1. **Санкт-Петербургский политехнический университет Петра Великого**
2. **Институт прикладной математики и механики**

**Высшая школа кибербезопасности и защиты информации**

**КУРСОВАЯ РАБОТА**

1. **Разработка WEB-приложения и Android приложения**
2. по дисциплине «Безопасность интернет-приложений»

|  |  |  |
| --- | --- | --- |
| 1. Выполнили 2. студенты гр. 3651003/60801 | *<подпись>*  *<подпись>*  *<подпись>*  *<подпись>*  *<подпись>* | А.В. Егорова  В.В. Еремук  К.Э. Косулин  В.А. Ромашов  М.В. Юнга |
|  | 1. *<подпись>* | А.А. Яковлева |
| Руководитель,   1. Ассистент | 1. *<подпись>* | * + - 1. А.Д. Дахнович |

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_202\_\_г.

1. Санкт-Петербург
2. 2020

СОДЕРЖАНИЕ

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Концепция приложения……………………………………………….. | |  | 3 |
| 1.1 Используемые технологии и инструменты……………………… | |  | 4 |
| 1.2 Взаимодействие между компонентами системы………………… | |  | 6 |
| 2. Распределение обязанностей в команде……………………………. | |  | 8 |
| 2.1 Роли в команде……………………………………………………. | |  | 8 |
| 2.2 Организация собраний и распределение задач…………………. | |  | 8 |
| 2.3 Сроки выполнения………………………………………………… | |  | 8 |
| 3. Разработка серверной части…………………………………………. | |  | 8 |
| 3.1 Разработка авторизации……………………………………………. | |  | 13 |
| 3.2 Spring Security………………………………………………………. | |  | 18 |
| 4. Разработка Web-приложения………………………………………….. | |  | 21 |
| 5. Разработка Android приложения………………………………………. | |  | 31 |
| 5.1 Реализация функциональности приложения………………………... | |  | 31 |
| 5.2 Работа приложения…………………………………………………… | |  | 32 |
| 5.3 Безопасность хранения данных……………………………………… | |  | 41 |
| 6. Разработка дизайна…………………………………………………….. | |  | 42 |
| 6.1 Material Design………………………………………………………… | |  | 42 |
| 6.2 Использованные инструменты…………………………………………. | |  | 44 |
| Заключение………………………………………………………………... | |  | 46 |
| Список используемых источников………………………………………. | |  | 47 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | |  |  |
|  | |  |  |

**1. Концепция приложения**

В настоящее время, всё большую актуальность приобретает проблема загрязнения мирового океана пластиком. По данным ООН, каждый год в океан попадает около 13 миллионов тонн пластиковых отходов. Широкое использование пластиковых изделий в промышленности и быту обусловило возникновение проблемы накопления связанных с ними отходов. Начиная с середины ХХ века отмечался ежегодный рост спроса на пластиковые изделия, составляя в настоящее время около 300 млн тонн, при этом 2/3 изделий из пластика – упаковочные материалы и предметы одноразового использования. Одна из основных причин этого – низкая стоимость полимеров, их малый вес, биоинертность, прочность и износостойкость. Важным шагом в решении данной проблемы является развитие культуры личного потребления, с целью планомерного сокращения производства одноразовых пластиковых изделий и развития процессов и инфраструктуры для сбора, переработки, транспортировки и вторичного использования отходов.

В настоящее время, на территории Санкт-Петербурга существуют предприятия общественного питания и магазины, предоставляющие скидки клиентам при использовании своей тары. Также, существуют пункты раздельного сбора отходов и организации, заинтересованные в использовании вторичного сырья. Кроме того, проводятся мероприятия, направленные на просвещение общественности в вопросах ответственного потребления и проблем экологии.

Основной задачей, которую ставит перед собой предлагаемый концепт приложения, является агрегация вышеуказанных организаций, пунктов и мероприятий, с целью предоставления информации о них заинтересованным лицам. Кроме того, концепт предлагает единую систему бонусов для поощрения и мотивации пользователей. Для проекта было выбрано рабочее название «Pocket Greta».

* 1. **Используемые технологии и инструменты**

Для разработки схем взаимодействия между отдельными компонентами и тестирования использовался инструмент Postman. Для реализации серверной части использовался язык программирования Java, а также технологии Spring Boot, Spring MVC, Spring Security, Spring Data JPA, Lombok, JsonWebToken. В качестве сервера баз данных был выбран PostgreSQL. Для реализации веб-приложения использовался язык программирования JavaScript, дополнительно использовались фреймворк для выполнения HTTP-запросов Axios, а также Google Maps API для отрисовки карты. Также, использовался CSS-фреймворк Materialize, для конфигурирования отображения некоторых элементов. Для организации доступа к серверной части, запущенной на локальной машине, использовался инструмент ngrok. Клиентская часть для Android была разработана с использованием языка программирования Kotlin. Для реализации веб-сервера был использован язык Python и язык Flask. Организация работы команды производилась с использованием инструментов Trello и Microsoft Teams, для хранения репозитория использовался GitHub. Для создания элементов графического интерфейса использовался графический редактор Adobe Illustrator. Использовались интегрированные среды разработки IntelliJ Idea, PyCharm, WebStorm.

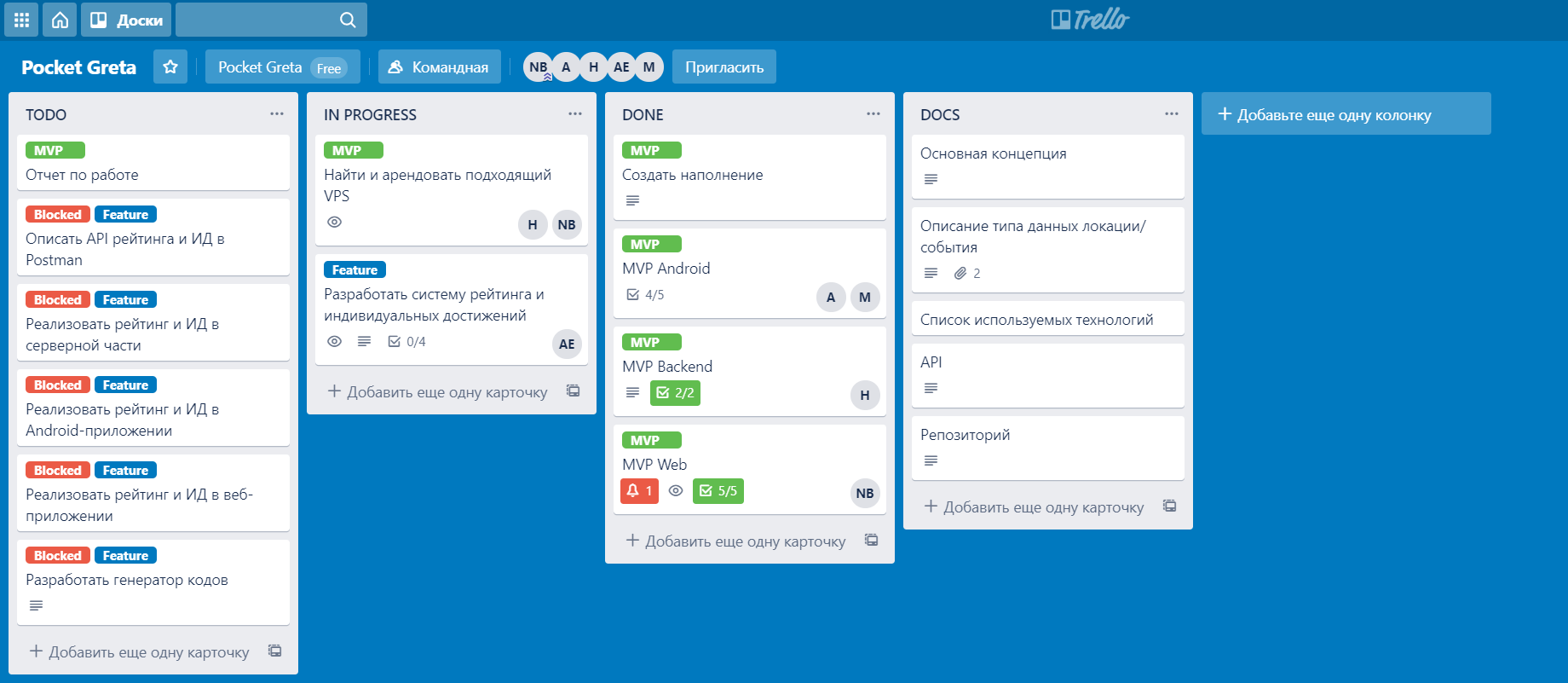


Рисунок 1 – Командная доска в Trello

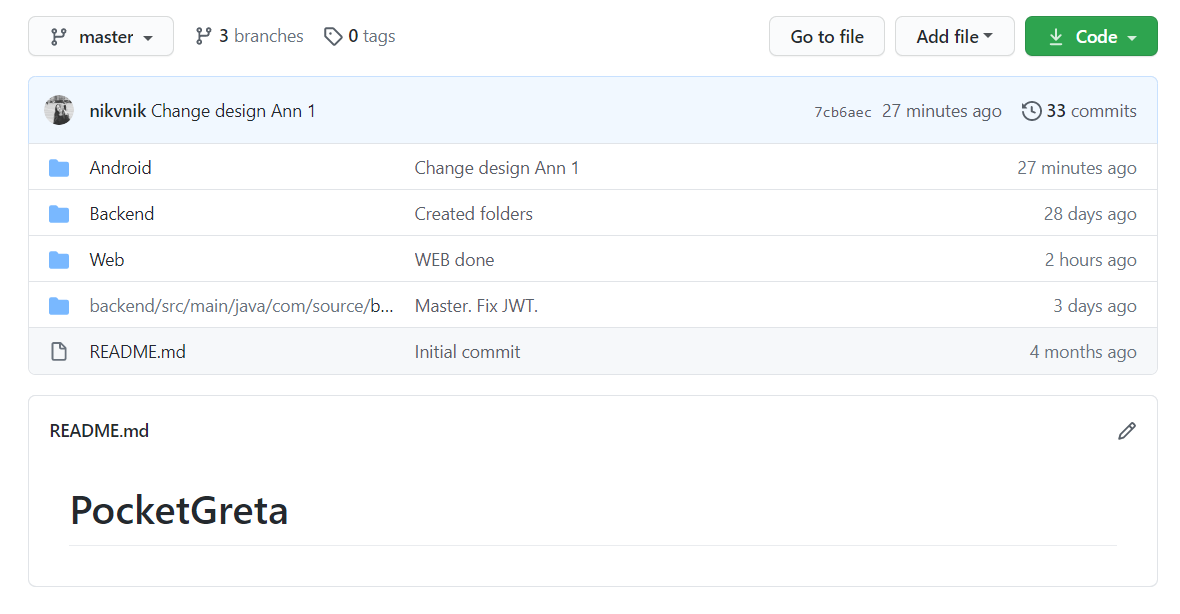


Рисунок 2 – Командный репозиторий на GitHub.

* 1. **Взаимодействие между компонентами системы**

Взаимодействие между отдельными компонентами системы производится при помощи HTTP-запросов. Информация передается в формате JSON со следующими полями:

|  |
| --- |
| { "latitude":"", - **широта** "longitude":"", - **долгота** "type":"", - **тип** "subtype":"", - **подтип** "descriptions":"", - **описание**, "details":"", - **детали** "id" : "" - **уникальный идентификатор** } |

Описание типов и подтипов локаций приведено на рисунке 3.



Рисунок 3 – Описание типов и подтипов локаций

Описание запросов к серверной части приведено в таблице 1.

