НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «*Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей*»

Виконав: студент *IV* курсу, групи *ІС-73*

Великий Данило Євгенович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Баклан Ігор Всеволодович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Сперкач Майя Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц., к.т.н. доц. Резніков Сергій Анатолійович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

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

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2021 р.

З А В Д А Н Н Я
на дипломний проєкт студенту

Великому Данилу Євгеновичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей*»

керівник проєкту Баклан Ігор Всеволодович*, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*11*” *05* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до

технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна варіантів використання

2. Схема бази даних

3. Схема прикладу коду для обрахування індексу небезпеки

4. Схема структурна класів програмного забезпечення

5. Схема структурна послідовності

6. Схема структурна компонентів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «*7*» *квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*12.02.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*20.02.2021*	
3.	*Постановка та формалізація задачі*	*03.03.2021*	
4.	*Розробка інформаційного забезпечення*	*16.03.2021*	
5.	*Алгоритмізація задачі*	*20.03.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*29.03.2021*	
7.	*Розробка програмного забезпечення*	*14.04.2021*	
8.	*Налагодження програми*	*22.04.2021*	
9.	*Виконання графічних документів*	*29.04.2021*	
10.	*Оформлення пояснювальної записки*	*04.05.2021*	
11.	*Подання ДП на попередній захист*	*14.05.2021*	
12.	*Подання ДП на основний захист*	*04.06.2021*	
13.	*Подання ДП рецензенту*	*07.06.2021*	

Студент Данило ВЕЛИКИЙ

Керівник Ігор БАКЛАН

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7302.00.000 ПЗ	Пояснювальна записка	88	
3	А4	ДП 7302.01.000 ТЗ	Технічне завдання	7	
4	А3	ДП 7302.02.000 ССВ	Схема структурна варіантів	1	
			використання		
5	А3	ДП 7302.03.000 СБД	Схема бази даних	1	
6	А3	ДП 7302.04.000 СПК	Схема прикладу коду	1	
			для обрахування індексу небезпеки		
7	А3	ДП 7302.05.000 ССКЛ	Схема структурна класів	1	
			програмного забезпечення		
8	А3	ДП 7302.06.000 ССП	Схема структурна послідовності	1	
9	А3	ДП 7302.07.000 ССКО	Схема структурна компонентів	1	
			програмного забезпечення		

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7302 00.000.00		
Розробн.		Великий Д.Є.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Баклан І.В.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73	
Н/контр.		Сперкач М.О.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: «Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей»

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 11 рисунків, 11 таблиць, 1 додаток, 29 джерел.

Дипломний проєкт присвячений розробці інформаційної системи з виявлення небезпечних місць скупчення великої кількості людей.

Основною ціллю розробки є попередження про скупчення людей, які можуть бути причиною захворювань вірусів, зокрема коронавірусу SARS-CoV-2 [1].

У розділі інформаційного забезпечення наведена структура вхідних та вихідних даних, описана структура бази даних.

Розділ математичного забезпечення присвячений опису алгоритмів та математичної моделі, які використовуються в системі.

У розділі програмного забезпечення наведені діаграми класів, послідовності, компонентів, специфікацію функцій. Наведений опис архітектури системи, програмного та апаратного забезпечення.

У технологічному розділі описана інструкція користувача та приведені результати тестування інформаційної системи.

КОРОНАВІРУС, ГЕОЛОКАЦІЯ, НАТОВП, ВАКЦИНАЦІЯ, ІНФОРМАЦІЙНА СИСТЕМА

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, containing 11 figures, 11 tables, 1 application, 29 sources.

The diploma project is devoted to the development of an information system for identification dangerous crowded places.

The main goal of the development is to prevent the crowds of people, which can cause diseases of viruses, including coronavirus SARS-CoV-2.

The section of the information support describes the structure of input and output data, provides structure of the database.

The section of mathematical support is devoted to the description of algorithms and mathematical model used in the system.

The software section contains diagrams of classes, sequences, components, specification of functions. Provided description of system architecture, software and hardware.

The technology section describes the user manual and the information system tests results are given.

CORONAVIRUS, GEOLOCATION, CROWD, VACCINATION, INFORMATION SYSTEM

ЗМІСТ

ВСТУП

Найголовнішою подією 21 століття, яка торкнулася кожної людини є пандемія коронавірусу COVID-19. Скоріш за все, вірус SARS-CoV-2 є випадковою мутацією свого попередника, який був вірусом, що наявний в організмах кажанів, проте не завдавав їм шкоди. Попередник SARS-CoV-2 існував протягом певного часу, поки не мутував і почав заражати людей, а згодом спричинив світову пандемію. Наслідки пандемії коронавірусу катастрофічні: світові економічні, соціальні та політичні кризи, серії локдаунів та карантинів у країнах, тотальний колапс медичних та фінансових систем, незчисленні смерті людей по всіх континентах.

Влади країн запровадили численні обмеження заради протидії поширення вірусу. В ці обмеження входять закриття муніципальних закладів, заборона виходити з дому під час карантину, вето на зібрання багатьох людей, зупинення роботи транспортної системи тощо. Оскільки немає вагомих протидій коронавірусу, люди проводять профілактичні стримуючі заходи: ходять в масках, не збираються в натовпи тощо. Наразі вакцини на стадії розвитку, хоч і декілька партій вже є у країнах.

Дипломний проєкт присвячений розробці інформаційної системи, яка попереджає про небезпечні місця скупчення великої кількості людей, які є найбільш загрозливими розповсюдженнями коронавірусу та інших вірусів, які передаються повітряно-крапельним шляхом.

Обрана тема дипломної роботи є актуальною, бо представлена інформаційна система зможе рятувати людські життя та попереджати ймовірні інфікування коронавірусом, а також запобігати ускладненням захворювання, що може призвести до складних наслідків.

Практичне значення одержаних результатів. Розроблено мобільний застосунок та інформаційну систему з визначення та попередження про небезпечні місця великої кількості людей.

Публікації. Результати роботи були опубліковані в 1 тезі доповіді на науково-технічній конференції [2].

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

У кожної сучасної людини є телефон, що має функції місцезнаходження або геолокацій. Таким чином можна передбачити великий натовп людей, який можна уникнути, запобігнувши ймовірного захворювання вірусами. Дана інформаційна система попереджає користувача щодо таких небезпечних скупчень людей.

1.1.1 Опис процесу діяльності

В даній інформаційній системи буде визначатися клас небезпеки скупчень людей. Для цього потрібно залучити користувачів до системи, щоб на основі їх даних обраховувати індекси небезпек. Кожний користувач зможе отримувати інформацію щодо небезпеки навколо себе і приймати рішення, як йому змінити маршрут в реальному житті. Таким чином, людина зможе оминати такі небезпечні скупчення, а отже спасати себе від можливого захворювання коронавірусом. Користувач буде користуватися мобільним застосунком, який буде передавати дані геолокацій користувача, а також сповіщати про небезпечні скупчення людей неподалік. Клас небезпеки та локації користувачів будуть змінюватись в реальному часі.

1.1.2 Опис функціональної моделі

Основним актором у системі є користувач мобільного застосунку. Користувач буде мати змогу переглядати карту, на якій будуть показані небезпечні скупчення, змінювати свій профіль, медичні показники та проходити автентифікацію у систему. Візуальне представлення схеми діаграми використання наведено в частині графічного матеріалу.

Відповідно до вище зазначених варіантів використання виявлені функціональні вимоги, встановлені їх пріоритети, результати яких наведено у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги

Актор	***Варіант використання***	***Функціональна вимога***	***Пріоритет***
Користувач мобільного застосунку	Реєстрація та авторизація	Система вимагає від користувача заповнення профілю, електронну пошту та пароль.	Важливий
	Перегляд мапи	Мобільний застосунок повинен показувати поточну геолокацію користувача, небезпечні місця на карту у виді теплових карт. Навколо карти повинен бути відображений клас небезпеки скупчень людей, який буде змінюватись відповідно до зміни стану системи.	Важливий
	Перегляд та редагування профілю	Користувач має змогу переглядати та змінювати свій профіль: ім’я, дату народження, стать.	Середній
	Перегляд та редагування медичних показників	Користувач має змогу переглядати та змінювати свої медичні показники.	Середній

1.2 Огляд наявних аналогів

Програми знаходжень великих скупчень людей були створені ще до пандемії коронавірусу. Цілі таких програм були різні – від моніторингу відвідувань закладу до обминання галасливих місць або заторів.

Як правило, об’єктом автоматизації були муніципальні заклади, як-от магазини, торгові центри чи готелі. Такі рішення були націлені на уникнення натовпів людей для зберігання часу, щоб не стояти в черзі чи проаналізувати час відвідування людей, наприклад щоб забезпечити заклад стабільним сервісом в часи пік або не витрачати зайві ресурси, коли не потрібно багато обслуговуючого персоналу. Реалізація цієї інформаційної системи відрізняється від інших насамперед тим, що націлена на уникнення коронавірусу та повітряно-крапельних інфекцій шляхом обминання натовпів людей.

1.2.1 populace.ai

Даний сервіс [3] має анонімну варіант роботи, простий інтерфейс, але працює тільки у браузері, не має ніяких налаштувань та неточно й несвоєчасно обробляє місцезнаходження користувача.

1.2.2 crowdless

Мобільний додаток crowdless [4] містить інтуїтивний інтерфейс та ним можна користуватися в якості гостя. Проте розрахунок ймовірного скупчення людей можливий тільки в певних геолокаціях, наприклад в торгових центрах чи магазинах.

1.2.3 density.io

Застосунок density.io [5] – це чудовий приклад корпоративного комерційного проєкту для великих компаній. Сервіс наділений великою кількістю точних метрик та статистики, має чудову систему повідомлень та

аналітики. Його мінуси: використання тільки у певних зонах, тобто прив'язка до певного місцезнаходження, наприклад готелю, складність використання, яка зумовлена кількістю налаштувань та телеметрії, та платність застосунку, тобто сервіс потребує значних кошт за встановлення обладнання та покупку апаратного та програмного забезпечення.

На основі аналогів можемо побудувати таблицю переваг та недоліків: табл. 1.2.

Таблиця 1.2 – Переваги та недоліки аналогів

Функціонал / Аналог	***populace.ai***	***crowdless***	***density.io***	***IC***
Моб. додаток	-	+	-	+
Безкоштовний функціонал	+	+	-	+
Апаратне забезпечення	-	-	+	-
Складність використання	-	-	+	-
Медичні показники	-	-	-	+

Створювана інформаційна система буде залучати до себе усі плюси аналогів, маючи мінімум їх мінусів.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки є створення мобільного застосунку та інформаційної системи з виявлення небезпечних місць скупчення великої кількості людей та розрахунку індикатора безпеки на випадок вірогідного захворювання вірусами, зокрема коронавірусом SARS-CoV-2.

1.3.2 Цілі та задачі розробки

Ціль розробки - попередження про скупчення людей, які можуть бути причиною захворювань вірусів, зокрема коронавірусу SARS-CoV-2

Задачі системи складаються з:

- розробки серверної частини, проектування бази даних;
- реалізації автентифікації та авторизації;
- розробки мобільного застосунку;
- реалізації передачі геоданих із застосунку;
- розробки інтерфейсу з виявленими місцями скупчення людей;
- реалізації алгоритму розрахунку індикатора безпеки на основі анонімних геоданих користувачів;
- відкритих масивах даних та розробки внесення медичних показників та загальних даних користувача.

Висновок до розділу

У ході написання розділу сформульовано постановку задачі, наведені очікування від програми, виявлені існуючі аналоги та досліджено їх сильні та слабкі сторони, наведений короткий вступ в предметну область, встановлена ціль виявлення потенційно небезпечних місць скупчень людей для попередження розповсюдження вірусів.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

На вхід до інформаційної системи дані поступають через мобільний застосунок. Користувач заповнює свій профіль та медичні показники, а його телефон (чи інший клієнт) передає дані геолокацій.

Профіль користувача містить наступні дані:

- електрона пошта;
- пароль;
- дата народження;
- ім’я;
- стать.

Медичні показники містять такі дані:

- статус – невідомий, інфікований, вакцинований, одужалий.

Запити геолокацій передають наступні дані:

- широта;
- довгота;
- радіус кола, в якому будуть обиратися натовпи людей.

2.2 Вихідні дані

Вихідні дані інформаційної системи відображаються мобільним застосунком на різних екранах додатку.

А саме:

- геолокація користувача;
- локації натовпів на карті у вигляді теплових карт;
- клас небезпеки у вигляді обода мапи;
- профіль та медичні показники користувача.

2.3 Опис структури бази даних

В базі даних основними сутностями є користувач, його медичні показники та його геолокацій. Отже створені 3 таблиці: User, UserLocation, UserHealthIndicator. Таблиця User має дані про користувача та його профіль, токен автентифікації. Таблиця UserLocation зберігає дані про останню локацію користувача, а UserHealthIndicator – про його медичний показник.

