

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

**Дисциплина**

Интернет за мобилни устройства

**Курсов проект**

**Тема 11:**

„Географска система“

**Студенти:**  **Преподавател:**

Илиян Костов гл.ас.д-р Велислав Колесниченко

фак.№ 23651224

Данаил Михайлов

фак.№ 23651231

Специалност : СИ

Катедра: Софтуерни и интернет технологии

**Съдържание**

[1. Задание 2](#_Toc1685831765)

[2. Анализ на проблема 5](#_Toc529428667)

[2.1 Функционални изисквания 5](#_Toc1606767388)

[2.2 Структура на проекта 5](#_Toc1140644588)

[2.3 Дефиниция на модулите на системата 7](#_Toc302102288)

[3. Проектиране на системата 9](#_Toc653145384)

[3.1 Проектиране на отделните модули 9](#_Toc785991634)

[3.2 UML 9](#_Toc723016889)

[3.3 Концептуален модел на базата от данни 10](#_Toc955824080)

[4. Реализация на системата 11](#_Toc1729807120)

[4.1 Реализация на базата от данни 19](#_Toc1269603202)

[4.2 Реализация на слоя за работа с базата данни 21](#_Toc1386471815)

[4.3 Реализация на бизнеслогика и графичен интерфейс 23](#_Toc964934417)

[4.4 Реализация на модул за регистриране на събития в системата 25](#_Toc34183164)

[5. Тестови резултати 25](#_Toc1967532896)

[5.1 JUnit tests 25](#_Toc1357625866)

[5.2 Функционални тестове 25](#_Toc1984846762)

[5.3 Интеграционни тестове 25](#_Toc884879274)

# **Задание**

Да разработите информационна система, което позволява на потребителите да инспектират опасностите и рисковете във всяка точка на света.

Данните се предоставят под формата на геопространствени файлове (ESRI shapefiles) и приложението ви трябва да може да визуализира тези данни на карта и да показва метаданните, свързани с всяка форма. Примерни данни за опасност, които могат да бъдат анализирани по този начин, са типовете почви за различни региони на света (в Северна България един от доминиращите почвени типове например са калциевите черноземи).

В допълнение към това, трябва да се създаде рамка за проверка за регистриране на различни видове събития от вашето приложение, които могат да бъдат експортирани по-късно при заявка на администратор. Тези събития представляват различни видове дейности, които потребителите могат да извършват, като вход, изход, заявка на информация за типа на почвата за дадено място. Важно е да следите какво правят вашите потребители в системата.

**Публична част**

Публичната част от вашите проекти трябва да бъде видима без удостоверяване.

Това включва началната страница на приложението, формулярите за вход и регистрация на потребителите. Хората, които не са удостоверени, не могат да взаимодействат с останалата част от приложението. Администраторите се предварително регистрирани, само обикновените потребители могат да се регистрират чрез регистрационната формата. Всеки администратор има и разрешенията на обикновените потребители. Събитие за одит за влизане и излизане трябва да се записват.

**Частна част (само за потребители)**

Приложението предоставя на потребителите потребителски интерфейс за проверка на различни видове опасности и рискове, налични на платформата. Трябва да бъдат представени опции за избират на типа данни за проверка („Тип почви“„Честота на земетресението“) и карта, на която да виждат информация за регион по свой избор. Действията, достъпни за потребителите, свързани с извличането на опасност / риск са:

Дълго кликване върху картата Извлича многоъгълна зона от категорията на опасност / риск, към която принадлежи кликнатото местоположение и се визуализира на картата. Например ако потребителят кликне двукратно върху София, когато избраният тип опасност / риск е „Тип почва”, тогава приложението ви трябва да извлече почвения регион, към който принадлежи София, и да го визуализира на картата.

Като част от задължителните проектни изисквания трябва да се поддържат само два типа данни - “Soil Type” and “Earthquake Frequency”. Уверете се, че формите за различни типове почви (поле DOMSOIL) в случай на “Soil Type” от различни типове (DN поле) в случай е "Честота на земетресението" са запълнени с различен цвят (различните типове почви са с различен цвят, еднаквите разбира се са с еднакъв цвят)

Щракнете или задръжте върху зона на картата, за която са извлечени вече данни след двойно щракване. След като имаме една или повече области, визуализирани на картата, потребителите биха искали да видят повече подробности, свързани с тази област. Например „Размер в квадратен км“ или „тип на почвата“ и т.н. · Изчистете картата с щракване с бутон или щракнете върху контекстното меню Премахва всичко визуализирано на картата от действията по-горе.

Изчистете картата, ако е избран друг тип данни

Трябва да се записват одиторски събития за всяко извличане на фигура с дълго щракване върху картата. Широчината и дължината на щракнатото местоположение и исканият тип данни трябва да бъдат записани като част от записа за проследяване действията на потребителя.

**Администраторска част**

Системните администратори трябва да имат административен достъп до системата и разрешения за администриране и преглед на информация, която не е достъпна за редовни потребители. Като част от задължителните функции, администраторите ще могат да преглеждат само отчет за одит за това, какви действия са извършени от потребителите (вижте по-горе за подробности за това, което трябва да бъде проследено). За да се покаже отчетът, администраторите трябва да предоставят период от време, за който се изисква отчет, и по желание низ за заявка за филтриране на записите. Не трябва да се разрешава на администраторите да експортират данни за период, по-дълъг от 3 месеца.

**Низът на заявката поддържа следния формат:**

<word1> <word2> <word3> Търси някоя от думите <word1>, <word2> или <word3> в потребителско име, IP и персонализираните данни, свързани със записите (пълно-търсене на текст, подобно на това, което прави Google).

„<word1> <word2> <word3>“ Търси цялата фраза „<word1> <word2> <word3>“ в потребителско име, IP и персонализирани данни, свързани със записите (пълно текстово търсене).

<Ключ>: <стойност> или <ключ>: "<стойност>" Търси <стойност>, използвайки пълно текстово търсене (вижте правилата по-горе), но само за полето <ключ>. Например „Потребителско име: krasi“ ще търси krasi в потребителско име, като използва пълно текстово търсене

<key> = <value> Търси точно съвпадение на <value> за полето <key>. Например„ DataType = почвата“ ще търси dataType, равен на „почвата“ (това ще върне всички записи с dataType „почвата“, но няма да съвпада с записи с dataType „почвен тип“, който „dataType: почвата“ ще съвпада)

Докладът трябва да бъде под формата на таблица, която включва следната

информация:

* Потребителско име
* IP
* Време на събитието
* Допълнителни подробности част от съобщението (зависи от действието, което се проследява)

Таблицата трябва да поддържа страници и сортиране по потребителско име, IP и време на събитието (сортирането по подразбиране е по време на регистрация на събитието). И администраторите също трябва да имат възможност да изтеглят отчета под формата на CSV файл.

**Супер Администраторска част**

Супер администратора е само един в системата, той има достъп да

функционалностите на потребителите и администраторите.

Като попълнение супер администратора може да прави регистрирани

потребители администратори и да блокира достъпа на регистрирани потребители

Само супер администратора вижда информация за действията правени от

Системните администратори в одита.

**REST API**

Основната логика на приложението (рамка за одитност + сървър на карти

(извличане на данни за формата)) трябва да се реализира като микро-услуги REST.

**База данни**

Предоставена е информацията за почвите и рисковите зони под формата на

Shape файлове, които трябва да се разархивират в база от данни. Може сами да изберете база за данните. PostgreSQLи PostGIS поддържа работа с географски обекти.

# **Анализ на проблема**

Проекта се основава на разработване на информационна система за визуализация на опасности и рискове чрез географска карта. Системата трябва да бъде интуативна както и за крайният потребител, така и за администраторите в приложението.

## **2.1 Функционални изисквания**

Основните функции на проекта са предназначени на 2 различни типа потребители: клиенти, администратори и супер администратори. Функционалностите изисквания са както следва:

1. Клиенти

* регистрация и влизане в системата
* изход от системата
* извличане на различни типове опасности (почви и земетресения), чрез свободен избор на географска карта

1. Администратори

* същите функционалности да притежават като на клиентите
* филтриране на отчет на действия, генерирани от клиентите
* сортиране на отчета на действия от клиентите
* експортиране на отчета в CSV (comma separated values) формат

1. Супер администратори

* същите функционалности да притежават като на администраторите
* филтриране на отчет на действия, генерирани от администраторите
* сортиране на отчета на действия от администраторите
* блокиране на достъпа на клиенти и администратори
* превръщане на потребители към администратори и обратното

## **2.2 Структура на проекта**

Структурата на проекта се основава на 2 различни типове системи: клиентска и сървърна.

**Клиентско приложение:**

Клиентското приложение се намира през Android мобилната платформа, като избран език за програмирането му е Kotlin. Потребителският интерфейс е написан изцяло с помощта на Jetpack Compose технологията, където целия интерфейс се описва изцяло като код.

За избор на софтуерната архитектура е следване принципа на Model-View-ViewModel (MVVM) моделът, където:

* View: визуалният интерфейс, предназначен за крайният потребител
* ViewModel: бизнес логика, която обработва потребителските действия и комуникира с външни системи (в нашия случай, със сървърът)
* Model: представлява структурата от данни, която се организира в зависимост от релациите с други структури от данни (примерно модел потребител има релация с тип на потребителя, типът риск с почви, земетресения и т.н).

**Сървърно приложение**

Сървърната страна е междуплатформено, т.е може да се внедри във всяка операционна система. Избраният език за писане е Java със Spring фрамелурка за разработване на REST API service, да може да обработва клиентските заявки съответно.

Софтуерната му архитектура е принципът на Model-View-Controller (MVC) моделът, където:

* Model: представлява структурата от данни, която се организира в зависимост от релациите с други структури от данни (примерно модел потребител има релация с тип на потребителя, типът риск с почви, земетресения и т.н).
* View: интерфейс с който комуникира с крайният потребител
* Controller: междинен компонент между model и view, който взима клиентската заявка от view, обработва ги с помощта на Model и връща крайният резултат обратно към потребителя

И в двете приложения се прилага принципът: *,,dependency injection’’* (DI), който е начин за описване на модули от кои други външни модули той зависи, за да може да обработи съответната задача за който е предназначен. Примерно при стандартна обектно-ориентирана система, начина на описване на зависимости между модули се описва когато първо се създават обекти в дефиницията на класа и след това се предава създадения обект към зависещия модул, докато с DI зависимите модули не се създават вътрешно в класа, а цялата тази отговорност се пренася на зависимия модул чрез използването на контейнер, или още наречен: *,,inversion of control’’* (IoC) .

Контейнерът е отговорен за управлението на жизненият цикъл на всички зависими модули и тяхната инициализация, според конфигурацията им. По отношение на реализация на проекта се използва Dagger контейнер на клиентският Андроид и Spring Framework на сървърният Java.

Има 3 различни имплементации за имплементиране на DI:

1. Constructor injection: зависимите модули се предават през конструктора

2. Setter injection: зависимите модули се предават през публичен метод за модификация на обект

3. Method injection: зависимите модули се предават през публични методи, които извършват и се нуждаят от външни зависими модули, за да могат да обработят услугата си.

## **2.3 Дефиниция на модулите на системата**

Модулите на системата са разделени между клиентското и сървърното приложение.

