05.2020*

7. Дата видачі завдання *15.12.2019*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*15.12.2019*	
2	*Вивчення та аналіз завдання*	*20.12.2019-02.02.2020*	
3	*Проведення огляду засобів та методів збору даних ЕЕГ*	*02.02.2020-01.03.2020*	
4	*Проведення аналізу засобів для розробки програмного забезпечення*	*01.03.2020-15.04.2020*	
5	*Розробка програмної реалізацію методів для збору мультимодальних даних*	*15.04.2020-13.05.2020*	
6	*Проведення моделювання та аналізу розробленої системи*	*13.05.2020-19.05.2020*	
7	*Оформлення матеріалів роботи*	*19.05.2020-31.05.2020*	
8	*Передзахист*		
9	*Захист*		

Студент _____ Гордієнко Н. Ю.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Роковий О. П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У роботі був розроблений прототип системи для збору мультимодальних даних з багатьох сенсорів та декількох джерел. Останнім часом швидко зростає потреба у дослідженнях фізичної та розумової активності людини та проведення миттєвої діагностики стан здоров'я на дому чи у польових умовах. Тому виникає необхідність у мобільній системі збору біометричних даних від різноманітних сенсорів носимої електроніки та отримання доступу до них у будь-який час. Запропонована система на прикладі інтеграції даних електроенцефалографії (ЕЕГ) дозволяє знімати такі мультимодальні ЕЕГ-дані з нейрокомп'ютерного інтерфейсу, контролювати, автоматично робити помітки на даних та зберігати їх на сервері. Таким чином ЕЕГ-дані можуть бути використані пізніше для подальшого аналізу, що може значно полегшити роботу дослідникам та спеціалістам у сфері охорони здоров’я. У порівнянні із наявними системами основними перевагами розробленої системи є її відкритість, мобільність, швидкість збору, передачі та безпечність використання персональних медичних даних.

ABSTRACT

A prototype system for collecting multimodal data from many sensors and several sources was developed. Recently, there is a growing need for research into physical and mental activity and instant diagnosis of health at home or in the field. Therefore, there is a need for a mobile system for collecting biometric data from a variety of wearable electronics sensors and accessing them at any time. The proposed system on the example of electroencephalography (EEG) data integration allows you to capture such multimodal EEG data from the neurocomputer interface, monitor, automatically make notes on the data and store them on the server. Thus, EEG data can be used later for further analysis, which can greatly facilitate the work of researchers and health professionals. Compared to existing systems, the main advantages of the developed system are its openness, mobility, speed of collection, transmission and security of personal medical data.

Технічне завдання до дипломного проекту
ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку методу інтеграції та обробки мультимодальних даних.

Область застосування: збір та аналіз мультимодальних даних, наприклад, для дослідження активності мозку, діагностика на основі даних електроенцефалографії (ЕЕГ).

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання розробки методу інтеграції та обробки мультимодальних даних, затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка методу інтеграції та обробки мультимодальних даних.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, довідники з програмованих логічних інтегральних схем, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Розробка інтерфейсу для користувача на основі мобільної платформи Android для контролю запису даних.
- Розробка серверного програмного забезпечення для збереження та надання доступу до даних.
- Виконання тестових вимірі за допомогою системи.

5.2. Вимоги до програмного забезпечення для мобільного додатку:

- Операційна система Android

5.3. Вимоги до програмного забезпечення для серверного програмного забезпечення:

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10
- Java EE 7 і вище

5.4. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 2 і вище
- Оперативної пам’яті не менше 512 Мбайт

Пояснювальна записка
до дипломного проєкту
на тему: «Метод інтеграції та обробки мультимодальних даних»

Київ – 2020 року

ЗМІСТ

ВСТУП

Сьогодні багато дослідників присвячують свою роботу дослідженню мозку задля використання альфа хвиль, які виробляються мозком під час розумової активності, у практичних цілях. Деякі наукові роботи присвячені вивченню розумової активності під час роботи задля визначення втрати уваги та випередження помилок, активності мозку під час вивчення мов програмування, аутентифікації за допомогою унікальності активності мозку. Наукові дослідження з активності мозку потребують обробки великої кількості різноманітних даних з різних видів сенсорів (таких як ЕЕГ, тонометр, термометр тощо), які відповідають різним типам діяльності (таких як зір, звук, дотик, запах, смак, тощо) на різних часових і просторових масштабних рівнях і практично у поточному часі. Тобто існує проблема збору даних з різних видів сенсорів та збереження даних для подальшого доступу. Комплексні системи зазвичай доступні тільки професіоналам чи в них відсутня мобільність, яка стає вирішальним фактором у світлі сучасних подій.

Актуальність теми дипломної роботи полягає в необхідності розробки і створення засобів реєстрації, обробки, аналізу та інтерпретації усієї сукупності таких мультимодальних даних, тобто методів та засобів інтеграції мультисенсорних (multisensory) або мультимодальних (multimodal) даних, які будуть доступними, легкими для використання та мобільними для проведення вимірювань майже у будь-яких умовах.

Практичним значенням побудованої системи є забезпечення способу збору та доступу до таких даних, що полегшить хід дослідження та дозволить накопичувати дані з різних частин світу. Таким чином дослідники зможуть ділитися зібраними даними та проводити більш масштабні дослідження, а спеціалісти отримають можливість вивчати та збирати дані пацієнтів у будь-якому місці.

Об'єктом даного дослідження є методи збору та обробки мультимодальних даних з різних сенсорів. Предметом даної роботи є

створення системи для полегшення збору таких даних та доступу для подальших досліджень і можливі шляхи вдосконалення такої системи.

Метою роботи є систематизація знань про розробку складних систем на основі мобільного додатку та хмарного сховища та створення пропозицій щодо вирішення поставленої задачі.

Задачі, що поставлення для досягнення мети даної роботи:

- Розглянути основні практики, що застосовуються для створення складних систем на основі мобільних додатків та хмарних сховищ.
- Обрати найкращі підходи для побудови системи збору даних з різних видів сенсорів.
- Створити систему збору і обробки мультимодальних даних.
- Проаналізувати продуктивність та захищеність розробленої системи та знайти шляхи щодо покращення та оптимізації представленого рішення.

РОЗДІЛ 1
ОГЛЯД ТЕОРЕТИЧНИХ ЗАСАД ЕЛЕКТРОЕНЦЕФАЛОГРАФІЇ ТА ТОКУ, ЩО ВИРОБЛЯЄТЬСЯ МОЗКОМ

1.1. Коротка інформація про електроенцефалографія

Електроенцефалографія (ЕЕГ) - метод електрофізіологічного моніторингу для реєстрації електричної активності мозку. Він, як правило, неінвазивний, тобто такий що не проникає в організм досліджуваної особи, з електродами, розміщеними вздовж шкіри голови, хоча інвазивні електроди іноді використовуються, як в електрокортикографії. ЕЕГ вимірює коливання напруги, що виникають внаслідок іонного струму в нейронах мозку. [1]

Цей метод є один з основних діагностичних тестів на епілепсію. При клінічних процедурах запис ЕЕГ зазвичай триває близько 30 хвилин, також використовується додатковий час для підготовки досліджуваної особи. Зазвичай ЕЕГ застосовується в клінічних обставинах для визначення змін у мозковій діяльності, які можуть бути корисними для діагностики порушень мозку, особливо епілепсії або іншого порушення припадків. ЕЕГ також може бути корисним для діагностики або лікування таких порушень [2]:

- Пухлина мозку
- Пошкодження мозку від травми голови
- Дисфункція мозку, яка може мати різні причини (енцефалопатія)
- Запалення мозку (енцефаліт)
- Інсульт
- Розлади сну

Також ЕЕГ починає використовуватися у різних видах застосунку для вимірювання рівня активності мозку у домашніх умовах і виконуються дослідження для використання даних зібраних під час ЕЕГ, як методу управління зовнішніми пристроями.

1.2. Процес вироблення токів мозком

Електричний заряд мозку підтримується мільярдами нейронів. Нейрони електрично заряджаються (або «поляризуються») мембранними

транспортними білками, які перекачують іони через їх мембрани. Нейрони постійно обмінюються іонами з позаклітинною середовищем, наприклад, для підтримки потенціалу спокою та для поширення потенціалу дії. Іони подібного заряду відштовхують один одного, і коли з багатьох нейронів одночасно виштовхується багато іонів, вони можуть хвилею штовхати своїх сусідів, які штовхають своїх сусідів тощо. Цей процес називається об'ємною провідністю. Коли хвиля іонів досягає електродів на шкірі голови, вони можуть виштовхувати або витягувати електрони на металі в електродах. Оскільки метал легко проводить ток, тобто забирає чи віддає електрони, різницю напруг між будь-якими двома електродами можна виміряти вольтметром. Запис цих напруг у часі з різних електродів встановлених по тілу і створює результат ЕЕГ. [3]

Електричний потенціал, що генерується окремим нейроном, занадто малий, щоб його можна було зафіксувати. Тому ЕЕГ-активність завжди відображає підсумок синхронної активності тисяч або мільйонів нейронів, які мають схожу просторову орієнтацію. Якщо клітини не мають подібної просторової орієнтації, їхні іони не вирівнюються і не створюють хвилі, які можуть бути виявлені електродами. Вважається, що пірамідальні нейрони кори створюють найбільш значну, а отже помітну, активність, яку можна зафіксувати за допомогою ЕЕГ. Головним поясненням такої особливості є розташування нейронів і їх одночасна та кооперативна робота. Оскільки градієнти поля напруги падають із квадратом відстані, активність від глибинних джерел важче виявити, ніж струми біля черепа. [4]

ЕЕГ активності шкіри голови показує коливання на різних частотах. Деякі з цих коливань мають характерні діапазони частот, просторовий розподіл і пов'язані з різними станами функціонування мозку (наприклад, пробудженням та різними стадіями сну). Ці коливання представляють синхронізовану активність по мережі нейронів. Нейронні мережі, що лежать в основі деяких з цих коливань є вже гарно досліджені і є зрозумілими для

лікарів та спеціалістів. З іншої сторони, природа багатьох інших не є зрозумілою і досі вивчається.

1.3. Основні діапазони частот та їх властивості

Наші мозкові хвилі змінюються відповідно до того, що ми робимо та відчуваємо. Коли домінуючі повільні мозкові хвилі, ми можемо відчувати втому, повільність, млявість або мрійливість. Більш високі частоти є домінуючими, коли ми відчуваємо напруженість чи тривогу.

Мозкові процеси є складним явищем, але умовно можна розділити на певні групи, які розглянуто детально у таблиці 1.1 (рис. 1.1 - 1.5). Швидкість мозкового хвилі вимірюється в герцах (цикли в секунду), і вони діляться на групи, що окреслюють певні типи хвиль, наприклад такі як повільні, помірні та швидкі хвилі. [5][6]

Таблиця 1.1.

Основні типи мозкових хвиль

Діапазон	**Частота**	**Фізіологічні властивості**
Дельта	до 4 Гц	Коливання амплітудою 20-30 мкВ можуть зустрічатися у ЕЕГ здорової притомної людини. Коливання більш високої амплітуди (40-300мкВ) у ЕЕГ притомної людини є патологічною ознакою, що може вказувати на різні види захворювань, такі як мозкові пухлини. Дельта-коливання стають вираженими під час певних фаз природного сну, наркотичного сну або у стані коми.
Тета	4 - 7 Гц	Коливання амплітудою до 40 мкВ можуть зустрічатися у ЕЕГ здорової притомної людини, але зростання їх частки є ознакою емоційної активації та інших типів мозкової активності. Наявність тета-коливань у більших кількостях пов'язана із патологічними станами або ж зміненими станами свідомості (сон, медитація та ін.).

Таблиця 1.1 (продовження)

Діапазон	Частота	Фізіологічні властивості
Альфа	8 – 13 Гц	Синусоїдальні коливання амплітудою до 100 мкВ у лобно-потиличному напрямку - найбільш виражені у ЕЕГ здорової притомної людини із закритими очима. У формі вираженого ритму реєструється у 80-90 % людей, пригнічується при відкриванні очей. Це пояснюється тим що активність мозку змінюється і переходить до активної діяльності, наприклад, аналізу інформації.
Бета	13 – 40 Гц	Коливання амплітудою 5-30 мкВ, які пов'язані з активним функціональним станом мозку. Зростання рівня активації головного мозку здебільшого супроводжується зменшенням частки альфа-коливань і зростанням частки бета-коливань. Наявність вираженого бета-ритму з амплітудою вище 40 мкВ є патологічною ознакою, що може свідчити про певні захворювання.
Гамма	вище 30 - 40 Гц	Коливання амплітудою до 10 мкВ - ознака когнітивних процесів і свідомості. Наявність такого виду коливань вище 15 мкВ є патологічною ознакою.

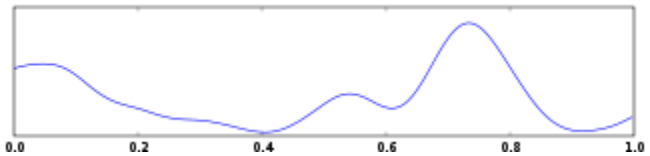


Рис. 1.1. Приклад візуалізації дельта хвиль

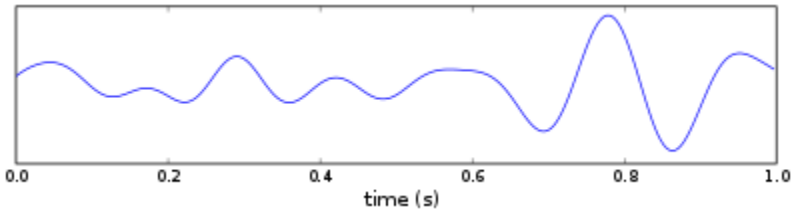


Рис. 1.2. Приклад візуалізації тета хвиль

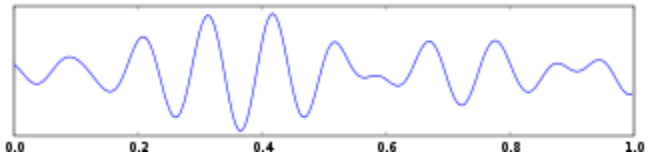


Рис. 1.3. Приклад візуалізації альфа хвиль

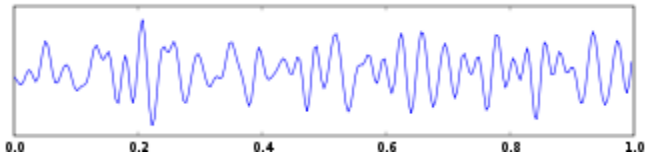


Рис. 1.4. Приклад візуалізації бета хвиль

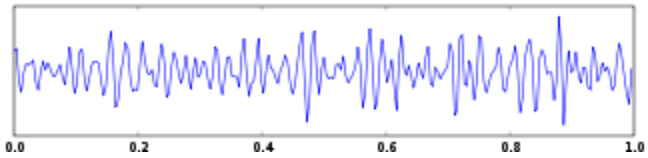


Рис. 1.5. Приклад візуалізації гама хвиль

1.4. Огляд технічних засобів для вимірювання

Новітні технології дозволяють створити апарати ЕЕГ різної форми, які мають різну точність та ціну для різного призначення. Дешеві апарати частіше використовуються в експериментах та для створення різних видів ігор та контролерів. Апарати середньої цінової категорії зазвичай мають більшу кількість каналів з більшою частотою. Професійни апарати мають найбільшу ціну та якість і зазвичай використовуються у медицині та професійних дослідженнях. Також варто відзначити форму різних апаратів. Професійні апарати зазвичай встановлюються стаціонарно в кілніках та лікарнях, а дешевші прилади є більш мобільними та менш точними.

Варто розглянути пристрої, які доступні для широкої аудиторії та які використовуються у дослідженнях у різних інноваційних сферах, таких як інформаційні технології тощо. Для деяких з пристроїв можна знайти статті огляди, що допоможе краще оцінити ринок і обрати найбільш підходящий пристрій для даного використання у мобільних застосунках.

Одним з пристроїв є MindFlex [7]. Це пристрій на основі ЕЕГ, основна задача якого є тренування мозку на основі технології ThinkGear,

розробленої NeuroSky. Пристрій містить блок керування та бездротову гарнітуру ЕЕГ (рис. 1.6). Ця гарнітура MindFlex ЕЕГ може бути закріплена на голові завдяки гумовій конструкції. Його енергозабезпечення забезпечується трьома 1,5 В батарейками АА. Мозкові сигнали виявляються металевим електродом, звуженим до чола монополярним методом, а нульовою точкою є електрод, затиснутий на мочці вуха. Блок обробки сигналів, також розроблений за допомогою технології ThinkGear NeuroSky за допомогою спеціального аналізу активності мозку, може визначити значення концентрації або уваги. Гарнітура NeuroSky MindFlex ЕЕГ передає оброблені сигнали до керованого блоку через бездротову мережу. На основі MindFlex створені іграшки, наприклад MindFlex Duel, для широкого ринку, головною ціллю яких є збільшення активності мозку за допомогою думок чи емоції, щоб перемогти суперника. [8]

Рис. 1.6. Пристрій ЕЕГ MindFlex з блоком керування та бездротовим зв'язком на 1 канал

Є й інші аналоги, що були використані у інших дослідженнях [9]. Наприклад, у одній з розглянутих робіт використовується доступна мобільна система, яка дозволяє використовувати одночасно від 8 до 16 каналів. Це система Ultracortex Mark 4 з розширенням Cyton Board та Daisy від OpenBCI (рис. 1.7.). [10] [11]

Це дає можливість робити заміри використовуючи або 8-каналів з частотою 250 ГГц ЕЕГ-дані, або 16-каналів з частотою 125 ГГц. Головними

перевагами цієї системи є її низька вартість, можливість виконувати заміри з 16 каналів, точність та простота виконання замірів. Використаний у цьому пристрої протокол передачі даних Bluetooth дає можливість збирати дані дистанційно, не використовуючи додаткових проводів. Це робить систему мобільною і дає можливість користувачеві вільно ходити та виконувати рухи під час дослідження. Ще однією перевагою є використання сухих електродів, які скорочують час налаштування до декількох хвилин. Через це погіршується якість даних через високу чутливість до артефактів руху та більш високий опір. Більш детальну інформацію про порівняння між сухими та вологими електродами можна знайти в кількох опублікованих дослідженнях. [12]

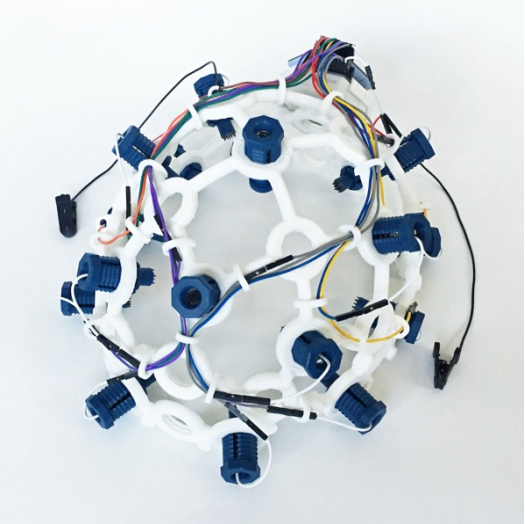


Рис. 1.7. Пристрій ЕЕГ Ultracortex Mark 4 з блоком керування Cyton Board та бездротовим зв'язком на 8-16 каналів

1.5. Методи аналізу

Існують різні методи аналізу отриманих з ЕЕГ даних. У цьому розділі будуть розглянуті різні види від самих простих, які використовувалися раніше і вже є застарівшими, до використання новітніх технологій для аналізу даних, таких як машинне навчання.

1.5.1. Мультимодальні дані

Злиття датчиків - це поєднання сенсорних даних або даних, отриманих з різних джерел, таким чином, що отримана інформація має меншу невизначеність, ніж це було б можливо, коли ці джерела використовувалися окремо. Дані зібрані з декількох різних джерел можна визначити як мультимодальні, так як вони мають різну природу і доповнюють один одного. Термін зменшення невизначеності в цьому випадку може означати більш точне, більш повне або більш надійне, або посилатися на результат виду, що виникає, наприклад, стереоскопічне бачення (обчислення інформації про глибину за допомогою комбінування двовимірних зображень з двох камер при трохи різних точки огляду). [13]

Джерела даних для процесу злиття не мають надходити від однакових датчиків. Можна виділити прямий синтез, непрямий синтез і синтез виходів попередніх двох. Пряме синтез - це злиття даних сенсорів із набору гетерогенних або однорідних датчиків, м'яких датчиків та значень історії сенсорних даних, тоді як непряме синтез використовує такі джерела інформації, як апріорні знання про навколишнє середовище та дані введені людиною. Злиття датчиків також відоме як (мультисенсорне) синтез даних і є підмножиною синтезу інформації.

На прикладі ЕЕГ різними датчиками можуть бути годинники, електрокардіограма серця, тонометр тощо. Тоді мультимодальними даними будуть дані про активність мозку, час вимірювань, дані про серцебиття, артеріальний тиск, а також введені користувачем дані про вид активності. Прикладами видів активності у цьому випадку можуть бути напруження

м’язів, підсилення мозкової активності шляхом уявлення певних дій чи виконання підрахунків, різні види емоцій тощо.