Таблица 1 – Описание запросов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип | URI | Параметры | Заголовки | Назначение |
| GET | rest/map/all |  |  | Получить список всех локаций |
| POST | rest/auth/login | email, password |  | Логин пользователя |
| GET | rest/acc/<user> |  | authorization, cookie | Получить количество бонусов |
| POST | rest/auth/refresh/token | refreshToken,  username | cookie | Обновить authenticationToken |
| POST | rest/auth/signup | username, email, password |  | Зарегистрировать пользователя |

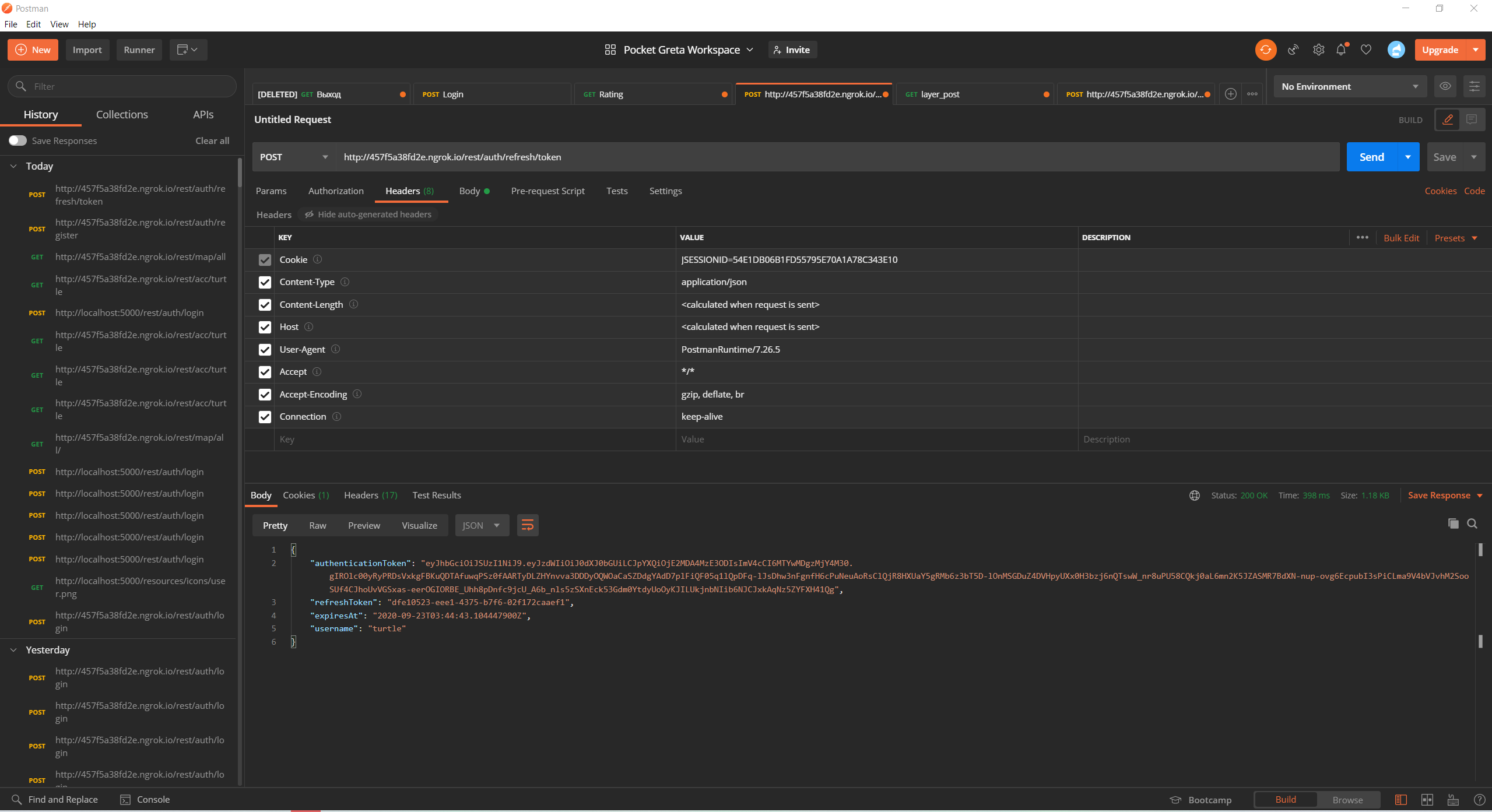


Рисунок 4 – Процесс отладки запросов в Postman

**2. Распределение обязанностей в команде**

* 1. Роли в команде

В работе над курсовой работой принимали участие шесть человек, где роли в команде были распределены следующим образом:

* Егорова Анна – дизайн Android-приложения, разработка логотипа проекта;
* Еремук Владимир – управление командой, распределение задач, разработка Web-приложения;
* Косулин Кирилл – разработка концепции взаимодействия, тестирование, создание отчета о работе.
* Ромашов Виктор – разработка серверной части
* Юнга Мария – разработка дизайна Web приложения, создание презентации проекта;
* Яковлева Анна – разработка Android-приложения.
  1. Организация собраний и распределение задач

В целях организации эффективной работы команда собиралась каждые две недели в течение двух месяцев для обсуждения текущего статуса задач и возникающих трудностей.

* 1. Сроки выполнения работы

Работа выполнялась с апреля по июнь 2020 года.

1. Разработка серверной части

Для разработки backend был выбран язык программирования Java.

Сервер поддерживает следующий функционал:

* Логин;
* Логаут;
* Выдать точки определенного типа;
* Выдать все точки;
* Выдать информацию об аккаунте.

Аспекты безопасности:

* Spring Security;
* Авторизация с помощью JWT токена;
* Шифрование паролей в базе данных;
* JWT токен не хранится в базе данных.

Использованные технологии:

* Spring Boot;
* Spring MVC;
* Spring Security;
* Spring Data JPA;
* Lombok;
* JsonWebToken;
* PostgreSQL.

Spring boot был выбран по причине того, что он позволяет быстро начать процесс разработки необходимого функционала, не отвлекаясь на начальную настройку Spring приложения.

Перед тем как описывать разработку сервера, стоит упомянуть, что в Spring используются bean'ы, которые представляют собой обьект класса, чаще всего в единственном числе, так же они используются для соблюдения Dependency Injection.

Первым делом был подключен PostgresSQL с необходимой базой данных. Далее она была подключена с помощью необходимых настроек в application.properties.

spring.datasource.initialization-mode=always

spring.datasource.platform=postgres

spring.datasource.url=jdbc:postgresql://localhost:5432/pocketGreta

spring.datasource.username=postgres

spring.datasource.password=postgres

spring.jpa.generate-ddl = true

spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true

В Spring существует аннотация @Controller. Класс, созданный с этой аннтоцией и аннотацией @RequestMapping, отвечает за маппинг запросов, который приходит от Dispatcher Servlet'а, отвечающего за исходного запроса.

После этого был реализован MapEndpoint, который отвечает за маппинг запросов от Dispatcher Servlet.

@RestController

@RequestMapping("rest/map")

@AllArgsConstructor

public class MapEndpoint

Пример:

@GetMapping("/{trashType}")

@ResponseBody

private ResponseEntity trashcanRequest(@PathVariable String trashType){

try{

return ResponseEntity.ok(mapService.getCoordinates(trashType));

}

catch (Exception e){

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(e.getMessage());

}

}

@PathVariable – значит, что переменная возьмет значение из адреса запроса, в данном случае – это trashtype.

@ResponseBody – Что можно вернуть только строку, не создавая ResponseEntity.

Далее из trashcanRequest идет вызов getCoordinates в mapService. Аннотация @Service означает, что класс является сервисом и наследует за собой определенные аннотации.

public Set<EcoUnit> getCoordinates(String type) throws Exception{

if (type.equals(Type.TRASHBIN.getType())) {

return getTrashcans();

}

if (type.equals(Type.SHOP.getType())) {

return getShops();

}

if (type.equals(Type.EVENT.getType())){

return getEvents();

}

throw new Exception("Type is incorrect");

}

public Set<EcoUnit> getTrashcans(){

return ecoUnitRepository.findByType(Type.TRASHBIN);

}

public Set<EcoUnit> getShops(){

return ecoUnitRepository.findByType(Type.SHOP);

}

public Set<EcoUnit> getEvents(){

return ecoUnitRepository.findByType(Type.EVENT);

}

Этот сервис обращается к EcoUnitRepository, который отвечает за запросы к базе данных. Благодаря тому, что данный интерфейс наследуется от JpaRepository, не надо писать обычные запросы с помощью sql. Достаточно написать в обьявлении метода findBy и перечислить переменные, например FindByType.

@Repository

public interface EcoUnitRepository extends JpaRepository<EcoUnit, Long> {

public Set<EcoUnit> findByType(Type type);

public List<EcoUnit> findAll();

}

@Repository – аннотация помечающая репозитории. Запрос в Spring проходит путь от контроллера до сервиса, а после до репозитория, если потребуется.

За точки отвечает класс EcoUnit.

@Data

@Entity

public class EcoUnit {

@Id

@GeneratedValue

private Long id;

@NotNull

@Enumerated(EnumType.ORDINAL)

private Type type;

private String subtype = "";

@NotNull

private Double latitude;

@NotNull

private Double longitude;

private String Details = "";

private String Descriptions = "";

}

Данный класс имеет две аннотации @Data и @Entity. Data - обьявляет класс, как компонент, автоматически генерирует hashcode/equals. Entity - показывает, что данный класс является таблицой в базе данных, с ID в виде @Id, которое генерируется по определенным правилам. @NotNull - из названия, выводит исключение, если это поле будет null, а так же @Enumerated - используется, если в Entity есть Enum тип, и можно выбрать в каком виде его записывать (по строкам, либо по числам, либо по полю, который есть в Enum).

3.1 Разработка авторизации

Был реализован AuthorizationEndpoint ("rest/auth") со следующим функционалом -

/signup - (Post запрос) Регистрация нового пользователя, принимает @RequestBody типа RegisterRequest

@Data

@AllArgsConstructor

@NoArgsConstructor

public class RegisterRequest {

private String email;

private String username;

private String password;

}

/accountVerification/{token} - (Get запрос) Не использованный функционал, активация пользователя по {token} после того, как он перейдет по ссылке в письме после регистрации.

/login - (Post запрос) Авторизация пользователя в систему, в качестве @RequestBody принимает LoginRequest

@Data

@AllArgsConstructor

@NoArgsConstructor

public class LoginRequest {

private String email;

private String password;

}

В качестве ответа в body отсылается AuthenticationResponse.

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class AuthenticationResponse {

private String authenticationToken;

private String refreshToken;

private Instant expiresAt;

private String username;

}

Поля данного класса будут расмотрены далее в пункте Spring Security.

/refresh/token - (Post запрос) Обновление refresh токена, принимает в @RequestBody RefreshTokenRequest.

@Data

@AllArgsConstructor

@NoArgsConstructor

public class RefreshTokenRequest {

@NotBlank

private String refreshToken;

private String username;

}

Entity Account приведен ниже.

@Data

@Entity

@Getter

public class Account {

@Id

@GeneratedValue

private Long id;

@NotNull

private String username;

@NotNull

private String password;

private String email;

private boolean Enabled;

private Integer bonuses;

@NotNull

private Instant registrationDate;

private String permission;

}

Данные запросы вызывают методы AuthorizationService. Регистрация приведена ниже.

public void signup(RegisterRequest registerRequest) {

Account account = new Account();

account.setUsername(registerRequest.getUsername());

account.setEmail(registerRequest.getEmail());

account.setPassword(passwordEncoder.encode(registerRequest.getPassword()));

account.setRegistrationDate(Instant.now());

account.setEnabled(true);

account.setBonuses(0);

account.setPermission("USER");

userRepository.save(account);

String token = generateVerificationToken(account);

}

Создается новый аккаунт, шифруется пароль, далее он сохраняется в userRepository, который отвечает за таблицу пользователей.

Более сложный метод Login приведен ниже.

public AuthenticationResponse login(LoginRequest loginRequest) throws Exception {

Optional<String> userOpt = userRepository.getUsernameByEmailOnly(loginRequest.getEmail());

String username = userOpt

.orElseThrow(() -> new UsernameNotFoundException("No user " +

"Found with email : " + loginRequest.getEmail()));

Optional<Account> accountOpt = userRepository.findByUsername(username);;

Account account = accountOpt

.orElseThrow(() -> new UsernameNotFoundException("No Account " +

"Found with email : " + loginRequest.getEmail()));

Authentication authenticate = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username,

loginRequest.getPassword()));

SecurityContextHolder.getContext().setAuthentication(authenticate);

String token = jwtProvider.generateToken(authenticate);

return AuthenticationResponse.builder()

.authenticationToken(token)

.refreshToken(refreshTokenService.generateRefreshToken(account).getToken())

.expiresAt(Instant.now().plusMillis(jwtProvider.getJwtExpirationInMillis()))

.username(username)

.build();

}

Сначала происходит поиск username по email с помощью

@Query("SELECT username FROM Account a WHERE email=?1")

public Optional<String> getUsernameByEmailOnly(String email);

Далее, если username был найден, создается обьект типа Account и используется authenticationManage.authenticate, отдельный bean, который будет упомянут в Spring Security пункте. После авторизации пользователя формируется AuthenticationResponse, с помощью builder() (Аннотация @Builder) и формируется JWT токен и RefreshToken, если он не существует.

Метод обновления токена -

public ResponseEntity<AuthenticationResponse> refreshTokens(@Valid @RequestBody RefreshTokenRequest refreshTokenRequest) {

try {

Optional<RefreshToken> refreshTokenOpt = refreshTokenRepository.findByToken(refreshTokenRequest.getRefreshToken());

RefreshToken refreshToken = refreshTokenOpt.orElseThrow(() -> new UsernameNotFoundException("No RefreshToken " +

"Found with Token : " + refreshTokenRequest.getRefreshToken()));

if(refreshTokenRequest.getUsername().equals(refreshToken.getAccount().getUsername())) {

AuthenticationResponse authenticationResponse = authService.refreshToken(refreshTokenRequest);

return new ResponseEntity<>(authenticationResponse, OK);

}

else {

return new ResponseEntity<>(HttpStatus.FORBIDDEN);

}

}

catch (Exception e){

return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

Применяется для обновления JWT токена, если он просрочен или необходимо по каким-либо причинам его заменить.

Сначала проверяется наличие токена в refreshTokenRepository, далее проверяется кому принадлежит токен и на какой username создан запрос на обновление JWT токена.

Также был создан AccountEndpoint ("rest/acc") - отвечающий за обработку дйствий над аккаунтами, допустим получение информации. Ниже приведен код endpoint'а, который обрабатывает запрос на получение информации об аккаунте.