В таблиці 2.1 показана структура БД у виді SQL:

Таблиця 2.1 – Структура БД та опис колонок

Назва таблиці	***Назва колонки***	***Тип даних***	***Опис***
User	id	serial	Первинний ключ, унікальний ідентифікатор
	name	varchar(256) not null	Ім’я користувача
	birthdate	date not null	Дата народження користувача
	email	varchar(256) not null	Електрона пошта користувача
	passwordHash	varchar(256) not null	Хеш паролю користувача
	token	varchar(256) not null	Токен авторизації користувача
	createdAt	timestamp with time zone not null	Дата створення рядку даних
	updatedAt	timestamp with time zone not null	Дата оновлення рядку даних
	sex	“enum_User_sex”(‘female’, ‘male’)	Стать користувача

Продовження таблиці 2.1

Назва таблиці	***Назва колонки***	***Тип даних***	***Опис***
UserHealthIndicator	userId	integer references id column from User	Первинний ключ, зовнішній ключ, ідентифікатор користувача
	status	"enum_UserHealthIndicator_status"(‘unknown’, ‘infected’, ‘recovered’, ‘vaccinated’) default 'unknown'::"enum_UserHealthIndicator_status" not null	Медичний статус користувача
	createdAt	timestamp with time zone not null	Дата створення рядку даних
	updatedAt	timestamp with time zone not null	Дата оновлення рядку даних

Продовження таблиці 2.1

Назва таблиці	***Назва колонки***	***Тип даних***	***Опис***
UserLocation	userId	integer references id column from User	Первинний ключ, зовнішній ключ, ідентифікатор користувача
	point	geometry(Point) not null	Геолокація користувача
	createdAt	timestamp with time zone not null	Дата створення рядку даних
	updatedAt	timestamp with time zone not null	Дата оновлення рядку даних

Візуальне представлення схеми бази даних наведено в частині графічного матеріалу.

В структурі БД також є неосновні таблиці, як-от rater_columns і us_lex. Це системні таблиці плагіну Postgis бази даних Postgres, які створені автоматично та оброблюються самою БД при зміні та при запитах до геометричних даних, тобто колонки point у таблиці UserLocation.

Висновок до розділу

В даному розділі детально описані вхідні та вихідні дані інформаційної системи, була визначена структура бази даних.

В рамках проектування БД створено 3 таблиці, які зберігають дані про користувачів, їх медичні показники та геолокацій.

Структури системи повністю відповідають вимогам технічного завдання, значить інформаційна система підходить до заявлених вимог.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Основна проблема розробки - визначення класу небезпеки грунтуючись на скупчення людей неподалік. Розробка математичної моделі, яка дозволяє вирахувати індекс небезпеки, щоб проінформувати агента моделі. На основі індексу формулюється клас небезпеки, який вже показується агенту моделі. Тобто мобільний застосунок (клієнт) передає дає геолокації, профілю та медичних показників користувача, а інформаційна система (сервер) оброблює дані та обраховує статус, а далі й визначає клас небезпеки, який буде показуватися користувачу та попереджувати його про небезпеку.

3.2 Математична постановка задачі

Нехай існують $n \in Z^+$ людей - користувачів в радіусі $R \in \mathrm{R}$ від агента моделі. Маємо їх геолокації в градусах, а саме широта $\mathrm{lat_i}$ та довгота $\mathrm{long_i}$, медичні показники $med_i \in \{\text{Невідомий}, \text{Інфікований}, \text{Одужалий}, \text{Вакцинований}\}$, віки $age_i \in N$ та статі $sex_i \in \{\text{Жінка}, \text{Чоловік}\}$, $i \in \overline{1, n}$.

На основі цих даних потрібно визначити індекс небезпеки місць скупчення $P \in N$, а також клас небезпеки $C \in \{\text{Небезпечний}, \text{Допустимий}, \text{Безпечний}\}$.

3.3 Обґрунтування методу розв’язання

Для розв’язання цієї конкретної задачі, а саме визначення класу небезпеки місць скупчення великої кількості людей було вирішено створити реалізацію алгоритму обраховування індексу, тобто створити формулу, яка брала б до уваги дані людей неподалік, тобто їх скупчення та обраховувала індекс небезпеки.

Для створення формули потрібно рахувати відстань від актора моделі до скупчень людей. Найлегший спосіб - евклідова відстань, проте вона не підходить по тій причині, що поверхня землі наближена до сфери або кулі, а ніяк до плоскої поверхні.

Для вирішення такої задачі потрібно прибігти до тригонометричних функцій, а саме до синус-верзуса. Він являє собою відстань від центральної точки дуги, вимірюваної подвоєним даними кутом, до центральної точки хорди, що стягує дугу.

$$versin\,\theta = 1 - \cos\theta = 2sin^2(\frac{\theta}{2})$$

Гаверсинус - це його половина, яка підходить для вирішення задач з пошуку відстаней:

$$hav\,\theta = \frac{versin\,\theta}{2} = sin^2(\frac{\theta}{2})$$

Тоді для будь-яких 2 точок на сфері гаверсинус центрального кута між ними обраховується по формулі:

$$hav\,\frac{d}{r} = hav(\varphi_2 - \varphi_1) + \cos\varphi_1 \cos(\varphi_2)\, hav(\lambda_2 - \lambda_1)$$

де r – радіус сфери, d – центральний кут між двома точками, які лежать на великому колі, φ_1 та φ_2 – широти точок в радіанах, λ_1 та λ_2 – довготи точок в радіанах [6].

3.4 Опис методів розв’язання

Перечислимо усі вхідні дані: широта та довгота користувача та радіус обробки скупчень, дані скупчень поблизу: вік, стать, медичний показник, широта та довгота.

Якщо позначити $h = hav\frac{d}{r}$ та використати обернені тригонометричні формули, то формула відстані до агента моделі прийме такий вигляд:

$$d = 2r \arcsin \sqrt{\begin{matrix} hav(\varphi_2 - \varphi_1) + \\ \cos(\varphi_2) * \cos(\varphi_1) * hav(\lambda_2 - \lambda_1) \end{matrix}} =$$

$$2r \arcsin \sqrt{\begin{matrix} sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \\ \cos(\varphi_2)\cos(\varphi_1)\, sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right) \end{matrix}}$$

Щоб використати формулу відстані, спочатку потрібно перевести широту й довготу в радіан із градусів:

$$\varphi = lat * \frac{\pi}{180}$$

$$\lambda = long * \frac{\pi}{180}$$

Тепер можна порахувати відстань від користувача до скупчень людей, взявши радіус сфери – приблизний радіус Землі, а саме 6371 км згідно з стандартом WGS84 [7].

Із-за того, що Земля - неідеальна сфера, погрішність таким методом є 0.5%.

Тепер обрахуємо індекс небезпеки P:

$$P = \sum_{i=1}^{n} c * \frac{k_{med_i}}{d_i} * k_{age_i} * k_{sex_i}$$

де k_{med_i}, k_{age_i}, k_{sex_i}- це коефіцієнти, що беруться відповідно до вхідних даних, c -константа, d – відстань в метрах від людини до агента моделі, яка визначена вище.

В цьому випадку

$$d_i = 2R_з \arcsin \sqrt{\begin{matrix} sin^2\left(\frac{\varphi_\alpha - \varphi_i}{2}\right) + \\ \cos(\varphi_\alpha)\cos(\varphi_i)\, sin^2\left(\frac{\lambda_\alpha - \lambda_i}{2}\right) \end{matrix}}$$

де позначення з поміткою і – дані скупчень, а з поміткою α – дані користувача.

На основі научних досліджень [8][9][10] коронавірусу були підібрані подальші показники та коефіцієнти, які зможуть еврістично відображати небезпеку.

Коефіцієнт k_{med_i} береться з таблиці 3.1 судячи по показнику med_i.

Таблиця 3.1 - Коефіцієнти медичного показника

Показник med	***Коефіцієнт*** $\boldsymbol{k_{med}}$
Невідомий	10
Інфікований	30
Одужалий	0.5
Вакцинований	0.01

Очевидно, що цей коефіцієнт залежить від того, чи людина інфікована або ж вакцинована, тоді й шанс зараження радикально змінюється.

Обраховування коефіцієнту віку задано формулою:

$$k_{age_i} = 0{,}002 * age_i + 0{,}95$$

де age_i – вік людини. Фактично константи цієї формули – це вирішення СЛАР, де показнику 20 років відповідає коефіцієнт 1, а 100 рокам – 1.2. Доведено, що літні люди найбільш непристосовані до коронавірусної інфекції[11][12].

Тепер визначимо коефіцієнт статі k_{sex_i} по таблиці 3.2.

Таблиця 3.2 - Коефіцієнти статі

Показник sex	***Коефіцієнт*** $\boldsymbol{k_{med}}$
Жінка	1
Чоловік	1.2

Були проведені дослідження, що жінки мають менший ризик померти або захворіти коронавірусом, ніж чоловіки.[13]

Оберемо таку константу c, щоб мати індекс небезпеки людсько-читабельним, наприклад 50.

Тоді визначимо інтервали класу небезпек C: індекс менше 50 будемо вважити безпечним класом, індекс від 50 до 140 – допустимим, а індекс більше 140 – небезпечним.

У частині графічного матеріалу представлений приклад коду для обраховування індексу небезпеки.

Висновок до розділу

У розділі математичного забезпечення було розглянуто змістовну постановку задачі та математичні постановку задачі. Був обраний та обґрунтований метод пошуку відстані через гаверсинус, а також сформульована формула для обраховування індексу, а також визначення класу небезпеки.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Програмне забезпечення, що використовується при розробці дипломного проєкту:

- операційна система: Linux Ubuntu [14] 20.04;
- середовища розробок: Webstorm [15] – розумна IDE для створення JS додатків та Android Studio [16]– IDE для мобільних застосунків;
- мови програмування: TypeScript [17] – узагальнена мова з типізацією, яка транспілюється в JavaScript для запуску в браузері чи на сервері, Dart [18] – ООП мова, використовується у мобільному застосунку;
- платформа: Node.js [19] – платформа для запуску JS на сервері;
- фреймворки: Nest.js [20] - фреймворк для створення потужних прогресивних масштабованих серверних веб-застосунків, Flutter [21] – фреймворк для створення мобільних додатків та веб застосунків, який використовує власний рушій для рендерингу;
- ORM - бібліотека: Sequelize [22];
- база даних: Postgres з плагіном, який надає географічні функції, Postgis [23][24];
- віртуалізація програми: Docker [25];

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Інформаційна система складається з багатьох користувацьких клієнтів (телефони чи ноутбуки) та серверу.

Вимоги до серверу:

- центральний процесор з частотою 1 ГГЦ чи потужніший;
- оперативна пам’ять об’ємом 512 МБ чи більше;
- 10 ГБ вільного диску.

На сервері повинен бути налаштований Docker, щоб запустити на ньому зібраний Docker - контейнер серверної частини додатку. Також в Docker'і можно запустити базу даних Postgres. Також можно налаштувати сервер без Docker'у, взявши таке ПЗ:

- node.js 14;
- npm 6 [26];
- postgreSQL з Postgis;
- ubuntu 16.04.

Користувач має мати телефон з GPS навігатором або ноутбук.

Вимоги до клієнта:

- sdk версія Android [27] – 16;
- deployment target версія IOS [28] – 9;
- google chrome [29].

4.3 Архітектура програмного забезпечення

Архітектура ПЗ складається з 3 компонент – пристрою користувача, серверу та БД.

На рисунку 4.1 зображена архітектура ПЗ.

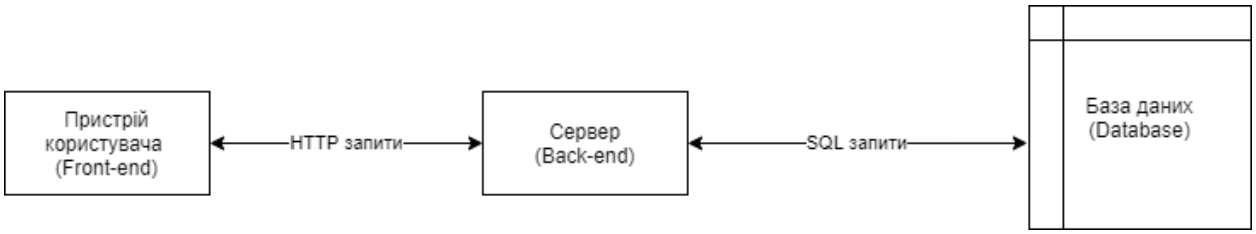


Рисунок 4.1 – архітектура ПЗ

4.3.1 Діаграма класів

Схема структурна класів програмного забезпечення наведена в частині графічного матеріалу.

4.3.2 Діаграма послідовності

Схема структурна послідовності для функцій автентифікації, взаємодією з профілем та медичними показниками та переглядом мапи наведена в частині графічного матеріалу. Компонент Front-end – це мобільний застосунок, на якому є основні екрани з наданим функціоналом. Back-end – це сервер, на якому є шар контролерів та сервісів, які мають модулі авторизації, локації, профілю та медичних показників. Також є база даних, у якій зберігаються усі дані.

4.3.3 Діаграма компонентів

Схема структурна компонентів представлена в частині графічного матеріалу для функцій автентифікації, взаємодією з профілем та медичними показниками та переглядом мапи наведена в частині графічного матеріалу.

4.3.4 Специфікація функцій

У таблиці 4.1 описані специфікації функцій.