1.5.2. Частотний аналіз

Сигнали записані з поверхневих електродів ЕЕГ шкіри голови, можуть бути представлені у часовому вимірі або з точки зору їхнього розкладання на синуси та косинуси в частотній вимірі. Мета частотного виміру полягає в опису сигналу за допомогою розкладання на синусоїди різної частоти. Тобто будь-який сигнал можна розглянути як суперпозицію трьох синусоїд різної частоти.

Перетворення Фур'є (FT) розкладає функцію (часто функцію часу або сигнал) на складові частоти. Існує чотири різних типи перетворень Фур'є, залежно від того, чи є сигнал безперервним чи дискретним, і від того, періодичний він чи ні. Виведення цих чотирьох перетворень можна знайти в підручниках з математики. [14]

Основна формула перетворення Фур'є має вигляд:

$$\boxed{F(w) = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt} \quad (1.1)$$

1.5.3. Часово-частотний аналіз

Одним з головних обмежень перетворення Фур'є є те, що воно не використовує час як характеристику зібраних даних. Для обчислення перетворення Фур'є вважається, що сигнал є нерухомим і, отже, активність на різних частотах є постійною протягом усього сигналу. Однак у багатьох випадках сигнали мають різні функції, які неможливо визначити за допомогою перетворення Фур'є. Це стосується музики, мови, звуків тварин, радіолокаційних даних та багатьох інших сигналів. Для сигналів ЕЕГ це обмеження є критичним при аналізі процесів, що змінюються в часі, такі як реакція на певний подразник або розвиток епілептичного припадку.

Можливо подолати нестачу часової роздільної здатності перетворення Фур'є, подрібнюючи дані на частини, а потім обчислюючи спектр потужності для кожної деталі або, ще краще, використовуючи вікно, що

розвивається в часі, для фокусування на різних сегментах даних. Ця процедура називається короткочасним перетворенням Фур'є (STFT) або віконним перетворенням Фур'є.

Важливим є вибір розміру вікна. Якщо вікно занадто вузьке, воно дасть хорошу часову роздільну здатність, але частоти не будуть добре підраховані. Навпаки, якщо вікно занадто велике, проаналізовані дані будуть мати хорошу частотну роздільну здатність, але локалізація часу буде втрачена. Існує компроміс між частотою та часовою роздільною здатністю. Аналогічно принципу невизначеності Гейзенберга в квантовій механіці, це називається принципом невизначеності аналізу сигналів: частота і час розв’язання не можуть бути зроблені довільно малими одночасно. Іншими словами, різка локалізація у часі та частоті взаємовиключна, оскільки для визначення частоти потрібно кілька точок даних.

Для кількісного визначення розподілу частоти в даний момент часу і, особливо, щоб побачити його еволюцію, можна обчислити ентропію спектру потужності. Ентропія - це міра випадковості або, іншими словами, інформаційного вмісту сигналу. Випадкові сигнали непередбачувані, і кожна нова точка даних дає нову інформацію. Навпаки, за упорядкованими сигналами нові точки даних можна передбачити з попередніх значень і, отже, несуть менше інформації. [14]

1.5.4. Вейвлети

Вейвлет - це математична функція, що дозволяє аналізувати різні частотні компоненти даних (рис. 1.8). Графік функції виглядає як хвилеподібні коливання з амплітудою, що зменшується до нуля далеко від початку координат. У загальному випадку аналіз сигналів проводиться в площині вейвлет-коефіцієнтів (масштаб - час - рівень) (Scale-Time-Amplitude). Вейвлет-коефіцієнти визначаються інтегральним перетворенням сигналу. Отримані вейвлет-спектрограми принципово відрізняються від звичайних спектрів Фур'є тим, що дають чітку прив'язку спектра різних особливостей сигналів на часі. Вейвлет-перетворення - це перетворення, що

розглядають функцію (взяту як функцією від часу) у термінах коливань, локалізованих за часом і частотою. Локальність у просторі означає, що енергія вейвлетів сконцентрована на скінченному інтервалі. Частотна локалізація означає, що перетворення Фур'є хвилі локалізоване. [15]

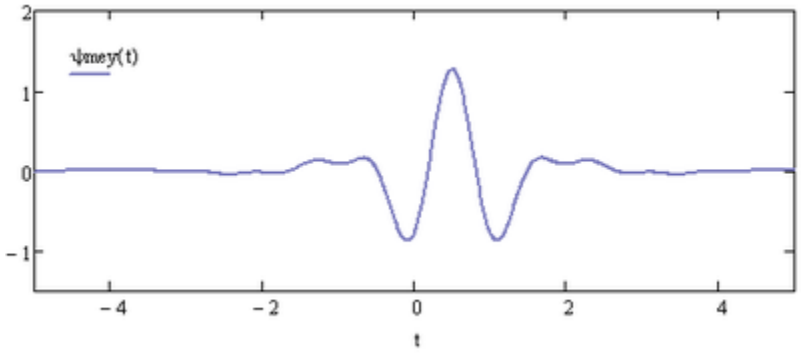


Рис. 1.8. Приклад візуалізації вейвлета Мейера

Цей метод є покращенням попереднього. Було помічено, що хороший компроміс між роздільною здатністю часу та частоти неможливий, оскільки високочастотні моделі мають меншу тривалість порівняно з низькочастотними. Отже, єдина величина для вікна для всіх частот не приносить гарних результатів. Тоді було запропоновано брати різні розміри вікон для різних частот, або точніше, брати косинусну функцію і стиснути або розтягнути її для отримання різних частот. Як результат, розмір вікна наслідує форму форму хвилі у різних масштабах, тобто має змінний розмір.

Ключова ідея вейвлетів - приймати різні розміри вікон для різних частот, що робиться шляхом стиснення або розтягування тієї ж функції, що називається материнською вейвлетом. Функція, яка використовується, не обов'язково повинна бути синусоїдою. Це дає другу перевагу: є словник вейвлет-функцій, який можна вибрати, залежно від їх властивостей та застосування.

1.5.5. Методи штучного інтелекту

Більшість сучасних ЕЕГ мозку засновані або використовують для аналіза даних алгоритми машинного навчання. Існує велика різноманітність типів класифікаторів, які використовуються в цій галузі. За роки існування цієї

сфери було розроблено багато різних технік та підходів до опрацювання даних, які розглянуті у різних матеріалах. [16]

В останні роки методи та техніки машинного навчання, які відомі як поглиблене навчання (Deep Learning) [17], набирають популярності та знаходить свій застосунок у різних сферах, таких як аналіз даних, машинне бачення, тощо.

Тому зараз проводяться дослідження, які використовують техніки поглибленого навчання для аналізу даних зібраних з ЕЕГ для їх подальшої класифікації та визначення потенційних хвороб. Ці техніки працюють краще на даних з великим обсягом інформації ніж інші методи машинного навчання і тому мають гарні перспективи для застосунків у майбутньому. До того ці техніки комп'ютерного аналізу даних мають потенціал і можуть замінити людський аналіз ЕЕГ даних. [18]

1.6. Використання у різних сферах

Варто розглянути дослідження у яких використовуються мозкові хвилі у різних сферах життя. Нижче будуть наведені деякі з цих досліджень. у яких використовуються як і професійні ЕЕГ, так і більш доступні аналоги.

Одне з досліджень вивчає процес засвоєння мови програмування та зручності її використання [19]. За думкою авторів вивчення процесу засвоєння є важливим для розуміння притаманної складності мов програмування. Це може допомогти зрозуміти міри розумових зусиль, заснованих на безпосередньому спостереженні за мозком під час роботи, та висвітлити характер роботи програмування. У своїй роботі дослідники збирали емпіричні дані з використанням перерізу студентів магістерської програми з інформатики та недорогого 14 канального пристрою від компанії Epoc з частотою 128 Гц для замірів активності мозку. У цьому дослідженні представлений зв’язок між знаннями та розумінням мови програмування, робиться висновки щодо спостережуваних показників когнітивних зусиль із використанням останніх когнітивних теорій та пропонуються вказівки щодо подальшої роботи, яка зараз можлива.

У цій роботі автори вивчали можливість визначення трьох основних емоцій у людей (задоволення, збудження та домінування) на основі даних мозкових хвиль основі даних з ЕКГ. Для аналізу вони використовували методи машинного навчання. У цьому дослідженні було застосовано експеримент, під час якого учасники опромінювали набір знімків з Міжнародної системи афективних зображень (International Affective Picture System IAPS), в той час як їх електрична мозкова активність реєструвалася за допомогою ЕЕГ.

У цій статті пропонується використовувати комп'ютерний інтерфейс, заснований на технологіях електроенцефалограми, як новий підхід для аутентифікації особи [20]. Автори розробили додаток для блокування екрана, який блокує та розблоковує екран комп'ютера за бажанням користувачів. Мозкові хвилі людини, записані в режимі реального часу, використовуються як пароль для розблокування екрана. У роботі дані збираються з 14 сенсорів гарнітури Emotiv.

У цій статті описано розробку та тестування системи інтерфейсів, за допомогою якої можна керувати зовнішніми пристроями, керуючи альфа-хвилями за допомогою руху очей [21]. Як вбачають дослідники, така система може використовуватися для контролю протезування, роботизованих озброєнь та зовнішніх пристроїв, таких як інвалідні коляски, за допомогою альфа-мозкових хвиль та ритму Mu. Відповідь, що генерується внаслідок руху очей, у формі мозкових хвиль збиралася та оброблялася для управління робототехнічними системами та управлінням розумного будинку.

ВИСНОВКИ ДО РОЗДІЛУ 1

Сьогодні ЕЕГ є досить поширеним тестом, який дозволяє фіксувати активність мозку людини під час різних видів активності, у медицині та інших сферах. Науковці активно проводять дослідження мозкових хвиль задля пошуку їх застосування у різних сферах: від освіти та медицини до інформаційних технологій. Для проведення досліджень використовуються пристрої різної цінової категорії, з різними кількостями каналів та різними частотами. Активність мозку є досить досліджуваною темою, яка знаходить застосування у різних сферах діяльності.

Варто відзначити особливості даних зібраних під час вимірів за допомогою ЕЕГ:

- дані мають мультимодальну структуру. Оскільки одночасно з ЕЕГ виконуються заміри часу кожного з вимірювань, створюється опис активності людини під час вимірів, а іноді дані можуть бути доповнені даними з інших сенсорів. Через це варто звертати увагу при їх збережені на зв'язки між різними видами даних.
- дані мають великий обсяг. Оскільки дані ЕЕГ зазвичай збираються з декількох каналів одночасно з великою частотою, то дослідження за допомогою ЕЕГ вимагає аналізу та збереження великого обсягу даних. Наприклад, один з розглянутих мобільних пристроїв давав змогу збирати дані з 8 каналів одночасно з частотою 250 Гц. Для ефективного збереження, збору і аналізу даних варто продумати систему, яка буде використовувати новітні технології та техніки.
- діяльність мозку має нелінійний характер взаємодії даних різних частин мозку, тому є складним предметом для аналізу. Через це аналітичні методи аналізу за допомогою спостережень та досвіду спеціалістів у багатьох випадках не є корисним. До того ж, досі деякі зв'язки, природа і особливості мозкової активності досі не вивчені і досі вивчаються. Саме

тому починають використовуватися чисельні методи аналізу даних, такі як машинне та поглиблене навчання.

Через це треба шукати інші підходи на основі найбільш перспективних сучасних методів та технологій для збору, збереження та обробки мультимодальних даних. Система на основі цих технологій може полегшити процес дослідження та збору великого об'єму даних.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ ДЛЯ РОЗРОБКИ СИСТЕМИ ЗБОРУ ТА ОБРОБКИ МУЛЬТИМОДАЛЬНИХ ДАНИХ

В більшості практичних випадків обробка мультимодальних даних мозку мусить відбуватися в поточному часі і режимі активного руху людини. Тому надзвичайно важливим є можливість застосування програмних інструментів збору та обробки даних із засобів носимої електроніки (wearable electronics), які не заважають активним рухам людини і роблять це в режимі поточного часу (real-time). С точки зору обробки та моніторингу таких даних надзвичайно важливим є моментальне відтворення зібраних даних на інших мобільних гаджетах, наприклад таких мобільних пристроях, як планшет, смартфон, смарт-годинник тощо. Розглянемо деякі програмні засоби (операційні системи, мови програмування, середовища розробки, бібліотеки, тощо) для створення організації такого режиму роботи.

2.1. Розробка мобільних застосунків для платформи Android

Android — операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux.

Хоча Android базується на ядрі Linux, він відділився від Linux-спільноти та Linux-інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik віртуальної машини Java, і все програмне забезпечення і застосування спираються на цю реалізацію Java. [22]

2.1.1. Android розробка мовою Java

Java - об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems у 1995, яка зараз належить Oracle. Java знайшла своє застосування у багатьох сферах, також і у розробці додатків для платформи Android.

Кількість розробників Java значно варіюється з року в рік. У 2003 році кількість розробників мовою Java склала від 1,5 до 3 мільйонів; У 2007 році,

коли Java стала мовою програмування з відкритим кодом, кількість розробників зросла до 6 мільйонів. Глобальне дослідження населення, проведене корпорацією Evans Data Corporation, повідомило, що кількість розробників Java у 2009 році становила 9 мільйонів, що робить Java однією з найбільш використовуваних мов програмування у всьому світі. До 2019 року, цей показник сягнув 7,6 мільйона. [23]

Віртуальна машина Java — набір комп'ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших комп'ютерних програм чи скриптів.

Java як і будь-яка інша мова програмування має свої плюси й мінуси, які варто розглянути [24]. Java має такі плюси як:

- це легка для вивчення та розуміння мова. Вона знайшла свій застосунок у різних сферах, тому за допомогою цієї мови можливо розробляти як і мобільні застосунки, так і серверне забезпечення.
- високу продуктивність як для нативних, так і для кросплатформних додатків.
- це мова на якій побудована платформа Android, тому існує великий спектр підтримуваних бібліотек. Також Java має широку екосистему з відкритим кодом.
- програми Java легші та компактніші навіть у порівнянні з програмами Kotlin, що призводить до швидшого використання додатків.
- Java також має швидший процес збирання (компілювання та лінкування) проєкту, дозволяючи вам кодувати більше за менший час.
- завдяки прискореній збірці з Gradle, збирання великих проєктів на Java стає простішим.

Але існують й мінуси Java:

- Java - мова, яка вимагає написання великої кількості коду, збільшуючи шанси помилок.

- у Java виникають деякі проблеми з дизайном API через існуючі обмеження.
- Java вимагає більше пам’яті в порівнянні з іншими мовами, що її сповільнює.

2.1.2. Android розробка мовою Kotlin

Kotlin (Котлін) — статично типізована мова програмування, що працює поверх JVM і розробляється компанією JetBrains. Автори ставили перед собою ціль створити лаконічнішу та безпечнішу для типів мову, ніж Java, і простішу, ніж Scala. Наслідками спрощення, порівняно з Scala стали також швидша компіляція та краща підтримка IDE. Мова розробляється з 2010 року, публічно представлена в липні 2011. З 17 травня 2017 року входить в список офіційно підтримуваних мов для розробки застосунків для платформи Android. З 7 травня 2019 року є рекомендованою мовою для розробки Android застосунків. [25]

Плюси Kotlin [24]:

- Kotlin має простіший синтаксис і отже більшу швидкість розробки. Це також означає, що помилок при розробці менше.
- Завдяки байт-коду Java, ви можете використовувати бібліотеки Java в Kotlin, роблячи перехід від Java до Kotlin менш проблемним.
- Котлін має null у своїй системі типів, якого за думкою користувачів не вистачало у Java. Android використовує null для відображення відсутності значення, а Kotlin дозволяє використовувати null, що значно полегшує цю больову точку.

Мінуси Kotlin:

- Його сильно виражений синтаксис, хоча і є великою перевагою, вимагає певного навчання заздалегідь.
- У більшості випадків Kotlin демонструє більш низьку швидкість компіляції, ніж Java, навіть незважаючи на те, що він перемагає Java в декількох випадках.

- Спільнота Kotlin ще молода, а навчальні ресурси обмежені, тому знайти відповіді на проблеми може бути складніше. Однак із зростанням популярності ресурси та громада з кожним днем розширюються.
- Деякі функції Android Studio, такі як автоматичне доповнення та компіляція, як правило, працюють у Kotlin повільніше порівняно з Java.

2.1.3. Обрана мова для розробки

Для розробки системи було обрано мову Java через її широкий застосунок у різних сферах та велику спільноту, яка може допомогти вирішити проблеми під час розробки. Використовуючи великий спектр існуючих бібліотек легко створити мобільний додаток для системи. Також мова Java буде використана для написання програмного забезпечення серверного сервісу.

2.2. Розробка серверного програмного забезпечення мовою JAVA за допомогою Spring

2.2.1. REST архітектура

REST (від англ. Representational State Transfer - «передача стану представлення») - це архітектурний стиль програмного забезпечення, який визначає набір обмежень та особливостей, які будуть використані для створення веб-служби [26]. Веб-сервіси, які відповідають архітектурному стилю REST, називаються послугами RESTful Web, забезпечують сумісність між комп'ютерними системами в Інтернеті. Веб-сервіси RESTful дозволяють системам, що виконують запити, отримувати доступ та маніпулювати текстовими поданнями веб-ресурсів за допомогою рівномірного та заздалегідь визначеного набору операцій, що не зберігають стану. Інші види веб-служб, такі як веб-сервіси SOAP, використовують власні довільні набори операцій.

"Веб-ресурси" вперше були визначені у Всесвітній павутині як документи або файли, визначені за їх URL-адресами. Однак сьогодні вони мають набагато більш загальне та абстрактне визначення, яке охоплює кожну річ, сутність чи дію, які можна будь-яким чином ідентифікувати,

назвати, вирішити, обробляти чи виконувати будь-якими способами. У веб-службі RESTful запити, що надсилаються до URI ресурсу, викликають механізми обробки цього запиту і отримують відповідь з відформатованим корисним навантаженням (наприклад, за допомогою HTML, XML, JSON тощо). Відповідь може підтвердити успішність запиту, вказати на помилку чи передати дані, до яких виконувався запит зі сторони клієнта. Якщо використовується протокол HTTP доступними операціями є такі методи як GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS та TRACE.

Використовуючи протокол без стану та стандартні операції, системи RESTful мають на меті швидку продуктивність, надійність та можливість зростати за рахунок повторного використання компонентів, якими можна керувати та оновлювати, не впливаючи на систему в цілому, навіть під час її роботи.

П’ять існуючих основних керівних обмежень визначають систему REST. Ці обмеження створені для визначення способів, якими сервер може обробляти та відповідати на запити клієнтів, завдяки чому, діючи в рамках цих обмежень, система набуває бажаних нефункціональних властивостей, таких як продуктивність, масштабованість, простота, модифікованість, видимість, портативність та надійність. Якщо система порушує будь-який з необхідних обмежень, її не можна вважати RESTful.

Формальні обмеження REST такі:

- архітектура клієнт-сервер. Принцип, що стоїть перед обмеженнями клієнт-сервер - це розділення рішень з користувальницьким інтерфейсом від рішень щодо зберігання даних. Це покращує переносимість інтерфейсів користувача на інші платформи і поширює можливості веб сервісу. Він також покращує масштабованість, спрощуючи серверні компоненти. Мабуть, найбільш важливим для мережі є те, що розділення дозволяє компонентам розвиватися незалежно, підтримуючи тим самим вимогу Інтернет-масштабів у кількох організаційних областях.

- відсутність збереження стану. Зв'язок клієнт-сервер обмежується тим, що на сервері між запитами не зберігається контекст, або стан, клієнта. Кожен запит будь-якого клієнта містить всю інформацію, необхідну для обслуговування запиту, і стан сеансу зберігається у клієнта. Клієнт починає надсилати запити, коли готовий здійснити перехід до нового стану.
- кешованість. Як і у всесвітній мережі, клієнти та посередники можуть кешувати відповіді. Відповіді повинні, неявно або явно, визначати себе як кешовані або не кешовані, щоб запобігти клієнтам надавати несвіжі або невідповідні дані у відповідь на подальші запити. Добре керований механізм кешування частково або повністю виключає деякі взаємодії клієнт-сервер, ще більше покращуючи масштабованість та продуктивність тим самим зменшуючи навантаження серверного програмно забезпечення та мережі загалом.
- багатошаровість системи. Клієнт не спроможний визначити, підключений він безпосередньо до кінцевого сервера чи до посередника по шляху. Якщо проксі-сервер або балансир завантаження розміщений між клієнтом і сервером, це не вплине на їх зв’язок і не потрібно буде оновлювати код клієнта або сервера. Посередницькі сервери можуть покращити масштабованість системи, ввімкнувши балансування навантаження та надання загальних кешів. Також безпеку можна додати як шар поверх веб-служб, а потім чітко відокремити бізнес-логіку від логіки безпеки. Додавання безпеки як окремого шару дає можливість застосовувати політику безпеки для різних груп користувачів. Нарешті, це також означає, що сервер може викликати кілька інших серверів, щоб генерувати відповідь клієнту, тобто використовувати мікросервісну архітектуру.