**Клиентско приложение**

* + App: Този модул съдържа Dagger контейнерът, който управлява всички зависими модули
  + config: конфигурационен файл, който съдържа сървърният api договор (още срещан като api contract) в YAML формат
  + Navigation: модул, който се използва за управлението на навигацията към различни компоненти
  + Helpers: помощен файл за извършване на общи функционалности през приложението. Няма обща структура.
  + Entity: модул, който описва всички структури с данни, от който приложението работи над тях. Той описва бизнес логиката на приложението.
  + Mapper: вид помощен модул, който преобразува/филтрира една структура към друга
  + Repository: модул, който обработва entity структурите от данни и ги предава на контролера
  + Controller: модул, който описва всичките API класове, чрез който се прави комуникация със сървърът
  + Storage: модул за съхраняване на потребителската информация след влизане към приложението
  + View: модул който служи за визуализация на потребителският интерфейс
  + ViewModel: модул за дефиниране логиката и обработване на потребителските събития, идващи от View

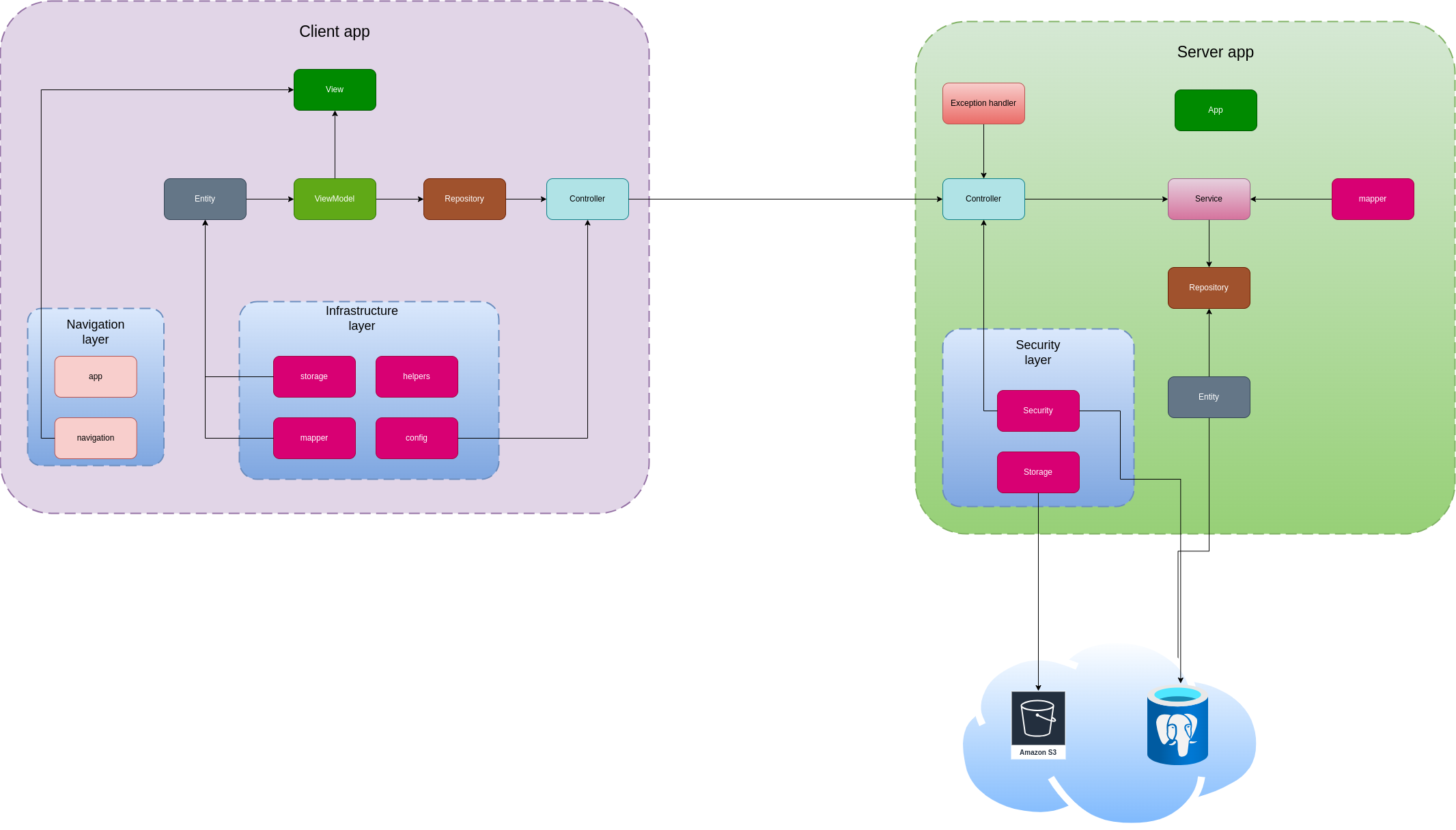
**Сървърно приложение**

* + app: модул, който е стартираща точка на сървърът
  + config: конфигурационен файл за дефиниране на автентикация и удостоверението на клиенти. Също така съдържа автентикация от сървърът към AWS S3
  + Controller: модул, който описва всичките API класове, чрез който се прави комуникация с клиента
  + Entity: модул, който описва всички структури с данни, от който приложението работи над тях.
  + Exception: модул за обработка на всички възможни грешки от страна на REST API.
  + Helpers: помощтен файл за извършване на общи функционалности през приложението. Няма обща структура.
  + Mapper: вид помощен модул, който преобразува/филтрира една структура към друга
  + Repository: модул, който обработва entity структурите от данни и ги предава на контролера
  + Service: модул, който дефинира бизнес правилата на приложението и предава крайният резултат към контролера.
  + Storage: модул за дефиниране на микросервис, който служи за съхраняване на потребителски отчети на събития в AWS S3 storage service

# **Проектиране на системата**

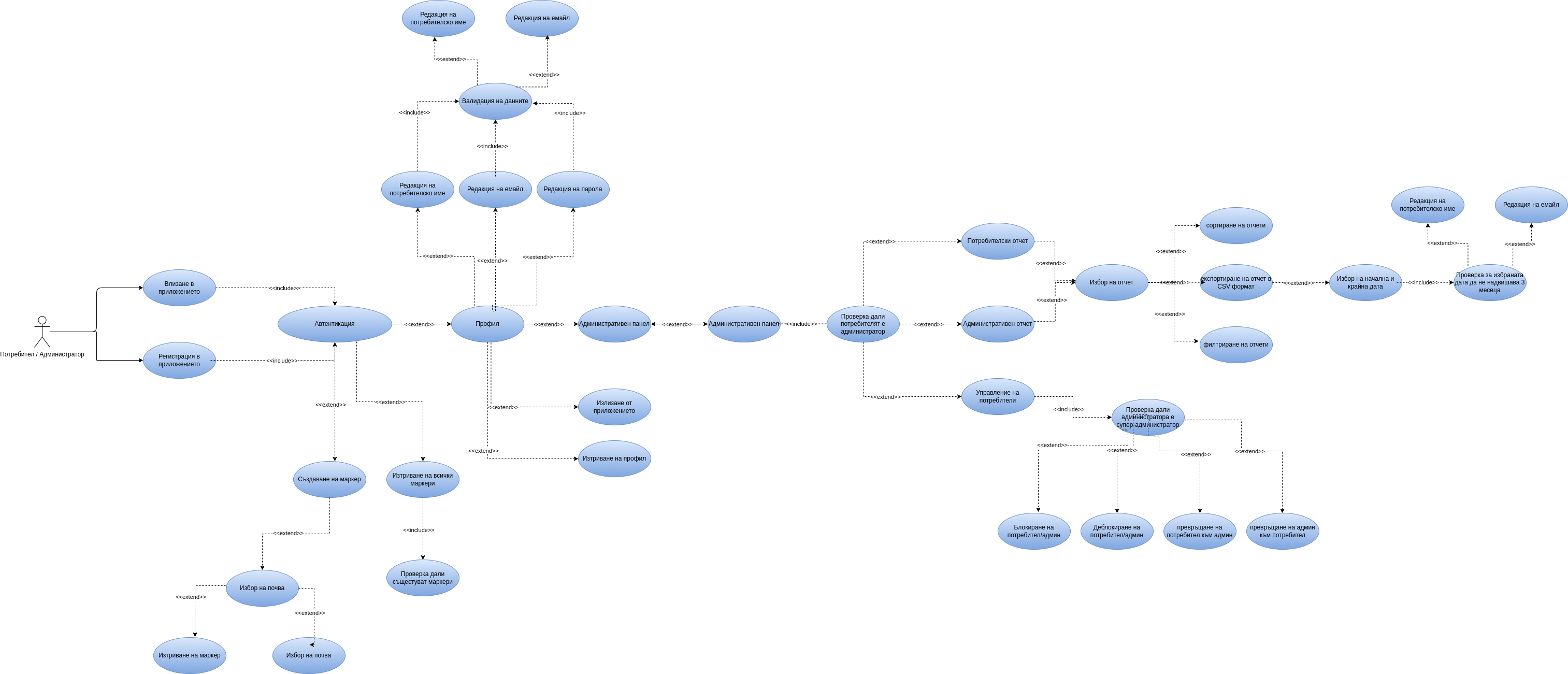
Целта на проекта е да се създаде мобилно и сървърно приложение, което да подпомогне на потребителите с информирането на различни видове почви и земетресенията около света, използвайки интерактивна географска карта, както и да предлага административна част за следене на потребителските действия.