Таблиця 4.1 – Специфікації функцій

Функція	***Опис***
calculateHealthStatusAndPoints(locations): {status, points, locations}	Обчислення індикатору та статусу небезпеки в цілому
getLocationPoints(distance, status, sex, age): number	Обчислення індикатора для конкретної людини
sexCoefficient(sex): number	Визначення коефіцієнту статі
ageCoefficient(age): number	Визначення коефіцієнту віку

Продовження таблиці 4.1

Функція	***Опис***
medCoefficient(status): number	Визначення коефіцієнту медичного показника
getLocationsInCircle(id, latitude, longtidute, radius): UserLocation[]	Отримання усіх людей в певному радіусі до користувача

4.4 Опис звітів

Результатом роботи є теплові карти інших людей на карті, які будуть відображені в мобільному застосунку, що є частиною інформаційної системи.

Висновок до розділу

В цьому розділі було описане ПЗ ІС. Основний стек ІС – Node.js та Flutter, а БД – Postgres. Були описані вимоги то ТЗ, які необхідні для вирішення поставлених задач. Також були описані архітектура ІС, наведені відповідні діаграми класів, послідовностей та компонентів. Наведені специфікації функцій класів.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

При першому запуску мобільного застосунку на екрану користувача відображатиметься стартова сторінка з кнопками реєстрації та логіну. На рис. 5.1 наведений стартовий екран.

Рисунок 5.1 – Стартовий екран

При переході на реєстрацію відобразиться екран з декількома текстовими полями. Необхідно заповнити пошту, пароль, ім’я, дату народження та стать. Екран реєстрації показаний на рис 5.2.

Рисунок 5.2 – Екран реєстрації

При переході на логін відобразиться екран з текстовими полями пошти та паролю, який наведений на рис. 5.3.

Рисунок 5.3 – Екран логіну

При неправильному вводу пароля чи некоректних даних буде відображатиметься помилка. На рис. 5.4 наведений екран з неправильним підтвердженням паролю.

Рисунок 5.4 – Екран реєстрації з некоректними даними

При вдалому логіні або реєстрації користувач попаде на головний екран, як показано на рис. 5.5.

Рисунок 5.5 – Головний екран застосунку з червоним ободком

На головному екрані мобільного застосунку є карта з поточною локацією користувача, скупчення людей у виді теплових карт та кольоровий ободок. Колір ободку вказує на клас небезпеки, на рис. 5.5 він червоний – це означає, що біля користувача наявні небезпечні скупчення людей.

З плином часу геолокації людей можуть змінюватися. На рис. 5.6 показано, що при достатньо великій дистанції від користувача до скупчень людей обід змінює колір на зелений, тобто клас небезпеки змінився з небезпечного на безпечний.

Рисунок 5.6 – Головний екран з зеленим ободком

При зміні класу небезпеки до мобільного телефону надходить сповіщення, як показано на рис. 5.7.

Рисунок 5.7 – Сповіщення щодо зміни класу небезпеки

Користувач може змінити поля профілю або медичні показники, або ж зовсім вилогінитися з застосунку. Для цього потрібно нажати на кнопку меню у лівому верхньому крайньому куту екрану. На рис. 5.8 наведений екран з меню застосунку.

Рисунок 5.8 – Екран меню

В меню є 4 вкладки: карта, профіль, медичні показники та вихід з додатку. При виході з додатку користувач попаде на стартовий екран додатку, його сесію буде видалено з застосунку.

При натисканні на вкладку профілю користувач перейде на екран профілю, як показано на рис 5.9.

Рисунок 5.9 – Екран профілю

На цьому екрані можна змінювати ім’я, стать та дату народження. Для збереження відповідних даних потрібно натиснути на кнопку «оновити».

На екрані медичних показників можна змінити поточний медичний стан. Вибір стану показаний на рис 5.10.

Рисунок 5.10 – Зміна медичного стану

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій інформаційної системи з виявлення небезпечних місць скупчення великої кількості людей вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

У результаті тестування був перевірений увесь мобільний застосунок та інформаційна система. У наступних таблицях наведено перелік випробувань функціональних можливостей.

Таблиця 5.1 – Логін

Мета тесту	***Перевірка функції «Логін»***
Початковий стан моделі	Відкритий екран логіну
Вхідні дані	Логін та пароль користувача
Схема проведення тесту	Ввести логін та пароль, під якими раніше користувач зареєструвався, натиснути кнопку «логін»
Очікуваний результат	Перехід до головного екрану
Стан моделі після проведення випробування	Перехід до головного екрану

Таблиця 5.2 – Реєстрація

Мета тесту	***Перевірка функції «Реєстрація»***
Початковий стан моделі	Відкритий екран реєстрації
Вхідні дані	Пошта, пароль, підтвердження паролю, ім’я, стать та дата народження
Схема проведення тесту	Ввести вхідні пошту, пароль та його підтвердження, ім’я, вибрати стать з випадаючого списку, обрати дату народження та натиснути «зареєструвати»
Очікуваний результат	Перехід до головного екрану
Стан моделі після проведення випробування	Перехід до головного екрану

Таблиця 5.3 – Зміна профілю

Мета тесту	***Перевірка функції «Зміна профілю»***
Початковий стан моделі	Відкритий екран профілю
Вхідні дані	Ім’я, стать та дата народження
Схема проведення тесту	Ввести вхідні ім’я, вибрати стать з випадаючого списку, обрати дату та натиснути «оновити»
Очікуваний результат	Оновлені дані збережені, відображена пошта при реєстрації
Стан моделі після проведення випробування	Оновлені дані збережені, відображена пошта при реєстрації

Таблиця 5.4 – Зміна медичних показників

Мета тесту	***Перевірка функції «Зміна медичних показників»***
Початковий стан моделі	Відкритий екран медичних показників
Вхідні дані	Медичний стан
Схема проведення тесту	Обрати медичний стан з випадаючого списку та натиснути «оновити»
Очікуваний результат	Оновлений медичний стан збережений
Стан моделі після проведення випробування	Оновлений медичний стан збережений

Таблиця 5.5 – Попередження про зміну класу небезпеки

Мета тесту	***Перевірка функції «Попередження про зміну класу небезпеки»***
Початковий стан моделі	Відкритий екран мапи
Вхідні дані	Геопозиція коистувача та скупчень людей
Схема проведення тесту	Підійти до скупчення людей. Відійти до безпечного інтервалу.
Очікуваний результат	Ободок мапи повинен змінити свій колір з червоного до жовтого і зеленого. Надійшло сповіщення щодо зміни класу небезпеки. Перемістилися геолокації та теплові карти на мапі

Продовження таблиці 5.5

Мета тесту	***Перевірка функції «Попередження про зміну класу небезпеки»***
Стан моделі після проведення випробування	Ободок мапи повинен змінити свій колір з червоного до жовтого і зеленого. Надійшло сповіщення щодо зміни класу небезпеки. Перемістилися геолокації та теплові карти на мапі

Висновок до розділу

У даному розділі було розглянуто детальну користувацьку інструкцію з приводу використання даного мобільного застосунку, було продемонстровано повний перелік функціоналу, який присутній в даній системі, за допомогою скріншотів і пояснень до них.

Зазначені мета випробувань та загальні положення для даної інформаційної системи.

Застосування було перевірено на тестових сценаріях, наведено опис тестів із детальними поясненнями. Тестування пройшло успішно, отже розроблений програмник продукт у повній мірі відповідає вимогам і розв’язує поставлені задачі.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті виконання дипломного проєкту розроблену інформаційну систему для виявлення небезпечних місць скупчення великої кількості людей.

Детально проаналізовані актуальність даного програмного продукту, існуючи підходи до вирішення задач, визначені цілі розробки та її призначення, описані функціональна модель системи, предметне середовище та процес діяльності.

Визначені вхідні та вихідні дані, наведена структура сутностей реляційної бази даних.

Сформульована змістовна та математична постановки задачі, обґрунтована доцільність використання формули гаверсинуса. Проаналізовані вхідні параметри моделі, були проведені дослідження для підбору правильних коефіцієнтів моделі. Була розроблена формула індексу небезпеки та визначення класу небезпеки на основі вхідних даних моделі.

В роботі також були описані технології, які використовувались для програмування даного продукту. В якості бази даних була обрана Postgres з плагіном Postgis. Використані мови програмування: Dart та Javascript. Наведені мінімальні вимоги до технічного забезпечення. Розглядані описи класів, які входять до архітектури розробки.

Приведене керівництво користувача, де описано, як користувач має працювати з програмним продуктом. Були створені та успішно пройдені тестові сценарії. Тестові випробування були детально описані та проаналізовані.

Розроблений програмний продукт є корисним для усіх людей, які не хочуть захворіти будь-якими інфекціями, які передаються повітряно-крапельним шляхом, як-от коронавірус SARS-CoV-2, а саме бути попередженими про небезпеку зараження від інших людей, або ж уникати такі великі скупчення людей.

ПЕРЕЛІК ПОСИЛАНЬ

1. SARS-CoV-2 [Електронний ресурс] - Режим доступу до ресурсу: https://www.who.int/health-topics/coronavirus
2. Великий Д.Є., Баклан І. В. Визначення небезпечних місць скупчення великої кількості людей // Матеріали VI всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2021) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 22-23 квітня 2021 р.
3. populace.ai [Електронний ресурс] - Режим доступу до ресурсу: http://populace.ai/
4. crowdless [Електронний ресурс] - Режим доступу до ресурсу: https://crowdlessapp.co/
5. density.io [Електронний ресурс] - Режим доступу до ресурсу: https://density.io/
6. U. S. Census Bureau Geographic Information Systems FAQ, What is the best way to calculate the distance between 2 points? [Електронний ресурс] - Режим доступу до ресурсу: http://www.movable-type.co.uk/scripts/gis-faq-5.1.html
7. WGS84 [Електронний ресурс] - Режим доступу до ресурсу: https://earth-info.nga.mil/
8. Edilson Crema Not even the air of empty spaces is coronavirus free (Two meters is not a safe distance) [Електронний ресурс] - Режим доступу до ресурсу: https://arxiv.org/abs/2006.08823
9. Stephen X. Zhang, Hao Huang, Feng Wei Geographical distance to the epicenter of Covid-19 predicts the burnout of the working population: Ripple effect or typhoon eye effect? [Електронний ресурс] - Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0165178120307691
10. Ne-Hooi Will Loh MBBS, Yanni Tan MBBS, Juvel Taculod BSRT, Billy Gorospe BSRT, Analine S. Teope BSN, Jyoti Somani MD, Addy Yong Hui Tan MBBS The impact of high-flow nasal cannula (HFNC) on coughing distance: implications on its use during the novel coronavirus disease outbreak [Електронний ресурс] - Режим доступу до ресурсу: https://link.springer.com/article/10.1007%2Fs12630-020-01634-3
11. Zuhua Chen, Hongjie Fan, Jian Cai, Yunjiang Li, Baoliang Wu, Yanchun Hou, Shufeng Xu, Fei Zhou, Yongguang Liu, Weiling Xuan, Hongjie Hu, Jihong Sun High-resolution computed tomography manifestations of COVID-19 infections in patients of different ages [Електронний ресурс] - Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S0720048X20301613

12. Amber L. Mueller, Maeve S. McNamara, David A. Sinclair Why does COVID-19 disproportionately affect older people? [Електронний ресурс] - Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7288963/
13. Biruk Beletew Abate, Ayelign Mengesha Kassie, Mesfin Wudu Kassaw, Teshome Gebremeskel Aragie, Setamlak Adane Masresha Sex difference in coronavirus disease (COVID-19): a systematic review and meta-analysis
[Електронний ресурс] - Режим доступу до ресурсу: https://bmjopen.bmj.com/content/10/10/e040129.abstract
14. Ubuntu [Електронний ресурс] - Режим доступу до ресурсу: https://ubuntu.com/
15. Webstorm [Електронний ресурс] - Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/webstorm/
16. Android studio [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/studio
17. Typescript [Електронний ресурс] - Режим доступу до ресурсу: https://www.typescriptlang.org/
18. Dart [Електронний ресурс] - Режим доступу до ресурсу: https://dart.dev/
19. Node.js [Електронний ресурс] - Режим доступу до ресурсу: https://nodejs.org
20. Nest.js [Електронний ресурс] - Режим доступу до ресурсу: https://nestjs.com/
21. Flutter [Електронний ресурс] - Режим доступу до ресурсу: https://flutter.dev/
22. Sequelize [Електронний ресурс] - Режим доступу до ресурсу: https://sequelize.org/
23. Postgres [Електронний ресурс] - Режим доступу до ресурсу: https://www.postgresql.org/
24. Postgis [Електронний ресурс] - Режим доступу до ресурсу: https://postgis.net/
25. Docker [Електронний ресурс] - Режим доступу до ресурсу: https://www.docker.com/
26. npm [Електронний ресурс] - Режим доступу до ресурсу: https://www.npmjs.com/
27. Android [Електронний ресурс] - Режим доступу до ресурсу: https://www.android.com/
28. Apple [Електронний ресурс] - Режим доступу до ресурсу: https://www.apple.com/
29. Google Chrome [Електронний ресурс] - Режим доступу до ресурсу: https://www.google.com/chrome

Додаток А

Тексти програмного коду

Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей

(Найменування програми (документа))

DVD-R

(Вид носія даних)

46 арк, 242 Кб

(Обсяг програми (документа) , арк., Кб)