2.2.2. Spring, як засіб розробки програмного забезпечення

2.2.2.1. Короткі відомості про Spring

Java Platform, Enterprise Edition, скорочено Java EE — обчислювальна корпоративна платформа Java. Платформа надає API та виконавче середовище для розробки і виконання корпоративного програмного забезпечення, включаючи мережеві та веб сервіси, та інші масштабовані, розподілені додатки. Java EE розширює стандартну платформу Java. J2EE є промисловою технологією і її використовують у високопродуктивних проєктах, у яких необхідна надійність, масштабованість, гнучкість. Компанія Oracle активно просуває Java EE у поєднанні зі своїми технологіями. [27]

Spring - це набір бібліотек та підходів, який полегшує створення серверного програмного забезпечення мовою Java [28]. Spring спрощує створення корпоративних програм Java. Як вказано у документації до Spring Framework 5.1, Spring потребує JDK 8+ (Java SE 8+) та забезпечує нестандартну підтримку JDK 11 LTS. Spring з'явився на світ у 2003 році як відповідь на складність ранніх специфікацій J2EE. Хоча деякі вважають, що Java EE та Spring є конкурентами, але Spring фактично доповнює Java EE. Модель програмування Spring не містить специфікації платформи Java EE та має наступні розширення:

- Servlet API (JSR 340)
- WebSocket API (JSR 356)
- Concurrency Utilities (JSR 236)
- JSON Binding API (JSR 367)
- Bean Validation (JSR 303)
- JPA (JSR 338)
- JMS (JSR 914)
- JTA / JCA для координації транзакцій.

Spring Framework також підтримує технічні характеристики ін'єкції залежностей (JSR 330) та загальні анотації (JSR 250), які розробники

додатків можуть вибрати замість специфічних для Spring механізмів, передбачених Spring Framework.

Ключовим елементів Spring Framework є Spring Container. Container створює об'єкти, пов'язує їх разом, налаштовує і управляє ними від створення до моменту знищення [29]. Для управління компонентами, з яких складається програма, Spring Container використовує Впровадження Залежностей (DI). Ці об'єкти називаються Spring Beans. Spring Container отримує інструкції які об'єкти створювати і як їх конфігурувати через метадані (рис. 2.1).

Метадані можуть бути отримані 3 способами:

- XML
- Анотації Java
- Java код

Далі будуть розглянуті основні розширення та методики, які полегшують процес розробки програмного забезпечення та роблять сервіс більш ефективним.

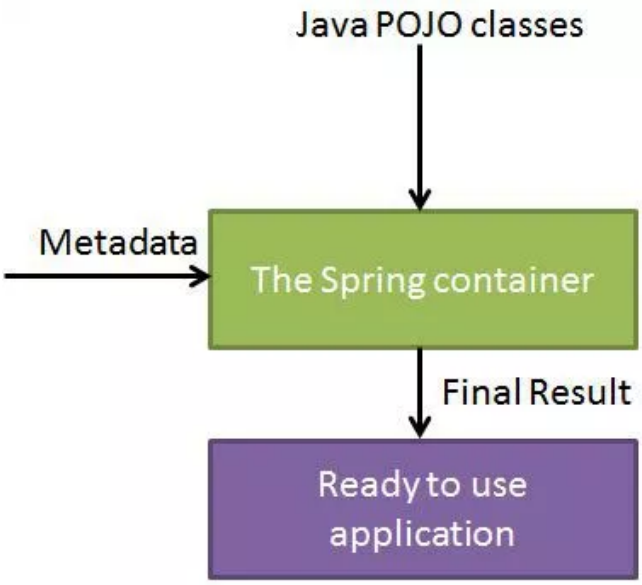


Рис. 2.1. Java POJO класи надходить в Spring Container, який на підставі інструкцій, отриманих через метадані, видає програму готову до використання.

2.2.2.2. JavaBeans

Біни - це об'єкти, які є основою програми та управляються Spring IoC контейнером [29]. Ці об'єкти створюються за допомогою конфігураційних метаданих, які вказуються в контейнері (наприклад, XML- <bean> ... </ bean>). Визначення біна містить метадані конфігурації, які необхідні керуючому контейнеру для отримання такої інформації:

- дії для створення та знищення біна;
- інформацію про життєвий цикл біна;
- залежності біна.

В Spring Framework існують такі властивості бінів:

- class - обов'язковий атрибут, що вказує на конкретний клас Java-додатка, який буде використовуватися для створення біна.
- name - унікальний ідентифікатор біна. Можливе використання властивостей "id" і/або "name" для зміни біна за допомогою файла налаштування.
- scope - це властивість, яка визначає зону видимості створюваних об'єктів.
- constructor-arg - атрибут, що визначає конструктор, який використовується для впровадження залежності.
- properties - визначає властивості впровадження залежності.
- initialization method - визначається метод ініціалізації біна
- destruction method - метод знищення біна, який буде використовуватися при знищенні контейнера, що містить бін.
- autowiring mode - визначає режим автоматичного зв'язування при впровадженні залежності.
- lazy-initialization mode - режим ледачою ініціалізації дає контейнеру IoC команду створювати екземпляр біна при першому запиті, а не під час запуску програми. Процес відкладання дає можливість розвантажити систему і зробити її більш ефективною.

2.2.2.3. Java Persistence API

JPA - це технологія, що забезпечує об'єктно-реляційне відображення простих JAVA об'єктів і надає API для збереження, отримання та управління такими об'єктами [30].

Сам JPA не вміє ні зберігати, ні управляти об'єктами, JPA тільки визначає основні механізми роботи: як щось буде діяти. JPA також визначає інтерфейси, які повинні будуть бути реалізовані провайдерами. Плюс до цього JPA визначає правила опису метаданих відображення і роботи провайдери. Далі, кожен провайдер, реалізуючи JPA визначає алгоритм дій для отримання, збереження і управління об'єктами. У кожного провайдера реалізація різна, тому розробник має можливість обрати провайдера на власний розсуд.

2.2.2.4. Java Transaction API

Java Transaction API (JTA) - це API, для підтримки транзакцій, який визначає взаємодію між менеджером транзакцій і іншими учасниками розподіленої транзакції системи [31]. Специфікація API розроблена в рамках Java Community Process як JSR 907.

Ця бібліотека забезпечує поділ кордонів транзакції, а також API, що описує взаємодію ресурсів в транзакціях. В архітектурі менеджер транзакцій або монітор обробки транзакцій координує операції до множинних ресурсів, таким як бази даних. У кожного ресурсу є свій власний менеджер. Менеджер ресурсу, як правило, має власний API для маніпулювання ресурсом. Крім того, менеджер ресурсу взаємодіє з монітором обробки транзакцій, для координації розподілених транзакцій між власним ресурсом і іншим ресурсами, а також взаємодіє з монітором обробки транзакцій для початку, передоручення або відкату транзакцій.

2.2.2.5. Сервлет

Сервлет є інтерфейсом Java, реалізація якого розширює функціональні можливості сервера [32]. Сервлет взаємодіє з клієнтами за допомогою принципу запит-відповідь. Хоча сервлети можуть обслуговувати будь-які

запити, вони зазвичай використовуються для розширення веб-серверів. Для таких додатків технологія Java Servlet визначає HTTP-специфічні сервлет класи. Пакети javax.servlet і javax.servlet.http забезпечують інтерфейси і класи для створення сервлетів.

Життєвий цикл сервлета складається з наступних кроків:

1. У разі відсутності сервлету в контейнері.

1.1. Клас сервлету завантажується контейнером.

1.2. Контейнер створює екземпляр класу сервлета.

1.3. Контейнер викликає метод init (). Цей метод створює сервлет і викликається в першу чергу, до того, як сервлет буде мати змогу обслуговувати запити. За весь життєвий цикл метод init() викликається тільки один раз.

2. Обслуговування клієнтського запиту. Кожен запит обробляється в своєму окремому потоці. Контейнер викликає метод service() для кожного запиту. Цей метод визначає тип запиту і розподіляє його в відповідний для цього типу метод обробки запиту. Розробник сервлету повинен надати реалізацію для цих методів. Якщо надійшов запит, метод для якого не реалізований, викликається метод батьківського класу і зазвичай завершується поверненням помилки ініціатору запиту.

3. У разі якщо контейнеру необхідно видалити сервлет, він викликає метод destroy(), який виводить сервлет з експлуатації знищуючи його. Подібно до методу init(), цей метод також викликається один раз за весь життєвий цикл сервлета.

2.3. Порівняння методів розробки баз даних

База даних - це система організованого збереження даних, які, як правило, зберігаються та доступні в електронному вигляді з комп'ютерної системи. Там, де бази даних складніші, вони часто розробляються з використанням формальних методів проектування та моделювання. [33]

Система управління базами даних (СУБД) - це програмне забезпечення, яке взаємодіє з кінцевими користувачами, програмами та самою базою

даних для збору та аналізу даних. Програмне забезпечення СУБД додатково охоплює основні засоби, надані для адміністрування бази даних. Загальна сума бази даних, СУБД та пов'язаних з ними програм може називатися "системою баз даних". Часто термін "база даних" також використовується для вільного посилання на будь-яку СУБД, систему бази даних або додаток, пов'язаний з базою даних.

За моделлю організації даних розрізняють такі бази даних:

- Ієрархічна. Ієрархічна база даних може бути представлена як дерево, що складається з об'єктів різних рівнів. Між об'єктами існують зв'язки типу «предок-нащадок». При цьому можлива ситуація, коли об'єкт не має нащадків або має їх декілька, тоді як у об'єкта-нащадка обов'язково тільки один предок.
- Мережна. Така база даних подібна до ієрархічної, за винятком того, що кожен об'єкт може мати більше одного предка.
- Реляційна. Реляційна база даних зберігає дані у вигляді таблиць. Найвживаніші СКБД використовують реляційну модель даних.
- Об'єктно-орієнтована. У базі даних цього виду дані оформляють у вигляді моделей об'єктів.

2.3.1. Реляційні бази даних

Реляційна база даних - це цифрова база даних, заснована на реляційній моделі даних, запропонована Е. Ф. Коддом у 1970 році [34]. Програмною системою, що використовується для підтримки реляційних баз даних, є система управління реляційними базами даних. Реляційна модель для управління базами даних - це підхід до управління даними за допомогою структури та мови, що відповідає логіці предикатів першого порядку, вперше описаному в 1969 році англійським вченим-комп’ютером Едгаром Ф. Коддом, де всі дані представлені у вигляді кортежів, згрупованих за допомогою відносин. База даних, організована з точки зору реляційної моделі, є реляційною базою даних.

Метою реляційної моделі є надання декларативного методу для визначення даних та запитів: користувачі безпосередньо заявляють, яку інформацію містить база даних та яку інформацію вони хочуть від неї отримати, і дозволяють програмному забезпеченню системи управління базами даних подбати про опис структур даних для зберігання, самі дані та механізми пошуку відповідей на запити.

2.3.2. Нормалізація схеми бази даних

Нормалізація схеми бази даних — це покроковий процес розбиття одного відношення відповідно до алгоритму нормалізації на декілька відношень [35]. Нормальна форма — властивість відношення в реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення. Далі будуть розглянуті декілька основних нормальних форм, які варто використовувати під час проектування схеми реляційної бази даних

Перша нормальна форма (1НФ, 1NF) є основною для структурованої схеми бази даних:

- Кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які унікально ідентифікують запис.
- Уникати дуплікатів та повторень правильно визначаючи неключові атрибути.
- Атомарність: кожен атрибут повинен мати лише одне значення, і не може містити.

Друга нормальна форма (2НФ, 2NF) вимагає, аби дані, що зберігаються в таблицях зі складним ключем, не залежали лише від частини ключа:

- Схема бази даних повинна відповідати вимогам першої нормальної форми.
- Дані, що повторно з'являються в декількох рядках повині бути винисені в окремі таблиці.

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

- Схема бази даних повинна відповідати всім вимогам другої нормальної форми.
- Будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

Будь-яке відношення в реляційній базі даних характеризується наступними властивостями.

1. Кожен кортеж містить тільки одне значення відповідного типу по кожному атрибуту (відношення нормалізовано). Кожному атрибуту в рамках кожного кортежу повинно бути поставлено у відповідність тільки одне значення.
2. Атрибути не є впорядкованими по якомусь правилу. Важливо пам'ятати, що упорядкування не є важливим і не враховується при роботі з базами даних.
3. Відсутність дублікатів кортежів. Розуміючи, що таблиця даних є фізичною реалізацією відносин в базі даних, існує можливість розміщення записів з однаковими значеннями. Розробник може забезпечити на рівні програмної логіки побудови бази даних можливість заборони дуплікатів.

Більшість реляційних баз даних використовують для визначення даних SQL та мову запитів; ці системи реалізують те, що можна розглядати як інженерне наближення до реляційної моделі.

2.3.3. SQL

SQL (англ. Structured query language - «мова структурованих запитів») - декларативна мова програмування, застосовувана для створення, модифікації та управління даними в реляційній базі даних, керованої відповідною системою управління базами даних. Головним призначенням SQL є опис, зміна і вилучення даних, що зберігаються в реляційних базах даних [36].

Таблиця в схемі бази даних SQL відповідає змінній предиката; вміст таблиці до відношення; ключові обмеження, інші обмеження та запити SQL відповідають предикатам. Схема містить таблиці, і кожна таблиця містить певну кількість стовпців. Це означає, що користувач не може оновлювати таблицю понад зазначену кількість стовпців, вказаних у таблиці. Це особливо корисно, якщо потрібно зберегти цілісність даних, а також переконатися в тому, який тип даних зберігається у базі даних.

Кожна таблиця в базі даних SQL може бути пов'язана. Тобто можливе створення зв’язків між таблицями. Ці відносини можуть бути один до багатьох, багато до багатьох або один до одного. Тип відносин залежить від поставленої задачі.

2.3.4. NoSQL

NoSQL (від англ. Not only SQL - не тільки SQL) - термін, що позначає ряд підходів, спрямованих на реалізацію систем управління базами даних, що мають суттєві відмінності від моделей, що використовуються в традиційних реляційних СУБД з доступом до даних засобами мови SQL. Застосовується до баз даних, в яких робиться спроба вирішити проблеми масштабованості та доступності за рахунок атомарности і узгодженості даних. Бази даних NoSQL були побудовані таким чином, щоб бути більш гнучкими, ніж SQL бази данних, а також містити більшу кількість даних [37].

У БД NoSQL немає заздалегідь визначеної схеми чи таблиць. Є колекції, і в кожній колекції є документи. Це дає змогу зберігати дані в різних формах у міру їх надходження. Ви можете вибрати в одній колекції декілька різних документів з різними полями. Також можливо створення відносин між колекціями. Однак вони не підходять для такої мети. Натомість ви можете зберегти все, що потрібно для одного запиту, в одній колекції.

2.3.5. Порівняння різних реалізацій баз даних

Різні підходи до проектування баз даних можуть бути корисними для різних задач. Варто проаналізувати дослідження щодо продуктивності

різних видів баз даних для збереження та надання доступу до даних сенсорів. Результати такого дослідження [38] виявили, що база даних, яка добре працює в режимі одиночного запису та зчитування декількох елементів, була б ідеальним кандидатом для зберігання даних сенсорів. З трьох перевірених дослідниками баз даних не існує єдиної бази даних, яка найкраще працює в обох випадках. Як виявило дослідження, MongoDB виграє в процесі єдиного запису, а PostgreSQL виграє в процесі виконання читання декількох елементів. Отже, від вимог програми та поставлених цілей залежить, яка база даних краще виконуватиме завдання. Багато проведених тестів показали, що продуктивність знижується при використанні віртуальної машини замість фізичного сервера. Однак деякі тести, що віртуалізація може позитивно вплинути на продуктивність. Дослідники припускають, що це пов'язано з кешуванням монітором віртуальної машини (VMM).

Порівнюючи використання трьох баз даних, можна зробити наступні висновки:

- Cassandra - NoSQL система, є найкращим вибором для великих застосувань збору критично важливих даних з датчиків. На його ефективність читання сильно впливає віртуалізація, як позитивно, так і негативно.
- MongoDB - NoSQL система, найкращий вибір для невеликого або середнього розміру некритичного сенсорного додатка, особливо коли важлива продуктивність запису. Під час використання віртуалізації спостерігається помірний вплив на продуктивність.
- PostgreSQL -SQL система, найкращий вибір, коли потрібні дуже гнучкі можливості запиту або важлива ефективність читання. Малі записи повільні, але позитивно впливають на віртуалізацію.

Для розробки невеликої системи, було обрано PostgreSQL через її найбільшу швидкість читання даних, оскільки швидкий доступ до даних є надзвичайно важливим критерієм для системи.

2.4. Протоколи передачі даних

Протокол передачі даних - це система правил, яка дозволяє двом або більше об'єктам системи створювати зв'язки та передавати інформацію через будь-який вид зміни фізичної величини. Протокол визначає правила, синтаксис, семантику та синхронізацію зв'язку та можливі методи відновлення помилок. Протоколи можуть бути реалізовані апаратним, програмним забезпеченням або їх комбінацією.

Комунікаційні системи використовують чітко визначені формати для обміну різними повідомленнями. Кожне повідомлення має точне призначення і очікує на одну з попередньо визначених відповідей для конкретної ситуації. Зазначена поведінка, як правило, не залежить від способу її здійснення. Протоколи комунікацій повинні бути узгоджені сторонами, що беруть участь. Зазвичай передача даних формується за допомогою стеку протоколів - сукупності протоколів, які взаємодіючи разом та розподіляючи функції забезпечують передачу даних.

2.4.1. HyperText Transfer Protocol

HTTP (протокол передачі гіпертексту) - це протокол, визначений в RFC 2616, який працює за принципом запитів та відповідей, який зазвичай використовує TCP протокол на транспортному рівні [39].

Протокол визначає, які повідомлення клієнт може надсилати на сервери і які отримувати відповіді. Заголовки запитів і відповідей використовують ASCII. Кожен запит складається з однієї або декількох рядків ASCII-тексту, причому перше слово в першому рядку є ім'ям методу, що має бути викликаний. За рядком запиту можуть слідувати інші рядки з додатковою інформацією. Вони називаються заголовками запиту (request headers). Цю інформацію можна використати з параметрами, які надаються при виклику процедури для додаткових опцій. Наприклад, локалізації чи визначення формату запиту чи відповіді. У свою чергу, відповіді можуть містити заголовки відповідей (response headers). Деякі заголовки можуть зустрічатися і у запиті, і у відповіді.

SSL (Secure Sockets Layer - протокол захищених сокетів) - протокол, який створює захищене з'єднання між двома сокетами, що дозволяє:

1) клієнту і серверу домовитися про використовувані параметрах;

2) провести аутентифікацію сервера клієнтом;

3) організувати таємне спілкування;

4) забезпечити захист цілісності даних.

При використані SSL між прикладним і транспортним рівнями з'являється новий рівень, який приймає запити від браузера і відсилає їх по TCP для передачі сервера. Після установки захищеного з'єднання основне завдання SSL полягає в використанні стиснення і шифрування. Комбінація SSL з HTTP має назву HTTPS (Secure HTTP - захищений HTTP), незважаючи на те що використовуються усі особливості та властивості HTTP. Втім, можливі деякі відмінності: доступ може здійснюватися через порт 443 замість стандартного для HTTP - 80.

2.4.2. Bluetooth

Bluetooth - протокол бездротової передачі даних заснованого на технології радіозв'язку. Стандартний Bluetooth включає в себе чимало протоколів, досить вільно розбиті на рівні. Також існують різні версії цього протоколу, деякі з яких стали досить популярними та широко використовуються у новітніх гаджетах для бездротової передачі даних, наприклад Bluetooth low energy [40].

На найнижчому рівні розміщений протоколи фізичного (радіотехнічного) рівня. За допомогою них описується та визначається радіозв'язок і методи модуляції. Багато з них створили, щоб зробити систему якомога доступнішою для широкого ринку. Головною метою наступного рівня управління канальним зв'язком є опис процесу управління головного узла тимчасовими інтервалами і як ці інтервали групуються в кадрі. Наступними є два протоколи, які використовують протокол управління каналом зв'язку. Протокол управління з'єднанням встановлює логічні канали між пристроями, керує режимами енерговикористання, підключенням і

шифровуванням, а також якісним обслуговуванням. Він розташований нижче лінії інтерфейсу хост-контролера. Цей інтерфейс створений для зручності під час реалізації: як правило, протоколи нижчі лінії реалізуються на чистому Bluetooth, а протоколи вище ліній - на пристрої Bluetooth, де розміщений чіп. Протокол канального рівня - це протокол L2CAP (протокол управління логічними каналами та узгодженням). Він збирає повідомлення про змінну довжину і при необхідності забезпечує надійність. L2CAP використовує багато протоколів для виконання своїх функцій. Протокол пошуків сервісів використовується для визначення місця розташування службових служб в межах мережі. Протокол RFcomm емулює роботу стандартного послідовного порту комп'ютера, до якого зазвичай підключаються клавіатура, миші та інші пристрої. На самому верхньому рівні знаходиться рівень додатків, який викорустовується різними видами застосунків.

2.5. Методи авторизації та аутентифікації

Аутентифікація - це метод, за допомогою якого процес впевнюється, що його співбесідник є саме тим, за кого він себе видає [41]. Перевірка справжності віддаленого процесу представляє собою єдину складну задачу і вимагає складних протоколів, заснованих на методах криптографії. Авторизація - це метод, за допомогою якого процес визначає, яку роль має клієнт і забезпечує надання відповідних дозволів аб привілеїв. Тобто, аутентифікація пов'язана з питанням справжності співбесідника, тоді як процес авторизації має справу з рішеннями та наданням доступу до певних матеріалів. Наприклад, Таненбаум наводить приклад кліентського процесу, який обробляє файли сервера і говорить: «Я обробляю Скотта, і я хочу передати файл cookbook.old». Сервер повинен вирішити наступні два питання.