## **3.1 Проектиране на отделните модули**

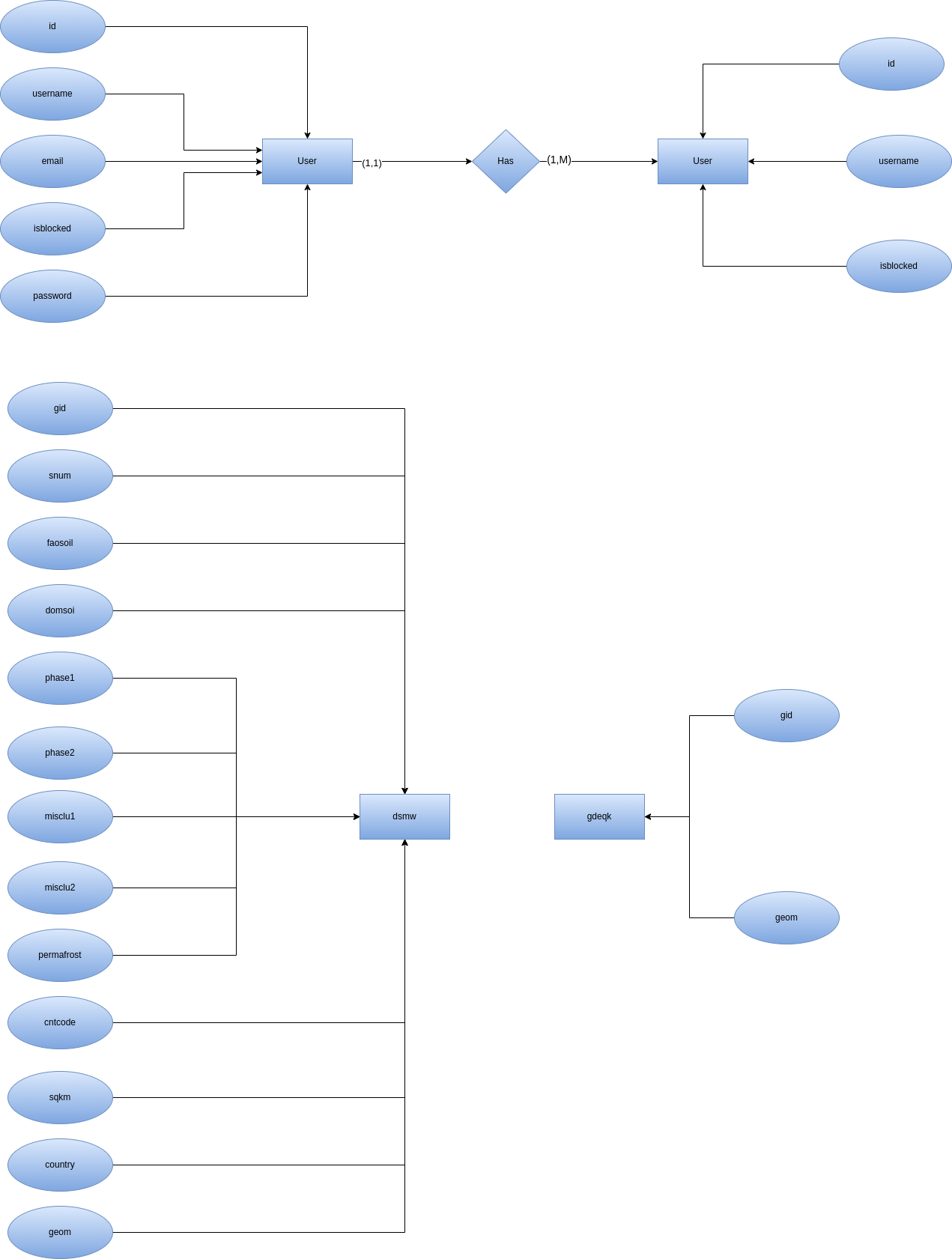


*Фиг. 1: Диаграма на отделни модули*

## **3.2 UML**

*Фиг. 2: Диаграма use case*

## **3.3 Концептуален модел на базата от данни**



*Фиг. 3: Модел на Чен диаграма*

# **Реализация на системата**

Клиентско приложение:

AppModule.kt

@Module  
@InstallIn(SingletonComponent::class)  
object AppModule {  
  
 private const val BASE\_URL = "<http://10.0.2.2:8080/api.tuvarna.geo.com/v1>"  
  
 @Provides  
 fun provideOkHttpClient(userSessionStorageProvider: Provider<UserSessionStorage>): OkHttpClient {  
 return OkHttpClient.Builder()  
 .addInterceptor(AuthInterceptor(userSessionStorageProvider))  
 .build()  
 }  
  
 @Singleton  
 @Provides  
 fun provideAuthControllerApi(client: OkHttpClient): AuthControllerApi {  
 return AuthControllerApi(BASE\_URL, client)  
 }  
  
 @Singleton  
 @Provides  
 fun provideRiskControllerApi(client: OkHttpClient): RiskControllerApi {  
  
 return RiskControllerApi(BASE\_URL, client)  
 }  
  
 @Singleton  
 @Provides  
 fun provideAdminControllerApi(client: OkHttpClient): AdminControllerApi {  
  
 return AdminControllerApi(BASE\_URL, client)  
 }  
  
 @Provides  
 @Singleton  
 fun provideUserSessionStorage(@ApplicationContext context: Context): UserSessionStorage {  
 return UserSessionStorage(context)  
 }  
}  
  
class AuthInterceptor(private val userSessionStorageProvider: Provider<UserSessionStorage>) :  
 Interceptor {  
  
 override fun intercept(chain: Interceptor.Chain): Response {  
 val originalRequest = chain.request()  
 val userSessionStorage = userSessionStorageProvider.get()  
 val accessToken = *runBlocking* **{** userSessionStorage.readAccessToken().first() **}**  
val requestWithAuthHeader =  
 originalRequest.newBuilder().header("Authorization", "Bearer $accessToken").build()  
 return chain.proceed(requestWithAuthHeader)  
 }  
}

AdminControllerApi.kt

*/\*\**  
 *\* Block user*  
 *\**  
 *\* @param email*  
 *\* @param blocked*  
 *\* @return RestApiResponseVoid*  
 *\* @throws IllegalStateException If the request is not correctly configured*  
 *\* @throws IOException Rethrows the OkHttp execute method exception*  
 *\* @throws UnsupportedOperationException If the API returns an informational or redirection response*  
 *\* @throws ClientException If the API returns a client error response*  
 *\* @throws ServerException If the API returns a server error response*  
 *\*/*  
@Suppress("UNCHECKED\_CAST")  
@Throws(IllegalStateException::class, IOException::class, UnsupportedOperationException::class, ClientException::class, ServerException::class)  
fun blockUser(email: kotlin.String, blocked: kotlin.Boolean) : RestApiResponseVoid {  
 val localVarResponse = blockUserWithHttpInfo(email = email, blocked = blocked)  
  
 return when (localVarResponse.responseType) {  
 ResponseType.*Success* -> (localVarResponse as Success<\*>).data as RestApiResponseVoid  
 ResponseType.*Informational* -> throw UnsupportedOperationException("Client does not support Informational responses.")  
 ResponseType.*Redirection* -> throw UnsupportedOperationException("Client does not support Redirection responses.")  
 ResponseType.*ClientError* -> {  
 val localVarError = localVarResponse as ClientError<\*>  
 throw ClientException("Client error : ${localVarError.statusCode} ${localVarError.message.*orEmpty*()}", localVarError.statusCode, localVarResponse)  
 }  
 ResponseType.*ServerError* -> {  
 val localVarError = localVarResponse as ServerError<\*>  
 throw ServerException("Server error : ${localVarError.statusCode} ${localVarError.message.*orEmpty*()} ${localVarError.body}", localVarError.statusCode, localVarResponse)  
 }  
 }  
}

*/\*\**  
 *\* Retrieve user logs*  
 *\**  
 *\* @param userType*  
 *\* @return RestApiResponseListLoggerDTO*  
 *\* @throws IllegalStateException If the request is not correctly configured*  
 *\* @throws IOException Rethrows the OkHttp execute method exception*  
 *\* @throws UnsupportedOperationException If the API returns an informational or redirection response*  
 *\* @throws ClientException If the API returns a client error response*  
 *\* @throws ServerException If the API returns a server error response*  
 *\*/*  
@Suppress("UNCHECKED\_CAST")  
@Throws(IllegalStateException::class, IOException::class, UnsupportedOperationException::class, ClientException::class, ServerException::class)  
fun getLogs(userType: kotlin.String) : RestApiResponseListLoggerDTO {  
 val localVarResponse = getLogsWithHttpInfo(userType = userType)  
  
 return when (localVarResponse.responseType) {  
 ResponseType.*Success* -> (localVarResponse as Success<\*>).data as RestApiResponseListLoggerDTO  
 ResponseType.*Informational* -> throw UnsupportedOperationException("Client does not support Informational responses.")  
 ResponseType.*Redirection* -> throw UnsupportedOperationException("Client does not support Redirection responses.")  
 ResponseType.*ClientError* -> {  
 val localVarError = localVarResponse as ClientError<\*>  
 throw ClientException("Client error : ${localVarError.statusCode} ${localVarError.message.*orEmpty*()}", localVarError.statusCode, localVarResponse)  
 }  
 ResponseType.*ServerError* -> {  
 val localVarError = localVarResponse as ServerError<\*>  
 throw ServerException("Server error : ${localVarError.statusCode} ${localVarError.message.*orEmpty*()} ${localVarError.body}", localVarError.statusCode, localVarResponse)  
 }  
 }  
}

ApiResponse.kt

enum class ResponseType {  
 *Success*, *Informational*, *Redirection*, *ClientError*, *ServerError*  
}  
interface Response  
  
abstract class ApiResponse<T>(val responseType: ResponseType): Response {  
 abstract val statusCode: Int  
 abstract val headers: Map<String,List<String>>  
}  
  
class Success<T>(  
 val data: T,  
 override val statusCode: Int = -1,  
 override val headers: Map<String, List<String>> = *mapOf*()  
): ApiResponse<T>(ResponseType.*Success*)  
  
class Informational<T>(  
 val statusText: String,  
 override val statusCode: Int = -1,  
 override val headers: Map<String, List<String>> = *mapOf*()  
) : ApiResponse<T>(ResponseType.*Informational*)  
  
class Redirection<T>(  
 override val statusCode: Int = -1,  
 override val headers: Map<String, List<String>> = *mapOf*()  
) : ApiResponse<T>(ResponseType.*Redirection*)  
  
class ClientError<T>(  
 val message: String? = null,  
 val body: Any? = null,  
 override val statusCode: Int = -1,  
 override val headers: Map<String, List<String>> = *mapOf*()  
) : ApiResponse<T>(ResponseType.*ClientError*)  
  
class ServerError<T>(  
 val message: String? = null,  
 val body: Any? = null,  
 override val statusCode: Int = -1,  
 override val headers: Map<String, List<String>>  
): ApiResponse<T>(ResponseType.*ServerError*)

Сървърно приложение:

AdminController.java

@RestController

@RequestMapping("/admin")

public class AdminController {

private static final Logger logger = LogManager.getLogger(AdminController.class.getName());

private AdminService adminService;

@Autowired

public AdminController(AdminService adminService) {

this.adminService = adminService;

}

@GetMapping("fetch/logs/{userType}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Retrieve user logs")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Logs retrieved!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<List<LoggerDTO>>> getLogs(@PathVariable("userType") String userType) {

logger.info("Received a request from the admin to query user logs!");

return new ResponseEntity<>(adminService.getLogs(userType), HttpStatus.OK);

}

@PostMapping("save/log/{userType}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Save log")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "User blocked!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<Void>> saveLog(@RequestBody LoggerDTO loggerDTO,

@PathVariable("userType") String userType) {

logger.info("Received a log to be saved later for the admins!: {}", loggerDTO);

return new ResponseEntity<>(adminService.saveLog(loggerDTO, userType), HttpStatus.OK);

}

@PutMapping("users/{email}/block/{blocked}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Block user")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "User blocked!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<Void>> blockUser(

@PathVariable("email") String email, @PathVariable("blocked") Boolean blocked) {

logger.info("Received a request from the admin to block={} a user: {}", blocked, email);

return new ResponseEntity<>(adminService.block(email, blocked), HttpStatus.OK);

}

@PutMapping("users/{email}/promote/{userType}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Promote or denote user to admin")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "User promoted/denoted!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<Void>> promoteUser(

@PathVariable("email") String email, @PathVariable("userType") String userType) {

logger.info("Received a request from the admin to promote={} a user: {}", userType, email);

return new ResponseEntity<>(adminService.promote(email, userType), HttpStatus.OK);

}

@GetMapping("fetch/users/{userType}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Get all users")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Users retrieved!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<List<UserInfoDTO>>> getUsers(@PathVariable("userType") String userType) {

logger.info("Received a request from the admin to fetch all users!");

return new ResponseEntity<>(adminService.getUsers(userType), HttpStatus.OK);