@ResponseBody

@RequestMapping("/{username}")

private ResponseEntity<AccountInfoDto> getAccountInfo(@PathVariable String username){

String usernameCheck = ((UserDetails)SecurityContextHolder.getContext().getAuthentication().getPrincipal()).getUsername();

if (usernameCheck.equals(username)){

return new ResponseEntity(accountService.getInfo(username), HttpStatus.OK);

}

else {

return new ResponseEntity("Now allowed", HttpStatus.FORBIDDEN);

}

}

Сервер получает Username из адреса запроса, далее, проверяет кто запросил информацию и выдает её через accountService.getInfo(). Ответ сервера -

@Data

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class AccountInfoDto {

private String email;

private String username;

private int bonuses;

}

3.2 Spring Security

В самом начале был создан класс SecurityConfiguration

@Configuration

@EnableWebSecurity

@AllArgsConstructor

@EnableGlobalMethodSecurity(prePostEnabled = true, proxyTargetClass = true)

public class SecurityConfiguration extends WebSecurityConfigurerAdapter implements WebMvcConfigurer

@Configuration - обозначает класс как конфигурацию Spring приложения. @EnableWebSecurity – включает Spring Security.

@EnableGlobalMethodSecurity – включает его глобально, с выбранными параметрами.

Далее был создан класс JwtProvider и сгенерирован Java KeyStore. Рассмотрим метод генерации токена:

public String generateToken(Authentication authentication) throws Exception{

User principal = (User) authentication.getPrincipal();

return Jwts.builder()

.setSubject(principal.getUsername())

.setIssuedAt(from(Instant.now()))

.signWith(getPrivateKey())

.setExpiration(from(Instant.now().plusMillis(jwtExpirationInMillis)))

.compact();

}

Токен создается с помощью builder, под пользователя который залогинился и подписывается закрытым ключем из jks файла и проставляется время жизни токена, равное 15 минутам, после этого пользователю нужно будет использовать refreshToken для генерации нового JWT токена.

Метод проверки токена с помощью открытого ключа из jks –

public boolean validateToken(String jwt) throws Exception{

parser().setSigningKey(getPublickey()).parseClaimsJws(jwt);

return true;

}

Далее был создан JwtAuthentiticationFilter, который является фильтром, обрабатывающим запросы до их поступления на конечные точки. Если адрес запроса совпадает с адресами, для которых не нужна авторизация, то проверка jwt токена не происходит, иначе, при отсутствии или наличии неправильного токена происходит остановка и проброс 403 кода в ответ с текстом "Token doesn't exist or doesn't match allowed token”.

Далее был переопределен метод configure –

@Override

protected void configure(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeRequests()

.antMatchers("/rest/auth/\*\*")

.permitAll()

.antMatchers("/rest/map/\*\*")

.permitAll()

.antMatchers("/rest/acc/\*")

.authenticated()

.anyRequest()

.authenticated();

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

}

Данный метод отвечает за настройку Spring Security, cors() – включает cors filter. Csrf().disable() – отключает защиту от csrf атак, т.к. используется jwt токен вместо этой настройки. Далее идут регулярные выражения, запросы по совпадающим адресам не будут заблокированы Spring Security и пойдут в обработку фильрами, в данном случае – это jwtAuthentiticationFilter, описанный раннее.

Также был создан bean AuthenticationManager -

@Bean(BeanIds.AUTHENTICATION\_MANAGER)

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

И добавлена вторая настройка –

@Autowired

public void configureGlobal(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {

authenticationManagerBuilder.userDetailsService(userDetailsService)

.passwordEncoder(encoder());

}

Данная настройка означает, что менеджер будет использовать userDetailsService.

Данный сервис был переопределен –

@Bean

public UserDetailsService userDetailsService(){

return new com.source.backend.service.UserDetailsServiceImpl();

}

В нем используется разработанный UserDetailsServiceImpl, в котором переопределен метод loadUserByUsername, –

@Override

@Transactional(readOnly = true)

public UserDetails loadUserByUsername(String username) {

Optional<Account> userOptional = userRepository.findByUsername(username);

Account account = userOptional

.orElseThrow(() -> new UsernameNotFoundException("No user " +

"Found with username : " + username));

return new org.springframework.security

.core.userdetails.User(account.getUsername(), account.getPassword(),

account.isEnabled(), true, true,

true, getAuthorities("USER"));

}

Данный метод использует AuthenticationManager при вызове authenticationManager.authenticate().

Также был разработан RefreshTokenService, отвечающий за генерации RefreshToken –

public RefreshToken generateRefreshToken(Account account) {

Optional<RefreshToken> refreshTokenOptional =refreshTokenRepository.findByAccount(account);

RefreshToken refreshToken = refreshTokenOptional.orElse(new RefreshToken());

if ( refreshToken.getAccount() != null){

return refreshToken;

}

refreshToken.setToken(UUID.randomUUID().toString());

refreshToken.setCreatedDate(Instant.now());

refreshToken.setAccount(account);

return refreshTokenRepository.save(refreshToken);

}

void validateRefreshToken(String token) throws Exception{

refreshTokenRepository.findByToken(token)

.orElseThrow(() -> new Exception("Invalid refresh Token"));

}

1. Разработка Web-приложения

Для разработки веб-приложения использовались технологии HTML, CSS, JavaScript, а также Google Maps API и фреймворки Axios, Materialize. Первоначально, было разработано приложение, реализующее отображение карты и добавление произвольной точки, а также, отображающее примерное расположение отдельных элементов. Для загрузки изображений посредством языка Python и фреймворка Flask был реализован простой веб-сервер. Результат разработки прототипа веб-приложения показан на рисунке 5.

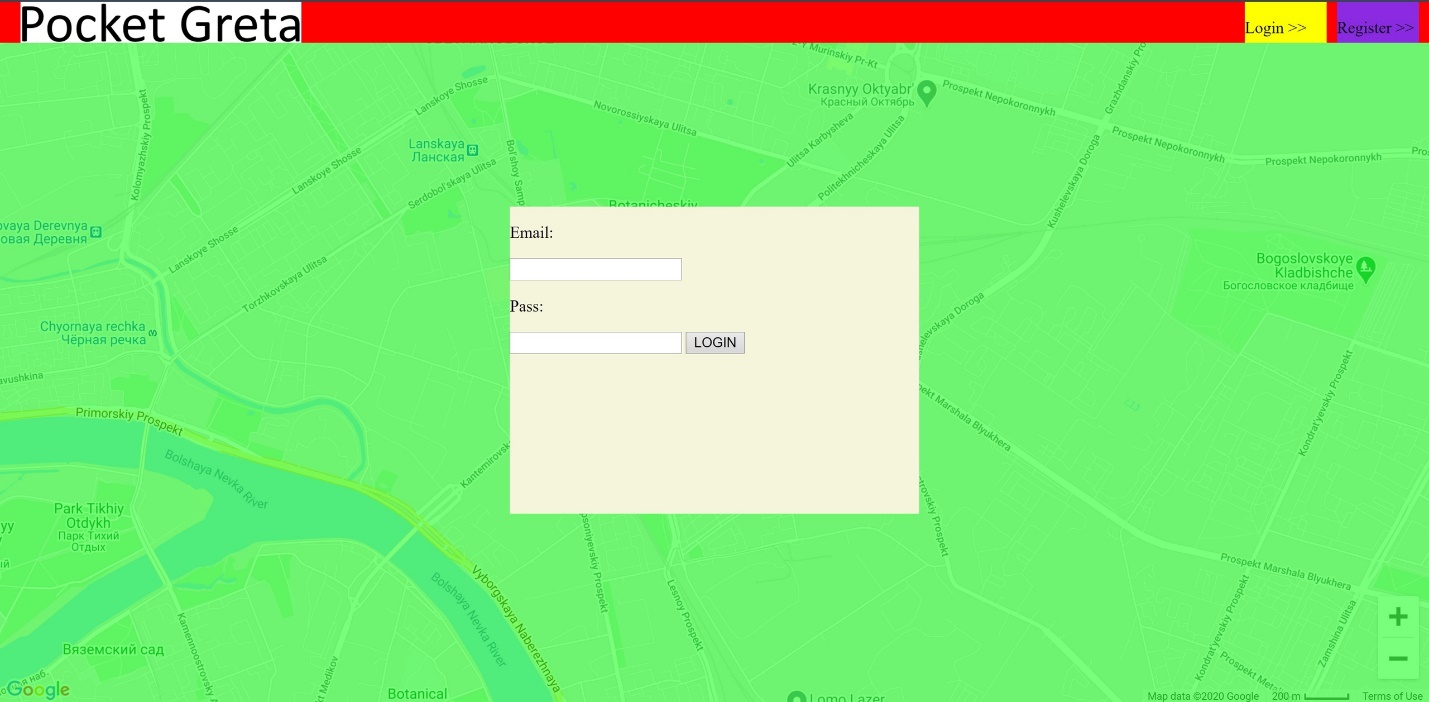


Рисунок 5 – Прототип веб-приложения

Далее, функционал добавления произвольной точки на карту был вынесен в отдельную функцию, принимающую на вход широту, долготу и тип локации.



Рисунок 6 – Добавление объекта на карту

Далее, был реализован функционал входа пользователя в систему, регистрации и получения баланса бонусов. Для этого, были реализованы функции doregister(), dologin() и checkLogin(), показанные на рисунках 7 - 9 соответственно.

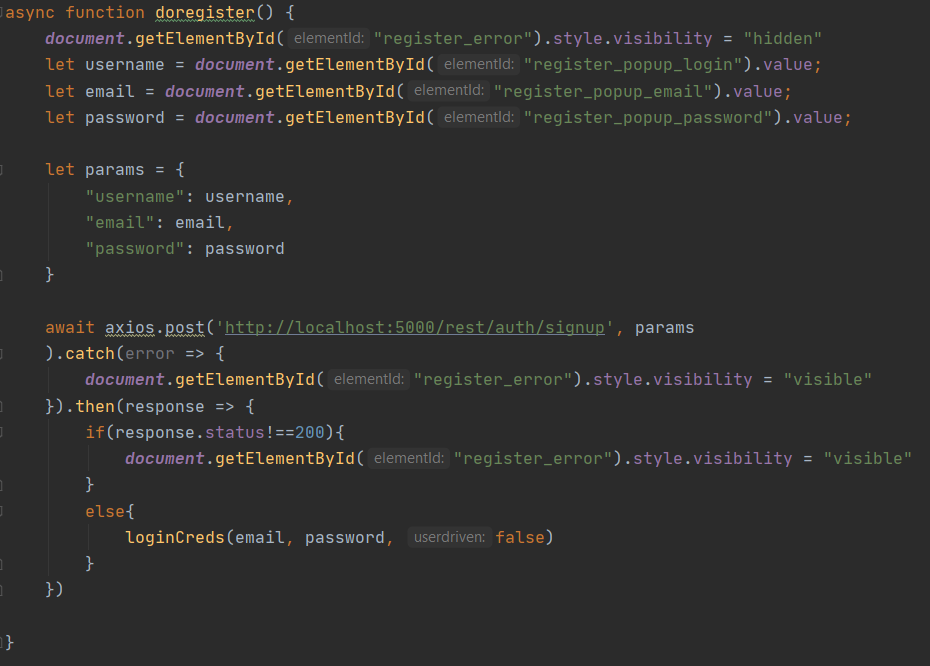


Рисунок 7 – Функция doregister()

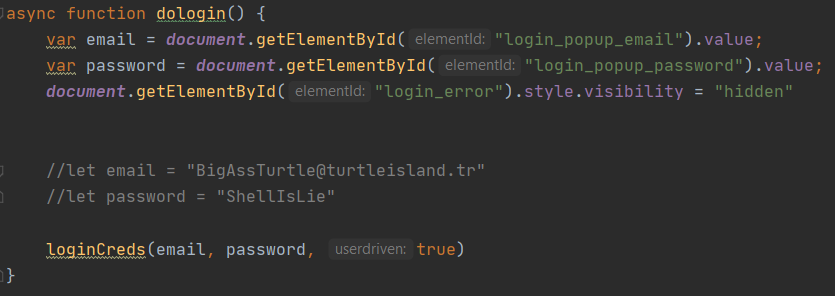


Рисунок 8 – Функция dologin()



Рисунок 9 – Функция checkLogin()

При входе пользователя в систему, authentificationToken, refreshToken, username и session cookie сохраняются в cookie-файлах браузера. Далее, вызывается функция checkLogin(), обновляющая authentificationToken (посредством вызова функции refreshToken()), получающая баланс бонусов и отображающая соответствующие данные на веб-странице. Функция checkLogin() также вызывается при каждой загрузке веб-страницы, обеспечивая обновление authentificationToken и отображая необходимые данные. При выходе из системы, вызывается функция logout(), очищающая файлы cookie и скрывающая данные пользователя, отображая вместо них кнопки входа и регистрации.