1. Чи дійсно це запит Скотта (аутентифікація)?
2. Чи є право скористатися файлом cookbook.old (авторизація)?

2.5.1. JSON Web Token

JSON Web Token (JWT) - це компактний, захищений від URL-адрес спосіб представлення даних ідентифікації клієнта, що передаються між двома сторонами [42]. Заявки в JWT кодуються як об'єкт JSON, який використовується як корисне навантаження структури веб-підпису JSON або як простий текст структури веб-шифрування JSON, що дозволяє інформації бути підписаною або цілістно захищеною з кодом аутентифікації повідомлення та/або зашифрованою.

JWT складається з трьох частин: заголовка, вмісту і підпису. Заголовок це JSON елемент, який описує до якого типу токену належить даний і які методи шифрування використовувались. Вміст токену складається є елементом JSON, який описує інформацію про клієнта. Останньою частиною є підпис. Щоб згенерувати підпис заголовок та вміст кодуються в Base64, записуються в один рядок через крапку, а потім цей рядок хешується визначеним методом. Сервер при отримані токену має змогу повторити операції та перевірити справжність отриманих даних. Останнім етапом формування токену є кодування усіх частин за допомогою Base64 і розділення всіх секцій токена крапкою.

ВИСНОВКИ ДО РОЗДІЛУ 2

Новітні технології дозволяють розробнику обирати потрібні засоби, механізми та методики розробки програмного забезпечення для створення найбільш продуктивної системи.

Для ефективної обробки та моніторингу мультимодальних даних в описаному в попередньому розділі контексті надзвичайно важливим є використання деяких найбільш придатних та ретельно підібраних програмних засобів , таких як операційні системи, мови програмування, середовища розробки, бібліотеки, тощо, для організації роботи в системі накопичення, збереження та аналізу таких мультимодальних даних. Обрана мова програмування дає змогу розроблювати програмне забезпечення як для мобільного додатку, так і для ефективного серверного забезпечення. Обрана REST архітектура серверного програмного забезпечення дозволяє ефективно розподіляти функціональні можливості та навантаження, роблячи серверну частину ефективною та масштабованою. Обраний фреймворк надає сукупність бібліотек та підходів, що полегшують процес розробки програмного забезпечення та його подальшої підтримки та модифікації. Розглянуті протоколи передачі даних дають змогу безпечно та легко обмінюватися даними у мережі інтернет та бездротово на невеликих відстанях, що зробить розроблену систему мобільною, зручною та дешевою. Були розглянуто різні види баз даних та підходи до збереження великих масивів даних та обрана модель зберігання даних для проекту. Для порівняння були використані теоретичні відомості про кожен вид баз даних та матеріали, які на основі практичних дослідів порівнювали ефективність різних баз даних. Серед різних методів авторизації та аутентифікації був обраний такий, який може бути підтриманий різними видами фронтендів на різних пристроях (браузерами через персональні комп'ютери, мобільними додатками на різних платформах) та має достатній рівень безпеки для забезпечення безпеки даних користувачів.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1. Опис системи

Система складається з таких основних компонентів (рис. 3.1):

- хмарного сховища;
- мобільного додатку;
- мозкового інтерфейсу для вимірювання активності мозку, який збирає інформацію з сенсорів.

Рис. 3.1. Схема цілісної системи збору та збереження даних ЕЕГ

Головна метою системи є збір даних з сенсорів за допомогою інтерфейсу і накопичення їх на мобільному пристрої для подальшого їх використання та аналізу. Після завершення сеансу вимірів дані можуть бути доповнені коментарем користувача або додатковими даними такими як геолокація, тощо та повинні бути збережені на хмарному сховищі для подальшого їх використання та аналізу. Тобто дані кожного сеансу будуть мультимодальними і доповнюватися не тільки різними сенсорами, а й введеними користувачем даними. Для зручності передачі даних з сенсорів до мобільного пристрою використовується протокол бездротової передачі

Bluetooth. Мобільний додаток для зв'язку та обміну повідомленнями повідомленнями з сервером буде використовувати протокол HTTP/HTTPS на прикладному рівні. Задля аутентифікації і авторизації мобільний додаток буде використовувати технологію JWT, яка забезпечує захищеність даних користувача. Мобільний додаток буде виступати в ролі інтерфейсу для користувача. Додаток буде дозволяти контролювати процес запису та переглядати збережені на хмарному сховищі сесії.

3.2. Обладнання

Щоб обрати потрібний пристрій варто переглянути технічну специфікацію та проведені за допомогою них дослідження. Огляд цих робіт дає можливість порівняти 2 системи. Оскільки система від компанії OpenBCI пропонує 16 каналів, більше можливостей щодо розміщення електродів і мобільність, яка може бути значною перевагою при розробці мобільної системи, для роботи була обрана система Ultracortex Mark 4 з Cyton Board (див. рис. 1.7).

Ultracortex - це гарнітура, яка вироблена за допомогою технологія 3D принта з відкритим кодом, призначена для роботи з будь-якою платою OpenBCI. Ця гарнітура розроблена лише для прийому сигналів ЕЕГ. Ultracortex Mark IV здатний прослуховувати до 16 каналів ЕЕГ з 35 доступних місць. Оскільки це проект з відкритим кодом, розробник має змогу робити модифікації кодової бази, змінюючи формати повідомлень чи виконуючи інші модифікації. Для цього проекту модифікації програмного забезпечення ЕЕГ пристрої виконані не будуть, тому що формат передачі даних і процес роботи задовольняє поставлені проектом цілі.

Cyton Board OpenBCI - це сумісний з Arduino 8-канальний нейронний інтерфейс з 32-бітовим процесором. Cyton Board OpenBCI реалізує мікроконтролер PIC32MX250F128B. Плата заздалегідь прошита завантажувальним пристроєм chipKIT ™ та останньою прошивкою OpenBCI. Дані дискретизуються з частотою 250 Гц на кожному з восьми каналів. Cyton Board OpenBCI може використовуватися для вибірки

мозкової активності (ЕЕГ), м’язової активності (ЕМГ) та серцевої діяльності (ЕКГ). Плата обмінюється даними з комп'ютером через USB передавач OpenBCI за допомогою радіомодулів RFDuino, використовуючи технологію Bluetooth.

RFduino - це модуль Bluetooth BLE з Bluetooth Low Energy 4.0 зі вбудованим модулем ARM Cortex-M0 та модулем SMT RFD22301.

OpenBCI Cyton використовує модулі RFDuino для бездротового з'єднання Bluetooth. Для досягнення максимальної швидкості передачі даних OpenBCI постачає USB передавач RFDuino, який підключається до комп'ютера. При використанні цього USB передавача досягається більш висока швидкість передачі даних у порівнянні зі стандартним Bluetooth 4.n BLE-з'єднанням.

USB передавач RFDuino ("хост" RFDuino) (рис. 3.2) з'єднаний з FTDI USB послідовним конвертором, налаштованим на комп'ютер, як ніби це стандартний послідовний порт, що працює зі швидкістю 115200 бод, використовуючи 8-N-1. 8-N-1 (скорочено 8N1) - це коротке позначення установки параметрів COM-порту комп'ютера або інтерфейсу UART (USART) в мікроконтролерах. Цифра 8 позначає кількість бітів інформації в пакеті, буква N вказує на відсутність службового біта перевірки на парність / непарність, цифра 1 позначає число стоп-бітів в кінці пакета.

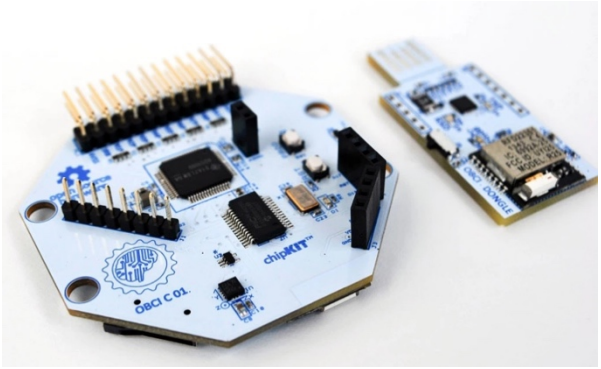


Рис. 3.2. Cyton Board OpenBCI та USB передавач RFDuino

3.3. Програмне забезпечення додатку та контролеру для збору даних

3.3.1. Формат передачі даних з OpenBCI Cyton

Дані передаються на пристрій у вигляді потоку байтів, які розділені на пакети. Оскільки кожний пакет має визначений формат, мобільний додаток має зчитувати поток байтів і розділяти його на пакети. Кожен пакет містить заголовок, лічильник зразків, дані з 8 каналів, а потім три значення осі акселерометра з подальшим колонтитулом. Дані акселерометра необов'язкові, і їх не потрібно надсилати з кожним пакетом при використанні. Якщо вони не використовуються, байти будуть рівні 0. Це дозволяє надіслати визначені користувачем допоміжні дані в останні шість байтів перед колонтитулом.

Заголовок:

- Байт 1: 0xA0;
- Байт 2: порядковий номер.

Дані ЕЕГ:

- Байти 3-5: Значення даних для каналу 1 ЕЕГ;
- Байти 6-8: Значення даних для каналу 2 ЕЕГ;
- Байти 9-11: значення даних для каналу 3 ЕЕГ;
- Байти 12-14: Значення даних для каналу 4 ЕЕГ;
- Байти 15-17: Значення даних для каналу 5 ЕЕГ;
- Байти 18-20: Значення даних для каналу 6 ЕЕГ;
- Байти 21-23: значення даних для каналу 6 ЕЕГ;
- Байти 24-26: значення даних для каналу 8 ЕЕГ.

Інші дані:

- Байти 27-32: 6 байт даних, визначених на основі нижнього колонтитулу нижче.

Футер (кінець пакету):

- Байт 33: 0xCX, де X приймає значення 0-F. Значення X може вказувати на модифікації формату передачі даних.

Драйвери повинні використовувати стоп байт, щоб визначити, як використати 6 додаткових байтів AUX. Позначення AX1-AX0 визначають: значення даних для каналу акселерометра X, а AY1-AY0: значення даних для каналу акселерометра Y, AZ1-AZ0: значення даних для каналу акселерометра Z.

Таблиця 3.1

Приклади різних версій протоколів зі значеннями стоп-байт та значеннями додаткових байт

Стоп байт	**Байт 27**	**Байт 28**	**Байт 29**	**Байт 30**	**Байт 31**	**Байт 32**	**Назва**
0xC0	AX1	AX0	AY1	AY0	AZ1	AZ0	Стандарт для акселерометра
0xC1	UDF	UDF	UDF	UDF	UDF	UDF	Стандарт для aux
0xC2	UDF	UDF	UDF	UDF	UDF	UDF	Визначено користувачем
0xC3	AC	AV	T3	T2	T1	T0	Дані з акселерометру та відмітки часу
0xC4	AC	AV	T3	T2	T1	T0	Дані з акселерометру та відмітки часу
0xC5	UDF	UDF	T3	T2	T1	T0	Дані з aux та відмітки часу
0xC6	UDF	UDF	T3	T2	T1	T0	Дані з aux та відмітки часу

3.3.2. Зчитування та зберігання даних у мобільному додатку

Дані зчитуються з підключеного USB передавача, ніби зі стандартного послідовного порта, що працює зі швидкістю 115200 бод. Для оптимізації

процесу зчитування та обробки, отримані пакети обробляються групами більше ніж X пакетів. Під час обробки даних наступні пакети продовжують накопичуватися на порті.

Кожна група пакетів розбивається на окремі пакети на основі значення стоп байта. Дані з кожного пакету послідовно обробляються та записуються до файлу з форматом csv для подальшої зручної обробки. Після завершення сеансу збору даних, вся інформація надсилається на сервер для подальшого збереження.

3.3.3. Інтерфейс мобільного додатку

Інтерфейс мобільного додатку заснований на декількох основних екранах, таких як екран входу, головне меню, екран зчитування та запису даних та екран перегляду зібраних даних.

Екран входу має основну функцію входу та реєстрації у системі. Також на цьому екрані знаходиться функціонал зміни мови в додатку (рис. 3.3). Механізм локалізації розроблений за порадами наданими Google щодо реалізації підтримки різних мов. Розробники Android рекомендують використовувати рамки ресурсів Android, щоб максимально відокремити локалізовані аспекти програми від основної функціональності на основі Java. Гарною практикою є розміщення всього вмісту користувальницького інтерфейсу програми у файлах ресурсів.

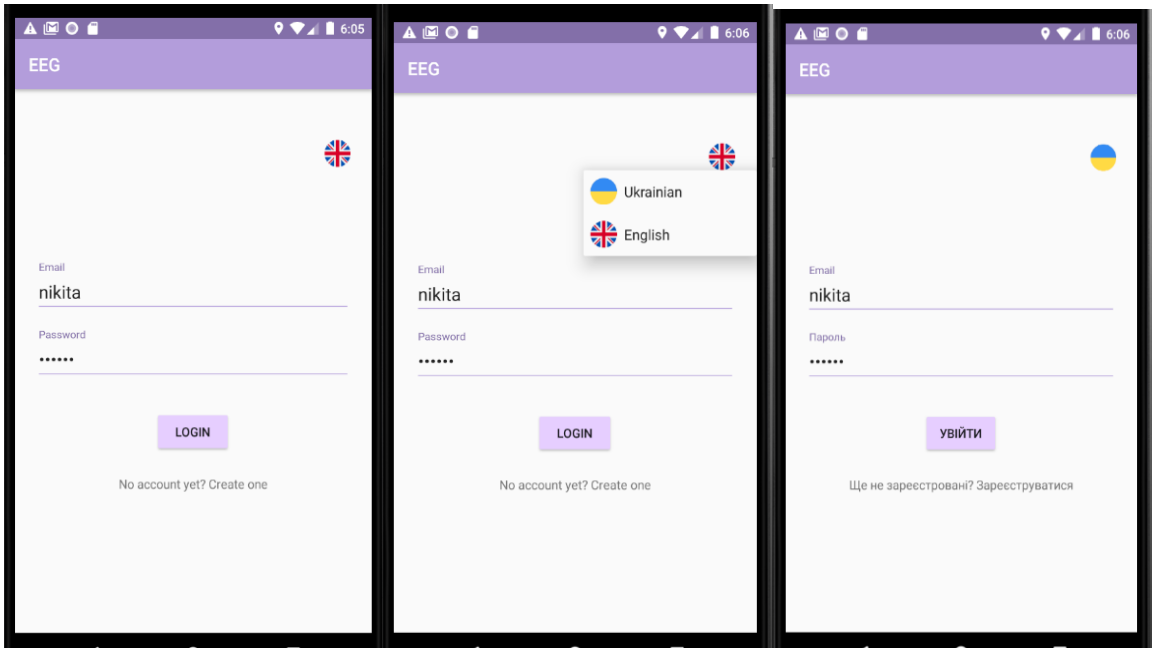


Рис. 3.3. Процес вибору мови локалізації на сторінці входу

Ресурси - це текстові рядки, макети, звуки, графіка та будь-які інші статичні дані, необхідні додатку Android. Додаток може включати кілька наборів ресурсів, кожен налаштований для іншої конфігурації пристрою. Коли користувач запускає додаток, Android автоматично вибирає та завантажує ресурси, які найкраще відповідають пристрою. При написанні програми, створюється ресурси за замовчуванням і альтернативні ресурси для використання додатком. Коли користувачі запускають додаток, система Android вибирає, які ресурси завантажувати, залежно від параметрів самого пристрою. Для створення ресурсів файли розміщуються у спеціально названих підкаталогах проекту.

Також важливими елементами є сторінки авторизації та реєстрації з валідацією даних (рис. 3.4). Авторизація та аутентифікація заснована на технології JWT, яка дає змогу надавати різні права та бути впевненим у справжності джерела та надісланих даних.

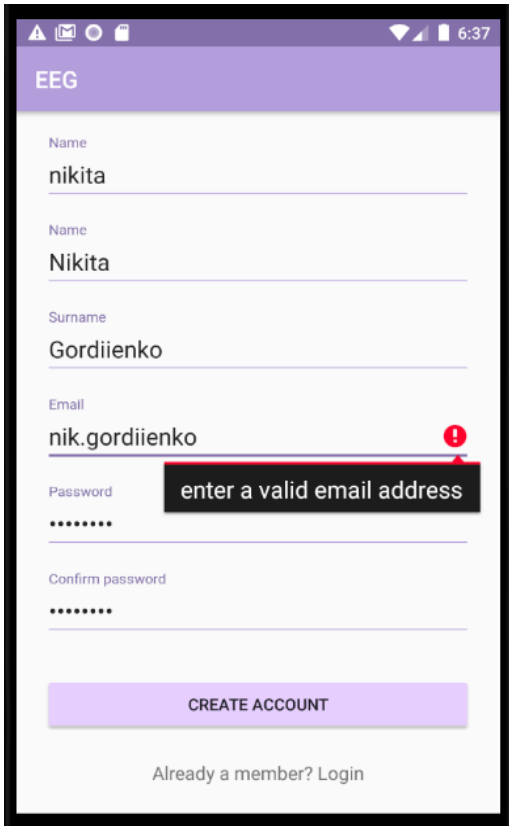


Рис. 3.4. Приклад валідації даних під час процесу реєстрації

Меню розташоване знизу та відображається на всіх головних сторінках для зручної навігації (рис. 3.5). Меню складається з 3 головних кнопок з іконками. Обраний розділ виділяється кольором, що сприяє інтуїтивному розумінню інтерфейсу. Основними розділами є головний розділ, на якому відображається екран перегляду записів, розділ запису, який дозволяє керувати апаратом ЕЕГ та збирати дані, та розділ налаштувань, у якому користувач має можливість обрати власне місце розташування та визначити які дані збираються кожним з каналів ЕЕГ.

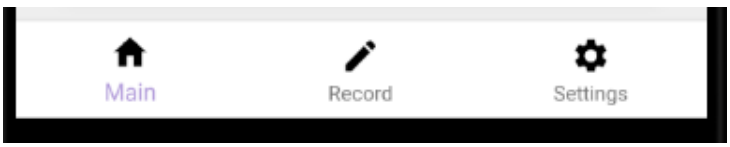


Рис. 3.5. Панель навігації на головних екранах додатку

На екрані перегляду даних у вигляді списку відображаються усі збережені на сервері сеансу (рис. 3.6).

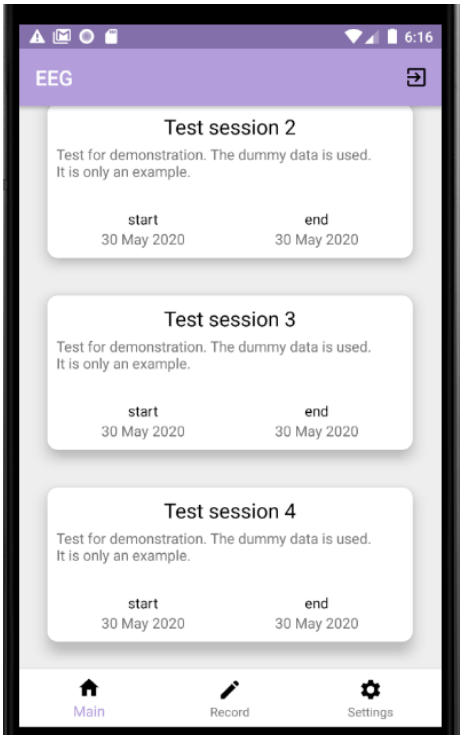


Рис. 3.6. Приклад інтерфейсу для відображення списку збережених сесій

Дані про сеанси розбиті на частини і надаються цими частинами. Для отримання даних збережених під час сеансу, потрібно натиснути на потрібний сеанс і дані будуть завантажені на мобільний пристрій задля подальшої обробки (рис. 3.7).

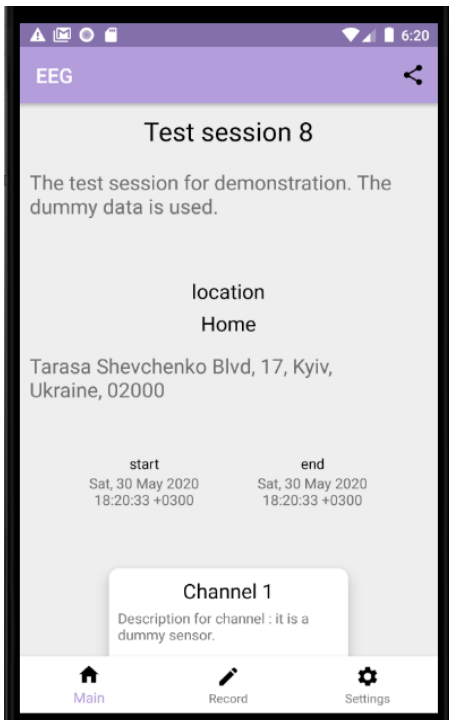


Рис. 3.7. Приклад інтерфейсу для відображення деталей однієї з збережених сесій

Після чого дані будуть відображені на екрані і буде надана можливість переглянути зібрані дані по кожному сенсору у вигляді динамічних графіків (рис. 3.8). Також можливе збереження сеансу у вигляді файлу формату csv, тобто наявна функція експорту.