}

}

SecurityConfig.java

@Configuration

@EnableWebSecurity

class SecurityConfig {

private JwtAuthenticationEntryPoint authenticationEntryPoint;

private final DefaultUserDetailsService defaultUserDetailsService;

@Autowired

protected SecurityConfig(JwtAuthenticationEntryPoint authenticationEntryPoint,

DefaultUserDetailsService defaultUserDetailsService) {

this.authenticationEntryPoint = authenticationEntryPoint;

this.defaultUserDetailsService = defaultUserDetailsService;

}

@Bean

protected JwtAuthenticationFilter jwtAuthenticationFilter(JWTTokenProvider jwtTokenProvider) {

return new JwtAuthenticationFilter(jwtTokenProvider, defaultUserDetailsService);

}

@SuppressWarnings({ "removal" })

@Bean

SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http,

JwtAuthenticationFilter authenticationFilter)

throws Exception {

http

.cors().disable()

.csrf().disable()

.addFilterBefore(authenticationFilter,

UsernamePasswordAuthenticationFilter.class)

.exceptionHandling().authenticationEntryPoint(authenticationEntryPoint).and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()

.authorizeHttpRequests()

.requestMatchers("/swagger-ui/\*\*", "/v3/\*\*")

.permitAll()

.requestMatchers(HttpMethod.POST, "/auth/\*\*")

.permitAll()

.requestMatchers("/delete/\*\*").hasRole("ADMIN")

.anyRequest().authenticated();

http.logout(logout -> logout.logoutUrl("/auth/logout")

.invalidateHttpSession(true));

return http.build();

}

@Bean

protected BCryptPasswordEncoder encodePassword() {

return new BCryptPasswordEncoder();

}

@Bean

protected AuthenticationManager authenticationManagerBean() throws Exception {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(defaultUserDetailsService);

authProvider.setPasswordEncoder(new BCryptPasswordEncoder());

return new ProviderManager(Collections.singletonList(authProvider));

}

}

UserServiceImpl.java

@Service

public class UserServiceImpl implements UserService {

private static final Logger logger = LogManager.getLogger(UserServiceImpl.class.getName());

private UserRepository userRepository;

private UserTypeRepository userTypeRepository;

private UserValidateService userValidateService;

private UserMapper userMapper;

private BCryptPasswordEncoder encodePassword;

private AuthenticationManager authenticationManager;

private JWTTokenProvider jwtTokenProvider;

@Autowired

public UserServiceImpl(UserRepository userRepository, UserTypeRepository userTypeRepository,

UserValidateService userValidateService, UserMapper userMapper, BCryptPasswordEncoder encodePassword,

AuthenticationManager authenticationManager, JWTTokenProvider jwtTokenProvider) {

this.userRepository = userRepository;

this.userTypeRepository = userTypeRepository;

this.userValidateService = userValidateService;

this.userMapper = userMapper;

this.encodePassword = encodePassword;

this.authenticationManager = authenticationManager;

this.jwtTokenProvider = jwtTokenProvider;

}

@Override

@Transactional

@SuppressWarnings("squid:S3457")

public RestApiResponse<Void> registerUser(RegisterUserDTO userDto) {

User user = userMapper.toEntity(userDto, encodePassword);

logger.info("Mapping user DTO to entity");

userValidateService.validateUserDoesNotExist(userDto.getEmail());

logger.info("User does not exist with given email: " + userDto.getEmail());

userValidateService.validateUserTypeExists(userDto.getUsertype());

user.setUserType(userTypeRepository.findByType(userDto.getUsertype()));

logger.info("User type exists: " + userDto.getUsertype());

userRepository.save(user);

logger.info("User %s registered successfully", userDto.getEmail());

return new RestApiResponse<>(null, "User registered successfully", 201);

}

@Override

@Transactional

@SuppressWarnings({ "squid:S3457", "squid:S2629" })

public RestApiResponse<LoggedInUserDTO> login(LoginUserDTO userDto) {

User user = userMapper.toEntity(userDto);

logger.info("Mapping user DTO to entity");

userValidateService.validateUserExists(user.getEmail());

logger.info("User exist with given email: " + userDto.getEmail());

User userFromDb = userRepository.findByEmail(user.getEmail());

userValidateService.validatePasswordMatch(encodePassword, user.getPassword(), userFromDb.getPassword());

userValidateService.validateIsUserBlocked(userFromDb.getIsBlocked());

logger.info("User logged in successfully. Sending credentials: %s", userFromDb.toString());

LoggedInUserDTO loggedUserDTO = new LoggedInUserDTO(userFromDb);

JWTAuthResponse jwtAuthResponse = createJWToken(userDto.getEmail(), userDto.getPassword());

logger.info("jwt token: {}", jwtAuthResponse.getAccessToken());

loggedUserDTO.setAccessToken(jwtAuthResponse.getAccessToken());

return new RestApiResponse<>(loggedUserDTO, "User logged in successfully", 201);

}

private JWTAuthResponse createJWToken(String email, String password) {

Authentication authentication = authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(

email, password));

SecurityContext sc = SecurityContextHolder.getContext();

sc.setAuthentication(authentication);

String token = jwtTokenProvider.generateToken(authentication);

JWTAuthResponse jwt = new JWTAuthResponse();

jwt.setAccessToken(token);

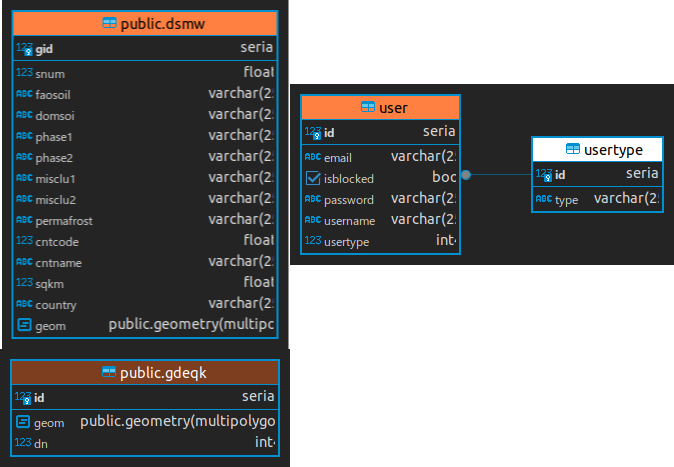
return jwt;

}

}

## **4.1 Реализация на базата от данни**

Таблиците на приложението са разпределени, според изискванията на заданието.



*Фиг. 4: Релационна схема*

Таблиците са следните:

* Почва
  + gid– идентификатор на почвата, тип цяло число
  + snum– последователен код на почвата, тип реално число
  + Faosoil – символ за картографиране на почвата, разделен на 4 компонента: доминираща почва, % картографираща единица на почвата, клас на текстура и наклон, тип низ
  + domsoi – тип доминираща почва, тип низ
  + Phase1, phase2 – подразделения на почвата, тип низ
  + Misclu1, мisclu2 - първият и вторият зависим тип наземна единица, тип низ
  + Permafrost – Показва зоните за вечна замръзване по вид, тип низ
  + Cntcode – уникален номер за всяка държава или област, тип реално число
  + Cntname – код с две букви, уникален за всяка държава или област, тип низ
  + Sqkm – налично количество на материала, тип цяло число
  + Country – името на страната или района, тип низ
  + Geom – коригираната степен на многоъгълника в квадратни километри въз основа на изчисление на равна площ, тип геопространствени данни
* Заметресение
  + Id – идентификатор на земетресението, тип цяло число
  + Geom – коригираната степен на многоъгълника в квадратни километри въз основа на изчисление на равна площ, тип геопространствени данни
* Потребител
  + Id – идентификатор на потребителя, тип цяло число
  + Username – име на потребителя, тип низ
  + email – емайл адрес на потребителя, тип низ
  + Isblocked – статус на блокиране на поробителя, тип булева стойност
  + usertype – идентификатор на таблица ‚ Тип потребител’ , тип цяло число
* Тип потребител
  + Id – идентификатор на типа потребител, тип цяло число
  + type– името на типа потребител, тип низ
  + Релацията между таблиците ‚ Тип потребител‘ и ‚Потребител‘ е 1:М

## **4.2 Реализация на слоя за работа с базата данни**

Избраната база данни за този проект е PostgreSQL с включване на допълнително разширение към него PostGIS, за да можем да работим с геопространствени данни.

За имплементиране на връзката между Java Spring сървърът PostgreSQL базата данни се използва Hibernate. Hibrenate e вид framework, който надгражда концепцията *Object-Relation-Mapping* (ORM). Тя ни позволява да асоциираме обикновени java класове (още наречени *plain-old-java-objects* POJO) със релационни таблици, намиращи се в базата данни.

Имплементацията на Hibernate се извършва с помощта на JPA (java Persistance API), която дефинира кои Pojo класове към кои таблици принадлежи. Заявките към тези класове се извършва с помощта на EntityManager API, който се дефинира в repository слоя. Този мениджър автомагически генерира най-често срещаните заявка за работа с данните (още наречени CRUD операции ( *Create-Read-Update-Delete*)), като използва JPQL (Java Persistant Query Language). В проекта също са използвани ръчно създадени JPQL за определни случай.

Пример за JPA дефиниция на Почва:

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Table(name = "\"dsmw\"")

public class Soil {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer gid;

@Column(nullable = false, unique = false)

private float snum;

@Column(nullable = false, unique = true)

private String faosoil;

@Column(nullable = false, unique = false)

private String domsoi;

@Column(nullable = true, unique = false)

private String phase1;

@Column(nullable = true, unique = false)

private String phase2;

@Column(nullable = true, unique = false)

private String misclu1;

@Column(nullable = true, unique = false)

private String misclu2;

@Column(nullable = true, unique = false)

private String permafrost;

@Column(nullable = true, unique = false)

private double cntcode;

@Column(nullable = true, unique = false)

private String cntname;

@Column(nullable = false, unique = false)

private double sqkm;

@Column(nullable = false, unique = false)

private String country;

@Transient

@Column(nullable = false, unique = false)

private MultiPolygon geom;

}

Пример за repository c EntityManager и ръчна имплементация на JPQL:

@Repository

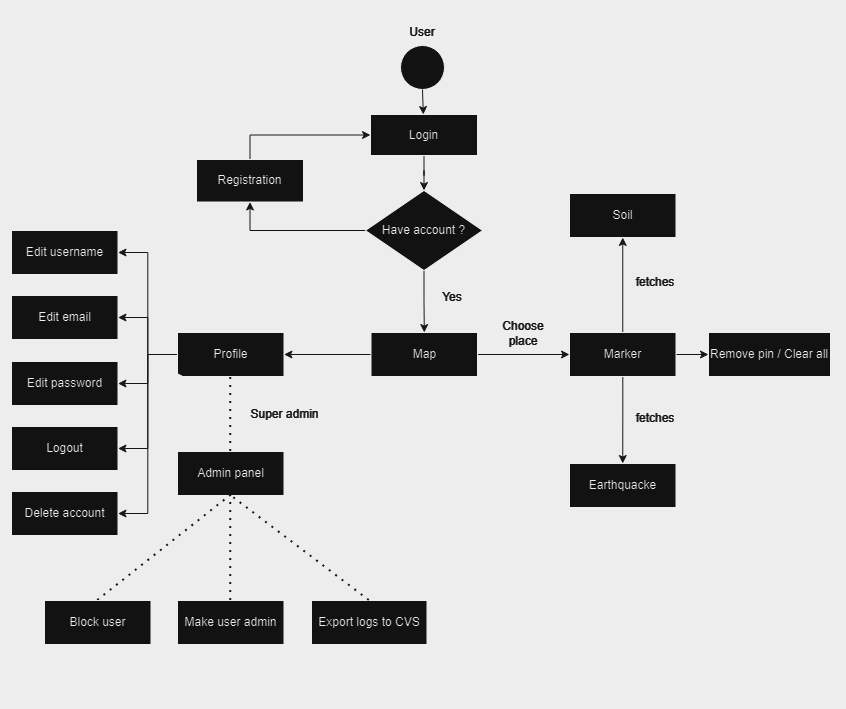
public interface SoilRepository extends JpaRepository<Soil, Integer> {