Рисунок 10 – функция logout()

Исходя из необходимости загружать ресурсы и отсутствия шифрования запросов при прямом обращении к серверной части, было решено реализовать веб-сервер посредством языка Python и фреймворка Flask. Для реализации шифрования, посредством openssl был создан корневой сертификат, а также сертификат доменного имени:

|  |
| --- |
| C:\OpenSSL-Win64\bin\openssl.exe req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048 -keyout RootCA.key -out RootCA.pem -subj "/C=US/CN=Example-Root-CA"  C:\OpenSSL-Win64\bin\openssl.exe x509 -outform pem -in RootCA.pem -out RootCA.crt  C:\OpenSSL-Win64\bin\openssl.exe req -new -nodes -newkey rsa:2048 -keyout localhost.key -out localhost.csr -subj "/C=US/ST=YourState/L=YourCity/O=Example-Certificates/CN=localhost.local"  C:\OpenSSL-Win64\bin\openssl.exe x509 -req -sha256 -days 1024 -in localhost.csr -CA RootCA.pem -CAkey RootCA.key -CAcreateserial -extfile domains.ext -out localhost.crt |

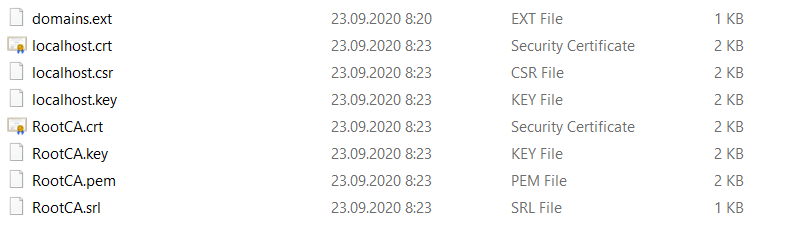


Рисунок 11 – Сертификаты

Корневой сертификат был установлен в систему, сертификаты доменного имени были утилизированы в веб-сервере.

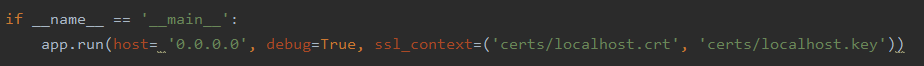


Рисунок 12 – Утилизация сертификатов в веб-сервере

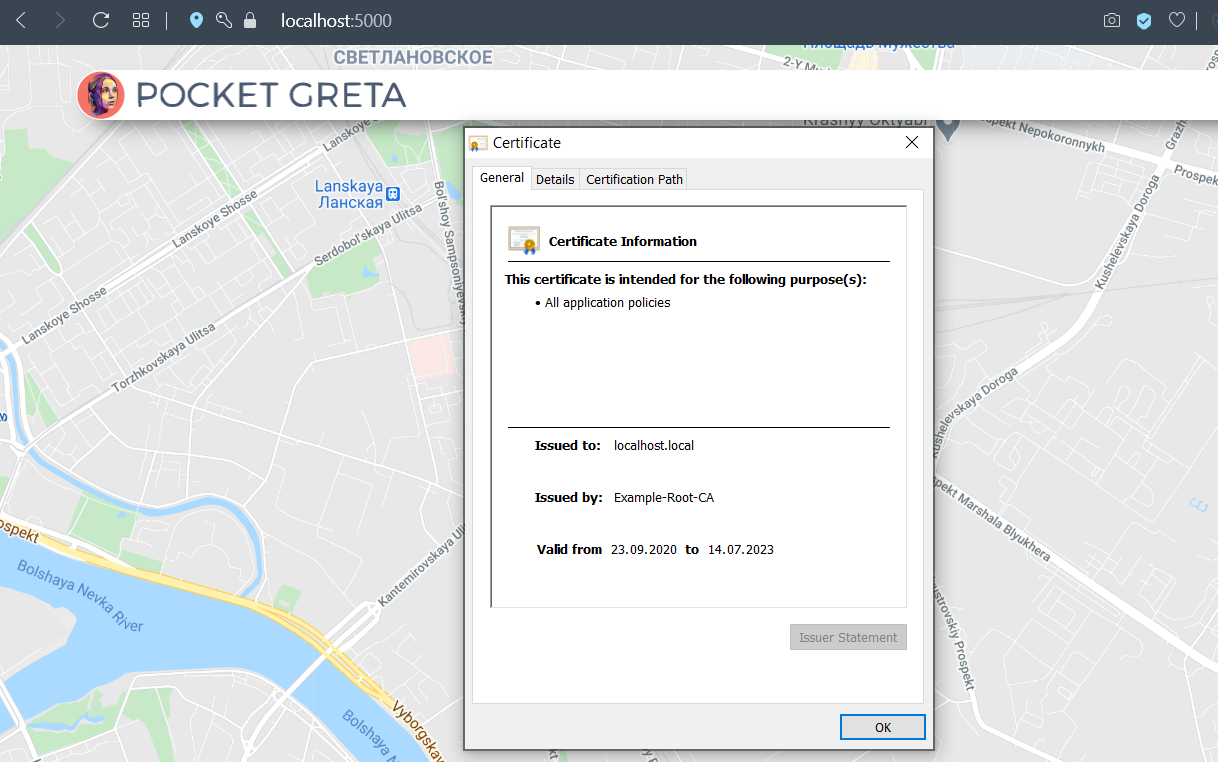


Рисунок 13 – результат подключения к веб-приложению после установки сертификатов.

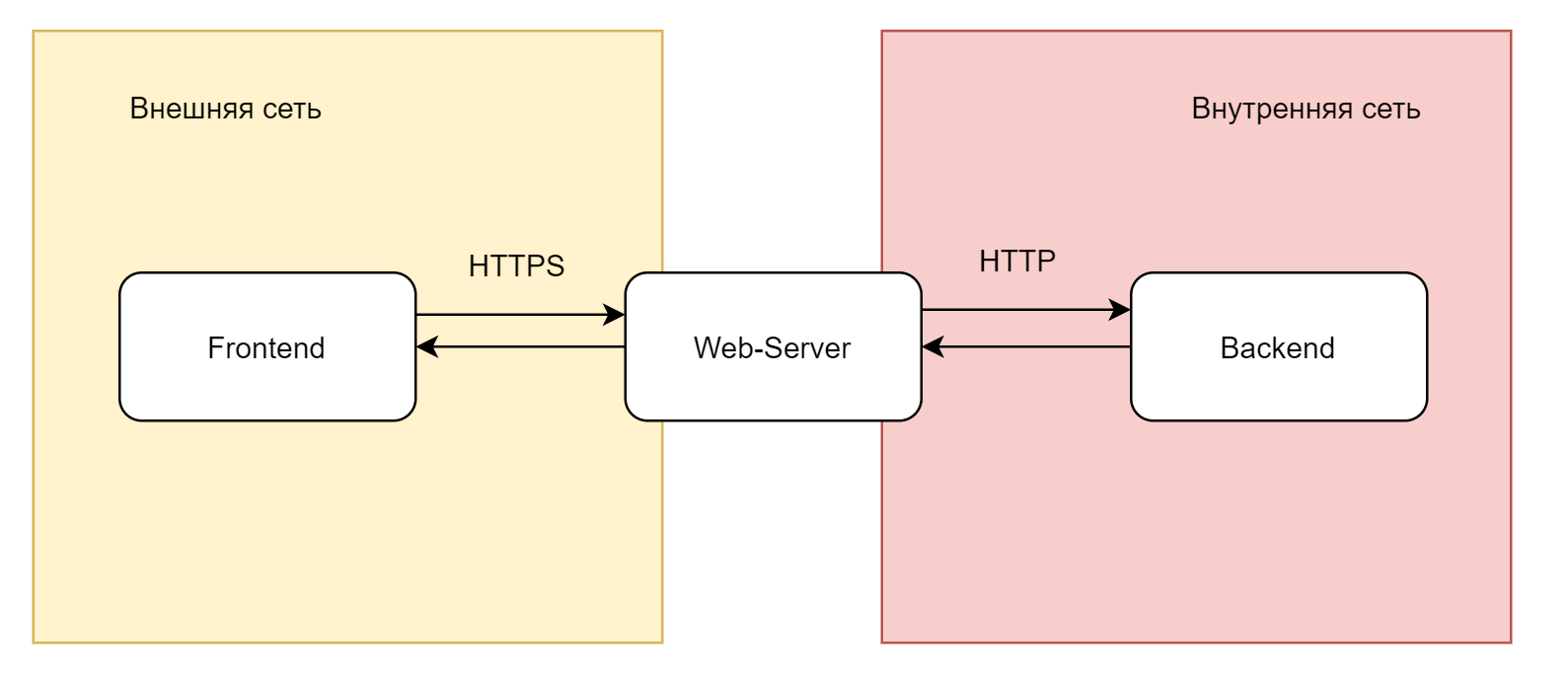


Рисунок 14 - Предлагаемая схема развертывания на реальном окружении с использованием веб-сервера

Далее, посредством CSS и фреймворка Materialize, был реализован дизайн веб-приложения. Работа готового приложения продемонстрирована на рисунках 15 – 21.