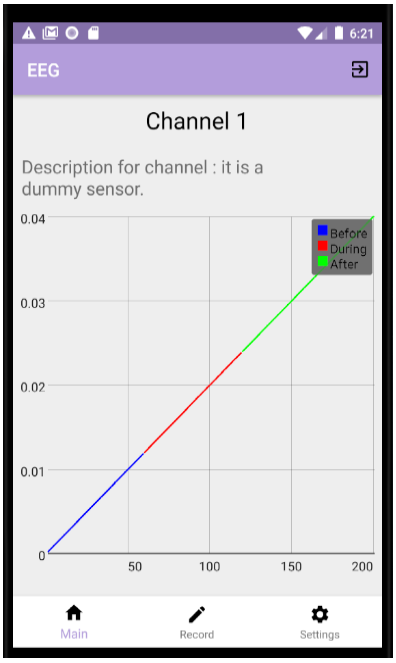


Рис. 3.8. Приклад інтерфейсу для відображення даних одного з сенсорів збереженої сесії

При записі та перегляді даних, важливо візуалізувати їх. Саме через візуалізацію людині легше простежити деякі патерни та тенденції. Оскільки дані будуть відображатися у вигляді графіку і цей графік має змінюватися у режимі реального часу, бібліотека має підтримувати такі функції. GraphView - це бібліотека для Android для програмного створення гнучких і красивих діаграм, яка була використана для візуалізації зібраних даних. Вона дозволяє створювати лінійні графіки, смугові графіки, точкові графіки або впроваджувати власні типи. Також графіки можна збільшувати чи зменшувати, а також переміщати, що дозволяє створювати зручні для користування графіки.

Екран налаштувань дозволяє обрати для кожного каналу з якого збираються дані потрібний сенсор та створювати нові сенсори (рис. 3.9). Ці сенсори дають змогу користувачеві збирати дані з будь-яких сенсорів,

ділянок мозку тощо. Тобто дані зібрані з одного з каналів будуть збережені на сервері як зібрані за допомогою обраного сенсора. Це у подальшому дасть змогу користувачеві фільтрувати дані та полегшить роботи з великою кількістю сенсорів.

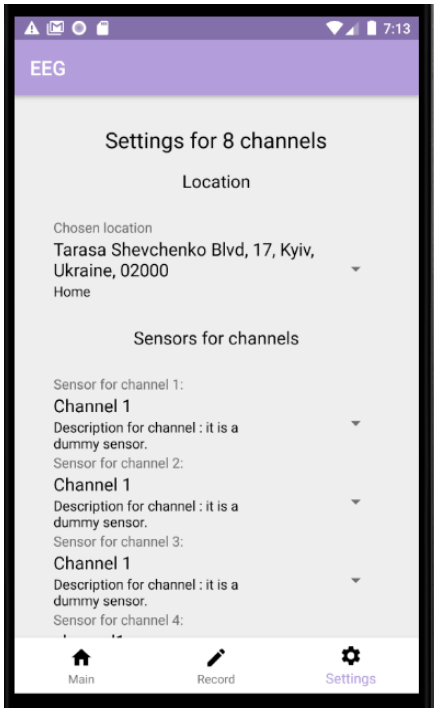


Рис. 3.9. Приклад інтерфейсу для налаштувань збору даних

Для створення та обрання локації збору даних використовується API Google Maps, яке дозволяє не тільки надавати користувачеві інтуітивний інтерфейс для вибору місцеположення, але й визначати адресу по визначеним координатам. На основі цієї технології було розроблене меню створення місцеположення (рис. 3.10).

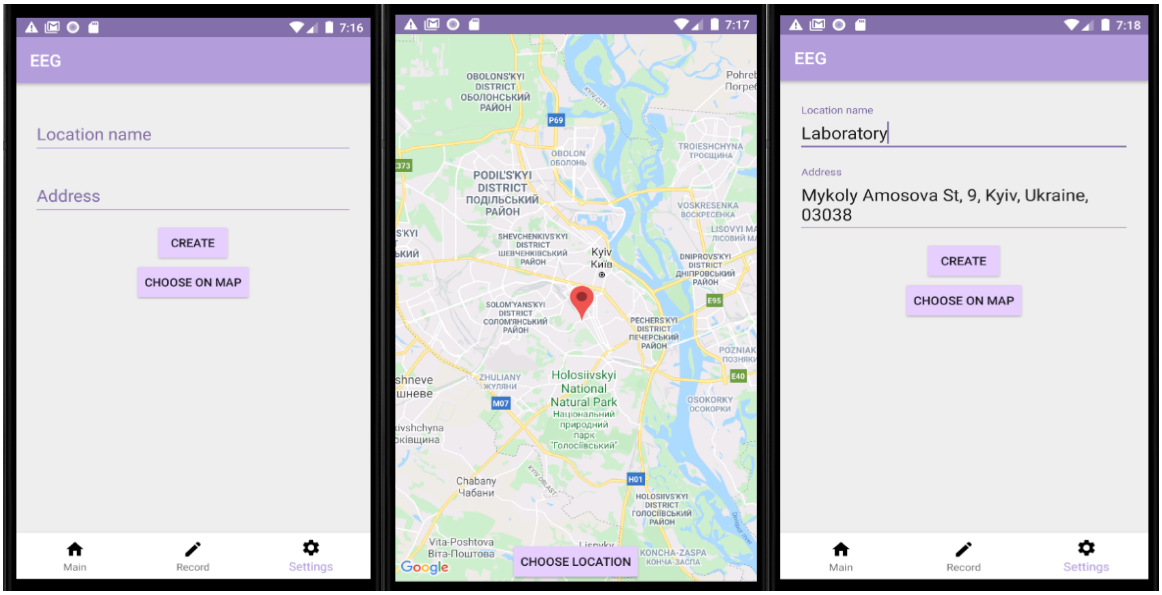


Рис. 3.10. Приклад інтерфейсу для створення нового місцезнаходження для запису даних з використанням Google Maps API

Екран запису містить декілька основних функцій, такі як визначення назви та опису сесії, підключення до пристрою й початок запису чи закінчення запису, і зміни стану (рис. 3.11). Зміна стану потрібна для поміток на даних. Було визначено 3 основних стани: до певної дії, під час дії та після певної дії. Таким чином спостерігач, той хто контролює запис сесії, може робити помітки на даних не відволікаючись від спостереження. Після закінчення запису тимчасовий файл створюється у файловій системі з накопиченими даними. Після успішного відвантаження даних на сервер цей файл видаляться з файлової системи.

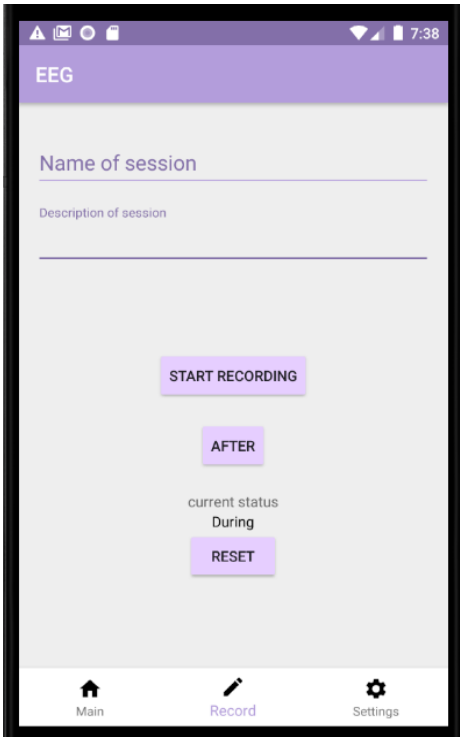


Рис. 3.11. Приклад інтерфейсу екрану запису

3.4. Серверне програмне забезпечення

3.4.1. Реєстрація, аутентифікація та авторизація

Процес реєстрації передбачає заповнення форм. Валідація проходить як зі сторони фронтенду (інтерфейсу користувача для вказування на невірний формат чи можливі помилки), так і зі сторони бек енду (серверного програмного забезпечення, яке відхиляє запити з неповною чи некоректною інформацією).

Збережена інформація зберігається у базі даних після успішної валідації. Після чого користувач має змогу зайти до свого акаунта. Процес авторизації та аутентифікації використовує такий механізм Spring як The Security Filter Chain (ланцюжок з фільтрів для забезпечення безпеки) [43]. Веб-інфраструктура Spring Security повністю заснована на стандартних фільтрах сервлетів. Він не використовує сервлетів або будь-які інші фреймворки на основі сервлетів (наприклад, Spring MVC), тому він не має міцних посилань на будь-яку конкретну веб-технологію. Він використовує HttpServletRequests і HttpServletResponses, і для нього не є важливим

джерело запитів. Наприклад, чи надходять запити від браузера, клієнта веб-сервісу, HttpInvoker або програми AJAX.

Spring Security підтримує внутрішній ланцюжок фільтрів, де кожен з фільтрів несе певну відповідальність, і фільтри додаються або видаляються з конфігурації залежно від того, які послуги потрібні. Упорядкування фільтрів важливо, оскільки між ними існують залежності. Також власні фільтри можуть бути створені та додані до ланцюги, чи навпаки - створені фільтри можуть бути прибрані з ланцюгу ччерез їх непотрібність. Деякі з фільтрів є надзвичайно важливими для процесу авторизації та аутентифікації.

Порядок визначення фільтрів є важливим елементом побудови ланцюга фільтрів. Незалежно від того, які фільтри фактично використовується узагальнений порядок є таким:

- ChannelProcessingFilter - фільтр, який може визначити протокол для обробки та перенаправити на інший протокол.
- SecurityContextPersistenceFilter - фільтр який відповідає за створення SecurityContext, який може бути встановлений у SecurityContextHolder на початку веб-запиту, а будь-які зміни в SecurityContext можуть бути скопійовані на HttpSession, коли веб-запит закінчується (готовий до використання з наступним веб-запитом).
- ConcurrentSessionFilter - фільтр, який використовує SecurityContextHolder і потребує оновлення SessionRegistry для відображення поточних.
- Механізми обробки автентифікації - UsernamePasswordAuthenticationFilter, CasAuthenticationFilter, BasicAuthenticationFilter тощо - фільтр, який змінює SecurityContextHolder таким чином, щоб містити дійсну інформацію про користувача, який надіслав запит.
- SecurityContextHolderAwareRequestFilter - фільтр, який може бути використаний для встановлення HttpServletRequestWrapper з Spring Security у свій контейнер сервлетів.

- RememberMeAuthenticationFilter - фільтр, який використовується у випадку якщо жоден попередній механізм обробки автентифікації не оновлював SecurityContextHolder, а запит представляє файли cookie, які дозволяють виконувати послуги запам'ятовування користувача. Тут також може бути розміщена дійсна інформація про користувача, який створив запит.
- AnonymousAuthenticationFilter - фільтр, який використовується у випадку якщо жоден раніше механізм обробки автентифікації не оновлював SecurityContextHolder. На цьому етапі може бути розміщений анонімний об’єкт аутентифікації, який вказує що користувач невідомий.
- ExceptionTranslationFilter - фільтр, мета якого фіксування будь-яких винятків та помилок Spring Security задля повернення відповіді про помилку HTTP або ж переадресації на відповідний AuthenticationEntryPoint.
- FilterSecurityInterceptor - фільтр, що захищає веб-URI та викликає винятки або помилки, коли доступ заборонено.

Оскільки для авторизації та аутентифікації використовується технологія JWT, був розроблений власний фільтр, який зчитує з заголовку JWT токен на валідує його. Як результат успішної валідації, інформація користувача додається до контексту, що дає змогу використовувати її під час обробки запиту. Це дає змогу побудувати RESTful сервіс без запам'ятовування стану і полегшує використання сервісу у додатку.

Для отримання токену потрібного для надсилання запитів використовуються два головних ендпоінти, тобто точки доступу що відповідають за певні функціональні можливості:

- для отримання токенів, який створює токен для доступу та токен для оновлення після отримання імені користувача та паролю.
- для поновлення токенів, який створює нові токени після отримання токену для оновлення.

Під час першого отримання токенів, які потрібні для доступу до даних та застосунку в загалому, користувач надсилає свій пароль та логін і отримує пару токенів: перший, що використовується для отримання доступу на короткий період та надсилається з кожним запитом; другий, що використовується для поновлення обох токенів, який валідний більш тривалий проміжок часу і надсилається тільки на ендпоінт оновлення токенів (рис. 3.12).

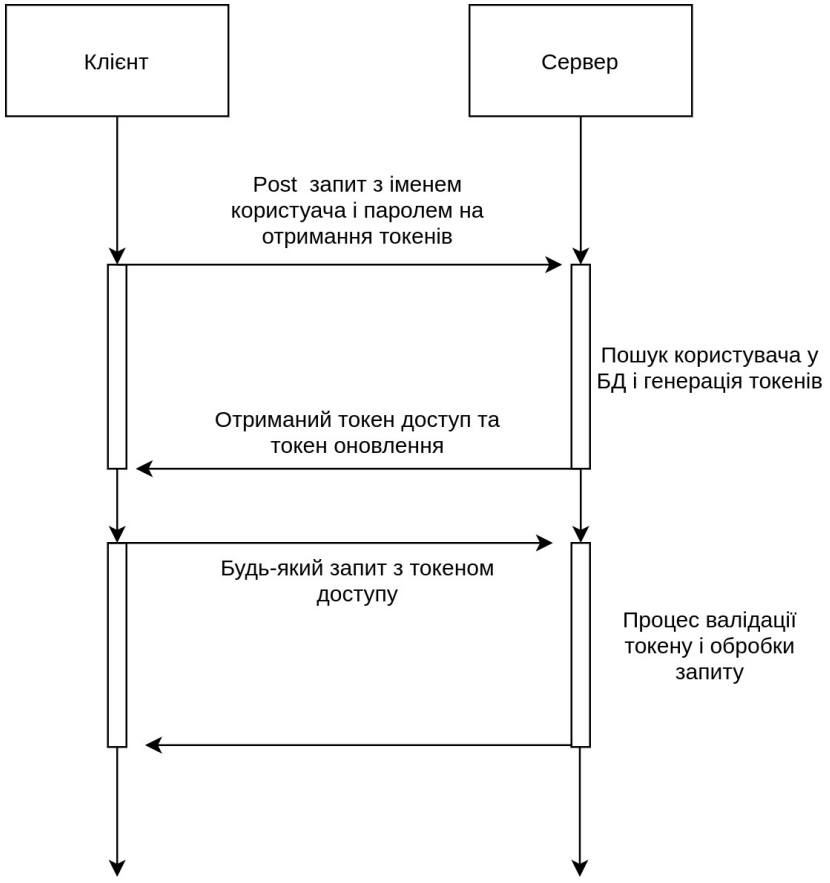


Рис. 3.12. Схема роботи аутентифікації: використання логіну та паролю та подальшим запитом з токеном доступу

Оскільки надсилання паролю та логіну є небезпечною операцією, через те що дані можуть бути перехоплені чи викриті якимось іншим чином, був створений токен для оновлення. У ньому зберігається випадкове унікальне число, яке зберігається за користувачем на стороні серверу.

Токен для доступу містить у собі ідентифікатор користувача, права користувача, які використовуються фронтендом (програмою, яка відповідає за представлення інтерфейсу для взаємодії з користувачем), тобто рівнем

представлення який у даній системі представлений мобільним додатком, для визначення можливих дій, дату створення, дату закінчення дії, інформацію про алгоритм шифрування та підпис.

- За допомогою ідентифікатору, сервер знаходить збережену інформацію про користувача та визначає його права.
- За допомогою дати створення та дати закінчення дії визначається придатність даного токену.
- За допомогою підпису та алгоритму шифрування визначається справжність надісланих даних.

Токен для оновлення містить ідентифікатор користувача, випадкове унікальне число, дату створення, дату закінчення дії, інформацію про алгоритм шифрування та підпис. При генерації токену випадкове унікальне число закріплюється за користувачем і коли користувач надсилає цей токен, сервер перевіряє справжність інформації за допомогою підпису, придатність за допомогою часу та чи закріплене унікальне число за визначеним користувачем. Якщо уся інформація правильна, то користувачу надаються два нових токени з новими термінами дії, закріплене число перезаписуються (рис. 3.13).

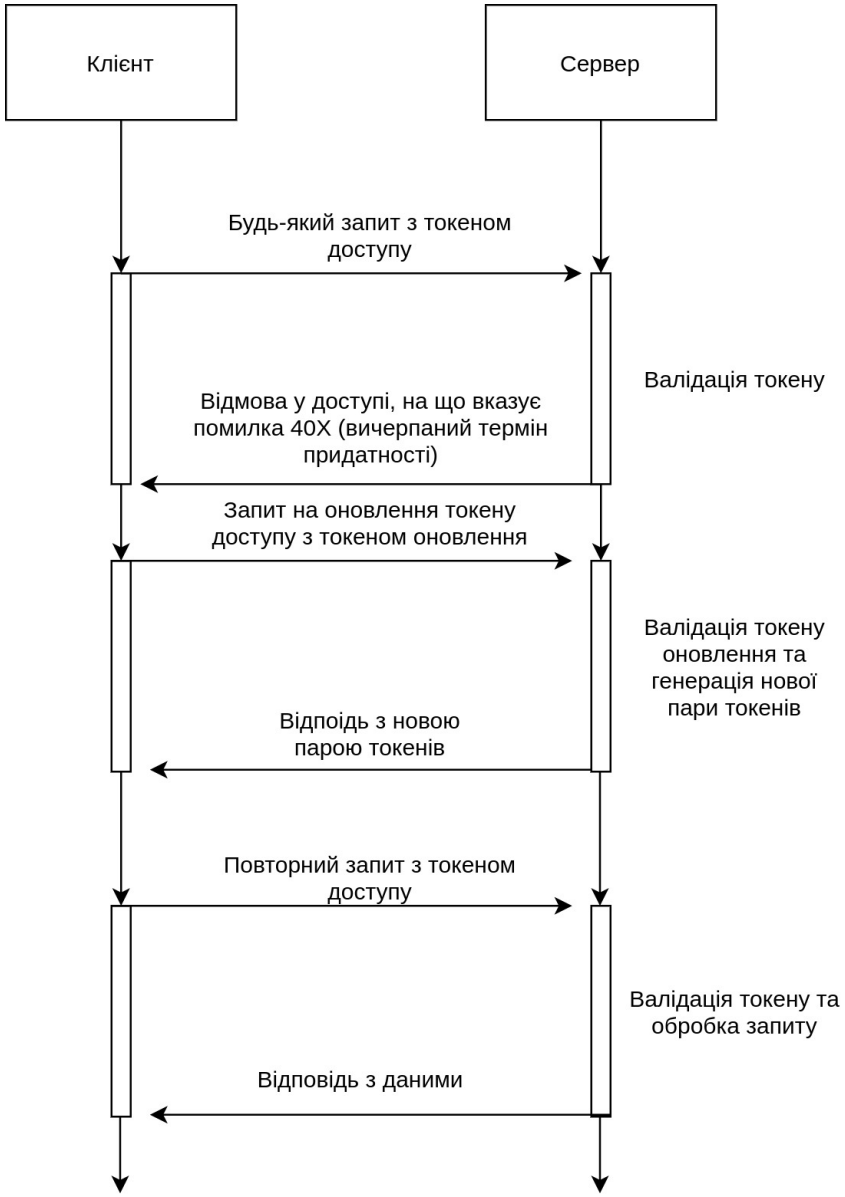


Рис. 3.13. Схема процесу аутентифікації: використання застарівшого токену доступу, його оновлення та повторний запит з поновленим токеном доступу

Така схема легко знаходить порушників та має просту схему використання:

- Якщо токен доступу став непридатним через закінчення терміну дії, сервер надішле помилку 401 і додаток зможе оновити його за допомогою токену оновлення.
- Якщо будь-яка інформація буде якось відредагована, перевірка підпису виявить це. Якщо токен доступу буде вкрадений, то він буде придатним

короткий проміжок часу і зловмисник не може продовжити термін його дії.

- Якщо токен оновлення вичерпав свій термін придатності, можливо отримати нову пару токенів за допомогою повторного входу в систему (рис. 3.14).