Київ – 2021 року

DCT Server/src/main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';
import { localsMiddleware } from 'common/middlewares/locals.middleware';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.use(localsMiddleware);
  app.useGlobalPipes(new ValidationPipe({ transform: true }));
  await app.listen(3000);
}
bootstrap();
```

DCT Server/src/app.module.ts

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { ApiModule } from 'api/api.module';
import { CommonModule } from 'common/common.module';
import { ScheduleModule } from '@nestjs/schedule';

@Module({
  imports: [
    ConfigModule.forRoot(),
    CommonModule,
    ApiModule,
    ScheduleModule.forRoot(),
  ],
})
export class AppModule {}
```

DCT Server/src/common/common.module.ts

```
import { HttpModule, Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { DatabaseModule } from 'common/database/database.module';
import { CoronavirusService } from 'common/coronavirus/coronavirus.service';

@Module({
  imports: [ConfigModule, DatabaseModule, HttpModule],
  exports: [DatabaseModule, CoronavirusService],
  providers: [CoronavirusService],
})
export class CommonModule {}
```

DCT Server/src/common/guards/auth.guard.ts

```
import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import { UserModel } from 'common/database/models';
import { LocalRequestInterface } from 'common/interfaces/local-request.interface';
import { UserInterface } from 'common/database/interfaces';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    @InjectModel(UserModel)
    private readonly userModel: typeof UserModel,
  ) {}

  public async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest<LocalRequestInterface>();
    const authHeader = request.headers.authorization;
    const token = (authHeader || '').replace('Bearer ', '');
    if (!token) {
      throw new UnauthorizedException('Unauthorized');
    }

    const user = await this.userModel.findOne({ where: { token } });
    if (!user) {
      throw new UnauthorizedException('Unauthorized');
    }

    request.locals.user = user.toJSON() as UserInterface;

    return true;
  }
}
```

DCT Server/src/common/database/database.module.ts

```
import { Module } from '@nestjs/common';
import { SequelizeModule } from '@nestjs/sequelize';
import { parse } from 'pg-connection-string';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { databaseProviders } from './database.providers';

@Module({
  imports: [
    SequelizeModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => {
        const url = configService.get<string>('DATABASE_URL');
        const { host, user: username, database, port, password } = parse(url);

        return {
          type: 'postgres',
          dialect: 'postgres',
          url,
          autoLoadModels: true,
          synchronize: true,
          username,
          password,
          database,
          host,
          port: +port,
          define: {
            freezeTableName: true,
          },
          logging: false,
        };
      },
      inject: [ConfigService],
    }),
    ...databaseProviders,
  ],
  exports: [...databaseProviders],
})
export class DatabaseModule {}
```

DCT Server/src/common/database/models/user.model.ts

```
import {
  Column,
  Model,
  Table,
  PrimaryKey,
  DataType,
  HasOne,
  AllowNull,
  Unique,
  AutoIncrement,
} from 'sequelize-typescript';
import { UserLocationModel } from './user-location.model';
import { UserHealthIndicatorModel } from './user-health-indicator.model';
import crypto from 'crypto';
import {
  UserHealthIndicatorInterface,
  UserInterface,
  UserLocationInterface,
} from 'common/database/interfaces';
import { SexEnum } from 'common/database/enums';

@Table({ tableName: 'User' })
export class UserModel extends Model<UserInterface> {
  @PrimaryKey
  @Unique
  @AutoIncrement
  @Column(DataType.INTEGER)
  id: number;

  @AllowNull(false)
  @Column(DataType.STRING(256))
  name: string;
```

```
  @AllowNull(false)
  @Column(DataType.DATEONLY)
  birthDate: string;

  @AllowNull(false)
  @Unique
  @Column(DataType.STRING(256))
  email: string;

  @AllowNull(false)
  @Column(DataType.STRING(256))
  passwordHash: string;

 @AllowNull(false)
  @Column(DataType.STRING(256))
  token: string;

  @AllowNull(false)
  @Column(DataType.ENUM('female', 'male'))
  sex: SexEnum;

  @HasOne(() => UserLocationModel)
  location: UserLocationInterface;

  @HasOne(() => UserHealthIndicatorModel)
  status: UserHealthIndicatorInterface;

 static makePasswordHash(password: string): string {
    return crypto
      .createHash('sha512')
      .update(password, 'utf-8')
      .digest('hex')
      .substring(0, 526);
  }

  static makeRandomToken(): string {
    return crypto.randomBytes(256).toString('hex').substring(0, 256);
  }
 }
```

DCT Server/src/common/database/models/user-location.model.ts

```
import {
  Column,
  Model,
  Table,
  ForeignKey,
  DataType,
  AllowNull,
  PrimaryKey,
  BelongsTo,
} from 'sequelize-typescript';
import { UserModel } from './user.model';
import {
  PointInterface,
  UserInterface,
  UserLocationInterface,
} from 'common/database/interfaces';

@Table({ tableName: 'UserLocation' })
export class UserLocationModel extends Model<UserLocationInterface> {
  @PrimaryKey
  @ForeignKey(() => UserModel)
  @Column(DataType.INTEGER)
  userId: number;

  @AllowNull(false)
  @Column(DataType.GEOMETRY('POINT'))
  point: PointInterface;

  @BelongsTo(() => UserModel)
  user: UserInterface;
}
```

DCT Server/src/common/database/models/user-health-indicator.model.ts

```
import {
  Column,
  Model,
  Table,
  ForeignKey,
  DataType,
  AllowNull,
  PrimaryKey,
  Default,
} from 'sequelize-typescript';
import { UserModel } from './user.model';
import { UserHealthIndicatorInterface } from 'common/database/interfaces';
import { HealthStatusEnum } from 'common/database/enums';

@Table({ tableName: 'UserHealthIndicator' })
export class UserHealthIndicatorModel extends Model<UserHealthIndicatorInterface> {
  @PrimaryKey
  @ForeignKey(() => UserModel)
  @Column(DataType.INTEGER)
  userId: number;

  @AllowNull(false)
  @Default(HealthStatusEnum.UNKNOWN)
  @Column(DataType.ENUM('unknown', 'infected', 'recovered', 'vaccinated'))
  status: HealthStatusEnum;
}
```

DCT Server/src/api/api.module.ts

```
import { Module } from '@nestjs/common';
import { HealthzController } from './healthz.controller';
import { AuthService } from './auth/auth.service';
import { AuthController } from './auth/auth.controller';
import { LocationsService } from 'api/locations/locations.service';
import { LocationsController } from 'api/locations/locations.controller';
import { ProfileController } from 'api/profile/profile.controller';
import { ProfileService } from 'api/profile/profile.service';
import { HealthController } from 'api/health/health.controller';
import { HealthService } from 'api/health/health.service';
import { StatsController } from 'api/stats/stats.controller';
import { StatsService } from 'api/stats/stats.service';
import { CommonModule } from 'common/common.module';

@Module({
  imports: [CommonModule],
  controllers: [
    HealthzController,
    AuthController,
    LocationsController,
    ProfileController,
    HealthController,
    StatsController,
  ],
  providers: [
    AuthService,
    LocationsService,
    ProfileService,
    HealthService,
    StatsService,
  ],
})
export class ApiModule {}
```

DCT Server/src/api/auth/auth.controller.ts

```
import { Body, Controller, Get, Post, UseGuards } from '@nestjs/common';
import { AuthService } from './auth.service';
import { LoginRequestDto, RegisterRequestDto } from 'api/auth/dtos';
import { AuthGuard } from 'common/guards/auth.guard';

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @UseGuards(AuthGuard)
  @Get()
  checkToken(): void {
    return;
  }

  @Post('login')
  public login(@Body() user: LoginRequestDto): Promise<string> {
    return this.authService.login(user.email, user.password);
  }

  @Post('register')
  public register(@Body() dto: RegisterRequestDto): Promise<string> {
    return this.authService.register(
      dto.email,
      dto.password,
      dto.name,
      dto.birthDate,
      dto.sex,
    );
  }
}
```

DCT Server/src/api/auth/auth.service.ts

```
import {
  ConflictException,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { UserModel } from 'common/database/models';
import { InjectModel } from '@nestjs/sequelize';
import { SexEnum } from 'common/database/enums';

@Injectable()
export class AuthService {
  constructor(
    @InjectModel(UserModel)
    private readonly userModel: typeof UserModel,
  ) {}

  private async findUser(email: string, password: string): Promise<string> {
    const user = await this.userModel.findOne({ where: { email } });
    if (!user) {
      return null;
    }
    if (user.passwordHash !== UserModel.makePasswordHash(password)) {
      return null;
    }
    user.token = UserModel.makeRandomToken();
    await user.save();
    return user.token;
  }

  public async login(email: string, password: string): Promise<string> {
    const token = await this.findUser(email, password);
    if (!token) {
      throw new UnauthorizedException('Access denied');
    }
    return token;
  }
```

```
  public async register(
    email: string,
    password: string,
    name: string,
    birthDate: string,
    sex: SexEnum,
  ): Promise<string> {
    const oldUser = await this.userModel.findOne({ where: { email } });
    if (oldUser) {
      throw new ConflictException('User already exists');
    }
    const passwordHash = UserModel.makePasswordHash(password);

    const user = await this.userModel.create({
      email,
      passwordHash,
      name,
      birthDate: new Date(birthDate).toISOString(),
      token: UserModel.makeRandomToken(),
      sex,
    });

    return user.token;
  }
}
```

DCT Server/src/api/health/health.controller.ts

```
import { Body, Controller, Get, Post, UseGuards } from '@nestjs/common';
import { HealthService } from './health.service';
import { HealthRequestDto, HealthResponseDto } from 'api/health/dtos';
import { GetUser } from 'common/decorators/current-user.decorator';
import { AuthGuard } from 'common/guards/auth.guard';

@Controller('health')
@UseGuards(AuthGuard)
export class HealthController {
  constructor(private readonly healthService: HealthService) {}

  @Get()
  public getHealth(@GetUser('id') userId: number): Promise<HealthResponseDto> {
    return this.healthService.getHealth(userId);
  }

  @Post()
  public async updateHealth(
    @GetUser('id') userId: number,
    @Body() { status }: HealthRequestDto,
  ): Promise<void> {
    await this.healthService.updateHealth(userId, status);
  }
}
```

DCT Server/src/api/health/health.service.ts

```
import { Injectable } from '@nestjs/common';
import { UserHealthIndicatorModel } from 'common/database/models';
import { InjectModel } from '@nestjs/sequelize';
import { HealthResponseDto } from 'api/health/dtos';
import { HealthStatusEnum } from 'common/database/enums';

@Injectable()
export class HealthService {
 constructor(
  @InjectModel(UserHealthIndicatorModel)
  private readonly healthIndicatorModel: typeof UserHealthIndicatorModel,
 ) {}

 public async getHealth(userId: number): Promise<HealthResponseDto> {
  const user = await this.healthIndicatorModel.findByPk(userId);
  if (!user) {
   return { status: HealthStatusEnum.UNKNOWN };
  }
  return {
   status: user.status,
  };
 }
 public async updateHealth(
  userId: number,
  status: HealthStatusEnum,
 ): Promise<void> {
  const user = await this.healthIndicatorModel.findByPk(userId);
  if (!user) {
   await this.healthIndicatorModel.upsert({
    status,
    userId,
   });
   return;
  }
  user.status = status;

  await user.save();
 }
}
```

DCT Server/src/api/locations/locations.controller.ts

```
import { Body, Controller, Get, Post, Query, UseGuards } from '@nestjs/common';
import { AuthGuard } from 'common/guards/auth.guard';
import { LocationsService } from 'api/locations/locations.service';
import {
  GetLocationsRequestDto,
  SaveLocationRequestDto,
} from 'api/locations/dtos/requests';
import { GetUser } from 'common/decorators/current-user.decorator';
import { GetLocationsResponseDto } from 'api/locations/dtos/responses';

@UseGuards(AuthGuard)
@Controller('locations')
export class LocationsController {
  constructor(private readonly locationsService: LocationsService) {}

  @Get()
  public async getLocations(
    @Query() { latitude, longitude, radius }: GetLocationsRequestDto,
    @GetUser('id') id: number,
  ): Promise<GetLocationsResponseDto> {
    return this.locationsService.getLocations(id, latitude, longitude, radius);
  }

  @Post()
  public async saveLocation(
    @Body() { latitude, longitude }: SaveLocationRequestDto,
    @GetUser('id') id: number,
  ): Promise<void> {
    await this.locationsService.saveLocation(id, latitude, longitude);
  }
}
```

DCT Server/src/api/locations/locations.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/sequelize';
import {
  UserHealthIndicatorModel,
  UserLocationModel,
  UserModel,
} from 'common/database/models';
import { PointInterface } from 'common/database/interfaces';
import sequelize, { Op } from 'sequelize';
import moment from 'moment';
import { GetLocationsResponseDto } from 'api/locations/dtos/responses';
import { CrowdStatusEnum } from 'api/locations/enums';
import { HealthStatusEnum, SexEnum } from 'common/database/enums';
import { LocationInterface } from 'api/locations/interfaces';