@Query(value = "SELECT \* FROM dsmw ds WHERE ST\_Contains(ds.geom, ST\_MakePoint(:longitude, :latitude)) = true", nativeQuery = true)

Soil findByLongitudeAndLatitude(@Param("longitude") double longitude, @Param("latitude") double latitude);

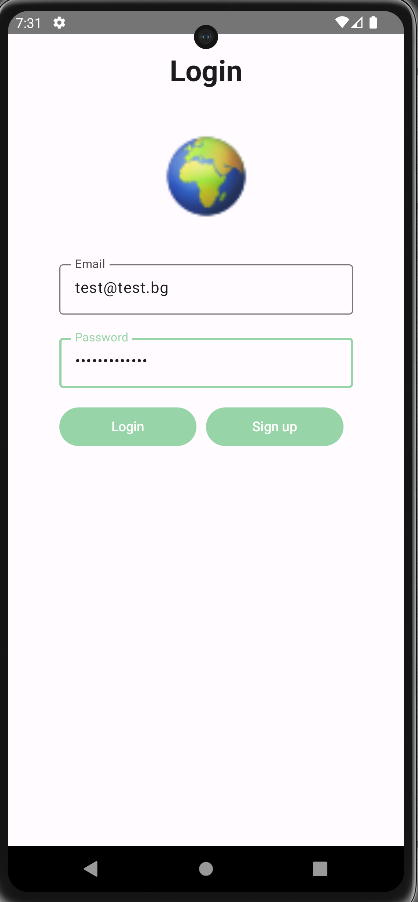
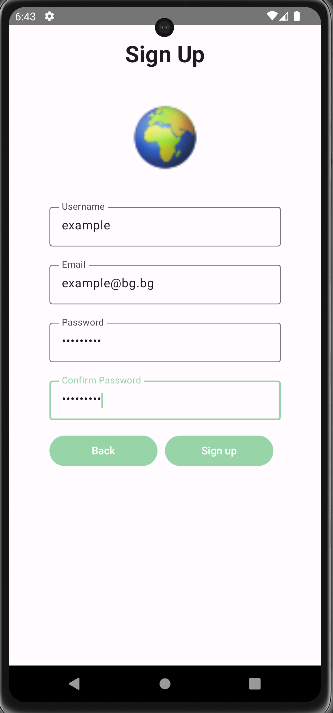
}

## **4.3 Реализация на бизнеслогика и графичен интерфейс**



*Фиг. 5: Диаграма на бизнес логика*

След успешно влизане или регистрация в системата, потребителят получава достъп до интерактивна карта, където може да поставя маркери. След това има възможност да избере какъв тип информация да вижда - за почви или земетресения. Накрая, той може да премахне своите маркери по желание. Всеки потребител разполага с личен профил, където може да редактира своите данни (имейл/парола) или да излезе от системата. Супер администраторът разполага със специален бутон за преглед на събитията в приложението и има право да блокира потребители.

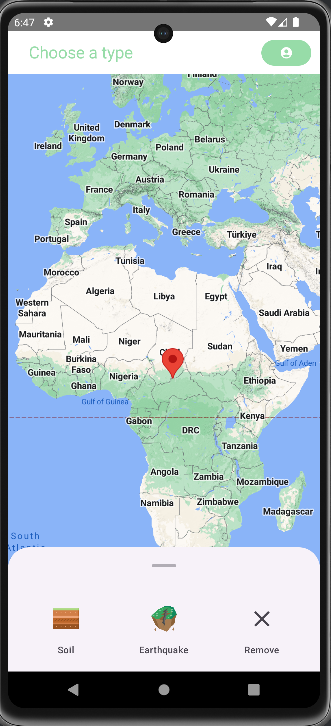
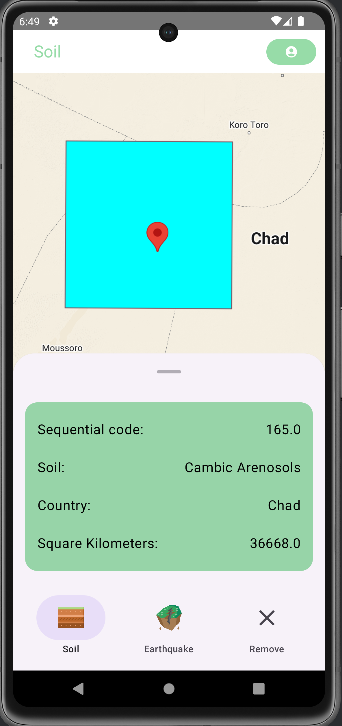
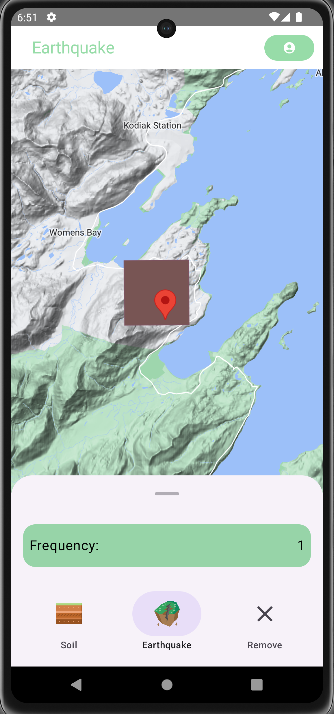
*Фиг. 6: Начален екран*

С бутоните:

* Login - Влизаме в системата
* Back - Връщаме се към предишната страница
* Sign up - Регистрираме се в системата

LoginView.kt

@Composable  
fun LoginView(navController: NavController) {  
 val loginViewModel = hiltViewModel<LoginViewModel>() val user by remember **{** *mutableStateOf*(UserEntity(0, "", "", "", false)) **}**  
LoadingIndicator(  
 stateFlow = loginViewModel.stateFlow.collectAsState().value,  
 navController = navController,  
 route = "home",  
 )  
 Box(contentAlignment = Alignment.Center, modifier = Modifier.*fillMaxWidth*()) **{**  
Column(  
 modifier = Modifier.*padding*(5.*dp*).*fillMaxWidth*(0.8f),  
 verticalArrangement = Arrangement.Center,  
 ) **{**  
Text(  
 text = "Login",  
 style = MaterialTheme.typography.bodyMedium,  
 modifier = Modifier.*padding*(16.*dp*).*align*(Alignment.CenterHorizontally),  
 fontWeight = FontWeight.Bold,  
 fontSize = 30.*sp*,  
 )  
  
 Spacer(modifier = Modifier.*height*(20.*dp*))  
  
 Image(  
 painter = painterResource(id = R.drawable.*earth*),  
 contentDescription = "Earth Icon",  
 modifier = Modifier.*size*(110.*dp*).*fillMaxWidth*().*align*(Alignment.CenterHorizontally),  
 )  
  
 Spacer(modifier = Modifier.*height*(20.*dp*))  
  
 LoginForm(user = user, loginViewModel = loginViewModel, navController = navController)  
 **}**  
 **}**  
}  
  
@Composable  
fun LoginForm(navController: NavController, user: UserEntity, loginViewModel: LoginViewModel) {  
  
 val isEmailValid = user.email.*isEmpty*() *// || Utils.isValidEmail(user.email)*  
val isPasswordValid = user.password.*isEmpty*() *// || Utils.isValidPassword(user.password)*  
val isSubmitBtnDisabled = isEmailValid || isPasswordValid  
  
 val textFieldModifier: Modifier = Modifier.*fillMaxWidth*().*padding*(8.*dp*).*height*(60.*dp*)  
 val errorTextModifier: Modifier = Modifier.*fillMaxWidth*().*padding*(start = 8.*dp*)  
  
 OutlinedTextField(  
 value = user.email,  
 isError = isEmailValid,  
 onValueChange = **{** user.email = **it }**,  
 label = **{** Text("Email") **}**,  
 keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Next),  
 modifier = textFieldModifier,  
 )  
  
 if (isEmailValid) {  
 Text(  
 text = "Email is invalid",  
 color = Color.Red,  
 style = MaterialTheme.typography.bodyMedium,  
 modifier = errorTextModifier,  
 )  
 }  
  
 OutlinedTextField(  
 value = user.password,  
 isError = isPasswordValid,  
 onValueChange = **{** user.password = **it }**,  
 label = **{** Text("Password") **}**,  
 visualTransformation = PasswordVisualTransformation(),  
 keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Next),  
 modifier = textFieldModifier,  
 )  
  
 if (isPasswordValid) {  
 Text(  
 text = "Password does not conform the rules",  
 color = Color.Red,  
 style = MaterialTheme.typography.bodyMedium,  
 modifier = errorTextModifier,  
 )  
 }  
  
 Row(  
 modifier = Modifier.*fillMaxWidth*().*padding*(8.*dp*),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 ) **{**  
Button(  
 onClick = **{** loginViewModel.login(user) **}**,  
 enabled = !isSubmitBtnDisabled,  
 modifier = Modifier.*weight*(1f).*padding*(end = 10.*dp*),  
 ) **{**  
Text("Login")  
 **}**  
Button(  
 onClick = **{** navController.navigate("signup") **}**,  
 modifier = Modifier.*weight*(1f).*padding*(end = 10.*dp*),  
 ) **{**  
Text("Sign up")  
 **}**  
 **}**  
}

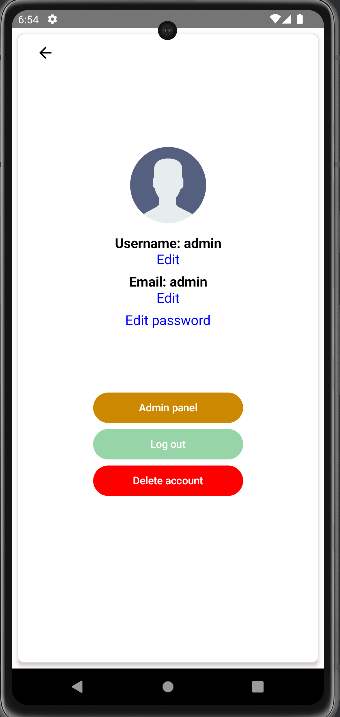
*Фиг. 7: Екран на географската карта*

С бутоните:

* Soil - Представяме информация за почвата
* Earthquacke - Представяме информация за земетресението
* Remove - Премахваме маркерът
* Profile - Преглеждаме нашият профил