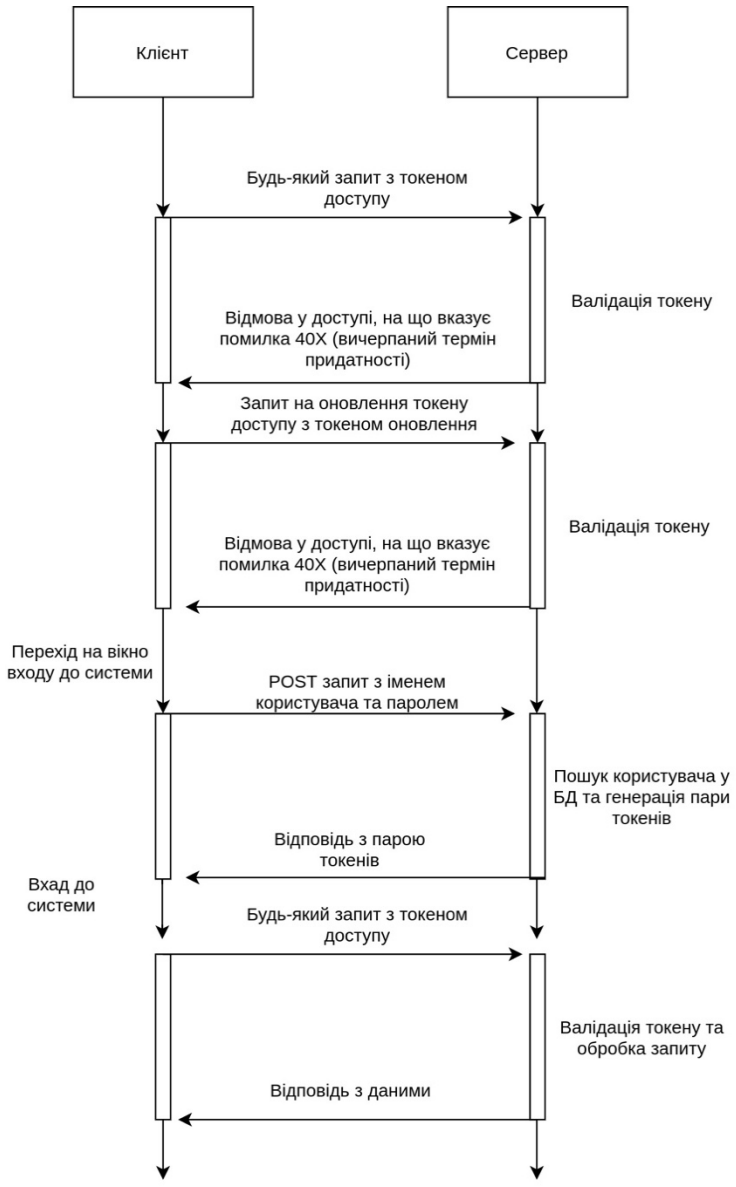


Рис. 3.14. Схема процесу аутентифікації: використання застарівшого токену доступа, спроба поновити його застарівшим токеном оновлення, отримання нових токенів за допомогою логіну з іменем користувача та паролем

3.4.2. Архітектура вебсервісу

Веб сервіс використовує архітектури REST отже приймає 4 типи основних запитів з параметрами.

Запити з методом GET надають дані користувачу перевіряючи чи є до них доступ у користувача. Усі дані надаються у форматі JSON і дані у вигляді списку передаються сторінками, на якій розміщена певна кількість об’єктів. Фронт-енд, який выдображає дані користувачу, керує сторінками та запитами на надання нової інформації. Це допомагає керувати списком і не робити зайвих непотрібних запитів надмірно навантажуючи сервер та мережу.

Запити з методом POST передають інформацію у тілі запиту для її використання чи збереження у базі даних.

Запити з методом PUT використовуються для редагування бази даних і у даному додатку доступні тільки адмінам та розробникам.

Запити з методом DELETE використовуються для видалення записів бази даних і у даному додатку доступні тільки адмінам та розробникам.

Приклади основних ендпоінтів та їхні функції:

- /auth/register, /auth/login, /auth/token - використовуються для реєстрації, авторизації та аутентифікації та до них дозволений анонімний доступ. Тобто вони не потребують валідного токену доступу для обробки запиту. Усі інші ендпоінти вимагають від користувача валідного токену доступу для обробки відправленого запиту.
- api_sessions/sessions, api_sessions/sessions/add, api_sessions/channel_data /add, api_sessions/session/{id}/sensor/{sensor_id} - основні ендпоінти для доступу до данних: надають доступ до сесій, дозволяють створювати сесії, дозволяють додавати дані та отримувати дані по певному сенсору та сесії.
- api_sensors/sensors/add, api_sensors/sensors, /api/locations, api/location/add - додаткові ендпоінти для створення додаткових даних необхідних для налаштування сесій, таких як місцезнаходження та сенсорів.

3.4.3. Збір та збереження даних

Дані зберігаються у базі даних, яка має декілька основних сутностей: користувач, сесія, доступні сенсори, дані кожного з сенсора.

- Користувач зберігає основну інформацію про користувача, яка використовується у додатку та збирається під час реєстрації.
- Сесія зберігає інформацію про сесію, пристрій який був використаний для збору інформації, тривалість збору інформації, дату початку та кінця вимірювання тощо.
- Дані сенсору використовують ідентифікатор сесії для встановлення сесії під час якої вони були зібрані.
- Доступні сенсори зберігають інформацію, про назву таблиці з даними, короткий опис даних та одиниць вимірювань. Використовується для опису даних на інтерфейсі користувача.

Таким чином можна додавати нові сенсори і збирати та зберігати з них не модифікуючи існуючу архітектуру бази даних.

ВИСНОВКИ ДО РОЗДІЛУ 3

Спроектована система дає змогу вільно збирати та зберігати дані. Система легка для використання і налаштування, тому що використовує технології бездротової передачі даних, що робить систему мобільною. Оскільки використовується мобільний додаток для системи Android у якості інтерфейсу, будь-який користувач у якого є пристрій з операційною системою Android, має змогу встановити додаток, зареєструватися у системі та почати збирати дані для дослідження. Розробники мають змогу легко розширювати систему збору даних додаючі нові характеристики чи сенсори для збору. Створена система полегшує процес збору даних для дослідників і полегшує їх використання. Наприклад, користувач може додати коментар, а потім легко відшукати потрібні дані для аналізу та завантажити їх у потрібному форматі. Розроблена система авторизації та аутентифікації впроваджує базисну функціональність, яку можна легко розширити додавши ролі адмінів чи модераторів. Розроблений додаток дає можливість розробити додаткові інтерфейси для налаштування більшої кількості каналів (наприклад, 16) і дає змогу не змінювати серверне програмне забезпечення, яке і зараз може підтримувати та зберігати дані з будь-якої кількості каналів. З іншого боку, під час розробки системи були виявлені можливі напрямки для поліпшення, такі як архівація даних для зменшення обсягу даних, що передаються через інтернет, можливість збереження даних на пристрої під час відсутності з'єднання і відправлення всіх сесій після відновлення з'єднання, та локалізація зі сторони серверного забезпечення. Тому хоча система повністю функціонує як було сплановано, можливі покращення взаємозв'язків між програмним забезпеченням для мобільного пристрою з сервером і зовнішнього вигляду додатку.

РОЗДІЛ 4
МОДЕЛЮВАННЯ І АНАЛІЗ РОБОТИ СИСТЕМИ

Для перевірки роботи варто перевірити як збираються та відображаються дані в інтерфейсі користувача. Як приклад для замірів був обраний алгоритм зберігання даних використаний у конкурсі [44]. Таким чином одна сесія збирання даних поділяється на 6 етапів під час яких записується активність мозку під час різних дій. У даному прикладі особа має виконати наступні рухи:

- HandStart - початок руху руки до об'єкту, наприклад, смартфону на столі.
- FirstDigitTouch - досліджувана особа має доторкнутися до об'єкта, наприклад, смартфону.
- BothStartLoadPhase - досліджувана особа має стиснути двома пальцями об'єкт, наприклад, смартфон на столі.
- LiftOff - досліджувана особа піднімає двома пальцями об'єкт у повітря, наприклад, смартфон зі столу на певну висоту.
- Replace - досліджувана особа повертає об'єкт двома пальцями назад, наприклад, повертає смартфон на стіл.
- BothReleased - досліджувана особа відпускає два пальці, наприклад, відпускає смартфон на столі.

Дані збираються з 8 каналів, які визначені міжнародними стандартами [45] (рис. 4.1), які описані в табл. 4.1.

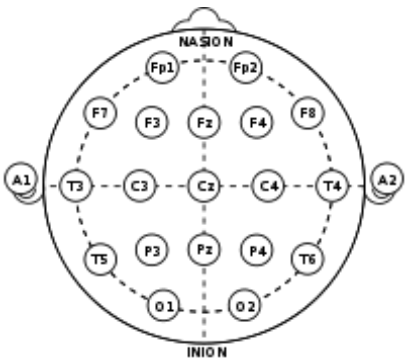


Рис. 4.1. Розміщення електродів ЕЕГ

Таблиця 4.1

Використані типи електродів та їх короткий опис

Електрод	Опис
FP2 FP1	Електроди, що розміщені у префронтальній корі. Ця частина кори головного мозку, яка охоплює передню частину лобової частки. Багато авторів вказують на цілісний зв’язок між волею людини до життя, особистістю та функціями префронтальної кори. Ця область мозку була залучена до планування складної когнітивної поведінки, самовираження особистості, прийняття рішень, модерування соціальної поведінки та модерування певних аспектів мови [46].
C4 C3	Електрод розміщений у центральній частині.
P8 P7	Електрод, що розміщений у тім'яній частині. Ця частина інтегрує сенсорну інформацію серед різних модальностей, включаючи просторовий сенс і навігацію. Основні сенсорні входи з шкіри (дотики, температури та больові рецептори) передаються через таламус до тім'яної частки. Також кілька областей тім'яної частки активно використовуються у обробці мови [47].
O2 O1	Електрод розміщений у потиличній частині. Це центр візуальної обробки мозку ссавців, що містить більшу частину анатомічної області зорової кори [47].

Були створені сенсори для кожного з каналів з детальним описом для полегшення використання. Для проведення збору даних для дослідження повинно бути залучено щонайменше 2 особи з різними ролями. Одна з осіб має бути наглядачем і буде використовувати інтерфейс для початку збору даних, а інша буде виконувати певні визначені дії за командами наглядача. Таке розподілення на ролі потрібно для того, щоб результати замірів не

були спотворені іншими видами мозкової діяльності та спостережувана особа не відволікалася на інші активності. Додатково були зібрані дані спокійного стану досліджуваної особи задля вимірювання рівня фону. Приклади виміряних даних у форматі CSV доступні у додатку до роботи. Усі зібрані дані доступні для перегляду в мобільному додатку одразу після збору.

Розроблений інтерфейс додатку полегшив хід роботи потенційного дослідження оскільки спостерігач мав змогу контролювати збір даних через мобільний пристрій не відволікаючись від спостерігача. Оскільки для успішного застосування методів машинного навчання та нейронних мереж, дані мають бути помічені, додана у додаток функція зміна стану виявилась досить корисною. Спостерігачеві був доступний вибір одного з трьох потенційних станів: до певної дії, під час певної дії і після певної дії. Кожного раз при отриманні данних мобільним пристроєм з пристрою ЕЕГ, додаток перевіряв обраний спостерігачем стан і додавав цю інформацію до даних зібраних з кожного сенсору. Таким чином дані з ЕЕГ пристрою доповнювалися даними користувача, обраним станом та сенсором, і даними з мобільного пристрою, такими як точний час і місцезнаходження, об'єднуючись у одну сесію і перетворюючись на мультимодальні. При використанні доступних методів збору інформації спостерігачеві довелося б використовувати персональний комп'ютер, який може бути відсутній у певних умовах, і самостійно помічати зібрані дані за допомогою спеціального програмного забезпечення чи вручну.

Також оскільки дані були передані та збережені на сервері, вони можуть бути отримані у будь-який час і у будь-якому місці при умові підключення до мережі. Додатково можуть бути розроблені клієнти для інших платформ, які зроблять можливим збір даних з більшої кількості пристроїв. Клієнт для браузерів полегшить отримання зібраних даних для подальшої обробки.

ВИСНОВКИ

У даній роботі була розглянута проблема збору та збереження мультимодальних даних з багатьох різних джерел. Було запропоноване рішення для цієї проблеми у вигляді мобільної системи для збору даних з декількох основних компонент: пристрою для збору, мобільного пристрою зі встановленим додатком та серверного програмного забезпечення. Перевагами даної системи перед конкурентами є наступні аспекти:

- мобільність - можливість виконувати запис даних у будь-якому місці, а не в спеціалізованих лабораторіях;
- інтегрованість - можливість збору даних з різних видів електродів і різного місця встановлення електродів, можливість збору даних з інших сенсорів;
- доступність - доступ до зібраних даних можливий у будь-якому місці з будь-яким пристроєм на основі ОС Android (у подальшому з будь-якої підтримуваної ОС);
- відкритість - на відміну від наявних закритих систем, дає можливість іншим розробникам (а не тільки автору цієї системи) створювати додатковий функціонал та набори даних (датасети) для практичного застосування;
- безпечність - безпечно інтегрувати та використовувати будь-які мультимодальні персональні біометричні дані.

У роботі були проаналізовані основні методи збору та аналізу ЕЕГ даних та основні пристрої для збору ЕЕГ даних, що доступні на ринку. Для створення зручної мобільної системи збору даних було обрано систему Ultracortex Mark 4 з Cyton Board контролером, мобільний пристрій з ОС Android та серверне програмне забезпечення розроблене мовою Java з використанням набору бібліотек Spring. Для передачі даних була використана бездротова технологія передачі даних Bluetooth, а для збереження даних була використана реляційна база даних. Для побудови архітектури серверного програмного забезпечення були використані основні

правила побудову REST веб-сервісів. Для авторизації та аутентифікації була розроблена система отримання доступу на основі JWT. Результатом роботи є прототип системи збору даних, за допомогою якого користувач може збирати дані з ЕЕГ пристрою Ultracortex Mark 4 та отримувати доступ до них на будь-якому пристрою за допомогою мобільного додатку. Основними функціями мобільного додатку є:

- створення та вхід до облікового запису;
- створення параметрів сесій (створення власних типів сенсорів, створення місць знаходження) та налаштування параметрів для кожної сесії запису даних;
- збір даних та автоматичне маркування даних (за допомогою лейблів або позначок, що можуть бути використані під час аналізу засобами машинного навчання та нейронних мереж);
- автоматичне надсилання та збереження даних на сервері;
- перегляд та отримання доступу до даних збережених на сервері;
- перегляд даних по будь-якому з сенсорів з візуалізацією у вигляді графіків;
- збереження даних на пристрою у форматі csv.

У ході роботи були виявлені напрямки для можливого покращення існуючої системи та для створення нових функціональних можливостей таких як: архівація даних для зменшення обсягу даних, що передаються через інтернет, можливість збереження даних на пристрої під час відсутності з'єднання і відправлення всіх сесій після відновлення з'єднання, та локалізація зі сторони серверного забезпечення. Серед нових функціональних можливостей можливо створення: фільтрація за певними ознаками такими як місцезнаходження, опис, сенсор, тощо, створення інтерфейсів для інших платформ та створення режиму запису даних з більшої кількості каналів.

Розроблена система може використовуватися як спеціалістами у сфері охорони здоров’я для проведення діагностики на дому в пацієнта чи у мобільних центрах, так і дослідниками для створення та зручного доступу до датасетів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Niedermeyer E. Electroencephalography: basic principles, clinical applications, and related fields / Niedermeyer E., da Silva F. L. (Ред.) // Lippincott Williams & Wilkins, 2005.
2. Larson D. E. Mayo Clinic family health book // W. Morrow, 1990.
3. Tatum W. O. Handbook of EEG Interpretation / Husain, A. M., Benbadis, S. R. // Demos Medical Publishing, 2008.
4. Nunez PL Srinivasan R. Electric fields of the brain: The neurophysics of EEG // Oxford University Press, 1981.
5. Empson J. Human brainwaves: The psychological significance of the electroencephalogram // Springer, 1986.
6. What are brainwaves? [Електронний ресурс]. - The Brainworks, 2020. – Режим доступу: https://brainworksneurotherapy.com/what-are-brainwaves.
7. J. Katona "Evaluation of the NeuroSky MindFlex EEG headset brain waves data," / J. Katona, I. Farkas, T. Ujbanyi, P. Dukan and A. Kovari // 2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI), Herl'any, 2014. - С. 91-94.
8. Mindflex Duel [Електронний ресурс]. - NeuroSky, 2020. – Режим доступу: https://store.neurosky.com/products/mindflex-duel.
9. Estermann B. Evaluation of Low-Cost EEG Hardware in a Motor Imagery Based Brain Computer Interface // 2017.
10. Ultracortex Mark IV [Електронний ресурс]. - OpenBCI, 2020. – Режим доступу: https://docs.openbci.com/docs/04AddOns/01-Headwear/MarkIV.
11. Frey J. Comparison of an open-hardware electroencephalography amplifier with medical grade device in brain-computer interface applications // 2016.
12. Hinrichs H. Comparison between a wireless dry electrode EEG system with a conventional wired wet electrode EEG system for clinical applications / Hinrichs, H., Scholz, M., Baum, A. K., Kam, J. W., Knight, R. T., & Heinze, H. J. // Scientific Reports, 2020.

13. Elmenreich W. Sensor Fusion in Time-Triggered Systems, PhD Thesis // Vienna University of Technology, 2002. - C. 173.
14. Buck J. R. Discrete-time signal processing / Buck J. R., Oppenheim A. V., Schafer R. W. // вид. Prentice Hall, 1999.
15. Капшій О. В. Вейвлет-перетворення у компресії та попередній обробці зображень / Капшій О. В., Коваль О. І., Русин Б. П. // Львів : Сполом, 2008. — 206 с.
16. Lotte F.. A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update / Lotte F., Bougrain L., Cichocki A., Cler, M., Congedo M., Rakotomamonjy A., Yger F. // Journal of neural engineering, 2018.
17. LeCun Y. Deep learning / LeCun Y., Bengio Y., Hinton G. // nature, 2015. - C. 436-444.
18. Schirrmeister R. T. Deep learning with convolutional neural networks for EEG decoding and visualization / Schirrmeister R. T., Springenberg J. T., Fiederer L. D. J., Glasstetter M., Eggensperger K., Tangermann M., Ball T. // Human brain mapping, 38(11), 2017. - C. 5391-5420.
19. Heraz A. Predicting the three major dimensions of the learner's emotions from brainwaves / Heraz A., Frasson C. // International Journal of Computer Science, 2007. - C. 187-193.
20. Mohanchandra K. Using brain waves as new biometric feature for authenticating a computer user in real-time / Mohanchandra K., Lingaraju G., Kambli P., Krishnamurthy V. // International Journal of Biometrics and Bioinformatics (IJBB), 2013.
21. Hundia, R. Brain computer interface-controlling devices utilizing the alpha brain waves // International Journal of Scientific & Technology Research, (2015). - C. 281-285.
22. Rogers R. Android application development: Programming with the Google SDK / Rogers R., Lombardo J., Mednieks Z., Meike B. // O'Reilly Media, Inc, 2009.

23. How Many Software Developers Are in the US and the World? [Електронний ресурс]. - Daxx Team, 2020. – Режим доступу: https://www.daxx.com/blog/development-trends/number-software-developers-world.
24. Java Vs Kotlin – Which Should You Choose For Android Development [Електронний ресурс]. - Hiral Atha, 2018. – Режим доступу: https://www.moveoapps.com/blog/java-vs-kotlin/.
25. Why Kotlin?[Електронний ресурс]. - Kotlin Foundation, 2020. – Режим доступу: https://kotlinlang.org/.
26. Fielding, R. T. Fielding dissertation: Chapter 5: Representational state transfer (rest) [Електронний ресурс] - Donald Bren School of Information and Computer Sciences, University of California, 2020. – Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
27. Java EE at a Glance [Електронний ресурс] - Oracle, 2020 - Режим доступу: https://www.oracle.com/java/technologies/java-ee-glance.html.
28. Johnson Rod Expert One-on-one J2EE Design and Development (First ed.) // Wrox Press, 2002. - С. 750.
29. The IoC container [Електронний ресурс]. – Spring Framework Reference Documentation, 2020. – Режим доступу: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html.
30. Spring Data JPA - Reference Documentation [Електронний ресурс]. – Spring Framework Reference Documentation, 2020. – Режим доступу: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference.
31. Java Transaction API (JTA) [Електронний ресурс]. – Oracle, 2020. – Режим доступу: https://www.oracle.com/java/technologies/jta.html.
32. Java Servlet Technology Overview [Електронний ресурс]. – Oracle, 2020. – Режим доступу: https://www.oracle.com/java/technologies/servlet-technology.html.

33. Beynon-Davies P. Database systems // Basingstoke, UK: Palgrave Macmillan, 2004 - С. 61.
34. Date C. J. Relational database writings, 1985-1989, Volume 1. Addison-Wesley, 1990.
35. Kent William A Simple Guide to Five Normal Forms in Relational Database Theory // Communications of the ACM, 1983. - С. 120–125.
36. Structured Query Language (SQL) [Електронний ресурс]. – Microsoft, 2020. – Режим доступу: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?redirectedfrom=MSDN&view=sql-server-ver15.
37. Core NOSQL Systems [Електронний ресурс]. – Hosting Data, 2015. – Режим доступу: https://hostingdata.co.uk/nosql-database/.
38. J. S. van der Veen Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual / J. S. van der Veen, B. van der Waaij, R. J. Meijer // 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, 2012. - С. 431-438.
39. Berners-Lee Tim HyperText Transfer Protocol // World Wide Web Consortium, 2010.
40. IPv6 over BLUETOOTH(R) Low Energy [Електронний ресурс]. – Internet Engineering Task Force (IETF), 2020. – Режим доступу: https://tools.ietf.org/html/rfc7668.
41. Andrew S. Tanenbaum Computer Networks 5th Edition / Andrew S. Tanenbaum , David J. Wetherall // Pearson, 2010 - С. 960.
42. JSON Web Token (JWT) [Електронний ресурс]. – Internet Engineering Task Force (IETF), 2020. – Режим доступу: https://tools.ietf.org/html/rfc7519#section-3.
43. The Security Filter Chain [Електронний ресурс]. – Spring Framework Reference Documentation, 2020. – Режим доступу: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html.