@Injectable()
export class LocationsService {
  public static readonly MAX_RADIUS = 5000; // 5000 meters
  private readonly MAXIMUM_UPDATE_AT_INTERVAL = 30; // 30 minutes
  private readonly FORMULA_CONSTANT = 50;
  private readonly STATUS_COEFFICIENTS = {
    UNKNOWN: 10,
    INFECTED: 30,
    RECOVERED: 0.5,
    VACCINATED: 0.01,
  };
  private readonly POINTS_LIMITS = {
    BAD: 140,
    OK: 50,
  };

  constructor(
    @InjectModel(UserModel)
    private readonly userModel: typeof UserModel,
    @InjectModel(UserLocationModel)
    private readonly userLocationModel: typeof UserLocationModel,
  ) {}
```

```
  public async saveLocation(
  userId: number,
  latitude: number,
  longitude: number,
 ): Promise<void> {
  const point: PointInterface = {
   type: 'Point',
   coordinates: [longitude, latitude],
  };

  await this.userLocationModel.upsert({ userId, point });
 }

public async getLocations(
  id: number,
  latitude: number,
  longitude: number,
  _radius?: number,
 ): Promise<GetLocationsResponseDto> {
  const radius = _radius ?? LocationsService.MAX_RADIUS;
  const _locations = await this.getLocationsInCircle(
   id,
   latitude,
   longitude,
   radius,
  );
  const { status, points, locations } = this.calculateHealthStatusAndPoints(
   _locations,
  );
  return {
   status,
   locations,
   radius,
   points,
  };
 }
```

```
private async getLocationsInCircle(
  id: number,
  latitude: number,
  longitude: number,
  radius: number,
): Promise<UserLocationModel[]> {
  const distanceAttribute = sequelize.fn(
    'ST_Distancesphere',
    sequelize.literal('point'),
    sequelize.literal(`ST_MakePoint(${longitude},${latitude})`),
  );

  return this.userLocationModel.findAll({
    attributes: [
      ...Object.keys(UserLocationModel.rawAttributes),
      [distanceAttribute, 'distance'],
    ],
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    where: [
      {
        userId: {
          [Op.not]: id,
        },
      },
      {
        updatedAt: {
          [Op.gte]: moment().subtract(this.MAXIMUM_UPDATE_AT_INTERVAL, 'm'),
        },
      },
      sequelize.where(distanceAttribute, {
        [Op.lte]: radius,
      }),
    ],
    order: [[sequelize.literal('distance'), 'ASC']],
```

```
      include: [
      {
       // eslint-disable-next-line @typescript-eslint/ban-ts-comment
       // @ts-ignore
       model: UserModel,
       // eslint-disable-next-line @typescript-eslint/ban-ts-comment
       // @ts-ignore
       include: [UserHealthIndicatorModel],
      },
    ],
  });
}

private medCoefficient(status: HealthStatusEnum): number {
  if (status === HealthStatusEnum.INFECTED) {
    return this.STATUS_COEFFICIENTS.INFECTED;
  }
  if (status === HealthStatusEnum.RECOVERED) {
    return this.STATUS_COEFFICIENTS.RECOVERED;
  }
  if (status === HealthStatusEnum.VACCINATED) {
    return this.STATUS_COEFFICIENTS.VACCINATED;
  }

  return this.STATUS_COEFFICIENTS.UNKNOWN;
}

private ageCoefficient(age: number) {
  // 100 y.o. - 1.2 coefficient
  // 20 y.o. - 1 coefficient
  return 0.002 * age + 0.95;
}

private sexCoefficient(sex: SexEnum) {
  if (sex === SexEnum.MALE) {
    return 1.2;
  }
  return 1;
}
```

```
  private getLocationPoints(
 distance: number,
  status,
  sex: SexEnum,
  age: number,
): number {
  const kMed = this.medCoefficient(status);
  const kSex = this.sexCoefficient(sex);
  const kAge = this.ageCoefficient(age);

  return Math.ceil(this.FORMULA_CONSTANT * (kMed / distance) * kAge * kSex);
}

  private getHealthStatus(pointSum: number): CrowdStatusEnum {
  if (pointSum > this.POINTS_LIMITS.BAD) {
    return CrowdStatusEnum.BAD;
  }
  if (pointSum > this.POINTS_LIMITS.OK) {
    return CrowdStatusEnum.OK;
  }

  return CrowdStatusEnum.GOOD;
}
```

```
  private calculateHealthStatusAndPoints(
  _locations: UserLocationModel[],
 ): {
  status: CrowdStatusEnum;
  points: number;
  locations: LocationInterface[];
 } {
  const locations: LocationInterface[] = _locations
   .map((location) => {
    const [longitude, latitude] = location.point.coordinates;
    return {
     distance: location.getDataValue('distance'),
     status: location.user.status?.status || HealthStatusEnum.UNKNOWN,
     sex: location.user.sex,
     age: moment().diff(moment(location.user.birthDate), 'years'),
     latitude,
     longitude,
    };
   })
   .map((l) => ({
    ...l,
    points: this.getLocationPoints(l.distance, l.status, l.sex, l.age),
   }));

  const pointSum = locations.reduce((acc, l) => acc + l.points, 0);

  const status = this.getHealthStatus(pointSum);

  return {
   status,
   points: pointSum,
   locations: locations.map((l) => ({
    points: l.points,
    distance: l.distance,
    latitude: l.latitude,
    longitude: l.longitude,
   })),
  };
 }
}
```

DCT Server/src/api/profile/ profile.controller.ts

```
import { Body, Controller, Get, Post, UseGuards } from '@nestjs/common';
import { ProfileService } from './profile.service';
import { ProfileRequestDto, ProfileResponseDto } from 'api/profile/dtos';
import { GetUser } from 'common/decorators/current-user.decorator';
import { AuthGuard } from 'common/guards/auth.guard';

@Controller('profile')
@UseGuards(AuthGuard)
export class ProfileController {
  constructor(private readonly profileService: ProfileService) {}

  @Get()
  public getProfile(
    @GetUser('id') userId: number,
  ): Promise<ProfileResponseDto> {
    return this.profileService.getProfile(userId);
  }

  @Post()
  public async updateProfile(
    @GetUser('id') userId: number,
    @Body() profile: ProfileRequestDto,
  ): Promise<void> {
    await this.profileService.updateProfile(
      userId,
      profile.name,
      profile.birthDate,
      profile.sex,
    );
  }
}
```

DCT Server/src/api/profile/ profile.service.ts

```
import { Injectable } from '@nestjs/common';
import { UserModel } from 'common/database/models';
import { InjectModel } from '@nestjs/sequelize';
import { ProfileResponseDto } from 'api/profile/dtos/responses';
import { SexEnum } from 'common/database/enums';

@Injectable()
export class ProfileService {
  constructor(
    @InjectModel(UserModel)
    private readonly userModel: typeof UserModel,
  ) {}

  public async getProfile(userId: number): Promise<ProfileResponseDto> {
    const user = await this.userModel.findByPk(userId);
    return {
      email: user.email,
      name: user.name,
      birthDate: new Date(user.birthDate).toISOString(),
      sex: user.sex,
    };
  }

  public async updateProfile(
    userId: number,
    name: string,
    birthDate: string,
    sex: SexEnum,
  ): Promise<void> {
    const user = await this.userModel.findByPk(userId);

    user.name = name;
    user.birthDate = new Date(birthDate).toISOString();
    user.sex = sex;

    await user.save();
  }
}
```

DCT Server/src/api/profile/ profile.service.ts

```
import { Controller, Get } from '@nestjs/common';
import { version } from 'package.json';

@Controller()
export class HealthzController {
  @Get()
  healthz(): { version: string } {
    return { version };
  }
}
```

DCT Client/lib/main.dart

```
import 'package:dct_client/config.dart';
import 'package:dct_client/constants.dart';
import 'package:flutter/material.dart';
import 'package:flutter_config/flutter_config.dart';

import 'jobs/main.dart';
import 'logger.dart';
import 'push_notifications.dart';
import 'routes.dart';
import 'services/navigation_service.dart';

Future<void> main() async {
  logger.v('Starting app');

  WidgetsFlutterBinding.ensureInitialized();
  await FlutterConfig.loadEnvVariables();
  Config.ensureVariables();
  await initPushNotifications();
  initJobs();

  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: appTitle,
      initialRoute: initialRoute,
      routes: routes,
      navigatorKey: NavigationService.navigatorKey,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
    );
  }
}
```

DCT Client/lib/routes.dart

```
import 'package:flutter/material.dart';

import 'auth/auth_widget.dart';
import 'auth/login_widget.dart';
import 'auth/register_widget.dart';
import 'health/health_widget.dart';
import 'home_widget.dart';
import 'profile/profile_widget.dart';
import 'splash_screen.dart';

const initialRoute = '/';
const homeRoute = '/home';
const authRoute = '/auth';
const loginRoute = '/auth/login';
const registerRoute = '/auth/register';
const profileRoute = '/profile';
const healthRoute = '/health';

Map<String, Widget Function(BuildContext)> routes = {
  initialRoute: (context) => SplashScreen(),
  homeRoute: (context) => HomeWidget(),
  authRoute: (context) => AuthWidget(),
  loginRoute: (context) => LoginWidget(),
  registerRoute: (context) => RegisterWidget(),
  profileRoute: (context) => ProfileWidget(),
  healthRoute: (context) => HealthWidget(),
};
```

DCT Client/lib/drawer_child_widget.dart

```
import 'package:dct_client/services/auth_service.dart';
import 'package:dct_client/services/navigation_service.dart';
import 'package:flutter/material.dart';
import 'package:flutter/painting.dart';

import 'constants.dart';
import 'routes.dart';

class DrawerChildWidget extends StatelessWidget {
  final String _header;

  const DrawerChildWidget([this._header = appTitle]);

  @override
  Widget build(BuildContext context) {
    return ListView(
      padding: EdgeInsets.zero,
      children: [
        SizedBox(
          height: 100,
          child: DrawerHeader(
            decoration: const BoxDecoration(
              color: Colors.blue,
            ),
            child: Text(
              _header,
              style: const TextStyle(
                color: Colors.white,
                fontSize: 22,
              ),
            ),
          ),
        ),
```

```
            ListTile(
          leading: const Icon(Icons.map_rounded),
          title: const Text('Map'),
          onTap: () {
            NavigationService.nonReturningNavigateTo(homeRoute);
          },
        ),
        ListTile(
          leading: const Icon(Icons.account_circle),
          title: const Text('Profile'),
          onTap: () {
            NavigationService.nonReturningNavigateTo(profileRoute);
          },
        ),
        ListTile(
          leading: const Icon(Icons.local_hospital ),
          title: const Text('Health'),
          onTap: () {
            NavigationService.nonReturningNavigateTo(healthRoute);
          },
        ),
        ListTile(
          leading: const Icon(Icons.logout),
          title: const Text('Logout'),
          onTap: () => AuthService.logout(),

        )
      ],
    );
  }
}
```

DCT Client/lib/auth/auth_widget.dart

```
import 'package:dct_client/constants.dart';
import 'package:flutter/material.dart';

import '../routes.dart';

class AuthWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        backgroundColor: Colors.lime,
        body: SizedBox(
            width: double.infinity,
            child: Padding(
              padding: const EdgeInsets.symmetric(vertical: 200),
              child: Column(
                children: [
                  Container(
                    height: 200,
                    decoration: const BoxDecoration(
                      image: DecorationImage(
                        image: AssetImage('assets/images/app_icon.png'),
                      ),
                    ),
                  ),
                  Text(
                    appTitle,
                    style: Theme.of(context).textTheme.headline5,
                  ),
                  ElevatedButton(
                    onPressed: () => Navigator.pushNamed(context, loginRoute),
                    child: const Text('Login'),
                  ),
                  ElevatedButton(
                    onPressed: () =>
                        Navigator.pushNamed(context, registerRoute),
                    child: const Text('Register'),
                  ),
                ],              ),            )));  }
}
```

DCT Client/lib/auth/login_widget.dart

```
import 'package:dct_client/services/auth_service.dart';
import 'package:dct_client/services/token_service.dart';
import 'package:flutter/material.dart';
import 'package:form_field_validator/form_field_validator.dart';

import '../logger.dart';
import '../routes.dart';
import '../utils.dart';
import 'dtos/login.dto.dart';

class LoginWidget extends StatefulWidget {
  @override
  _LoginState createState() => _LoginState();
}

class _LoginState extends State<LoginWidget> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _passwordVisible = false;

  static const _sizedBoxHeight = 30.0;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
```

```
Future<void> _login(BuildContext context) async {
  try {
   if (!_formKey.currentState.validate()) {
    return;
   }

   final dto = LoginDto(
    _emailController.text,
    _passwordController.text,
   );

   final response = await AuthService.login(dto);
   if (!Utils.isStatusCodeOk(response.statusCode)) {
    ScaffoldMessenger.of(context).showSnackBar(
     const SnackBar(
      content: Text(
       'Failed to login',
       style: TextStyle(color: Colors.red),
      ),
     ),
    );
    return;
   }

   await TokenService.saveToken(response.body);

   Navigator.pushReplacementNamed(context, homeRoute);
  } catch (e) {
 logger.e(e);

   ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
     content: Text(
      'Try again later',
      style: TextStyle(color: Colors.red),
     ),
    ),
   );
  }
}
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Form(
        key: _formKey,
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 10),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              //todo: make own logo
              const FlutterLogo(
                size: 200,
              ),
              const SizedBox(height: _sizedBoxHeight),
              TextFormField(
                  decoration: const InputDecoration(
                      border: OutlineInputBorder(),
                      labelText: 'Email',
                      hintText: 'Enter your email'),
                  autofocus: true,
                  controller: _emailController,
                  validator: EmailValidator(errorText: 'Invalid email')),

              const Divider(),
              TextFormField(
                obscureText: !_passwordVisible,
                decoration: InputDecoration(
                  border: const OutlineInputBorder(),
                  labelText: 'Password',
                  hintText: 'Enter your password',
                  suffixIcon: IconButton(
```

```
icon: Icon(
                    _passwordVisible
                        ? Icons.visibility
                        : Icons.visibility_off,
                    color: Theme.of(context).primaryColorDark,
                  ),
                  onPressed: () {
                    setState(() {
                      _passwordVisible = !_passwordVisible;
                    });
                  },
                ),
              ),

              controller: _passwordController,
              validator: RequiredValidator(errorText: 'Invalid password'),

            ),
            const Divider(),

            ElevatedButton(
              style: ButtonStyle(
                  backgroundColor: MaterialStateProperty.all(Colors.blue)),
              onPressed: () => _login(context),
              child: const Text(
                'Login',
                style: TextStyle(
                  color: Colors.black,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

DCT Client/lib/auth/register_widget.dart

```
import 'package:dct_client/profile/enums/sex_enum.dart';
import 'package:dct_client/services/auth_service.dart';
import 'package:dct_client/services/token_service.dart';
import 'package:flutter/material.dart';
import 'package:form_field_validator/form_field_validator.dart';

import '../logger.dart';
import '../routes.dart';
import '../utils.dart';
import 'dtos/register.dto.dart';

class RegisterWidget extends StatefulWidget {
  @override
  _RegisterState createState() => _RegisterState();
}

class _RegisterState extends State<RegisterWidget> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _nameController = TextEditingController();
  DateTime _birthDate;
  Sex _sex;
  bool _passwordVisible = false;
  bool _repeatedPasswordVisible = false;

  static const sizedBoxHeight = 30.0;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    _nameController.dispose();
    super.dispose();
  }
```

```
Future<void> _selectDate(BuildContext context) async {
  const maxHumanLifeInYears = 150;
  final DateTime timePicked = await showDatePicker(
    context: context,
    initialDate: _birthDate ?? DateTime.now(),
    firstDate: DateTime(DateTime.now().year - maxHumanLifeInYears),
    lastDate: DateTime.now(),
  );
  if (timePicked != null && timePicked != _birthDate) {
    setState(() {
      _birthDate = timePicked;
    });
  }
}

Future<void> _register(BuildContext context) async {
  try {
    if (!_formKey.currentState.validate()) {
      return;
    }

    final dto = RegisterDto(_nameController.text, _emailController.text,
        _passwordController.text, _birthDate.toIso8601String(), _sex);

    final response = await AuthService.register(dto);

    if (!Utils.isStatusCodeOk(response.statusCode)) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text(
            'Failed to register',
            style: TextStyle(color: Colors.red),
          ),
        ),
      );
      return;
    }
```

```
    await TokenService.saveToken(response.body);

    Navigator.pushReplacementNamed(context, homeRoute);
   } catch (e) {
    logger.e(e);

    ScaffoldMessenger.of(context).showSnackBar(
     const SnackBar(
      content: Text(
       'Try again later',
       style: TextStyle(color: Colors.red),
      ),
     ),
    );
   }
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    body: Center(
     child: Form(
      key: _formKey,
      child: Padding(
       padding: const EdgeInsets.symmetric(horizontal: 10),
       child: ListView(
        children: [
         //todo: make own logo
         const FlutterLogo(
          size: 200,
         ),
         const SizedBox(height: sizedBoxHeight),
```

```
TextFormField(
          decoration: const InputDecoration(
            border: OutlineInputBorder(),
            labelText: 'Email',
            hintText: 'Enter your email'),
          autofocus: true,
          controller: _emailController,
          validator: EmailValidator(errorText: 'Invalid email'),
        ),
        const Divider(),
        TextFormField(
          obscureText: !_passwordVisible,
          decoration: InputDecoration(
            border: const OutlineInputBorder(),
            labelText: 'Password',
            hintText: 'Enter your password',
            suffixIcon: IconButton(
              icon: Icon(
                _passwordVisible
                    ? Icons.visibility
                    : Icons.visibility_off,
                color: Theme.of(context).primaryColorDark,
              ),
              onPressed: () {
                setState(() {
                  _passwordVisible = !_passwordVisible;
                });
              },
            ),
          ),
          controller: _passwordController,
          validator: RequiredValidator(errorText: 'Invalid password'),
        ),
```

```
const Divider(),
            TextFormField(
              obscureText: !_repeatedPasswordVisible,
              decoration: InputDecoration(
                border: const OutlineInputBorder(),
                labelText: 'Repeat password',
                hintText: 'Repeat your password',
                suffixIcon: IconButton(
                  icon: Icon(
                    _repeatedPasswordVisible
                        ? Icons.visibility
                        : Icons.visibility_off,
                    color: Theme.of(context).primaryColorDark,
                  ),

const Divider(),
            TextFormField(
              decoration: const InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Name',
                  hintText: 'Enter your name'),
              controller: _nameController,
              validator: RequiredValidator(errorText: 'Invalid name'),
            ),
            const Divider(),
            Row(children: [
              const Text(
                'Sex',
                style: TextStyle(fontSize: 24),
              ),
              const SizedBox(
                width: 20,
              ),
```

```
    DropdownButton<Sex>(
   value: _sex,
   icon: const Icon(Icons.arrow_downward),
   onChanged: (Sex newValue) {
    setState(() {
     _sex = newValue;
    });
   },
   items: Sex.values.map<DropdownMenuItem<Sex>>((Sex value) {
    return DropdownMenuItem<Sex>(
     value: value,
     child: Text(value.name),
    );
   }).toList(),
  ),
 ]),
 const Divider(),


 // date picker

 Center(
  child: Text(
   _birthDate == null
     ? 'Pick date'
     : '${_birthDate.day}/${_birthDate.month}/${_birthDate.year}',
   style: const TextStyle(
     fontSize: 24, fontWeight: FontWeight.bold),
  ),
 ),
 const SizedBox(
  height: sizedBoxHeight,
 ),
```

```
ElevatedButton(
                style: ButtonStyle(
                  backgroundColor:
                      MaterialStateProperty.all(Colors.lightBlueAccent)),
                onPressed: () => _selectDate(context),
                child: const Text(
                 'Select birth date',
                 style: TextStyle(
                  color: Colors.black,
                 ),
                ),
               ),

               ElevatedButton(
                 style: ButtonStyle(
                  backgroundColor:
                      MaterialStateProperty.all(Colors.lightGreen)),
                 onPressed: () => _register(context),
                 child: const Text(
                  'Register',
                  style: TextStyle(
                   color: Colors.black,
                   fontWeight: FontWeight.bold,
                  ),
                 ),
               ),
             ],
           ),
         ),
       ),
     ),
   );
  }
}
```