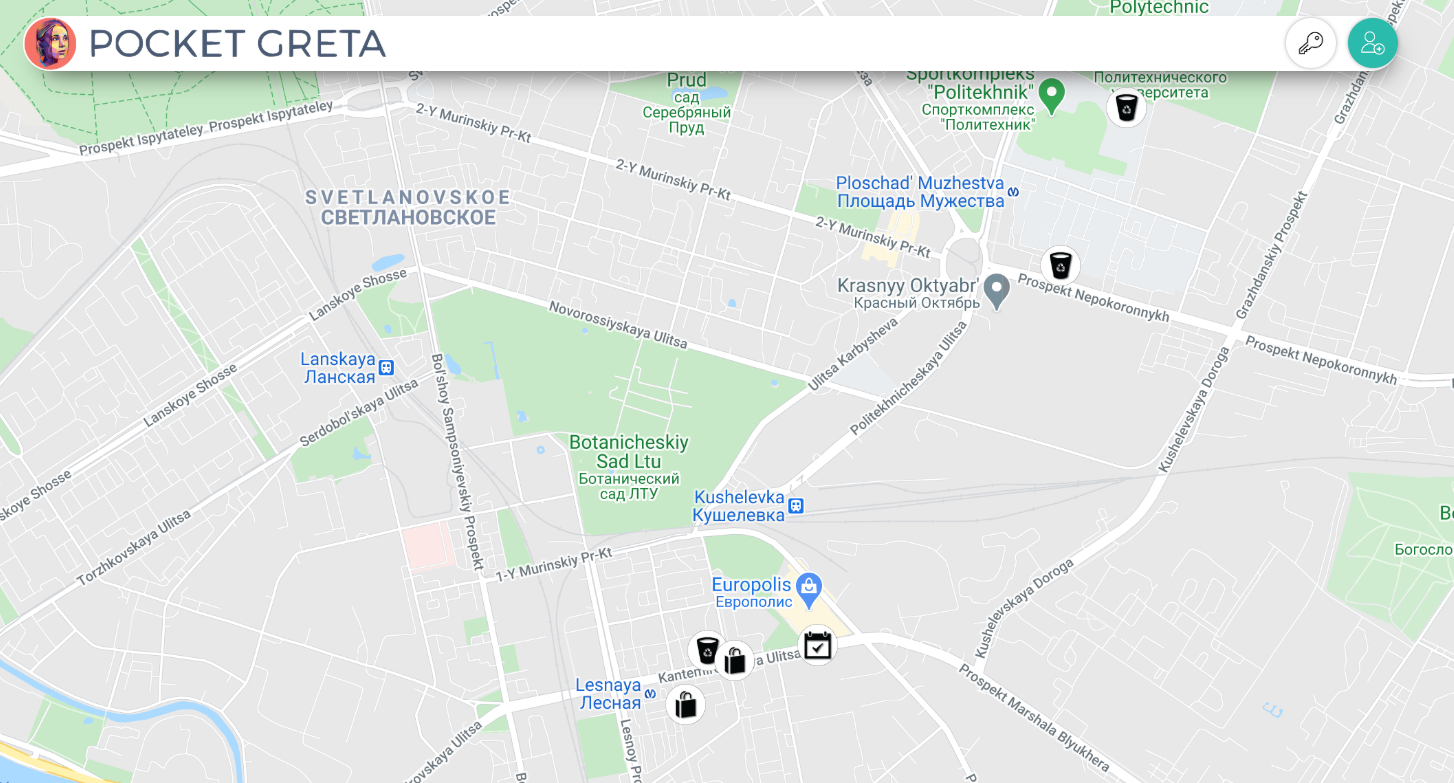


Рисунок 15 – Состояние приложение после запуска

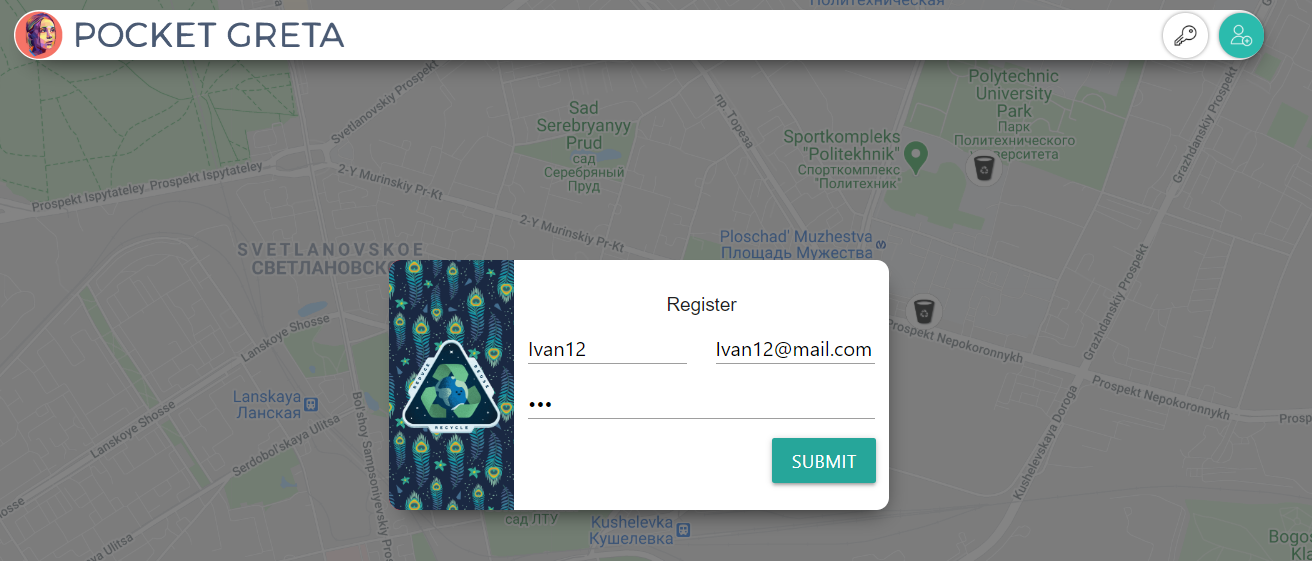


Рисунок 16 – Форма регистрации

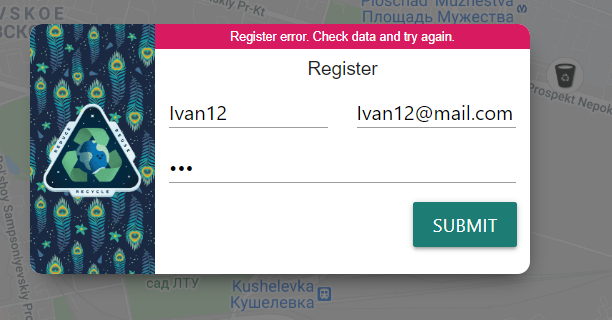


Рисунок 17 – Форма регистрации с сообщением об ошибке

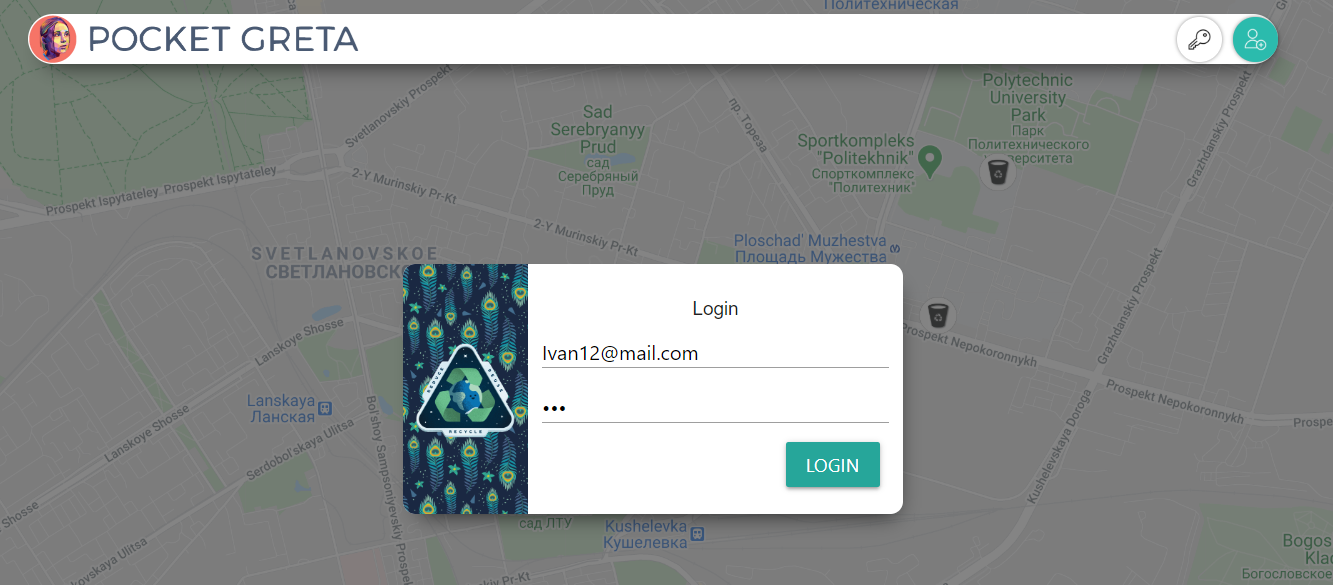


Рисунок 18 – Форма входа

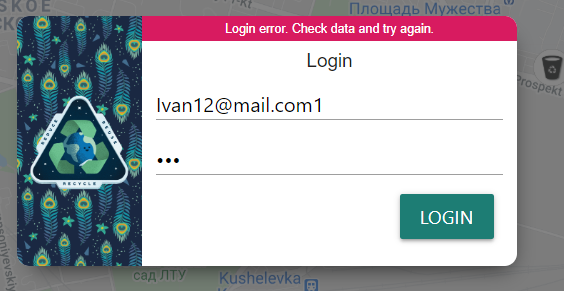


Рисунок 19 – Форма входа с сообщением об ошибке

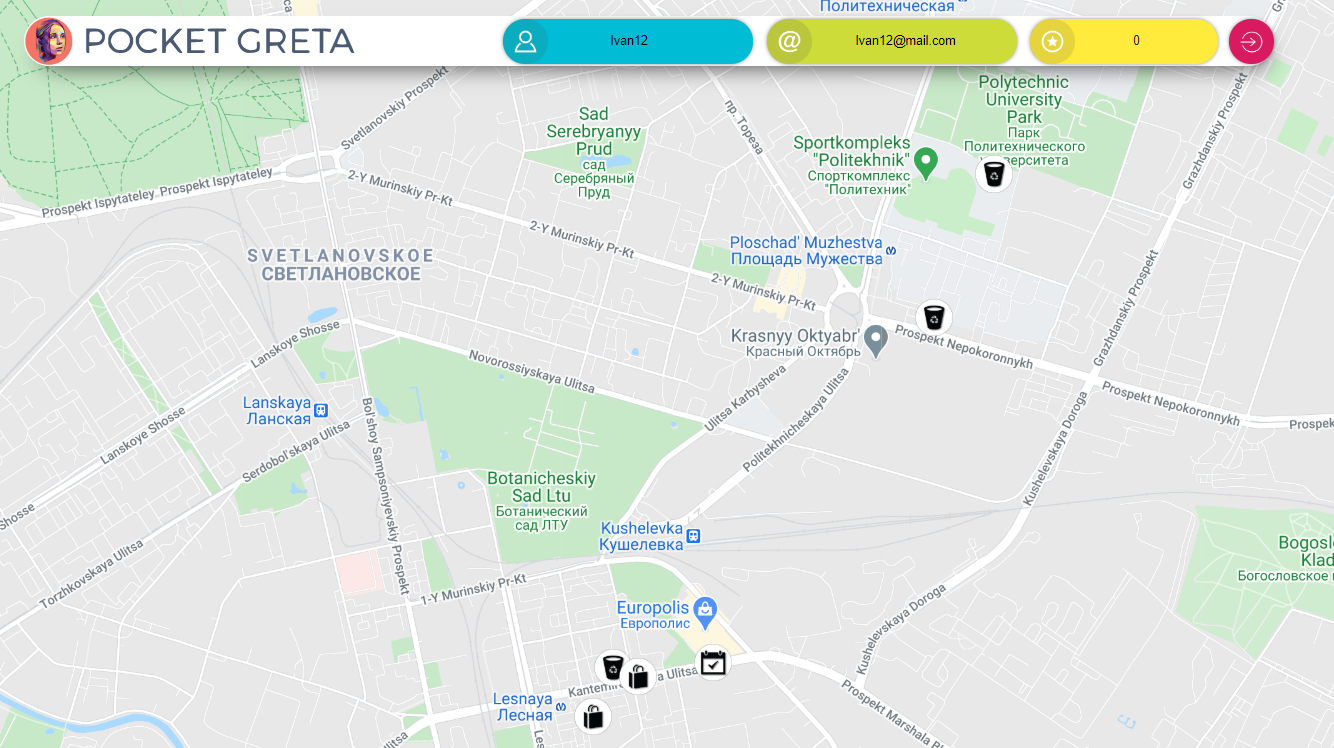


Рисунок 20 – Состояние после входа

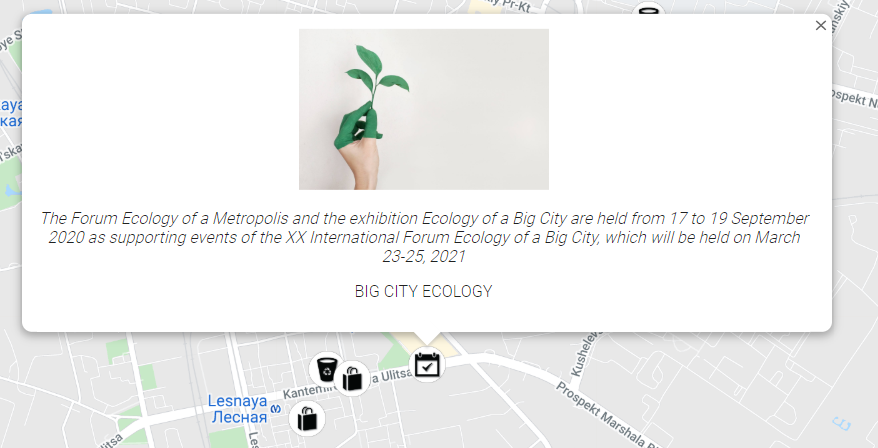


Рисунок 21 – Отображение карточки локации

Пример cookie-файлов после входа пользователя в систему:

|  |
| --- |
| [  {  "domain": "localhost",  "hostOnly": true,  "httpOnly": false,  "name": "cookie\_sessionid",  "path": "/",  "sameSite": "unspecified",  "secure": false,  "session": true,  "storeId": null,  "value": "A2DC430C904E22DFE5BD4E0C1A35B6C4"  },  {  "domain": "localhost",  "hostOnly": true,  "httpOnly": false,  "name": "username",  "path": "/",  "sameSite": "unspecified",  "secure": false,  "session": true,  "storeId": null,  "value": "Ivan12"  },  {  "domain": "localhost",  "hostOnly": true,  "httpOnly": false,  "name": "refreshToken",  "path": "/",  "sameSite": "unspecified",  "secure": false,  "session": true,  "storeId": null,  "value": "452ef069-bf4a-44a8-927e-d1b7dfd5e714"  },  {  "domain": "localhost",  "hostOnly": true,  "httpOnly": false,  "name": "authenticationToken",  "path": "/",  "sameSite": "unspecified",  "secure": false,  "session": true,  "storeId": null,  "value": "eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJJdmFuMTIiLCJpYXQiOjE2MDA4MzQ0NzYsImV4cCI6MTYwMDgzNTM3Nn0.O5H55g9YF2tAA1YxrhwggIW1BqI\_Q1XdkBkJGYP6FjorUwcqMqaH9UCpr3pZKOO\_adJCvNEwm6ebccdPDrM52CpJ8qXiDUnGMwEAoYZsMvdJDlX\_I2v8daLR0mwwRmxIegdye8MCKl0E-ziWhq-U92vL8GkKYDmmhoBFyqbt4b65dwIx7TT3\_NsL4eRgh5CXXcjVns2wnMSYq9MTgY55-UEbCME3P-2eHBzNuk6rBlhxBVA6qhRttTj19GmjQLlN1QuI0sBX0Hj6SJrE66eesvHSbYmp4bI9bTXrsPIhb71DlRCVxaijEF8QHuKIHxpo\_xFhwIZYNsPRFOpCKaRhTw"  }  ] |

1. Разработка Android приложения

5.1 Реализация функциональности приложения

Android приложение было разработано на языке программирования Java.

Данное приложение поддерживает следующий функционал:

* Выгрузка точек с сервера и установка их на карту;
* Регистрация пользователя;
* Вход пользователя в аккаунт;
* Получение информации о бонусах залогиненного пользователя в системе.

Аспекты безопасности:

* Хранение Refresh токенов в защищённой области ОС, доступной только данному приложению;
* Хранение username залогиненного пользователя в защищённой области ОС, доступной только данному приложению;
* Ограничение допустимого при регистрации пароля (длина не менее 8 символов);
* Регистрация по почте;
* Пароль и почта не хранится в памяти приложения.

Технологии, использованные в процессе создания приложения:

* AndroidStudio
* Gradle

**5.2 Работа приложения**

При первом входе в приложение отображается окно с картами Google maps. При запуске приложения отправляется GET запрос на сервер при помощи сторонней библиотеки com.androidnetworking.AndroidNetworking. Запрос и обработка полученных данных в формате JSONArray, приведено на рисунке 22.

AndroidNetworking.*get*(url)  
 .setPriority(Priority.*LOW*)  
 .build()  
 .getAsJSONArray(new JSONArrayRequestListener() {  
 @Override  
 public void onResponse(JSONArray response) {  
 for (int i = 0; i < response.length(); i++) {  
 JSONObject employee = null;  
 try {  
 employee = response.getJSONObject(i);  
  
 Double latitude = employee.getDouble("latitude");  
 Double longitude = employee.getDouble("longitude");  
 String type = employee.getString("type");  
 String subtype = employee.getString("subtype");  
 String description = employee.getString("descriptions");  
 String details = employee.getString("details");  
 int uid = employee.getInt("id");  
 Log.*d*("JSON",type+subtype+description+details);  
  
 onMapReadyPoint(latitude, longitude, type, subtype, description, details, uid);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 @Override  
 public void onError(ANError error) {  
 // handle error  
 }  
 });

Рисунок 22 – Отправка запроса на получение точек

При получении JSONArray от сервера приложение проводит парсинг и записывает долготу, ширину, тип объекта, подтип объекта, описание, детали и его id в переменные. Полученные данные передаются в функцию, реализующую добавление точки на карту.