44. Grasp-and-Lift EEG Detection [Електронний ресурс]. – Kaggle, 2020. – Режим доступу: https://www.kaggle.com/c/grasp-and-lift-eeg-detection/data.
45. Jaakko Malmivuo Bioelectromagnetism. 13. Electroencephalography // Principles and Applications of Bioelectric and Biomagnetis Fields, Oxford University Press, 1995. - С. 247-264.
46. Yang Y. Prefrontal structural and functional brain imaging findings in antisocial, violent, and psychopathic individuals: a meta-analysis / Yang Y., Raine A. // Psychiatry Research, 2009.
47. SparkNotes: Brain Anatomy: Parietal and Occipital Lobes [Електронний ресурс]. – Sparknotes, 2020. – Режим доступу: https://web.archive.org/web/20071231064003/http://www.sparknotes.com/psychology/neuro/brainanatomy/section5.rhtml.

ДОДАТОК А. Код програми.

1. Приклад розробленого контролеру для одного з ендпоінтів.

```
@RestController
@RequestMapping("/api_sensors")
public class SensorController {

    @Autowired
    private SensorService service;

    @GetMapping("/sensors")
    public List<SensorDto> getEvents() {
        return service.findAll();
    }

    @GetMapping(path = "/sensors/{id}")
    public SensorDto getSensorById(@PathVariable("id") long sensorId) {
        return service.find(sensorId);
    }

    @PostMapping("/sensors/add")
    public SensorDto createMeasurementSession(@RequestBody ChannelSensor sensor) {
        return service.save(sensor);
    }

    @PutMapping(path = "/sensors/update/{id}")
    public SensorDto updateMeasurementSession(@PathVariable("id") long sensorId,
@RequestBody ChannelSensor newSensor) {
        return service.update(sensorId, newSensor);
    }

    @DeleteMapping(path = "/sensors/delete/{id}")
    public void deleteSessions(@PathVariable("id") long id) {
        service.delete(id);
    }

}
```

2. Приклад сервісу для обробки даних користувача.

```
@Service
public class UserService implements IService<User> {

    @Autowired
    private BcryptPasswordEncoder encoder;

    @Autowired
```

```
private UserRepository repository;

@Autowired
private JwtTokenUtil jwtTokenUtil;

@Autowired
private UserMapper userMapper;


@Override
public List<User> findAll(int size, int page) {
   return repository.findAll(new PageRequest(page, size)).getContent();
}

@Override
public User save(User entity) {
   if (repository.findByLogin(entity.getLogin()).isPresent()) {
      throw new RuntimeException();
   }
   entity.setRefreshId(UUID.randomUUID().toString());
   return repository.save(entity);
}

public FullUserDto getUser(long id) {
   return userMapper.convertToFullUserDto(find(id));
}

@Override
public User find(long id) {
   return repository.findById(id)
         .orElseThrow(UserNotFoundException::new);
}

@Override
public User update(long id, User entity) {
   return null;
}

@Transactional
public TokenResponse login(User user) {
   User persistedUser = repository
         .findByLogin(user.getLogin())
         .orElseThrow(() -> new RuntimeException(«No user with given login exists»));
```

```
        if (!encoder.matches(user.getPassword(), persistedUser.getPassword())) {
            throw new RuntimeException(«No user with given login and pass exists»);
        }
        System.out.println(persistedUser);
        TokenResponse token = new
TokenResponse(jwtTokenUtil.generateToken(persistedUser),
                jwtTokenUtil.generateRefresh(persistedUser));
        repository.save(persistedUser);

        return token;


    }

    @Transactional
    public SmallUserDto register(RegisterDTO user) {
        if (!user.getConfirmPassword().equals(user.getPassword())) {
            throw new RuntimeException(«Password is incorrect»);
        }
        User userEntity = User.builder()
                .email(user.getEmail())
                .firstName(user.getFirstName())
                .secondName(user.getSecondName())
                .login(user.getLogin())
                .password(encode(user.getPassword()))
                .priveleges(Arrays.asList(«READ_SESSIONS», «WRITE_SESSIONS»))
                .build();

        save(userEntity);
        return userMapper.convertToSmallUserDto(userEntity);
    }

    @Transactional
    public User findByRefreshToken(RefreshToken tokenIn) {

        long id = 0;
        String token = tokenIn.getToken();
        String refreshId = «»;
        if (token != null) {
            try {
                refreshId = jwtTokenUtil.getRefreshIdFromToken(token);
            } catch (IllegalArgumentException e) {
                System.out.println(«an error occured during getting username from token» + e);
            } catch (SignatureException e) {
                System.out.println(«Authentication Failed. Username or Password not valid.»);
```

```
            }
        } else {
            System.out.println(«couldn’t find bearer string, will ignore the header»);
        }
        System.out.println(refreshId);
        User user = repository.findByRefreshId(refreshId);

        repository.save(user);
        return user;
    }

    @Transactional
    public TokenResponse refresh(RefreshToken tokenIn) {


        String token = tokenIn.getToken();
        User user = null;
        try {
            user = findByRefreshToken(tokenIn);
        } catch (Throwable throwable) {
            System.out.println(«error» + throwable.getMessage());
            throw new UserNotFoundException(«Invalid token»);
        }
        if (!jwtTokenUtil.validateToken(token, user))
            throw new RuntimeException();
        if (user == null) {
            throw new RuntimeException();
        }
        System.out.println(user);
        TokenResponse tokenResponse = new
TokenResponse(jwtTokenUtil.generateToken(user),
                jwtTokenUtil.generateRefresh(user));

        return tokenResponse;
    }

    private String encode(String s) {
        return encoder.encode(s);
    }

    public List<RegisteredUserDtoWithToken> getUsers(Pageable pageable) {
        return repository.findAll(pageable)
                .stream()
                .map(userMapper::convertToRegisteredUserDtoWithToken)
                .collect(Collectors.toList());
```

```
    }


    public FullUserDto getUserInfo() {
        User userRequester = getRequester();
        User persistedUser =
repository.findByLogin(userRequester.getLogin()).orElseThrow(UserNotFoundException::new
);
        return userMapper.convertToFullUserDto(persistedUser);
    }

    public List<FullUserDto> find(String search, Pageable pageable) {
        return repository.searchUsers(search.toUpperCase(), pageable)
                .stream()
                .map(user -> userMapper.convertToFullUserDto(user))
                .collect(Collectors.toList());
    }

    public User getRequester() {
        return (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    }

}
```

3. Приклад фільтру обробки JWT для ланцюга фільтрів.

```
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    public static final String USERNAME_FROM_TOKEN = "an error occured during getting
username from token";
    public static final String EXPIRED_TOKEN = "the token is expired and not valid anymore";
    @Autowired
    private UserRepository repository;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain
chain) throws IOException, ServletException {
        String token = jwtTokenUtil.resolveToken(req);
        boolean isCont = true;
        String username = null;
```

```
        long id = 0;
        if (token != null) {
            try {
                id = jwtTokenUtil.getIdFromToken(token);
            } catch (IllegalArgumentException e) {
                logger.error(USERNAME_FROM_TOKEN, e);
            } catch (ExpiredJwtException e) {
                logger.warn(EXPIRED_TOKEN, e);
                ((HttpServletResponse) res).sendError(HttpServletResponse.SC_UNAUTHORIZED,
"The token is not valid.");
                res.getWriter().write("{\"msg\":\"token is not valid\"}");
                isCont = false;
            } catch (SignatureException e) {
                logger.error("Authentication Failed. Username or Password not valid.");
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            logger.warn("couldn't find bearer string, will ignore the header");
        }
        if (id != 0 && SecurityContextHolder.getContext().getAuthentication() == null) {

            User user = repository.findById(id)
                    .orElseThrow(UserNotFoundException::new);

            if (jwtTokenUtil.validateToken(token, user)) {
                TokenAuthentication authentication = new TokenAuthentication(token,
user.getAuthorities(), true, user);
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(req));
                logger.info("authenticated user " + user + ", setting security context");
                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        }
        if (isCont) {
            chain.doFilter(req, res);
        }
    }
}
```

4. Приклад сервісу для створення та обробки даних JWT.

```
@Component
public class JwtTokenUtil implements Serializable {
```

```
    @Value("${accessToken.validity}")
    private String ACCESS_TOKEN_VALIDITY_SECONDS;

    @Value("${refreshToken.validity}")
    private String REFRESH_TOKEN_VALIDITY_SECONDS;

    public long getIdFromToken(String token) throws ExpiredJwtException {
        return Long.parseLong(getClaimFromToken(token, Claims::getSubject));
    }

    public String getRefreshIdFromToken(String token) {
        final Claims claims = getAllClaimsFromToken(token);
        System.out.println(claims);
        return (String) claims.get("refreshId");
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver)
throws ExpiredJwtException {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) throws ExpiredJwtException {
        return Jwts.parser()
                .setSigningKey(SIGNING_KEY)
                .parseClaimsJws(token)
                .getBody();
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }

    public String generateRefresh(User user) {

        Claims claims = Jwts.claims().setSubject(Long.toString(user.getId()));
        claims.put("refreshId", user.getRefreshId());

        return Jwts.builder()
                .setClaims(claims)
```

```
                .setIssuer("https://devglan.com")

                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() +
Long.parseLong(REFRESH_TOKEN_VALIDITY_SECONDS) * 1000))
                .signWith(SignatureAlgorithm.HS256, SIGNING_KEY)
                .compact();

    }

    public String generateToken(User user) {
        return doGenerateToken(user);
    }

    private String doGenerateToken(User user) {

        Claims claims = Jwts.claims().setSubject(Long.toString(user.getId()));
        claims.put("scopes", user.getAuthorities()
                .stream()
                .map(x -> x.getAuthority())
                .collect(Collectors.toList()));
        claims.put("name", user.getLogin());

        return Jwts.builder()
                .setClaims(claims)
                .setIssuer("https://devglan.com")
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() +
Long.parseLong(ACCESS_TOKEN_VALIDITY_SECONDS) * 1000))
                .signWith(SignatureAlgorithm.HS256, SIGNING_KEY)
                .compact();
    }

    public Boolean validateToken(String token, User userDetails) {
        try {
            final long id = getIdFromToken(token);
            return id == userDetails.getId() && !isTokenExpired(token);
        } catch (Throwable throwable) {
            System.out.println("PIZDEC");
            return false;
        }
    }

    public String resolveToken(HttpServletRequest req) {
        String bearerToken = req.getHeader("Authorization");
```

```
        if (bearerToken != null && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }

}
```

5. Приклад адаптеру для відображення списку у мобільному додатку.

```
public class RVAdapter extends AMyAdapter<SessionDto> {


    private List<SessionDto> sessionDtos;

    public void setList(List<SessionDto> l){
        sessionDtos = l;
    }


    @Override
    public void addList(List<SessionDto> l) {
        sessionDtos.addAll(l);
    }

    public List<SessionDto> getList(){
        return sessionDtos;

    }
    WeakReference<Context> context;
    WeakReference<Activity> activity;
    WeakReference<Application> app;

    public RVAdapter(Application app, Activity activity) {
        this.sessionDtos = new ArrayList<>();
        this.app = new WeakReference<Application>(app);
        this.activity = new WeakReference<>(activity);
    }



    @Override
    public int getItemCount() {
        return sessionDtos == null ? 1 : sessionDtos.size() + 1;
    }
```

```
    @Override
    public void onBindViewHolder(@NonNull final RecyclerView.ViewHolder
personViewHolder, int i) {
        super.onBindViewHolder(personViewHolder, i);
        // Loader ViewHolder


    }

    RecyclerView.ViewHolder createViewHolderMy(ViewGroup viewGroup){
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.session_card,
viewGroup, false);
        return new SessionViewHolder(v);
    }

    @Override
    void renderPVH(final RecyclerView.ViewHolder personViewHolder, int i){
        SessionViewHolder svh = (SessionViewHolder) personViewHolder;
        svh.name.setText(sessionDtos.get(i).getTitle());
        svh.desc.setText(sessionDtos.get(i).getDescription());
        svh.id.setText(Long.toString(sessionDtos.get(i).getId()));

        svh.start.setText(Constants.mFormatterDateShort.format(new
Date(sessionDtos.get(i).getStartDate())));
        svh.end.setText(Constants.mFormatterDateShort.format(new
Date(sessionDtos.get(i).getEndDate())));
        final int var = i;
        svh.cv.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                SessionDto sessionDto = sessionDtos.get(var);
                ((MyApplication) app.get()).setSelectedSession(sessionDto);

                Bundle bundle = new Bundle();
                bundle.putSerializable("session", sessionDto);
                DetailsDeciderFactory.create(sessionDto).call(bundle,
((AppCompatActivity)activity.get()).getSupportFragmentManager());

            }
        });

        svh.cv.setId((int) sessionDtos.get(i).getId());

    }
```

```
    @Override
    public void onAttachedToRecyclerView(RecyclerView recyclerView) {
        super.onAttachedToRecyclerView(recyclerView);
    }

    @Override
    public void notifyList(){
        notifyDataSetChanged();
    }


    @Override
    public SessionDto findById(long id) {
        for(SessionDto u: sessionDtos){
            if(u.getId() == id){
                return u;
            }
        }
        return null;
    }



    @Override
    long getYourItemId(int position){
        return sessionDtos.get(position).getId();
    }


}
```

6. Приклад сервісу для визначення статусу відповіді та оновлення токенів.

```
public class TokenAuthenticator implements Authenticator {
    WeakReference<Context> c;

    TokenAuthenticator(Context c){
        this.c = new WeakReference<>(c);
    }
    @Override
    public Request authenticate(Route route, Response response) throws IOException {
        if (response.code() == 401) {
            if(!Auth.isRefreshing()) {
                Auth.setRefreshing(true);
```

```
            RefreshTokenService taskService =
ServiceGenerator.createService(RefreshTokenService.class);
            HashMap<String, String> hm = new HashMap<>();
            hm.put("token", Auth.getInstance(c.get()).getRefreshToken());
            Call<LoginResponse> call = taskService.getToken(new
RefreshToken(Auth.getInstance(c.get()).getRefreshToken()));


            retrofit2.Response<LoginResponse> refreshResponse = call.execute();

            Auth.setRefreshing(false);
            if (refreshResponse != null && refreshResponse.code() == 200) {
               LoginResponse r = refreshResponse.body();
               Auth.update(c.get(), r);
               Auth.getInstance(c.get()).setAccessToken(r.getAccessToken());

               Auth.getInstance(c.get()).setRefreshToken(r.getRefreshToken());
               return response.request().newBuilder()
                     .header("Authorization", "Bearer " + r.getAccessToken())
                     .build();
            } else {
               Auth.setRefreshing(false);
               return null;
            }
         }
         while(Auth.isRefreshing()){

         }
         return response.request().newBuilder()
               .header("Authorization", "Bearer " +  Auth.getInstance(c.get()).getAccessToken())
               .build();
      }
      return null;
   }
}
```

7. Приклад класу для візуалізації інформації про сесію запису.

```
public class DetailsRender {

   LinearLayoutManager llm;

   private WeakReference<Activity> activityWeakReference;
   private View root;
```

```
    private SessionDto sessionDtoSelected;
    public DetailsRender(Activity a, View root){
        activityWeakReference = new WeakReference<>(a);
        this.root = root;
    }

    public void render(SessionDto sessionDto) {
        sessionDtoSelected = sessionDto;
        renderText(sessionDtoSelected);
        renderButtons(sessionDtoSelected);
    }

    private void renderText(SessionDto sessionDto) {
        TextView descTv = root.findViewById(R.id.desc);
        TextView startDateTv = root.findViewById(R.id.start_date);
        TextView endDateTv = root.findViewById(R.id.end_date);
        TextView title = root.findViewById(R.id.title);


        TextView placeTv = root.findViewById(R.id.location_name);
        TextView placeTv2 = root.findViewById(R.id.location_desc);

        title.setText(sessionDto.getTitle());
        descTv.setText(sessionDto.getDescription());

        startDateTv.setText(Constants.mFormatterDate.format(new
Date(sessionDto.getStartDate())));
        endDateTv.setText(Constants.mFormatterDate.format(new
Date(sessionDto.getEndDate())));
        placeTv.setText(sessionDto.getUserLocation().getDescription());
        placeTv2.setText(sessionDto.getUserLocation().getName());
    }

    private void renderButtons(final SessionDto sessionDto) {
        final Activity activity = activityWeakReference.get();
        llm = new LinearLayoutManager(activity.getApplicationContext());
        RecyclerView recyclerView = root.findViewById(R.id.rv);
        final RVAdapterSensorDto rvAdapter = new
RVAdapterSensorDto(activity.getApplication(), activity, sessionDto);
        recyclerView.setLayoutManager(llm);

        //rv.setNestedScrollingEnabled(false);
        recyclerView.setHasFixedSize(true);
        recyclerView.stopScroll();
```

```
        recyclerView.stopNestedScroll();
        recyclerView.setAdapter(rvAdapter);
        rvAdapter.setList(sessionDto.getSensors());

    }

}
```

8. Приклад класу для налаштування типів сенсорів для каналів запису.

```
public class ElevenChannelsService implements IChannelService {

    ElevenChannelsService() {
        channelsList = new ArrayList<>();
        channelsList.add(new Channels(R.id.ch1, "ch1"));
        channelsList.add(new Channels(R.id.ch2, "ch2"));
        channelsList.add(new Channels(R.id.ch3, "ch3"));
        channelsList.add(new Channels(R.id.ch4, "ch4"));

        channelsList.add(new Channels(R.id.ch5, "ch5"));
        channelsList.add(new Channels(R.id.ch6, "ch6"));
        channelsList.add(new Channels(R.id.ch7, "ch7"));
        channelsList.add(new Channels(R.id.ch8, "ch8"));

        channelsList.add(new Channels(R.id.ch9, "ch9"));
        channelsList.add(new Channels(R.id.ch10, "ch10"));
        channelsList.add(new Channels(R.id.ch11, "ch11"));
    }
    List<Channels> channelsList;
    @Override
    public int getViewId() {
        return R.layout.activity_settings;
    }

    @Override
    public List<Channels> getChannels() {
        return channelsList;
    }

    @Override
    public List<ChannelDataDto> getChannelDataDtos() {
        List<ChannelDataDto> channelDataDtos = new ArrayList<>();
        for (Channels channels: channelsList) {
            channelDataDtos.add(channels.getChannelDataDto());
        }
        return channelDataDtos;
```

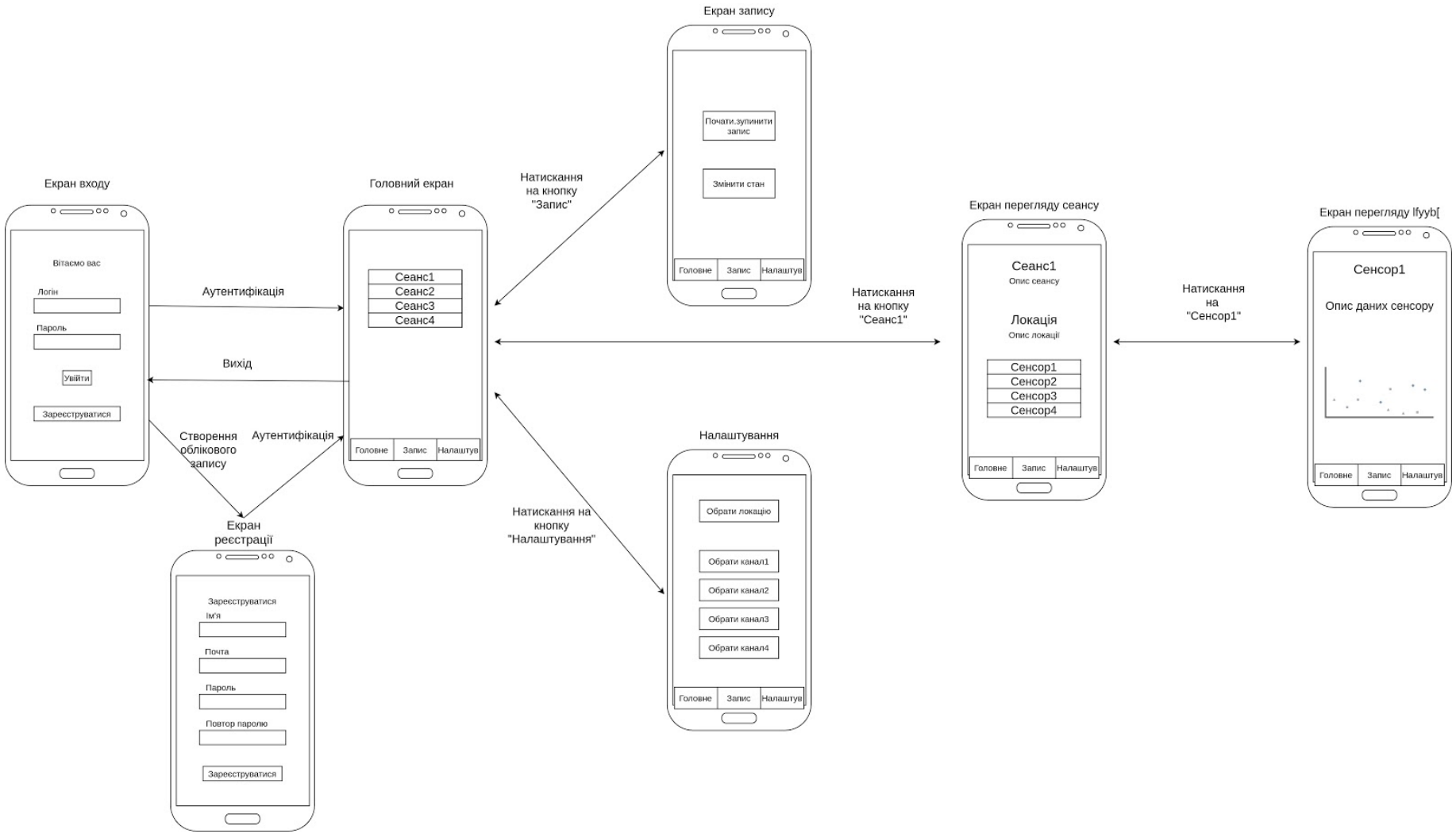
```
    }

    @Override
    public String getCsvHeaderChannels() {
        return null;
    }

    @Override
    public void clear() {
        for(Channels channels: channelsList) {

channels.setChannelDataDto(ChannelDataDto.builder().id(channels.getSensorDtoId()).build());
        }
    }

    @Override
    public void initializeWithAdapters(View v, Context context, List<SensorDto> sensorDtos) {
        for(Channels channels: channelsList) {
            Spinner spinner = v.findViewById(channels.getSpinnerViewId());
            ChooseSensorAdapter chooseSensorAdapter = new ChooseSensorAdapter(context,
sensorDtos);
            if(sensorDtos.size() > 0) {
                channels.getChannelDataDto().setSensor(chooseSensorAdapter.getItem(0));
            } else {
                channels.getChannelDataDto().setSensor(new SensorDto());
            }
            spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
                @Override
                public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l)
{
                    channels.getChannelDataDto().setSensor(chooseSensorAdapter.getItem(i));
                }
                @Override
                public void onNothingSelected(AdapterView<?> adapterView) {

                }
            });
            spinner.setAdapter(chooseSensorAdapter);
        }
    }

    @Override
    public void writeValue(String[] nextLine) {
        Integer timestamp = Integer.parseInt(nextLine[0]);
        Integer statusBefore = Integer.parseInt(nextLine[1]);
```

```
        Integer statusDuring = Integer.parseInt(nextLine[2]);
        Integer statusAfter = Integer.parseInt(nextLine[3]);
        DataStatus dataStatus = DataStatus.BEFORE;
        if (statusBefore == 1) {
            dataStatus = DataStatus.BEFORE;
        } else if (statusDuring == 1) {
            dataStatus = DataStatus.DURING;
        } else if (statusAfter == 1) {
            dataStatus = DataStatus.AFTER;
        }

        Integer number = Integer.parseInt(nextLine[4]);

            for(int i = 6; i < 17; i++) {
                channelsList.get(i-6).getChannelDataDto().getData().add(DataDto.builder()
                        .ts(timestamp)
                        .d(dataStatus.getSet_i())
                        .n(number)
                        .v(Double.parseDouble(nextLine[i])).build());
            }

    }

    public List<SensorDto> getSensorDtos() {
        Set<SensorDto> sensorDtos = new HashSet<>();
        for (Channels channels: channelsList) {
            System.out.println("Sensor + " + channels.getChannelDataDto().getSensor());
            sensorDtos.add(channels.getChannelDataDto().getSensor());
        }

        List<SensorDto> sensorDtosList = new ArrayList<>(sensorDtos);
        System.out.println(sensorDtosList);
        return sensorDtosList;
    }

    public boolean isInitialized() {
        for(Channels channels: channelsList) {
            if (channels.getChannelDataDto().getSensor() == null) {
                return false;
            }
        }
        return true;
    }
```