GeoMap.kt

@SuppressLint("RestrictedApi")  
@Composable  
fun GeoMap(navController: NavController) {  
 val cameraPositionState = rememberCameraPositionState **{}** val homeViewModel = hiltViewModel<HomeViewModel>()  
 val risk by homeViewModel.riskLocations.collectAsState()  
  
 Box(Modifier.*fillMaxSize*()) **{**  
MapsTheme **{**  
GoogleMap(  
 modifier = Modifier.*matchParentSize*(),  
 cameraPositionState = cameraPositionState,  
 properties = MapProperties(mapType = MapType.*TERRAIN*),  
 uiSettings = MapUiSettings(compassEnabled = false),  
 onMapLongClick = **{** latLng **->**  
*// Check if user is trying to click to an existing marker*  
val nearLocation = homeViewModel.getNearestLocation(latLng)  
 *userMarkerState*.clickedMarker.value =  
 if (nearLocation != null) {  
 nearLocation  
 } else {  
 homeViewModel.addRiskByLocation(latLng)  
 latLng  
 }  
 cameraPositionState.move(CameraUpdateFactory.newLatLng(latLng))  
 *userMarkerState*.changeTitle("Choose a type")  
 **}**,  
 ) **{**  
risk.*forEach* **{** (position, datatype) **->**  
val pointEntity = PointEntity(position)  
 when (datatype.riskState) {  
 RiskChoices.*Soil* -> {  
 val soil: Soil = datatype.soil  
 if (soil.sqkm != null && soil.domsoi != null) {  
 PolygonDrawing(pointEntity, soil.sqkm, pointEntity.getColorToSoilType(soil.domsoi))  
 }  
 }  
 RiskChoices.*Earthquake* -> {  
 val earthquake = datatype.earthquake  
 if (earthquake.dn != null) PolygonDrawing(pointEntity, earthquake.dn \* 10.0)  
 }  
 else -> {}  
 }  
  
 Marker(  
 state = rememberMarkerState(position = position),  
 onClick = **{**  
*userMarkerState*.clickedMarker.value = position  
 cameraPositionState.move(CameraUpdateFactory.newLatLng(position))  
 true  
 **}**,  
 title = "Soil/Earthquake",  
 anchor = *Offset*(0.10f, 0.10f),  
 )  
 **}**  
 **}**  
 **}**  
 **}**  
TopBottomBar(homeViewModel, navController)  
}  
  
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
private fun TopBottomBar(homeViewModel: HomeViewModel, navController: NavController) {  
 val bottomSheetState = rememberBottomSheetScaffoldState()  
  
 BottomSheetScaffold(  
 scaffoldState = bottomSheetState,  
 topBar = **{**  
TopAppBar(  
 colors =  
 TopAppBarDefaults.topAppBarColors(  
 containerColor = Color.White,  
 titleContentColor = Color.Black,  
 ),  
 modifier = Modifier.*height*(56.*dp*),  
 title = **{**  
Row(  
 horizontalArrangement = Arrangement.SpaceBetween,  
 modifier = Modifier.*fillMaxWidth*().*padding*(11.*dp*),  
 verticalAlignment = Alignment.CenterVertically,  
 ) **{**  
Text(*userMarkerState*.topBarTitleText.value, color = *uiColorStyle*)  
  
 Button(  
 colors = ButtonDefaults.buttonColors(containerColor = *uiColorStyle*),  
 onClick = **{**  
homeViewModel.logUserViewNavigation("Profile")  
 navController.navigate("profile")  
 **}**,  
 ) **{**  
Icon(Icons.Filled.*AccountCircle*, contentDescription = null)  
 **}**  
 **}**  
 **}**,  
 )  
 **}**,  
 sheetContent = **{** BottomBar(homeViewModel = homeViewModel, bottomSheetState = bottomSheetState) **}**,  
 ) **{**  
LoadingIndicator(  
 stateFlow = homeViewModel.stateFlow.collectAsState().value,  
 navController = null,  
 )  
 **}**  
}  
  
@SuppressLint("StateFlowValueCalledInComposition")  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun BottomBar(homeViewModel: HomeViewModel, bottomSheetState: BottomSheetScaffoldState) {  
 if (*userMarkerState*.hasUserClickedMarker()) {  
 BottomBarContent(homeViewModel)  
 Spacer(modifier = Modifier.*height*(56.*dp*))  
 }  
 Row(  
 horizontalArrangement = Arrangement.SpaceBetween,  
 modifier = Modifier.*fillMaxWidth*().*padding*(16.*dp*),  
 ) **{**  
Spacer(modifier = Modifier.*height*(30.*dp*))  
 if (*userMarkerState*.hasUserClickedMarker()) {  
 LaunchedEffect(key1 = Unit) **{** bottomSheetState.bottomSheetState.expand() **}**  
NavigationBar(containerColor = Color.Transparent) **{**  
NavigationBarItem(  
 icon = **{**  
Image(  
 painter = painterResource(R.drawable.*soil*),  
 contentDescription = null,  
 modifier = Modifier.*padding*(8.*dp*).*size*(35.*dp*),  
 )  
 **}**,  
 label = **{** Text("Soil") **}**,  
 selected = *userMarkerState*.userChoiceForDataType.value == RiskChoices.*Soil*,  
 onClick = **{**  
*userMarkerState*.userChoiceForDataType.value =  
 if (*userMarkerState*.userChoiceForDataType.value == RiskChoices.*Soil*) RiskChoices.*None*  
else RiskChoices.*Soil*  
**}**,  
 )  
 NavigationBarItem(  
 icon = **{**  
Image(  
 painter = painterResource(R.drawable.*earthquake*),  
 contentDescription = null,  
 modifier = Modifier.*padding*(8.*dp*).*size*(35.*dp*),  
 )  
 **}**,  
 label = **{** Text("Earthquake") **}**,  
 selected = *userMarkerState*.userChoiceForDataType.value == RiskChoices.*Earthquake*,  
 onClick = **{**  
*userMarkerState*.userChoiceForDataType.value =  
 if (*userMarkerState*.userChoiceForDataType.value == RiskChoices.*Earthquake*)  
 RiskChoices.*None*  
else RiskChoices.*Earthquake*  
**}**,  
 )  
 NavigationBarItem(  
 icon = **{**  
Icon(  
 Icons.Filled.*Clear*,  
 contentDescription = null,  
 modifier = Modifier.*padding*(8.*dp*).*size*(35.*dp*),  
 )  
 **}**,  
 label = **{** Text("Remove") **}**,  
 selected = false,  
 onClick = **{**  
homeViewModel.removeRiskByLocation(*userMarkerState*.clickedMarker.value)  
 *userMarkerState*.resetMarkerState()  
 **}**,  
 )  
 if (homeViewModel.riskLocations.value.size > 1) {  
 NavigationBarItem(  
 icon = **{**  
Icon(  
 Icons.Filled.*Delete*,  
 contentDescription = null,  
 modifier = Modifier.*padding*(8.*dp*).*size*(35.*dp*),  
 )  
 **}**,  
 label = **{** Text("Clear all") **}**,  
 selected = false,  
 onClick = **{**  
homeViewModel.purgeAllRisks()  
 *userMarkerState*.resetMarkerState()  
 **}**,  
 )  
 }  
 **}**  
} else {  
 Text(text = "Nothing to see here...")  
 *userMarkerState*.changeTitle("Geo")  
 }  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
 **}**  
}  
  
@SuppressLint("StateFlowValueCalledInComposition")  
@Composable  
fun BottomBarContent(homeViewModel: HomeViewModel) {  
 val riskLocations: RiskHierarchy?  
 var soil: Soil? = null  
 var earthquake: Earthquake? = null  
  
 when (*userMarkerState*.userChoiceForDataType.value) {  
 *// If the user hasn't chosen a data type yet*  
RiskChoices.*None* -> {  
 Box(modifier = Modifier.*fillMaxWidth*()) **{** *userMarkerState*.changeTitle("Choose a type") **}**  
}  
 *// If user has chosen a data type*  
RiskChoices.*Soil* -> {  
 riskLocations = homeViewModel.getRiskByLocation(*userMarkerState*.clickedMarker.value)  
  
 *// First check if the user clicked on a marker with already fetched risk, in order to avoid*  
 *// making a redundant api call*  
if (  
 riskLocations != null &&  
 riskLocations.riskState != RiskChoices.*None* &&  
 riskLocations.soil != Soil()  
 ) {  
 homeViewModel.updateRiskStateByLocation(  
 *userMarkerState*.clickedMarker.value,  
 RiskChoices.*Soil*,  
 )  
 soil = riskLocations.soil  
 } else  
 *// If no existing risk data is found, make an api call*  
{  
 LaunchedEffect(key1 = Unit) **{**  
homeViewModel.retrieveSoil(  
 RiskDTO(  
 *userMarkerState*.clickedMarker.value.latitude,  
 *userMarkerState*.clickedMarker.value.longitude,  
 )  
 )  
 **}**  
}  
 *// In here put all of the available data types to visualize the respective table*  
soil?.*let* **{**  
SoilTable(soil = **it**)  
 *userMarkerState*.changeTitle("Soil")  
 **}**  
}  
 RiskChoices.*Earthquake* -> {  
 riskLocations = homeViewModel.getRiskByLocation(*userMarkerState*.clickedMarker.value)  
 *// First check if the user clicked on a marker with already fetched risk, in order to avoid*  
 *// making a redundant api call*  
if (  
 riskLocations != null &&  
 riskLocations.riskState != RiskChoices.*None* &&  
 riskLocations.earthquake != Earthquake()  
 ) {  
 homeViewModel.updateRiskStateByLocation(  
 *userMarkerState*.clickedMarker.value,  
 RiskChoices.*Earthquake*,  
 )  
 earthquake = riskLocations.earthquake  
 } else  
 *// If no existing risk data is found, make an api call*  
{  
 LaunchedEffect(key1 = Unit) **{**  
homeViewModel.retrieveEarthquake(  
 RiskDTO(  
 *userMarkerState*.clickedMarker.value.latitude,  
 *userMarkerState*.clickedMarker.value.longitude,  
 )  
 )  
 **}**  
}  
 *// In here put all of the available data types to visualize the respective table*  
earthquake?.*let* **{**  
EarthquakeTable(earthquake = **it**)  
 *userMarkerState*.changeTitle("Earthquake")  
 **}**  
}  
 }  
}