В зависимости от типа устанавливает изображение для точки. На картине 23 приведен экран приложение после добавления точек с сервера. На рисунке 24 приведен пример информации о точке.

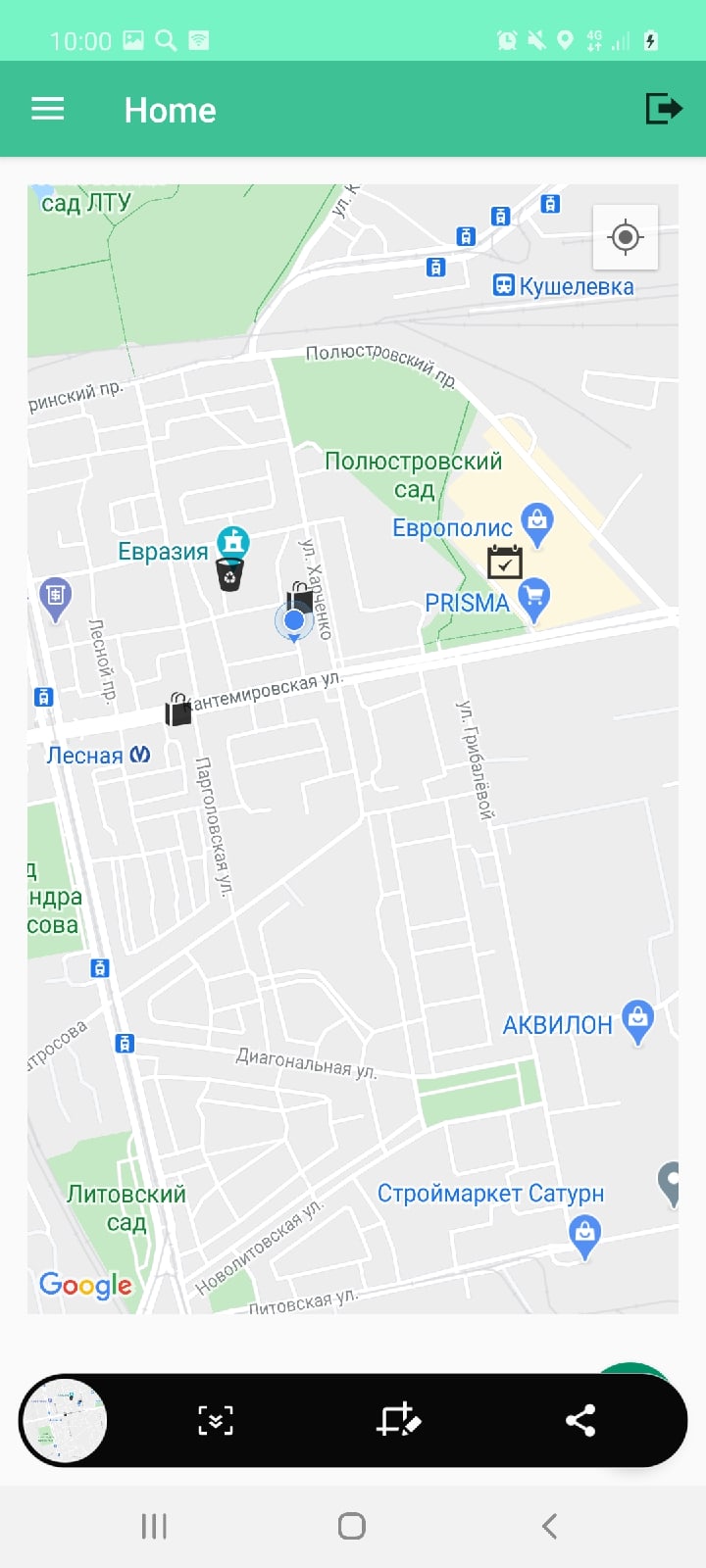


Рисунок 23 – Отображение точек на карте

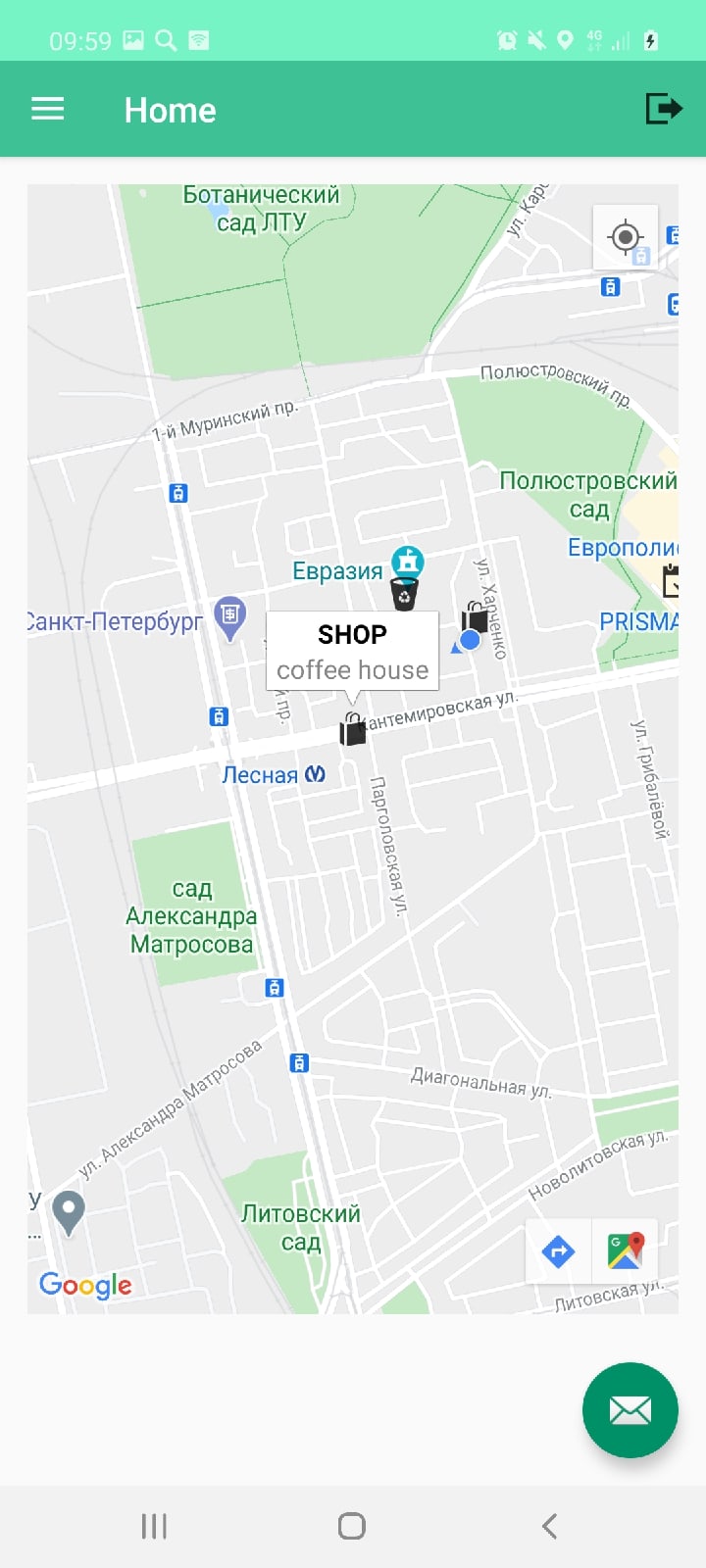


Рисунок 24 – Пример информации о точке

В опциональном меню есть возможность зарегистрироваться. На рисунке 25 приведено активити регистрации.

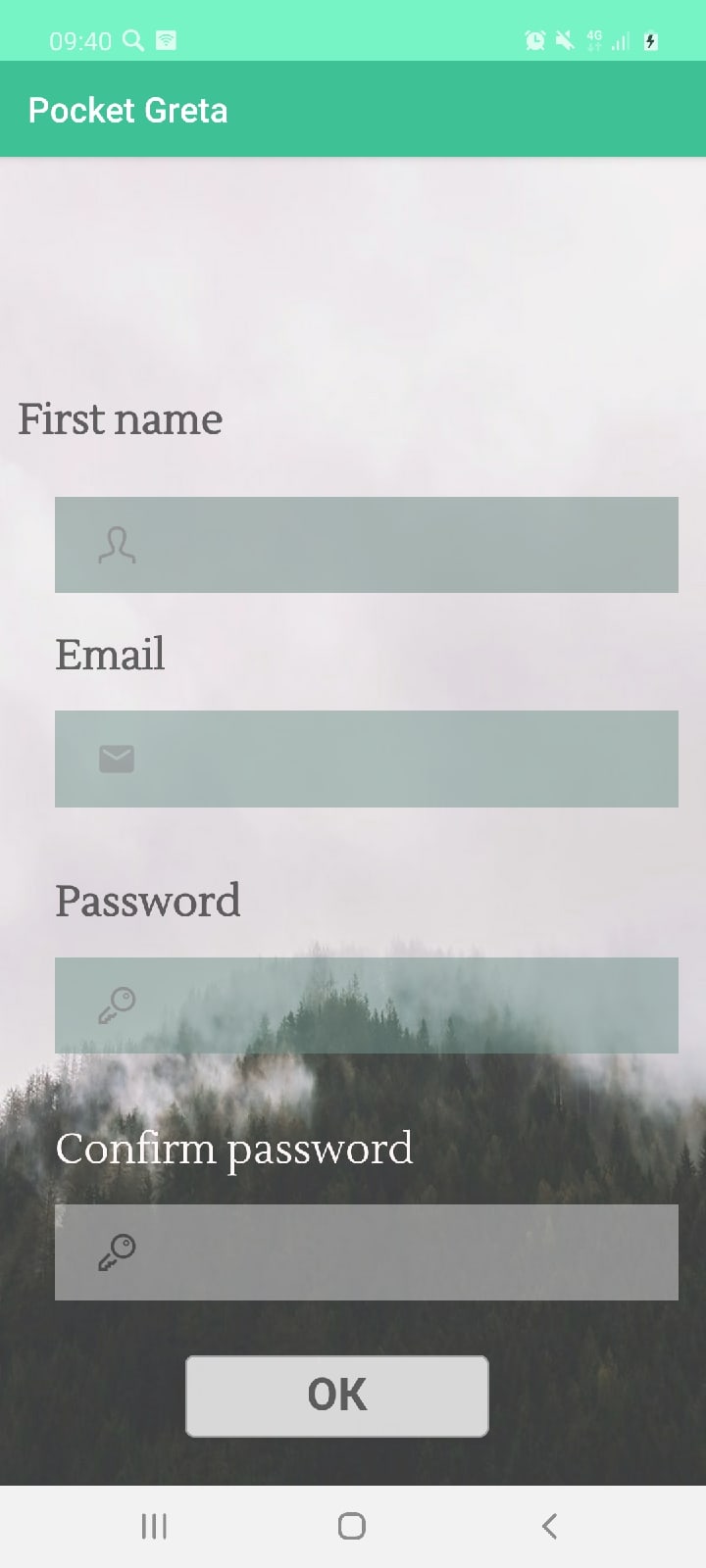


Рисунок 25 – Окно регистрации

При регистрации при помощи POST запроса, приведенного на рисунке 26.

AndroidNetworking.*post* (url)  
 .addJSONObjectBody(jsonObject)  
 .setPriority (Priority.*MEDIUM*)  
 .build ()  
 .getAsJSONObject (new JSONObjectRequestListener() {  
 @Override  
 public void onResponse (JSONObject response) {  
 Log.*d*("Registration","Login Succesfull");  
 }  
 @Override  
 public void onError (ANError error) {  
 Log.*d*("Registration","Error");  
 }  
 });  
}

Рисунок 26 – Отправка запроса на регистрацию пользователя

JSON object отправляемый на сервер в теле данного запроса выглядит следующим образом:

{

“username”:”username”,

“password”:”password”,

“email”:”email”

}

Если регистрация проходит успешно, то приложение переходит на страницу входа.

Для входа пользователя приложение отправляет на сервер следующий POST запрос:

AndroidNetworking.*post* (url)  
 .addJSONObjectBody(jsonObject)  
 .setPriority (Priority.*MEDIUM*)  
 .build ()  
 .getAsJSONObject (new JSONObjectRequestListener() {  
 @Override  
 public void onResponse (JSONObject response) {  
 Log.*d*("Login","Login Succesfull");  
 try {  
 access\_token = response.getString("authenticationToken");  
 response\_token = response.getString("refreshToken");  
 username = response.getString("username");  
 if(Singleton.*getInstance*()==null){  
 Singleton.*getInstance*(access\_token, username);  
 }  
 else  
 {  
 Singleton singleton = Singleton.*getInstance*();  
 singleton.setTokenAuth(access\_token);  
 singleton.setUsername(username);  
 }  
 SharedPreferences pref = getApplicationContext().getSharedPreferences("MyPref",*MODE\_PRIVATE*);  
 SharedPreferences.Editor editor = pref.edit();  
 editor.putString("tokenRefresh",response\_token);  
 editor.putString("username",username);  
 editor.putBoolean("FLAGUS", true);  
 editor.apply();  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 @Override  
 public void onError (ANError error) {  
 Log.*d*("Login","Login Error");  
 }  
 });

Рисунок 27 – Отправка запроса на вход

JSON object отправляемый на сервер в теле данного запроса выглядит следующим образом:

{

“email”:”email”,

“password”:”password”

}

При успешном входе сервер присылает клиенту RefreshToken, AuthenticationToken и username. Приложение сохраняет AuthenticationToken и username в singleton. Так же RefreshToken и username сохраняется в память ОС, доступную только данному приложению.