DCT Client/lib/location/map_widget.dart

```
import 'dart:async';

import 'package:dct_client/jobs/get_status_and_locations.job.dart';
import 'package:dct_client/jobs/send-geodata.job.dart';
import 'package:dct_client/logger.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:geolocator/geolocator.dart';

// import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:google_maps_flutter_heatmap/google_maps_flutter_heatmap.dart';
import 'package:wakelock/wakelock.dart';

import '../constants.dart';
import '../push_notifications.dart';
import 'dtos/get_locations.dto.dart';
import 'enums/crowd_status_enum.dart';

class MapWidget extends StatefulWidget {
  const MapWidget();

  @override
  _MapWidgetState createState() => _MapWidgetState();
}

class _MapWidgetState extends State<MapWidget> {
  StreamSubscription<Position> _positionStreamSubscription;
  Position _position;
  CrowdStatus _status;

  final Set<Heatmap> _heatmaps = {};

  final Set<Circle> _circles = {};
```

```
  Color _resolveStatusColor() {
  switch (_status) {
    case CrowdStatus.good:
      return Colors.green;

    case CrowdStatus.ok:
      return Colors.yellow;

    case CrowdStatus.bad:
      return Colors.red;

    default:
      return Colors.red;
  }
 }

@override
 void dispose() {
   if (_positionStreamSubscription != null) {
     _positionStreamSubscription.cancel();
     _positionStreamSubscription = null;
   }
   Wakelock.disable();
   super.dispose();
 }

 static Set<Heatmap> _createHeatmapsFromLocations(
     List<LocationDto> locations) {
   return locations.asMap().entries.map((entry) {
     final l = entry.value;
     final i = entry.key;
```

```
Future<void> _sendNotification(CrowdStatus newStatus) async {
  try {
    if (_status != null && newStatus != _status) {
      await scheduleNotification('Crowd status changed!');
    }
  } catch (e) {
    logger.e(e);
  }
}

Future<void> _geodataListener(Position position) async {
  if (!mounted) return;
  if (position == null) return;

  sendGeoData();
  final dto = await getStatusAndLocations();
  if (dto == null) return;

  await _sendNotification(dto.status);

  final heatmaps = await compute(_createHeatmapsFromLocations, dto.locations);

        intervalDuration: sendGeodataDurationInverval)
      .listen(_geodataListener);
}
```

```
  static List<WeightedLatLng> _createPoints(LatLng location) {
    final List<WeightedLatLng> points = <WeightedLatLng>[];
    for (var i = -1; i <= 1; i++) {
      for (var j = -1; j <= 1; j++) {
        points.add(_createWeightedLatLng(
            location.latitude + i, location.longitude + j, 1));
      }
    }

    return points;
  }
setState(() {
      _position = position;
      _status = dto.status;

      _heatmaps.clear();
      _heatmaps.addAll(heatmaps);

      _circles.clear();
      _circles.add(Circle(
          strokeColor: Colors.blue,
          center: LatLng(position.latitude, position.longitude),
          radius: dto.radius,
          circleId: CircleId('radius-circle')));
    });
  }
```

```
  @override
  void initState() {
    super.initState();

    Wakelock.enable();
    _positionStreamSubscription = Geolocator.getPositionStream(
          desiredAccuracy: LocationAccuracy.best,

static WeightedLatLng _createWeightedLatLng(
      double lat, double lng, int weight) {
    return WeightedLatLng(point: LatLng(lat, lng), intensity: weight);
  }

  @override
  Widget build(BuildContext context) {
    if (_position == null) {
      return const CircularProgressIndicator(
          valueColor: AlwaysStoppedAnimation<Color>(Colors.blue));
    }

    return Scaffold(
        body: Container(
      color: _resolveStatusColor(),
      child: Padding(
        padding: const EdgeInsets.all(12.0),
        child: GoogleMap(
          initialCameraPosition: CameraPosition(
            target: LatLng(_position.latitude, _position.longitude),
            zoom: defaultMapsZoom,
          ),
          myLocationEnabled: true,
          circles: _circles,
          heatmaps: _heatmaps,
        ),
      ),
    ));
  }
}
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Ігор Баклан*
(підпис) (вл. ім’я, прізвище)
“5” квітня 2021 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“6” квітня 2021 р.

«Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей»

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7302.01.000 ТЗ*

на 7 сторінках

Київ – 2021 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повне найменування системи: «Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей».

Умовне позначення: «Dangerous crowds tracker».

Скорочена назва: «DCT».

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовником проекту є кафедра автоматизованих систем обробки інформації і управління факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Представник замовника: Баклан Ігор Всеволодович.

Адреса замовника: м. Київ, пр. Перемоги 37, корп. 18.

Розробником системи є студент групи ІС-73 кафедри Автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» Великий Данило Євгенович.

1.3 Перелік документів на підставі яких створюється система

Завдання на дипломний проєкт є підставою для розробки системи «Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей».

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку робіт над системою – 18.04.2021.

Плановий термін завершення робіт над системою – 04.06.2021.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Призначенням розробки є створення мобільного застосунку та інформаційної системи з виявлення небезпечних місць скупчення великої кількості людей та розрахунку індикатора безпеки на випадок вірогідного захворювання вірусами, зокрема коронавірусом SARS-CoV-2.

2.2 Цілі створення системи

Ціль розробки - попередження про скупчення людей, які можуть бути причиною захворювань вірусів, зокрема коронавірусу SARS-CoV-2

Задачі системи складаються з:

- розробки серверної частини, проектування бази даних;
- реалізації автентифікації та авторизації;
- розробки мобільного застосунку;
- реалізації передачі геоданих із застосунку;
- розробки інтерфейсу з виявленими місцями скупчення людей;
- реалізації алгоритму розрахунку індикатора безпеки на основі анонімних геоданих користувачів;
- відкритих масивах даних та розробки внесення медичних показників та загальних даних користувача.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес визначення класу небезпеки великих скупчень людей. По завершенню роботи отримаємо систему, яка дозволить виявляти скупчення людей та сигналізувати про небезпеку.

4 ВИМОГИ ДО СИСТЕМИ

4.1 Вимоги до системи в цілому

Для функціонування системи необхідно мати сервер та декілька клієнтів зі скачаним застосунком.

Система повинна оброблювати всі виключні ситуації, що пов’язані з некоректними отриманими даними. Усі дані у системі є конфіденційною інформацією і не можуть бути розголошеними.

Система повинна адекватно реагувати на помилки застосування, видавати відповідні повідомлення користувачі та логувати їх.

Система має токен авторизації, який автентифікує користувача.

4.2 Вимоги до функціональних характеристик

Система повинна виконувати наступні функції:

- автентифікація користувачів;
- перегляд мапи з тепловими картами небезпек;
- перегляд та редагування профілю;
- перегляд та редагування медичних показників.

4.3 Вимоги до видів забезпечення

Інформаційна система складається з багатьох користувацьких клієнтів (телефони чи ноутбуки) та серверу.

Вимоги до серверу:

- Центральний процесор з частотою 1 ГГЦ чи потужніший;
- оперативна пам’ять об’ємом 512 МБ чи більше;
- 10 ГБ вільного диску.

Користувач має мати телефон з GPS навігатором або ноутбук.

Вимоги до клієнта:

- sdk версія Android – 16;
- deployment target версія IOS – 9;
- google chrome.

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

У таблиці 5.1 наведено календарний план робіт та терміни їх виконання.

Таблиця 5.1 – Календарний план виконання робіт

№	*Назва етапу*	*Термін виконання*
1	Вивчення рекомендованої літератури	12.02.2021
2	Аналіз існуючих аналогів	20.02.2021
3	Постановка та формалізація задачі	03.03.2021
4	Розробка інформаційного забезпечення	16.03.2021
5	Алгоритмізація задач	20.03.2021
6	Обґрунтування використовуваних технічних засобів	29.03.2021
7	Розробка програмного забезпечення	14.04.2021
8	Налагодження програми	22.04.2021
9	Виконання графічних документів	29.04.2021
10	Оформлення пояснювальної записки	04.05.2021
11	Подання ДП на попередній захист	14.05.2021
12	Подання ДП на основний захист	04.06.2021
13	Подання ДП рецензенту	07.06.2021

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

Для перевірки роботи програмного забезпечення інформаційної системи буде виконано ряд функціональних тестів. Робота цих тестів націлена на перевірку правильності роботи компонентів системи та застосування її в цілому. Також перевіряється виконання передумов тестових сценаріїв, прогонка юніт та e2e тестів.

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008158041

Дата перевірки:
03.06.2021 14:43:50 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2021 17:27:27 EEST

ID користувача:
77149

Назва документа: **Velikij_bachelor_is73_v2**

Кількість сторінок: **30** Кількість слів: **4430** Кількість символів: **32773** Розмір файлу: **667.48 KB** ID файлу: **1008224492**

18%
Схожість

Найбільша схожість: **6.57%** з джерелом з Бібліотеки (ID файлу: **1008224483**)

8.69% Джерела з Інтернету	198	Сторінка 32
15.2% Джерела з Бібліотеки	583	Сторінка 34

0.7% Цитат

Цитати	1	Сторінка 35

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

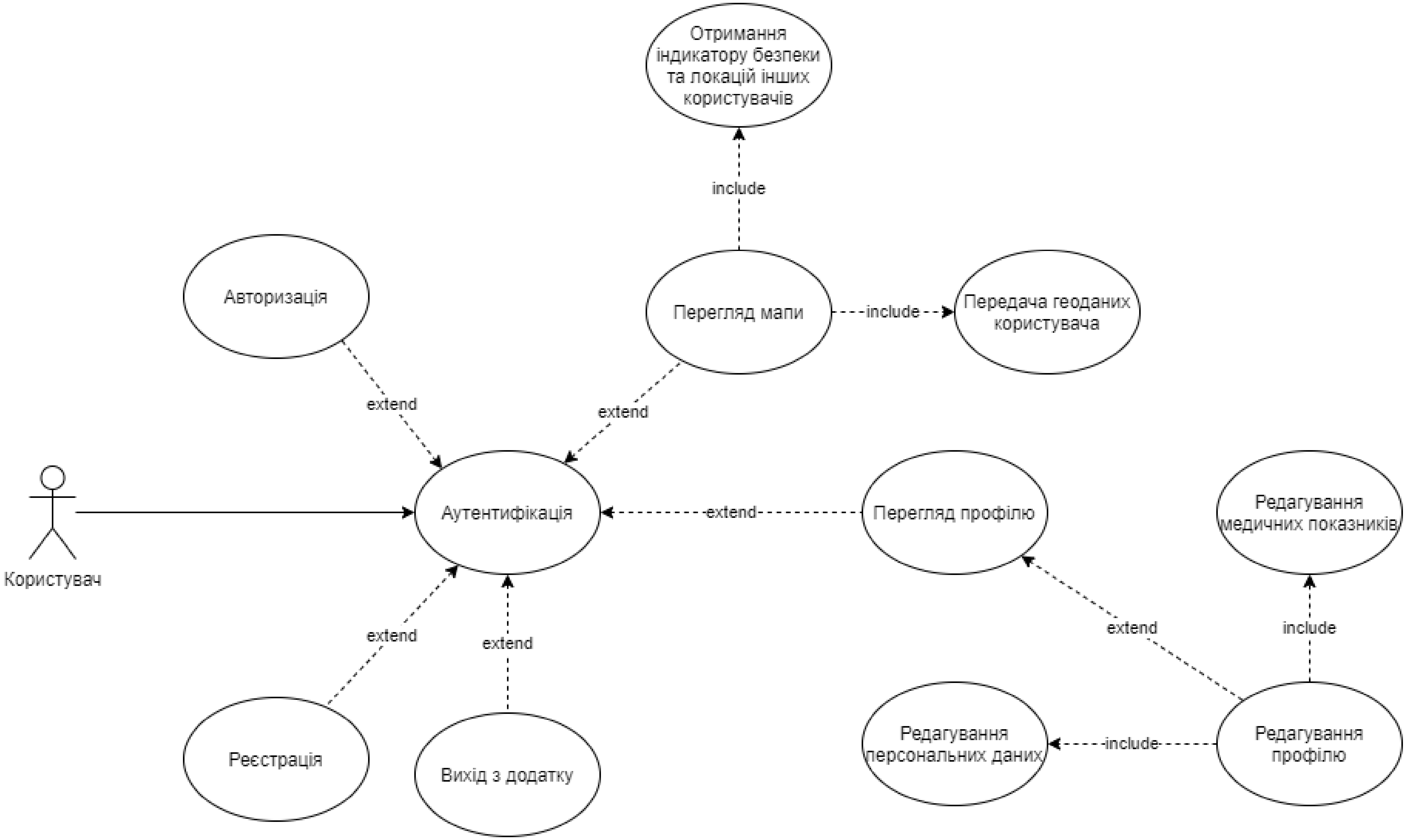
Замінені символи	51

Графічний матеріал
до дипломного проєкту

на тему: «Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей»

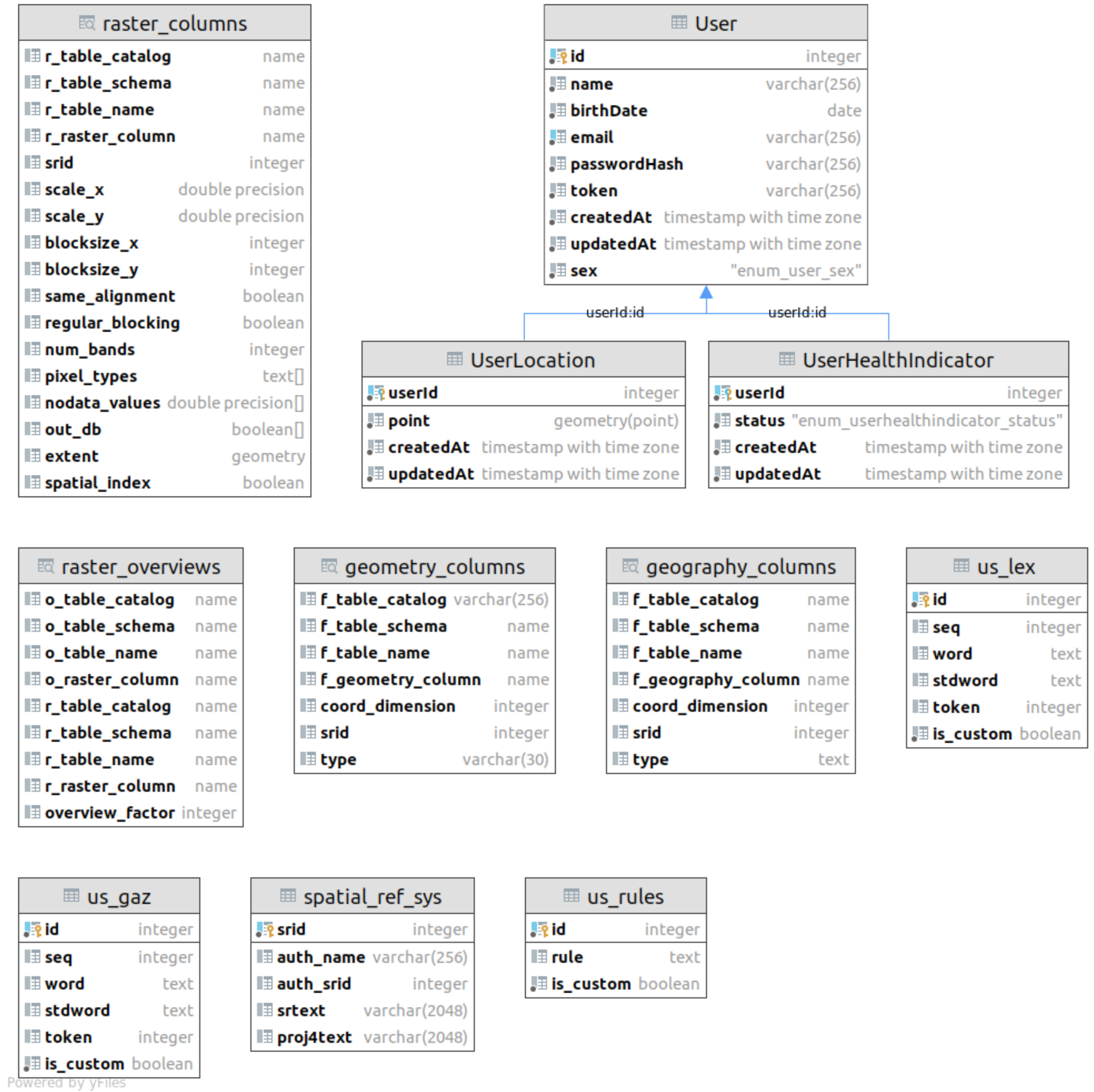
Київ – 2021 року

ДП 7302.02.000 ССВ



Зм.	Арк.	№ документа	Підпис	Дата	ДП 7302.02.000 ССВ			
Розробив		Великий Д. Є.			*Схема структурна варіантів використання*	Літера	Маса	Масштаб
Перевірив		Баклан І. В.				Аркуш 1	Аркушів 1	
Консульт.					*Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей*	*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73*		
Н. кон.		Сперкач М. О.						
Затвердив		Павлов О. А.						

ДП 7302.03.000 СБД



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Великий Д. Є.		
Перевірив		Баклан І. В.		
Консульт.				
Н. кон.		Сперкач М. О.		
Затвердив		Павлов О. А.		

ДП 7302.03.000 СБД

Схема бази даних

Літера | Маса | Масштаб

Аркуш 1 | Аркушів 1

Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей

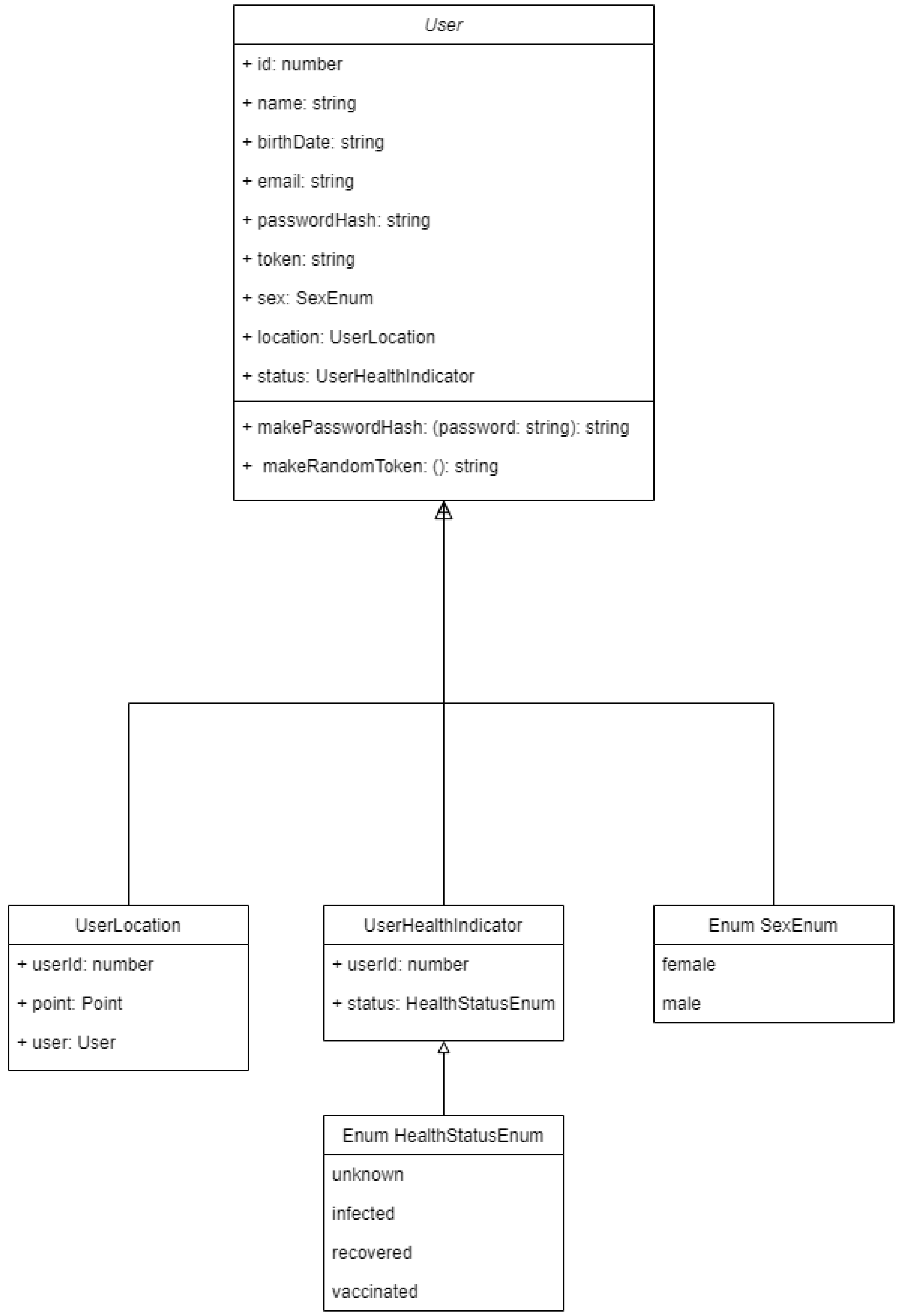
КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-73

ДП 7302.04.000 СПК