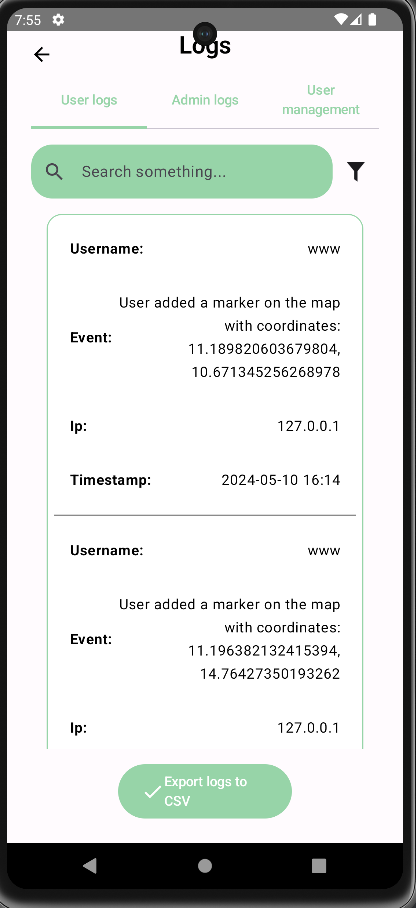
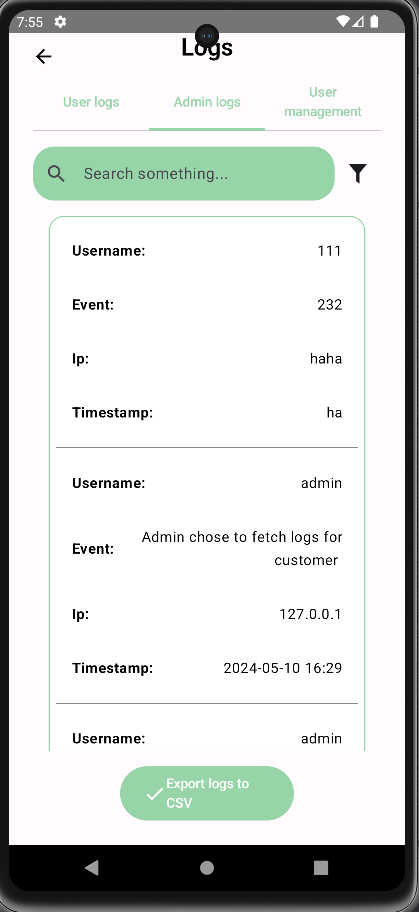
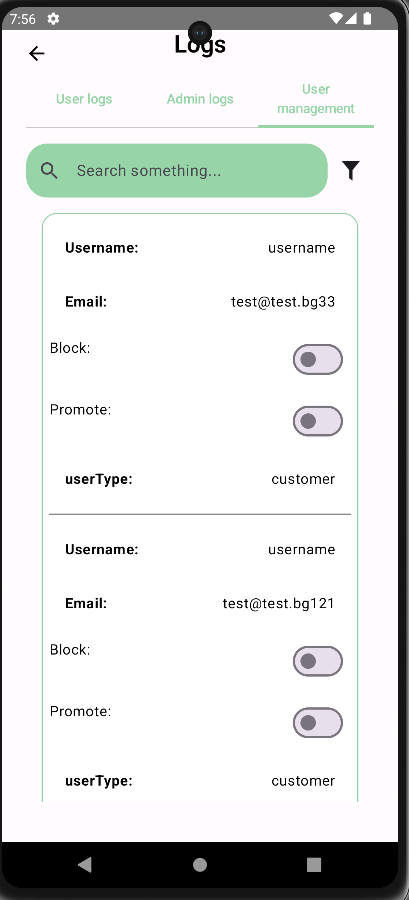


*Фиг. 8: Екран на профила*

С бутоните:

* Edit - Редактираме информацията за име/имейл/парола
* Admin panel - Преглеждаме събитията (само за супер администратори)
* Log out - Излизаме от профилът
* Delete account - Изтриваме профилът

ProfileView.kt  
@Composable  
fun ProfileView(navController: NavController, userSessionStorage: UserSessionStorage) {  
 val profileViewModel = hiltViewModel<ProfileViewModel>()  
 val userProps: UserStorage =  
 userSessionStorage.readUserProps().collectAsState(initial = UserStorage(0, "", "", "")).value  
 val user by remember **{** *mutableStateOf*(UserEntity(0, "", "", "", false)) **}**  
  
user.id = userProps.id  
 user.username = userProps.username  
 user.email = userProps.email  
 user.usertype.type = userProps.userType  
  
 val showAlertDialog = remember **{** *mutableStateOf*(false) **}**  
val showDeleteAccountDialog = remember **{** *mutableStateOf*(false) **}**  
val showRenameUsernameDialog = remember **{** *mutableStateOf*(false) **}**  
val showRenameEmailDialog = remember **{** *mutableStateOf*(false) **}**  
val showChangePasswordDialog = remember **{** *mutableStateOf*(false) **}**  
  
Box(modifier = Modifier.*fillMaxSize*(), contentAlignment = Alignment.TopCenter) **{**  
Card(  
 modifier =  
 Modifier.*padding*(0.*dp*)  
 .*fillMaxSize*()  
 .*padding*(8.*dp*)  
 .*wrapContentHeight*()  
 .*align*(Alignment.TopCenter),  
 elevation = CardDefaults.elevatedCardElevation(defaultElevation = 4.*dp*),  
 colors =  
 CardDefaults.cardColors(containerColor = Color.White, contentColor = Color.LightGray),  
 shape = *RoundedCornerShape*(8.*dp*),  
 ) **{**  
Button(  
 onClick = **{**  
profileViewModel.logUserViewNavigation("GeoMap")  
 navController.popBackStack()  
 **}**,  
 colors =  
 ButtonDefaults.buttonColors(  
 contentColor = Color.Black,  
 containerColor = Color.Transparent,  
 ),  
 ) **{**  
Icon(imageVector = Icons.Default.*ArrowBack*, contentDescription = "Back")  
 **}**  
  
Column(  
 modifier = Modifier.*fillMaxSize*(),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 ) **{**  
Spacer(modifier = Modifier.*height*(100.*dp*))  
  
 Image(  
 painter = painterResource(id = R.drawable.*profile*),  
 contentDescription = user.username,  
 modifier = Modifier.*size*(100.*dp*, 100.*dp*),  
 )  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 Text(  
 text = "Username: " + user.username,  
 style = TextStyle(fontSize = 18.*sp*, fontWeight = FontWeight.Bold, color = Color.Black),  
 modifier = Modifier.*align*(Alignment.CenterHorizontally),  
 )  
  
 Text(  
 text = "Edit",  
 style = TextStyle(fontSize = 18.*sp*, color = Color.Blue),  
 modifier =  
 Modifier.*align*(Alignment.CenterHorizontally).*clickable* **{**  
showRenameUsernameDialog.value = true  
 **}**,  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 Text(  
 text = "Email: " + user.email,  
 style = TextStyle(fontSize = 18.*sp*, fontWeight = FontWeight.Bold, color = Color.Black),  
 modifier = Modifier.*align*(Alignment.CenterHorizontally),  
 )  
  
 Text(  
 text = "Edit",  
 style = TextStyle(fontSize = 18.*sp*, color = Color.Blue),  
 modifier =  
 Modifier.*align*(Alignment.CenterHorizontally).*clickable* **{**  
showRenameEmailDialog.value = true  
 **}**,  
 )  
  
 Spacer(modifier = Modifier.*height*(8.*dp*))  
  
 Text(  
 text = "Edit password",  
 style = TextStyle(fontSize = 18.*sp*, color = Color.Blue),  
 modifier =  
 Modifier.*align*(Alignment.CenterHorizontally).*clickable* **{**  
showChangePasswordDialog.value = true  
 **}**,  
 )  
  
 Spacer(modifier = Modifier.*height*(80.*dp*))  
 if (userProps.userType != *FORBIDDEN\_USER\_TYPE\_ADMIN\_PANEL*)  
 Button(  
 onClick = **{**  
profileViewModel.logUserViewNavigation("Admin")  
 navController.navigate("admin")  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(0.5f).*align*(Alignment.CenterHorizontally),  
 colors = ButtonDefaults.buttonColors(containerColor = *Color*(204, 136, 0)),  
 ) **{**  
Text(text = "Admin panel")  
 **}**  
  
Button(  
 onClick = **{**  
profileViewModel.logUserViewNavigation("Logout -> Login")  
 navController.navigate("login")  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(0.5f).*align*(Alignment.CenterHorizontally),  
 ) **{**  
Text("Log out")  
 **}**  
  
Button(  
 onClick = **{**  
profileViewModel.logUserViewNavigation("Delete account")  
 showAlertDialog.value = true  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(0.5f).*align*(Alignment.CenterHorizontally),  
 colors = ButtonDefaults.buttonColors(containerColor = Color.Red),  
 ) **{**  
Text("Delete account")  
 **}**  
if (showAlertDialog.value) {  
 AlertDialogManager(  
 title = "Delete Account",  
 message = "Are you sure you want to delete your account?",  
 onConfirm = **{**  
*CoroutineScope*(Dispatchers.Main).*launch* **{**  
showAlertDialog.value = false  
 showDeleteAccountDialog.value = true  
 **}**  
 **}**,  
 onDismiss = **{** showAlertDialog.value = false **}**,  
 )  
 }  
  
 // Delete account dialog  
 if (showDeleteAccountDialog.value) {  
 DialogConfirmDeleteAccount(  
 title = "Delete Account",  
 message = "Please enter your password to confirm the deletion of your account",  
 onConfirm = **{** password **->** profileViewModel.deleteAccount() **}**,  
 onDismiss = **{** showDeleteAccountDialog.value = false **}**,  
 )  
 }  
  
 // Rename username dialog  
 if (showRenameUsernameDialog.value) {  
 DialogRenameUsername(  
 title = "Rename username",  
 message = "Please enter your username",  
 userUsername = user.username,  
 onConfirm = **{** username **->**  
*CoroutineScope*(Dispatchers.Main).*launch* **{**  
profileViewModel.renameUsername()  
 showRenameUsernameDialog.value = false  
 **}**  
 **}**,  
 onDismiss = **{** showRenameUsernameDialog.value = false **}**,  
 )  
 }  
  
 // Rename email dialog  
 if (showRenameEmailDialog.value) {  
 DialogRenameEmail(  
 title = "Rename email",  
 message = "Please enter your email",  
 userEmail = user.email,  
 onConfirm = **{** email **->**  
*CoroutineScope*(Dispatchers.Main).*launch* **{**  
profileViewModel.renameEmail(email)  
 showRenameEmailDialog.value = false  
 **}**  
 **}**,  
 onDismiss = **{** showRenameEmailDialog.value = false **}**,  
 )  
 }  
  
 // Change password dialog  
 if (showChangePasswordDialog.value) {  
 DialogChangePassword(  
 title = "Change password",  
 onConfirm = **{** oldPassword, newPassword **->**  
*CoroutineScope*(Dispatchers.Main).*launch* **{**  
profileViewModel.changePassword(oldPassword, newPassword)  
 showChangePasswordDialog.value = false  
 **}**  
 **}**,  
 onDismiss = **{** showChangePasswordDialog.value = false **}**,  
 )  
 }  
 **}**  
 **}**  
 **}**  
}

*Фиг. 9: Екран на админ панела*

С бутонът:

* Export logs to CSV- Експортираме събитията
* User logs - Показва заявките от потребителите
* Admin logs - Показва заявките от админите
* User managment - управлява потребителите
* Block - блокира потребител
* Promote - Повишава потребител

UserLogsTable.kt

@Composable  
fun UserLogsTable(userLogs: List<LoggerDTO>) {  
 Box(  
 modifier =  
 Modifier.*padding*(16.*dp*)  
 .*fillMaxWidth*()  
 .*verticalScroll*(rememberScrollState())  
 .*background*(color = Color.White, shape = *RoundedCornerShape*(16.*dp*))  
 .*border*(*BorderStroke*(1.*dp*, *Color*(151, 212, 168)), shape = *RoundedCornerShape*(16.*dp*))  
 .*padding*(8.*dp*)  
 ) **{**  
Column(modifier = Modifier.*fillMaxWidth*()) **{**  
userLogs.*forEachIndexed* **{** index, userLog **->**  
TableKeyValueTextRow("Username:", userLog.username!!)  
 TableKeyValueTextRow("Event:", userLog.event!!)  
 TableKeyValueTextRow("Ip:", userLog.ip!!)  
 TableKeyValueTextRow("Timestamp:", userLog.timestamp!!)  
  
 if (index < userLogs.size - 1)  
 Divider(  
 color = Color.Gray,  
 thickness = 1.*dp*,  
 modifier = Modifier.*padding*(vertical = 8.*dp*),  
 )  
 **}**  
 **}**  
 **}**  
}

## **4.4 Реализация на модул за регистриране на събития в системата**

Модулът за регистрация на събития се намира в viewModel/states слоя, като на всеки viewmodel се инжектира в техният конструктор. Обекта се извиква sendLog всеки път при извършване на api операции от view modelът.