На рисунке 28 приведено контекстное меню. Есть возможность перехода на карты, а также в личный кабинет пользователя.

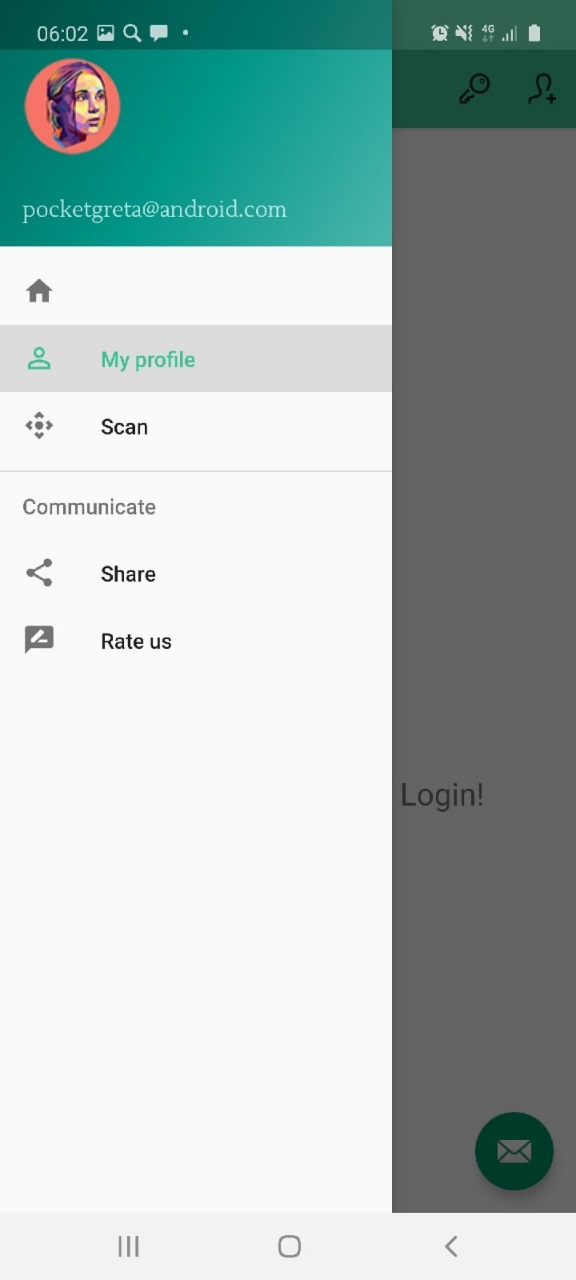


Рисунок 28 – Контекстное меню приложения

При переходе в личный кабинет пользователю, если пользователь еще не выполнил вход в систему, то выводится меня приведенное на рисунке 29.

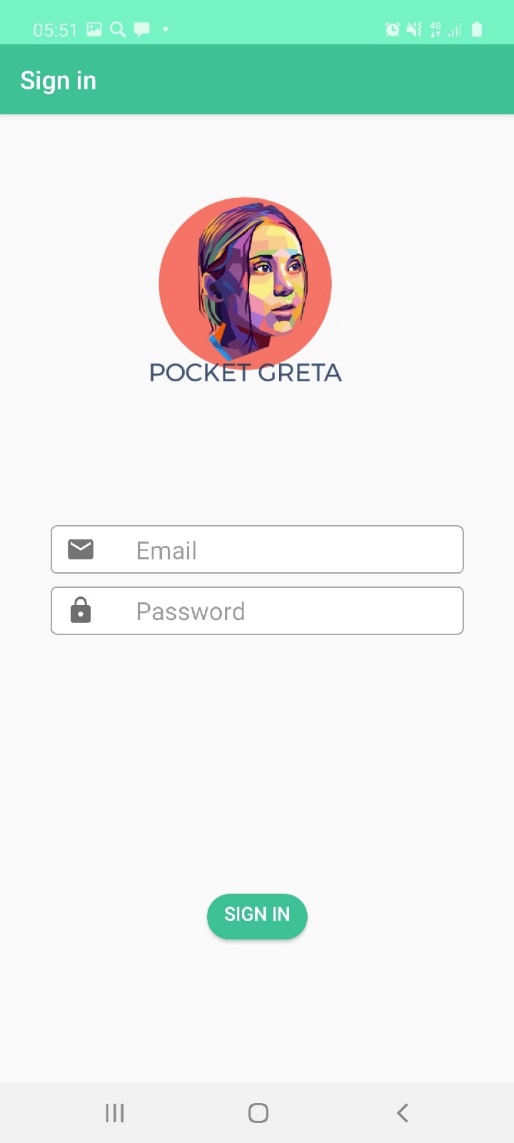


Рисунок 28 – Окно входа в систему

Пользователю предлагается произвести вход в систему. После чего переводит на экран входа в систему.

При переходе в личный кабинет залогиненного пользователя на сервер отправляется POST запрос. Отправка запроса приведена на рисунке 29.

AndroidNetworking.*post* (url+username)  
 .addHeaders("Authorization", "Bearer "+tokenAuth)  
 .setPriority (Priority.*MEDIUM*)  
 .build ()  
 .getAsJSONObject (new JSONObjectRequestListener() {  
 @Override  
 public void onResponse (JSONObject response) {  
 try {  
 bonus = response.getInt("bonuses");  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 @Override  
 public void onError (ANError error) {  
 Log.*d*("Registration",error.toString());  
 }  
 });

Рисунок 29 – Отправка запроса на доступ к бонусам.

JSON object отправляемый на сервер в теле данного запроса выглядит следующим образом:

{

“authenticationToken”:” authenticationToken”,

“username”:” username”

}

В ответ на данный запрос сервер присылает приложению JSONObject в теле которого передаются накопленные бонусы пользователя.

На экране пользователя выводится его username и количество накопленных бонусов, рисунок 30.

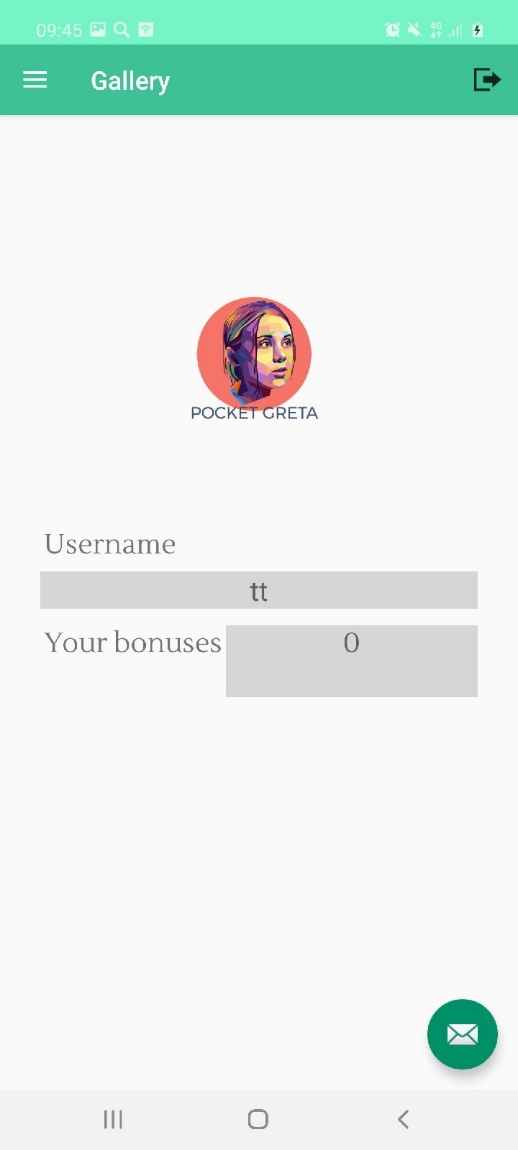


Рисунок 30 – Окно личного кабинета пользователя

При закрытии приложения, если пользователь не выполнил выхода из аккаунта, в ОС сохраняется RefreshToken и username. При повторном запуске приложения на сервер отправляется POST запрос, приведенный на рисунке 31.

AndroidNetworking.*post* (url)  
 .addJSONObjectBody(jsonObject)  
 .setPriority (Priority.*MEDIUM*)  
 .build ()  
 .getAsJSONObject (new JSONObjectRequestListener() {  
 @Override  
 public void onResponse (JSONObject response) {  
 try {  
 tokenAuth = response.getString("authenticationToken");  
 Singleton.*getInstance*(tokenAuth, username);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 @Override  
 public void onError (ANError error) {  
 }  
 });

Рисунок 31 – Отправка запроса на получение токена аутентификации

JSON object отправляемый на сервер в теле данного запроса выглядит следующим образом:

{

“refreshToken”:” refreshToken”,

“username”:” username”

}

В ответ на данный запрос сервер присылает JSONObject содержащий токен аутентификации. Данный токен сохраняется в singleton.

Singleton также хранит флаг, типа Boolean, который установлен в true, если пользователь вошел в систему, и в false в противном случае.

Данный флаг используется приложение для проверки, какое из двух меню выводить пользователю.

Если пользователь выполнил вход в систему, то ему выводится опциональное меню, приведенное на рисунке 32, в противном случае, как на рисунке 33.