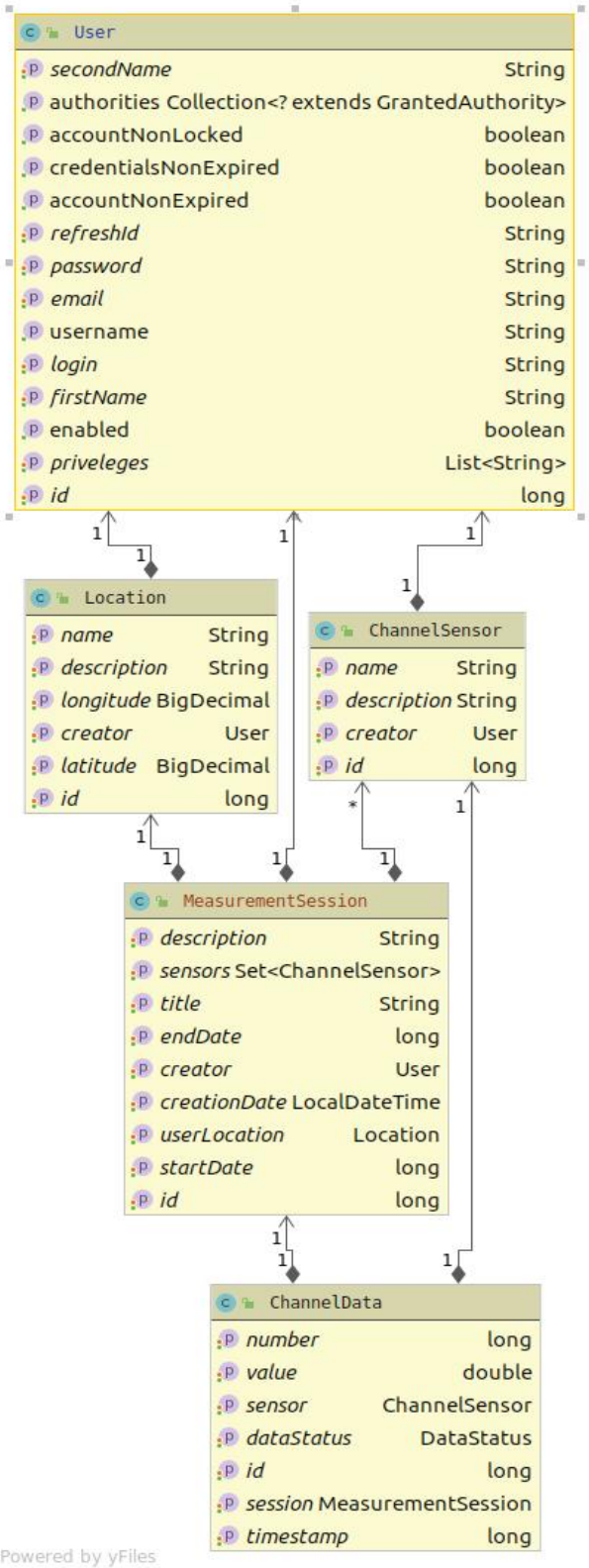
```
<?xml version="1.0" encoding="utf-8"?>

<ScrollView android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:paddingTop="56dp"
        android:paddingLeft="24dp"
        android:paddingRight="24dp"
        android:focusable="true"
        android:focusableInTouchMode="true">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:layout_gravity="end">


            <androidx.appcompat.widget.AppCompatImageButton
                android:id="@+id/languages"
                android:layout_width="30dp"
                android:layout_height="30dp"
                android:src="@drawable/ic_unitedkingdom"
                android:background="#00ffffff"/>

        </LinearLayout>

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="72dp"
            android:layout_marginBottom="24dp"
            android:layout_gravity="center_horizontal"/>

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"

            android:layout_marginTop="8dp"
            android:theme="@style/TextLabel"
            android:layout_marginBottom="8dp">
```

```
    <EditText android:id="@+id/editLogin"

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"
        android:text="nikita"
        android:hint="@string/email"/>

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:theme="@style/TextLabel"
    android:layout_marginBottom="8dp">

    <EditText

        android:id="@+id/editPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:text="nikita"
        android:hint="@string/password"/>

</com.google.android.material.textfield.TextInputLayout>

<Button
    android:id="@+id/buttonLogin"
    android:layout_gravity="center_horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="12dp"
    android:layout_marginTop="24dp"
    android:layout_marginBottom="24dp"
    android:text="@string/login"
    />
<Button
    android:visibility="gone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="skip"
    android:text="@string/skip"/>
<TextView
```

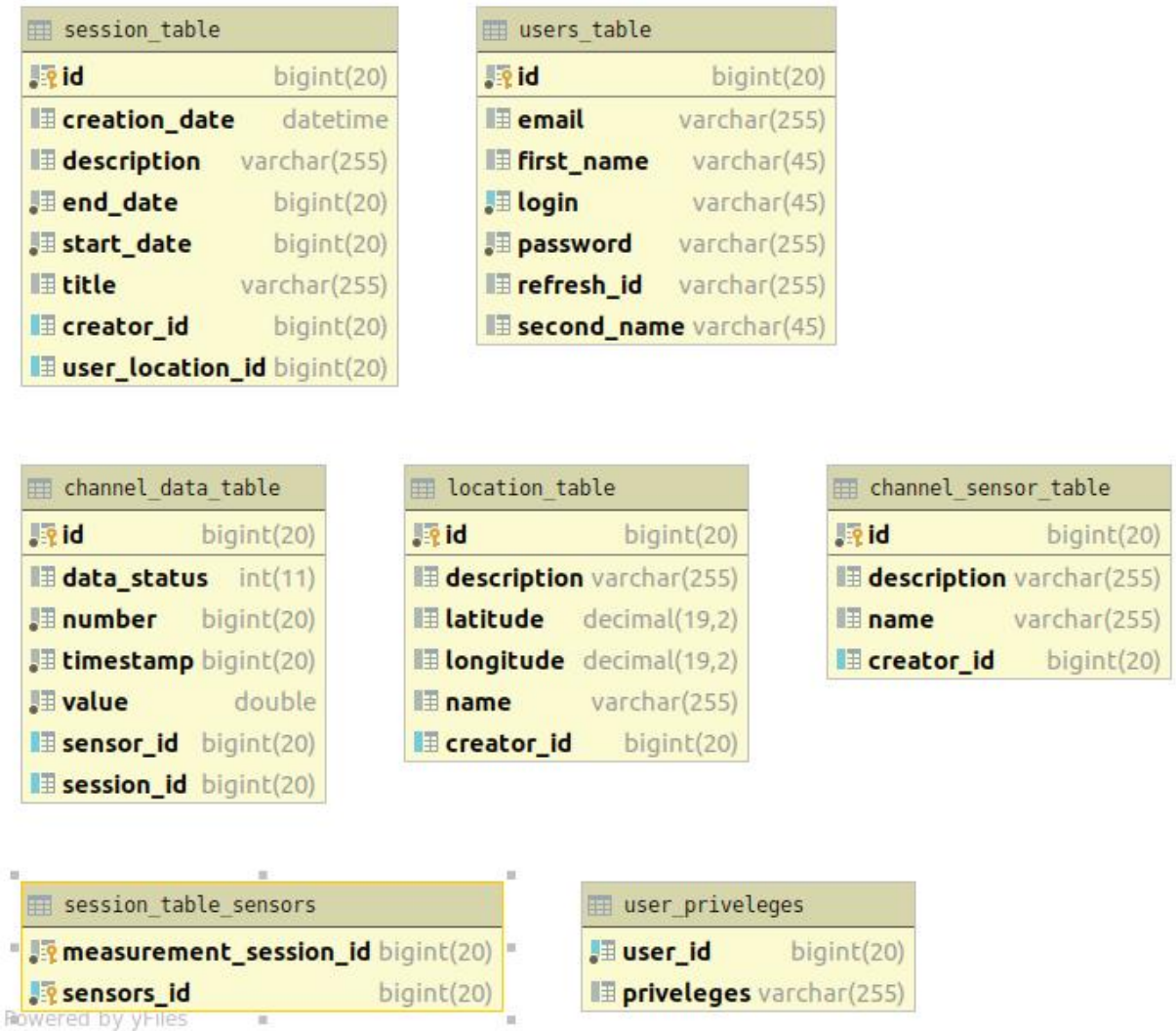
```
            android:clickable="true"
            android:focusable="true"
            android:onClick="signUpActivity"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="24dp"
            android:gravity="center"
            android:text="@string/no_account"
            />

    </LinearLayout>

</ScrollView>
```



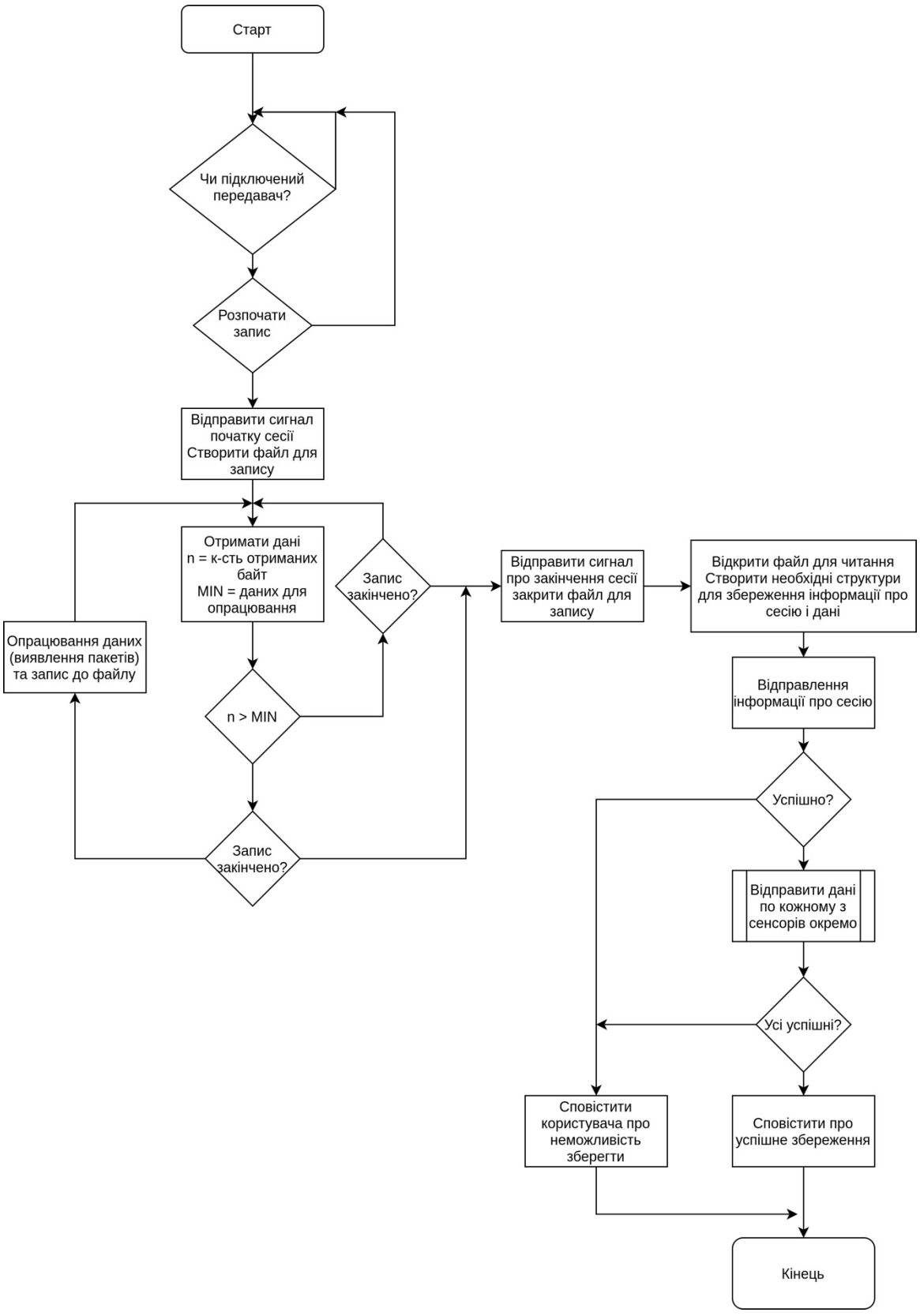
					ДП 6404. 00.004 Д1	
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*		
		Гордієнко Н.Ю.			*Схема інтерефейсу мобільного додатку.*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив.		*Роковий О.П.*				1 / 1
.						
Н. Контр.		*Сімоненко В.П*				*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*
Затвердив.						



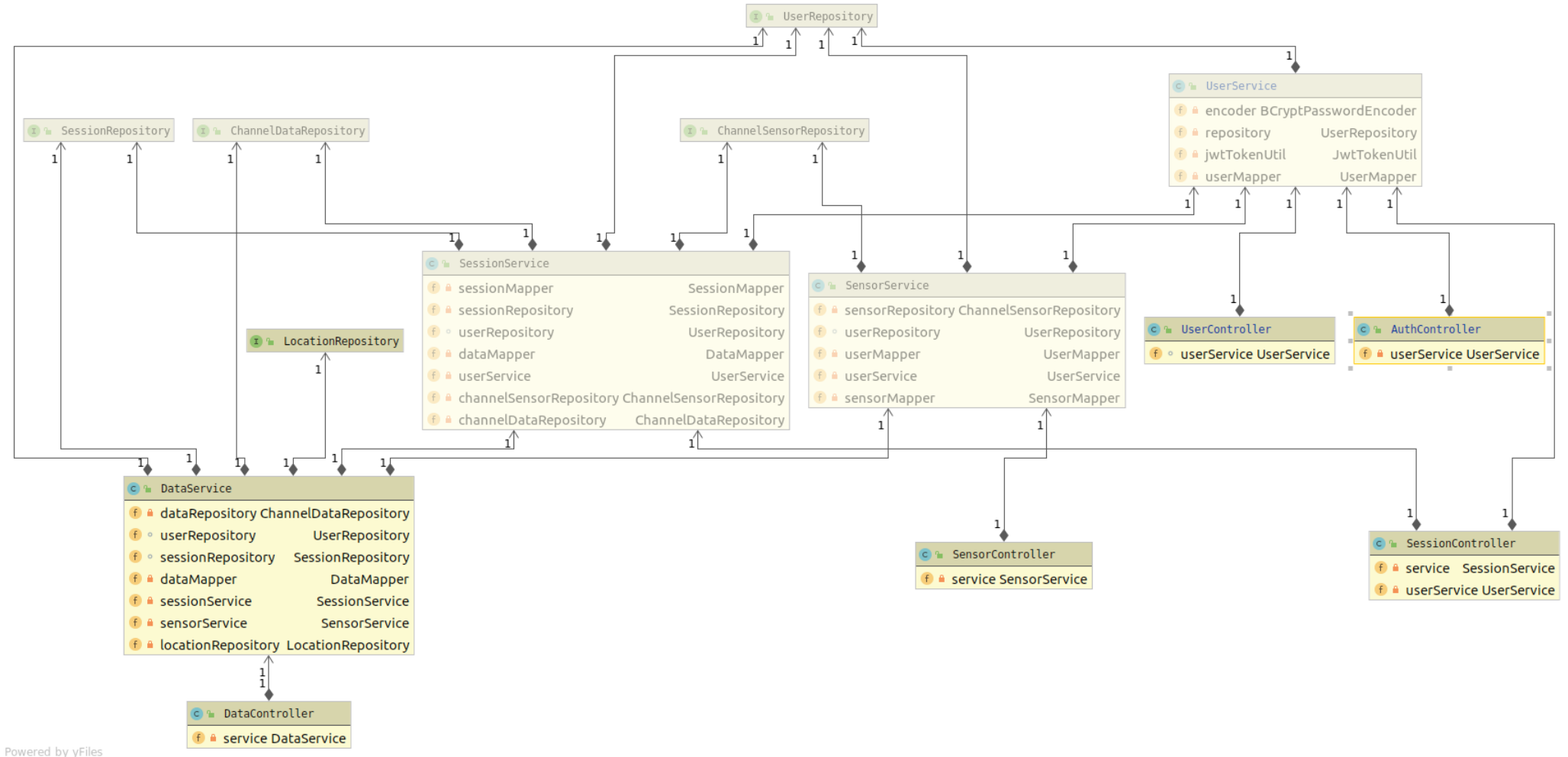
					ДП 6404. 00.005 Д2	
Зм.	*Арк*	*№ докум.*	*Підпис*	*Дата*		
		Гордієнко Н.Ю.			*Діаграма класів сутностей використаних зі сторони серверного ПЗ.*	*Літ.* / *Аркуш* / *Аркушів*: 1 / 1
Перевірив.		*Роковий О.П.*				
.						*НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64*
Н. Контр.		*Сімоненко В.П*				
Затвердив.						



					ДП 6404. 00.006 Д3		
Зм.	Арк	№ докум.	Підпис	Дата			
		Гордієнко Н.Ю.			Діаграма таблиць у створеній базі даних.	Літ.	Аркуш 1 / Аркушів 1
Перевірив.		Роковий О.П.					
.						НТУУ “КПІ ім. Ігоря Сікорського” ФІОТ гр. ІП-64	
Н. Контр.		Сімоненко В.П.					
Затвердив.							



					ДП 6404. 00.007 Д4	
Зм.	Арк	№ докум.	Підпис	Дата		
		Гордієнко Н.Ю.			Принципова схема реалізації методу.	Літ. / Аркуш 1 / Аркушів 1
Перевірив.		Роковий О.П.				
.						НТУУ “КПІ ім. Ігоря Сікорського” ФІОТ гр. ІП-64
Н. Контр.		Сімоненко В.П				
Затвердив.						



					ДП 6404. 00.008 Д5	
Зм.	Арк	№ докум.	Підпис	Дата		
		Гордієнко Н.Ю.			Структурна схема серверного програмного забезпечення.	Літ. / Аркуш 1 / Аркушів 1
Перевірив.		Роковий О.П.				НТУУ "КПІ ім. Ігоря Сікорського" ФІОТ гр. ІП-64
.						
Н. Контр.		Сімоненко В.П.				
Затвердив.						