Клиентска страна:

LoggerManager.kt

class LoggerManager @Inject constructor(private val adminRepository: AdminRepository) {  
 suspend fun sendLog(username: String, userType: String, event: String) {  
 withContext(Dispatchers.IO) **{**  
val timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm"))  
 val clientIP = InetAddress.getLocalHost().*hostAddress*  
Timber.d("%s is sending log %s to the server! Moving on...", userType, username)  
 adminRepository.sendLog(userType, LoggerDTO(username, event, clientIP, timestamp))  
 **}**  
}  
}

Извикване от viewmodel:

@HiltViewModel  
class HomeViewModel  
@Inject  
constructor(  
 private val loggerManager: LoggerManager,  
 private val userSessionStorage: UserSessionStorage,  
 private val riskRepository: RiskRepository,  
) : UIState() {

fun retrieveSoil(point: RiskDTO) {  
 Timber.d(  
 "User clicked the map with coordinates %.6f, %.6f and chose a soil type! Moving on...",  
 point.latitude,  
 point.longitude,  
 )  
 *viewModelScope*.*launch* **{**  
mutableStateFlow.value = UIFeedback(state = UIFeedback.States.*Waiting*)  
 val message =  
 when (val result: ApiPayload<Soil> = riskRepository.getSoil(point)) {  
 is ApiPayload.Success -> {  
 val riskLocations: Soil = result.data!!  
 Timber.d("Soil type received! Payload received from server %s", riskLocations)  
 val risk = RiskHierarchy()  
 risk.soil = riskLocations  
 updateRiskByLocation(LatLng(point.latitude!!, point.longitude!!), risk)  
  
 loggerManager.sendLog(  
 userSessionStorage.readUsername(),  
 userSessionStorage.readUserType(),  
 "User choose soil datatype to be retrieved for the coordinates: " +  
 point.latitude +  
 ", " +  
 point.longitude,  
 )  
 returnStatus = UIFeedback.States.*Success*  
result.message!!  
 }  
 is ApiPayload.Failure -> {  
 loggerManager.sendLog(  
 userSessionStorage.readUsername(),  
 userSessionStorage.readUserType(),  
 "User encountered an issue when trying to retrieve soil datatype for the coordinates: " +  
 point.latitude +  
 ", " +  
 point.longitude,  
 )  
 returnStatus = UIFeedback.States.*Failed*  
result.message  
 }  
 }  
 mutableStateFlow.value = UIFeedback(state = returnStatus, message = message)  
 **}**  
}

Сървърна страна:

В сървърната страна лога се обслужва чрез POST api ендпоинт и след това се съхранява в AWS S3 bucket, като се организира според датата на съхраняване:

Обслужване на заявката за съхраняване на лог:

@RestController

@RequestMapping("/admin")

public class AdminController {

@PostMapping("save/log/{userType}")

@ResponseStatus(HttpStatus.OK)

@Operation(summary = "Save log")

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "User blocked!"),

@ApiResponse(responseCode = "400", description = "Bad request"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

public ResponseEntity<RestApiResponse<Void>> saveLog(@RequestBody LoggerDTO loggerDTO,

@PathVariable("userType") String userType) {

logger.info("Received a log to be saved later for the admins!: {}", loggerDTO);

return new ResponseEntity<>(adminService.saveLog(loggerDTO, userType), HttpStatus.OK);

}

@Override

@Transactional

@SuppressWarnings({ "squid:S3457", "squid:S2629" })

public RestApiResponse<Void> saveLog(LoggerDTO loggerDTO, String userType) {

userValidateService.validateUseTypeExistWithoutSuperAdmin(userType);

s3Service.store(loggerDTO, userType);

return new RestApiResponse<>(null, "Log saved!", 201);

}

Съхраняване на лог в S3:

@Component

public class S3Service {

@Value("${spring.cloud.aws.s3.bucket}")

private String bucket;

@Autowired

private AWSConfig awsConfig;

private final Logger logger = LogManager.getLogger(S3Service.class.getName());

private LocalDate currentDate = LocalDate.now();

private String logPath = currentDate.getYear() + "/" + currentDate.getMonth() + "/" + currentDate.getDayOfMonth()

+ "/logs.json";

public void store(LoggerDTO loggerDTO, String userType) {

List<LoggerDTO> existingLogs = getLog(userType + "/" + logPath);

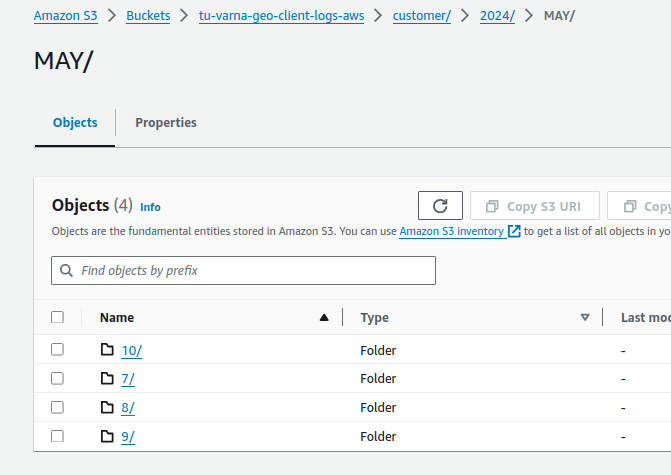
existingLogs.add(loggerDTO);

awsConfig.s3Template().store(bucket,

userType + "/" + logPath,

existingLogs);

}



*Фиг. 10: Дъво на директории на логове в AWS S3*

# **5. Тестови резултати**

За имплементиране на тестове за имплементирани основни от страна на Java Spring сървърът. Те са извършвани с помощта на Mockito.

## **5.1 JUnit tests**

@SpringBootTest(classes = Main.class)

@AutoConfigureMockMvc

class UserServiceImplTest {

private UserRepository userRepository;

private UserTypeRepository userTypeRepository;

private UserValidateService userValidateService;

private UserMapper userMapper;

private BCryptPasswordEncoder encoder;

private UserServiceImpl userService;

AuthenticationManager authenticationManager;

JWTTokenProvider jwtTokenProvider;

@BeforeEach

void setup() {

userRepository = Mockito.mock(UserRepository.class);

userTypeRepository = Mockito.mock(UserTypeRepository.class);

userValidateService = Mockito.mock(UserValidateService.class);

userMapper = Mockito.mock(UserMapper.class);

encoder = Mockito.mock(BCryptPasswordEncoder.class);

authenticationManager = Mockito.mock(AuthenticationManager.class);

jwtTokenProvider = Mockito.mock(JWTTokenProvider.class);

userService = new UserServiceImpl(userRepository, userTypeRepository, userValidateService, userMapper, encoder,

authenticationManager, jwtTokenProvider);

}

@Test

void registerUserTest() {

RegisterUserDTO userDto = new RegisterUserDTO("dummy", "dummy", "123",

false, "customer");

User user = new User();

when(userMapper.toEntity(any(RegisterUserDTO.class), eq(encoder))).thenReturn(user);

when(userRepository.save(any(User.class))).thenReturn(user);

when(userTypeRepository.findByType(userDto.getUsertype())).thenReturn(new UserType());

RestApiResponse<Void> registerResponse = userService.registerUser(userDto);

assertNotNull(registerResponse);

assertEquals(201, registerResponse.getStatus());

verify(userRepository).save(any(User.class));

}

}

Резултат:

%TESTC 1 v2

%TSTTREE2,com.tuvarna.geo.UserServiceImplTest,true,1,false,1,UserServiceImplTest,,[engine:junit-jupiter]/[class:com.tuvarna.geo.UserServiceImplTest]

%TSTTREE3,registerUserTest(com.tuvarna.geo.UserServiceImplTest),false,1,false,2,registerUserTest(),,[engine:junit-jupiter]/[class:com.tuvarna.geo.UserServiceImplTest]/[method:registerUserTest()]

%TESTS 3,registerUserTest(com.tuvarna.geo.UserServiceImplTest)

%TESTE 3,registerUserTest(com.tuvarna.geo.UserServiceImplTest)

%RUNTIME9644

## **5.2 Функционални тестове**

## **5.3 Интеграционни тестове**

@ExtendWith(SpringExtension.class)

@ActiveProfiles

@AutoConfigureMockMvc

@SpringBootTest(classes = { Main.class }, webEnvironment = SpringBootTest.WebEnvironment.MOCK)

class AuthControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testLogin() throws Exception {

LoginUserDTO userDto = new LoginUserDTO("dummy", "dummy");

RestApiResponse<LoggedInUserDTO> expectedResult = new RestApiResponse<LoggedInUserDTO>(

"User logged in successfully", 201);

when(userService.login(userDto)).thenReturn(expectedResult);

this.mockMvc.perform(post("/auth/login")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString((userDto)))).andExpect(status().isOk())

.andExpect(content().string(containsString("User logged in successfully")));

}

@Test

void testRegister() throws Exception {

RegisterUserDTO userDto = new RegisterUserDTO("username", "[email@email.com](mailto:email@email.com)", "123", false, null);

RestApiResponse<Void> expectedResult = new RestApiResponse<Void>(

"User registered successfully", 201);

when(userService.registerUser(userDto)).thenReturn(expectedResult);

this.mockMvc.perform(post("/auth/register")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString((userDto)))).andExpect(status().isCreated())

.andExpect(content().string(containsString("User registered successfully")));

}

}

Резултат:

%TESTC 2 v2

%TSTTREE2,com.tuvarna.geo.AuthControllerTest,true,2,false,1,AuthControllerTest,,[engine:junit-jupiter]/[class:com.tuvarna.geo.AuthControllerTest]

%TSTTREE3,testLogin(com.tuvarna.geo.AuthControllerTest),false,1,false,2,testLogin(),,[engine:junit-jupiter]/[class:com.tuvarna.geo.AuthControllerTest]/[method:testLogin()]

%TSTTREE4,testRegister(com.tuvarna.geo.AuthControllerTest),false,1,false,2,testRegister(),,[engine:junit-jupiter]/[class:com.tuvarna.geo.AuthControllerTest]/[method:testRegister()]

%TESTS 3,testLogin(com.tuvarna.geo.AuthControllerTest)

%TESTE 3,testLogin(com.tuvarna.geo.AuthControllerTest)

%TESTS 4,testRegister(com.tuvarna.geo.AuthControllerTest)

%TESTE 4,testRegister(com.tuvarna.geo.AuthControllerTest)

%RUNTIME8505