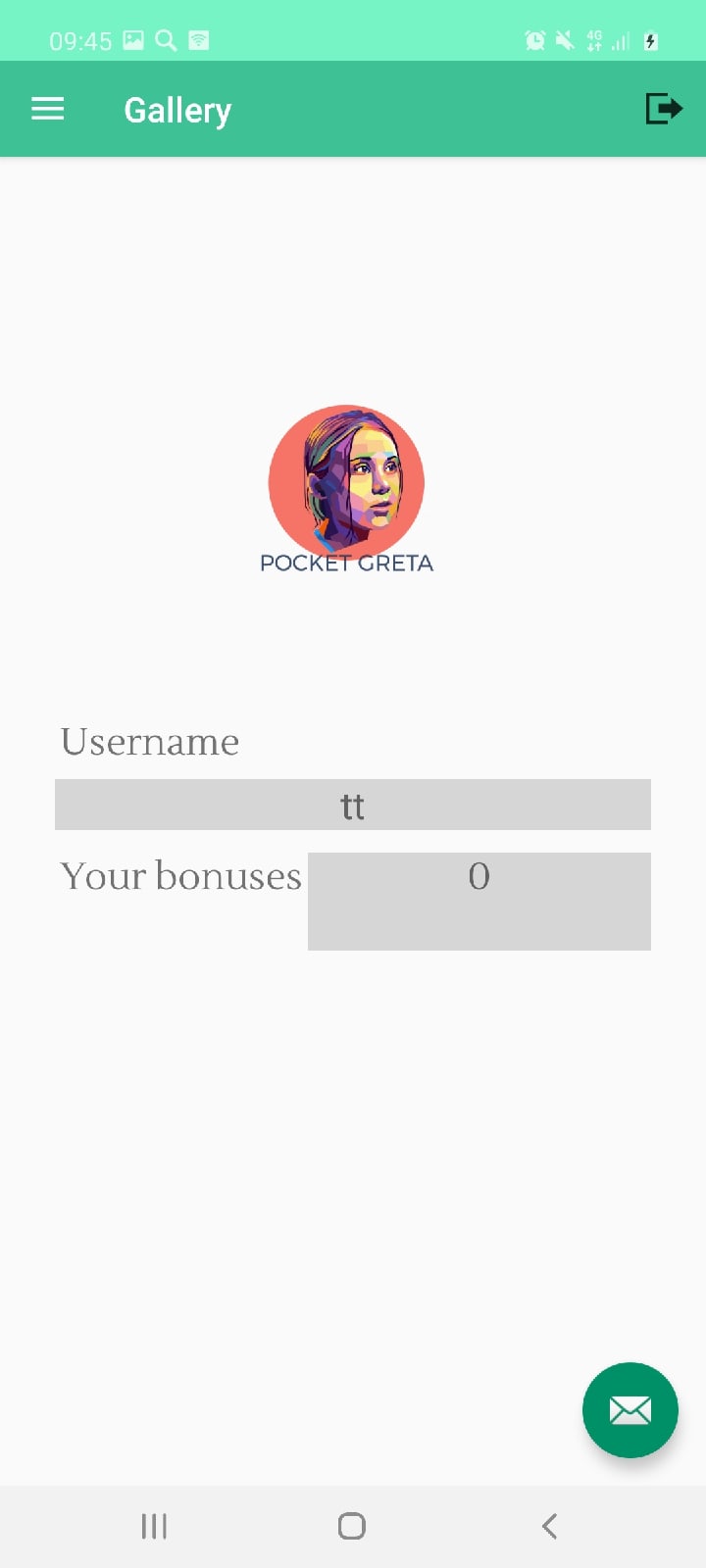


Рисунок 32 – Опциональное меню

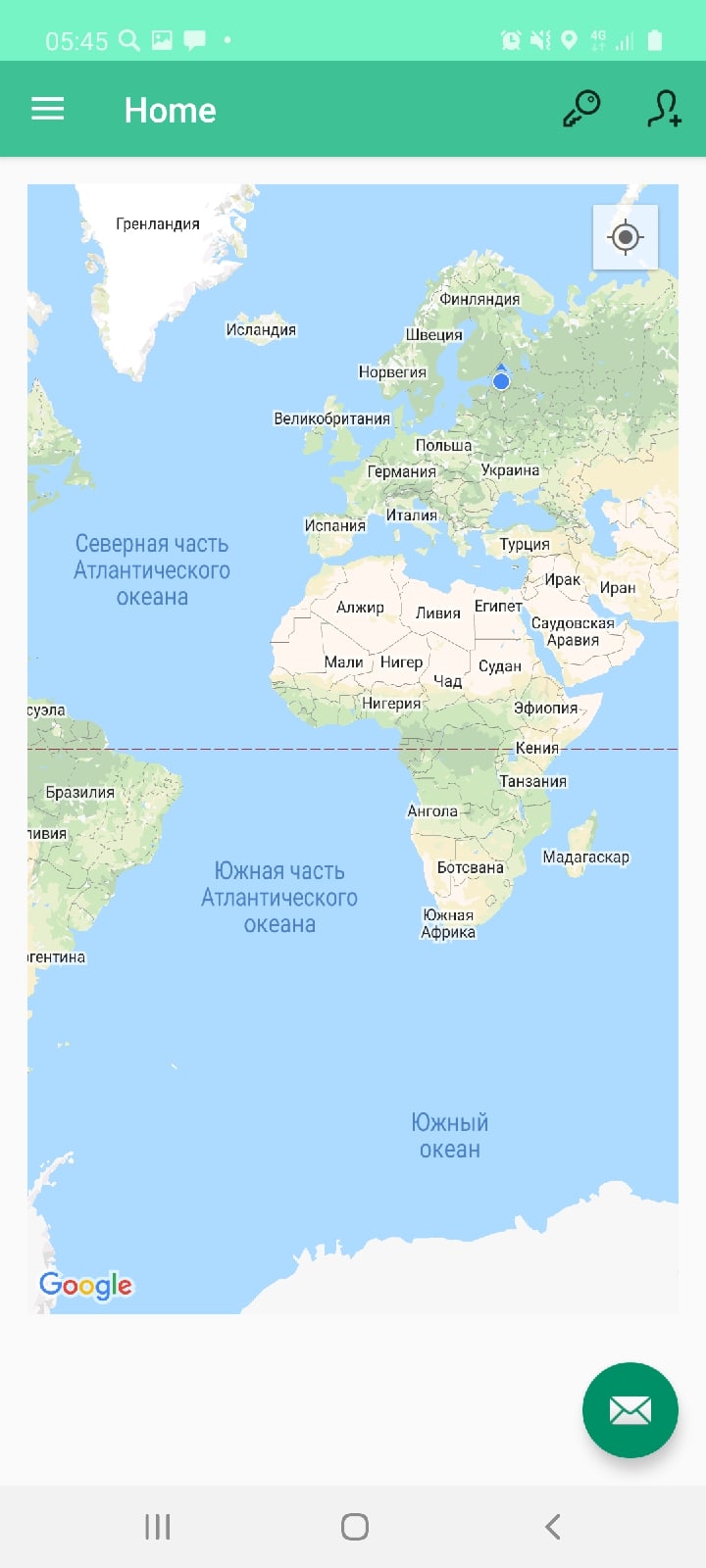


Рисунок 33 – Опциональное меню

**5.3 Безопасность хранения данных**

Для обеспечения безопасности на устройстве не хранятся чувствительные данные пользователей. Для общения с сервером через REST API необходимы username и RefreshToken. Для этих целей наиболее подходящим средством является компонент Android Shared Preferences (аналог Local Storage). В оригинальном решении данные хранятся в открытом в формате key-value в XML виде. Доступ к директории с файлами имеет только приложение, единственным способом получить доступ к файлам является наличие root-прав на устройстве.

**5.4 Обфускация исходного кода**

Для усложнения reverse-engineering’а была применена обфускация с помощью ProGuard. В результате обфускации размер APK файла уменьшился более, чем в 2 раза. Результаты представлены на рисунке 34.

**6 Разработка дизайна**

**6.1 Material Design**

Material Design — стиль графического дизайна интерфейсов программного обеспечения и приложений, разработанный компанией Google.

Material Design позволяет более объективно подходить к принятию дизайн-решений: как что-то выглядит, как что-то работает, как осуществляется анимация и тому подобное. Она задает разумные рамки, но не излишние ограничения. Новый подход задаёт вектор движения, а не создаёт чёткие границы. UI рассматривается как живой объект со всеми присущими ему свойствами.

Главные преимущества концепции:

* простота восприятия;
* унифицированный интерфейс на всех платформах;
* логичное поведение объектов;
* адаптивность.

Material Design строется на четырех основных принципах:

1. Тактильные поверхности. В Material Design интерфейс складывается из осязаемых слоёв так называемой «цифровой бумаги». Эти слои расположены на разной высоте и отбрасывают тени друг на друга, что помогает пользователям лучше понимать анатомию интерфейса и принцип взаимодействия с ним.
2. Полиграфический дизайн. Если считать слои кусками «цифровой бумаги», то в том, что касается «цифровых чернил» (всего того, что изображается на «цифровой бумаге»), используется подход из традиционного графического дизайна: например, журнального и плакатного.
3. Осмысленная анимация. В реальном мире предметы не возникают из ниоткуда и не исчезают в никуда — такое бывает только в кино. Поэтому в Material Design мы всё время думаем о том, как с помощью анимации в слоях и в «цифровых чернилах» давать пользователям подсказки о работе интерфейса.
4. Адаптивный дизайн. Речь идет о том, как мы применяем предыдущие три концепции на разных устройствах с разными разрешениями и размерами экранов.

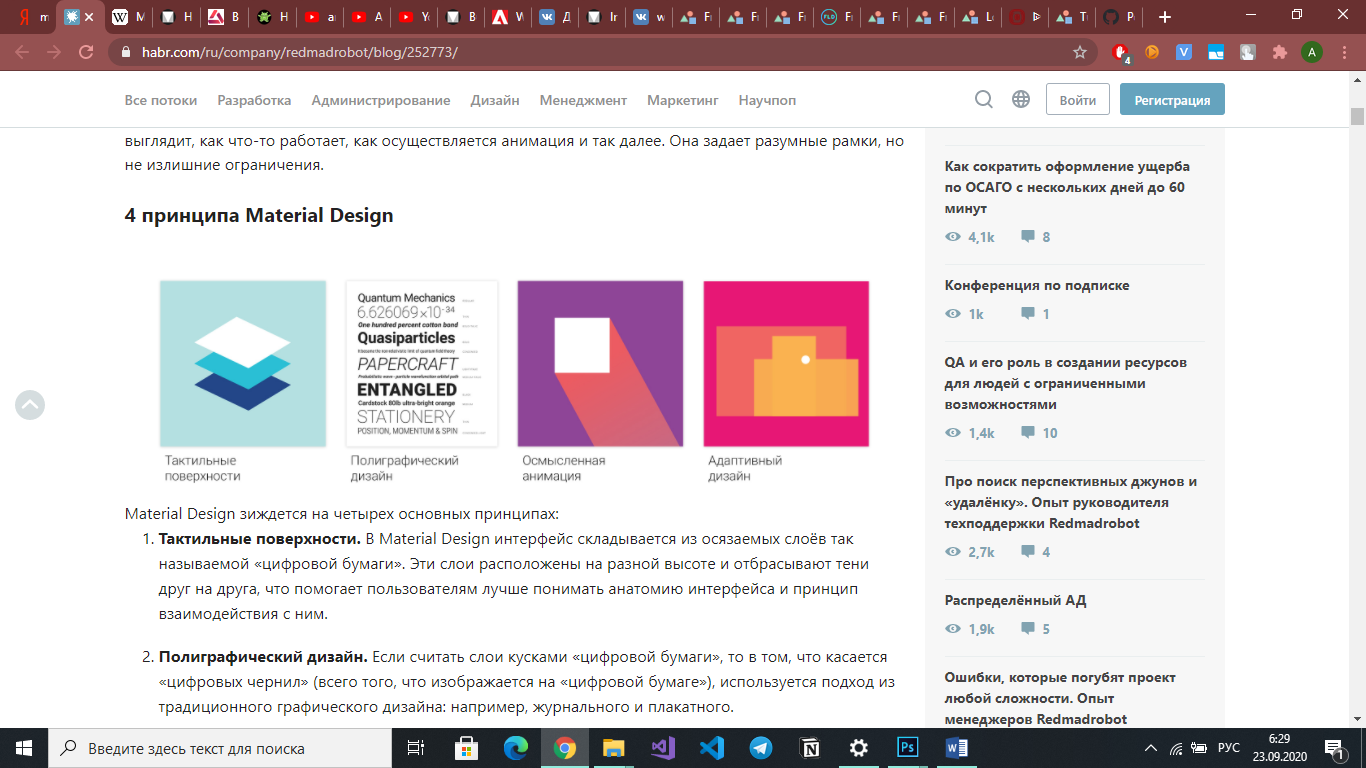


Рисунок 34 – 4 принципа Material Design

В дизайне интерфейсов цвет является важным средством выразительности. В прежнем Android цвет был чем-то дополнительным, теперь же он играет более заметную роль. В Material Design стандартная цветовая палитра приложения состоит из основного и акцентного цветов. Цветовые темы разработаны так, чтобы быть гармонично сочетаться друг с другом, обеспечивать хорошую читабельность текста и отличать элементы пользовательского интерфейса и поверхности друг от друга.

Базовая цветовая тема в Material Design - встроенная тема, которую можно использовать как есть. Сюда входят цвета по умолчанию: основной и второстепенный цвета. Дополнительные цвета пользовательского интерфейса используются для цвета фона, поверхностей, ошибок, типографики и значков. Все эти цвета можно настроить для приложения.

Основной цвет – это цвет, который чаще всего отображается на экранах и компонентах вашего приложения. Основной используется для больших областей вроде action bar, а в его более тёмную вариацию красится status bar. Дополнительный цвет дает больше возможностей для того, чтобы сделать акцент на чем-либо. Вторичный цвет не является обязательным, и его следует применять с осторожностью, чтобы выделить отдельные части вашего пользовательского интерфейса. Если у вас нет второстепенного цвета, ваш основной цвет также можно использовать для выделения элементов. Вторичные цвета лучше всего подходят для: плавающих кнопок, элементов управления выбором, такие как ползунки и переключатели, обозначение выделенного текста, индикаторов выполнения, а также для ссылок и заголовков. Более яркий акцентный цвет используется точечно в элементах управления, кнопках, полосках, индикаторах и т.д. Акцентный цвет призван привлекать внимание пользователя к ключевым элементам, таким как плавающая кнопка.

Акценты ставятся точечно, в небольшом количестве. Для раскрашивания остальной части интерфейса есть простое базовое правило. Когда текста много, например, это список почты, стоит оставить app bar стандартного размера и покрасить его, чтобы позволить пользователю сосредоточиться на содержимом. Если контента не так много, например, детальный просмотр отдельного элемента, фото или калькулятор, то это отличная возможность для применения больших цветных плашек — 2х или 3х высоты app bar.

Android поддерживает библиотеку под названием Palette, которая позволяет экстрагировать цвет из фотографий. То есть имеется возможность динамически красить интерфейс, исходя из фотоиллюстраций в приложении.

**6.2 Использованные инструменты**

Для создания дизайна приложения были использован следующие инструменты: Adobe Photoshop, Adobe XD, Figma и Color Pallete.

В Adobe Photoshop был создан логотип приложения. Adobe XD - это единое решение, позволяющее разрабатывать пользовательские интерфейсы, создавать интерактивные прототипы, отправлять работы редакторам, поэтапно дорабатывать проекты и просматривать обновления в режиме реального времени. и Figma – это графический онлайн-редактор для совместной работы. В нём можно создать прототип сайта, интерфейс приложения и обсудить правки с коллегами в реальном времени. Таким образом эти программные продукты были использованы для создания интерфейса приложения. Color Pallete использовалась для подбора наиболее гармоничной цветовой гаммы, которую мы использовали в своем дизайне.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения данной работы были изучены этапы разработки современных клиент-серверных приложений, ориентированных на работу в браузерах и на мобильных устройствах. Исследованы способы обеспечения безопасности компонентов распределенного приложения, взаимодействующих посредством протокола HTTP. Изучены подходы к построению системы авторизации, идентификации и аутентификации пользователей распределенного приложения.

# Список использованных источников

1. Руководство по CSS // MDN Web Docs. – 2020. – URL: <https://developer.mozilla.org/ru/docs/Web/CSS/Reference> - (дата обращения 01.05.2020).
2. Использование Axios для доступа к API // Vue JS. – 2020. – URL: <https://ru.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html> - (дата обращения 07.06.2020).
3. Cправочник по JavaScript // MDN Web Docs. – 2020. – URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference> - (дата обращения 11.06.2020).
4. How to create an HTTPS certificate for localhost domains // GitHub. – 2019. – URL: <https://gist.github.com/cecilemuller/9492b848eb8fe46d462abeb26656c4f8> - (дата обращения 22.07.2020).
5. Documentation for app developers // Developers – 2020. – URL: <https://developer.android.com/docs?hl=ru> - (дата обращения 22.08.2020).