```
const arr = [
  {
    latitude: 50.4531823,
    longitude: 30.47838182,
    sex: SexEnum.FEMALE,
    status: HealthStatusEnum.VACCINATED,
    age: 20,
  },{
    latitude: 50.4539958,
    longitude: 30.47819293,
    sex: SexEnum.MALE,
    status: HealthStatusEnum.UNKNOWN,
    age: 17,
  }
];

const agentLocation = {
  latitude: 50.4535832,
  longitude: 30.47833843,
};

private medCoefficient(status: HealthStatusEnum): number {
  if (status === HealthStatusEnum.INFECTED) {
    return this.STATUS_COEFFICIENTS.INFECTED;
  }
  if (status === HealthStatusEnum.RECOVERED) {
    return this.STATUS_COEFFICIENTS.RECOVERED;
  }
  if (status === HealthStatusEnum.VACCINATED) {
    return this.STATUS_COEFFICIENTS.VACCINATED;
  }

  return this.STATUS_COEFFICIENTS.UNKNOWN;
}

private ageCoefficient(age: number) {
  return 0.002 * age + 0.95;
}

private sexCoefficient(sex: SexEnum) {
  if (sex === SexEnum.MALE) {
    return 1.2;
  }
  return 1;
}

private radian(deg: number): number {
  return deg * Math.PI / 180;
}

private distance(
  latitude: number,
  longitude: number,
  agentLocation): number {
  const agentLatitudeRadian = this.radian(agentLocation.latitude);
  const agentLongtitudeRadian = this.radian(agentLocation.longitude);
  const latRadian = this.radian(latitude);
  const longRadian = this.radian(longitude);

  return 2 * this.EARTH_RADIUS * Math.arcsin(
    Math.sqrt(
      Math.pow(
        Math.sin((agentLatitudeRadian - latRadian)/2)
        ,2) + Math.cos(agentLatitudeRadian) + Math.cos(latRadian) * Math.pow(
        Math.sin((agentLongtitudeRadian - longRadian)/2)
        ,2)
      )
    );
}

private getLocationPoints(
  latitude: number,
  longitude: number,
  status,
  sex: SexEnum,
  age: number,
  agentLocation
): number {
  const distance = this.distance(latitude, longitude, agentLocation);
  const kMed = this.medCoefficient(status);
  const kSex = this.sexCoefficient(sex);
  const kAge = this.ageCoefficient(age);

  return Math.ceil(this.FORMULA_CONSTANT * (kMed / distance) * kAge * kSex);
}

private getHealthStatus(pointSum: number): CrowdStatusEnum {
  if (pointSum > this.POINTS_LIMITS.BAD) {
    return CrowdStatusEnum.BAD;
  }
  if (pointSum > this.POINTS_LIMITS.OK) {
    return CrowdStatusEnum.OK;
  }

  return CrowdStatusEnum.GOOD;
}

private calculateHealthStatusAndPoints(
  data, agentLocation
): {
  status: CrowdStatusEnum;
  points: number;
} {
  const locations = data.map((l) => ({
    ...l,
    points: this.getLocationPoints(l.latitude, l.longitude, l.status, l.sex, l.age, agentLocation),
  }));

  const pointSum = locations.reduce((acc, l) => acc + l.points, 0);

  const status = this.getHealthStatus(pointSum);

  return {
    status,
    points: pointSum,
  };
}

const result = this.calculateHealthStatusAndPoints(arr, agentLocation);
```

Зм.	Арк.	№ документа	Підпис	Дата	ДП 7302.04.000 СПК		
Розробив		Великий Д. Є.			*Схема прикладу коду для обраховування індексу небезпеки*	Літера	Маса	Масштаб
Перевірив		Баклан І. В.				Аркуш 1	Аркушів 1
Консульт.					*Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей*	*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73*	
Н. кон.		Сперкач М. О.					
Затвердив		Павлов О. А.					

ДП 7302.05.000 ССКЛ



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Великий Д. Є.		
Перевірив		Баклан І. В.		
Консульт.				
Н. кон.		Сперкач М. О.		
Затвердив		Павлов О. А.		

ДП 7302.05.000 ССКЛ

Схема структурна класів програмного забезпечення

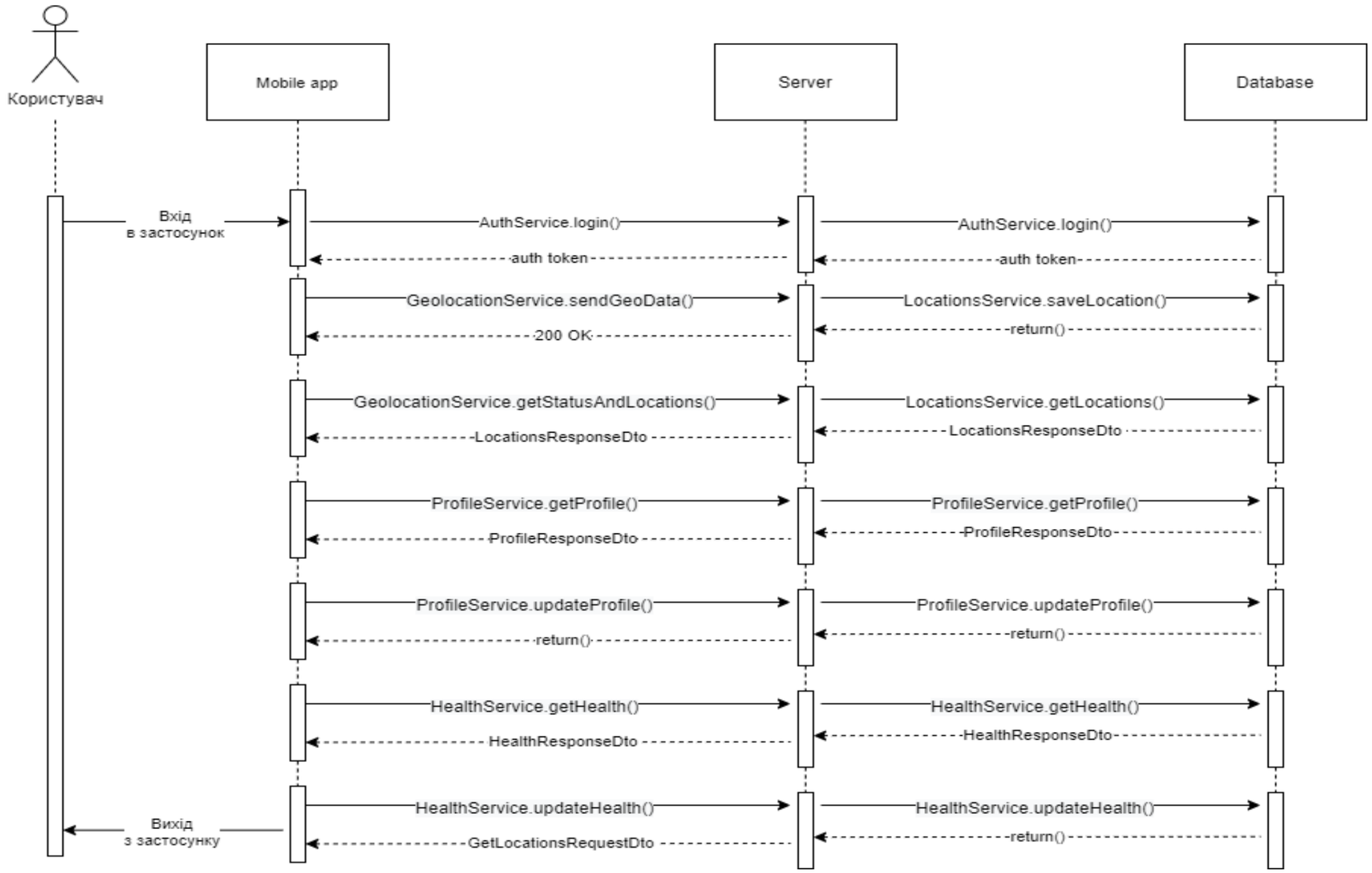
Літера	Маса	Масштаб

Аркуш 1 | Аркушів 1

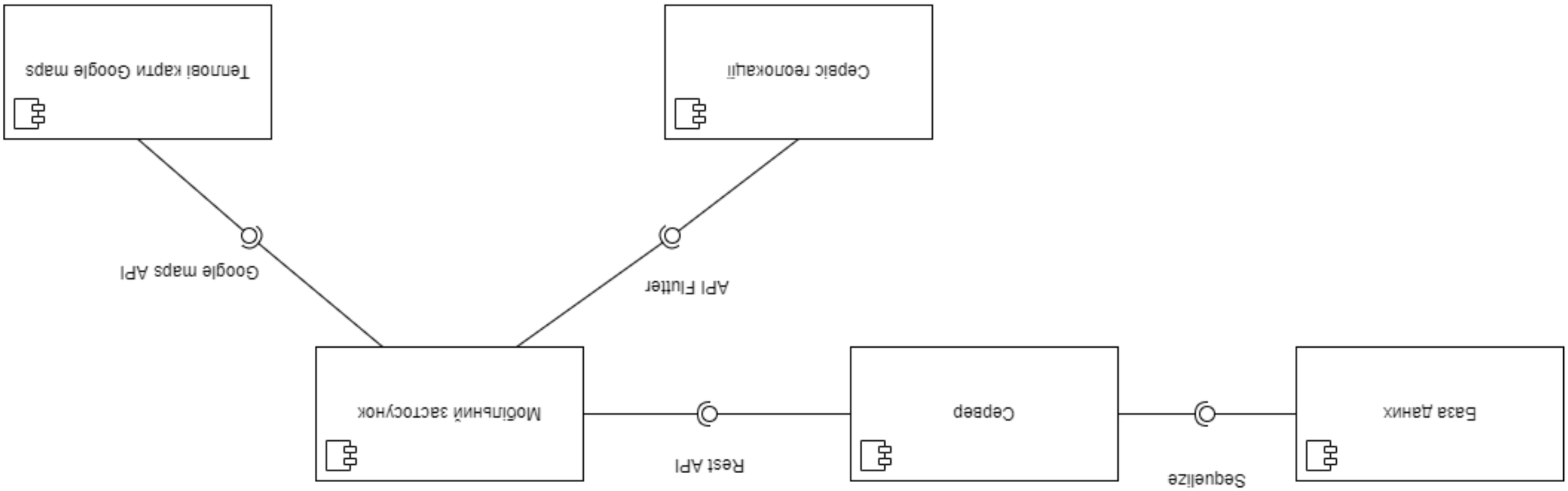
Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей

КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-73

ДП 7302.06.000 ССП



					ДП 7302.06.000 ССП	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності	Літера / Маса / Масштаб
Розробив		Великий Д. Є.				
Перевірив		Баклан І. В.				Аркуш 1 / Аркушів 1
Консульт.					Інформаційна система з виявлення небезпечних місць скупчення великої кількості людей	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73
Н. кон.		Сперкач М. О.				
Затвердив		Павлов О. А.				



Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна компонентів програмного забезпечення*				
Розробив		Великий Д. Є.							
Перевірив		Бакпан . В.						Аркуш 1	Аркушів 1
Консульт.					*Інформаційна система з вдосконаленням небезпечних місць скупчення великої кількості людей*				
Н. кон.		Снєжкч М. О.						*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73*	
Затвердив		Павлов О. А.							

ДП 7302.07.000 ССКО