ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Приглашенный преподаватель факультета компьютерных наук департамента программной инженерии

Г. М. Сосновский
«10» апреля 2023 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия»

В. В. Шилов
«11» апреля 2023 г.

ФОРУМ ДЛЯ РАЗМЕЩЕНИЯ ГАЙДОВ (сервер)

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.10-01 81 01-1-ЛУ

Исполнитель:
студент группы БПИ204

/ Д. А. Тасбауова /
«10» апреля 2023 г.

Москва 2023

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.05.10-01 81				

УТВЕРЖДЕН
RU.17701729.05.10-01 81 01-1-ЛУ

Пояснительная записка

RU.17701729.05.10-01 81 01-1

Листов 80

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.05 -01 81				

Москва 2023

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведена пояснительная записка к программе ««HSE Apple» образовательная социальная сеть для НИСа по iOS разработке»

В разделе «Введение» указано наименование программы, краткое наименование программы, документы, на основании которых ведется разработка, а также организация, утвердившая данный документ.

В разделе «Назначение и область применения» указано функциональное назначение программы, эксплуатационное назначение программы и краткая характеристика области применения программы.

В разделе «Технические характеристики» содержатся следующие подразделы:

1) постановка задачи на разработку программы;

2) описание алгоритма и функционирования программы с обоснованием выбора схемы алгоритма решения задачи и возможные взаимодействия программы с другими программами;

3) описание и обоснование выбора метода организации входных и выходных данных;

4) описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101–77 Виды программ и программных документов [1];

2) ГОСТ 19.102–77 Стадии разработки [2];

3) ГОСТ 19.103–77 Обозначения программ и программных документов [3];

4) ГОСТ 19.104–78 Основные надписи [4];

5) ГОСТ 19.105–78 Общие требования к программным документам [5];

6) ГОСТ 19.106–78 Требования к программным документам, выполненным печатным способом [6];

7) ГОСТ 19.404–79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603–78 [8], ГОСТ 19.604–78 [9].

Перед прочтением данного документа рекомендуется ознакомиться с терминологией, приведенной в Приложении 1.

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование: «Форум для размещения гайдов (Сервер)».

Наименование на английском языке: «Guide Forum»

1.2. Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1. Функциональное назначение

Основными целями создания приложения являются поддержка обмена знаниями, обучение и создание сообщества путем предоставления пользователям платформы для размещения, обмена и доступа к статьям и руководствам в различных областях человеческой деятельности.

Основные функции приложения:

1. Облегчение обмена знаниями: Приложение позволяет пользователям делиться своими знаниями и опытом в различных областях с другими людьми, облегчая другим возможность учиться и извлекать пользу из их опыта.

2. Контент, создаваемый пользователями: Приложение предоставляет пользователям платформу для размещения и обмена собственными статьями и руководствами, что делает его источником пользовательского контента, который может быть полезным и информативным для других.

3. Организация: Приложение организует статьи и руководства по различным категориям и темам, облегчая пользователям поиск нужной информации.

4. Доступность: Приложение предоставляет пользователям централизованную и легкодоступную платформу для доступа и чтения статей и руководств, что делает удобным для пользователей процесс обучения и расширения своих знаний.

Функциональным назначением серверной составляющей проекта является обработка запросов, предоставление данных для клиентской части и хранение данных приложения.

Функции серверной части:

1. Прием запросов от приложений-клиентов.
2. Интерпретация запросов.
3. Оптимизация и выполнение запросов к БД.
4. Отправка результатов приложению-клиенту.
5. Обеспечение системы безопасности и разграничение доступа.
6. Управление целостностью БД.

2.1.2. Эксплуатационное назначение

Эксплуатационное назначение серверной части приложения "Форум для размещения гайдов (Сервер)" заключается в предоставлении технологии и инфраструктуры, необходимых для поддержки функциональных возможностей приложения и обеспечения надежности, масштабируемости и безопасности платформы. Основные задачи серверной части могут быть обобщены следующим образом:

1. Хранение данных: На сервере хранятся все статьи, руководства, информация о пользователях и другие данные, генерируемые приложением.
2. Обработка данных: Сервер обрабатывает все запросы данных и операции, инициированные пользователями, такие как создание, обновление и удаление статей и руководств, управление аутентификацией и обратной связью пользователей.
3. Поиск данных: Сервер извлекает данные из базы данных и возвращает их в пользовательский интерфейс по мере необходимости, например, когда пользователь ищет статьи и руководства или запрашивает просмотр собственной информации.
4. Безопасность: Сервер реализует меры безопасности для защиты данных и информации, хранящихся на платформе, такие как шифрование и контроль доступа.
5. Производительность: Сервер оптимизирован для обеспечения производительности, что гарантирует плавную и эффективную работу приложения даже при умеренной нагрузке.

Конечным пользователем является клиентская часть приложения (Android, Web).

2.2. Краткая характеристика области применения

«Форум для размещения гайдов» - серверная часть приложения, предназначающегося для размещения пользовательских статей и руководств в различных сферах деятельности человека. Функциональным назначением серверной части приложения является обработка запросов, предоставление данных для клиентской части и работа с базой данных (её просмотр, хранение, редактирование и добавление данных).

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Программа должна обеспечивать возможность выполнения следующих функций:

Функции разделены на несколько блоков, имеющих общий логический контекст:

1) Контроль сеанса работы пользователя (регистрация, авторизация, log in/log out)
2) Профиль аккаунта и пользовательские настройки приложения
3) Навигация и лента статей
4) Статьи
5) Категории и тэги
6) Реакции и подписки пользователей

1. Функционал для Контроля аккаунтов и сеанса работы аккаунтов
   1.1. Регистрация аккаунта и каталог аккаунтов.
   1.2. Авторизация аккаунта
2. Функционал Профиля аккаунта и пользовательских настроек приложения
   2.1. Добавление/изменение данных профиля
   2.2. Предоставление данных профиля
   2.3. Изменение настроек пользователя
   2.4. Предоставление данных настроек
3. Функционал Навигации и ленты событий.
   3.1. Предоставление данных статей в ленте пользователя
   3.2. Предоставление данных статей по подписке в ленте пользователя
4. Функционал Статей.
   4.1. Добавление/изменение/удаление статьи
   4.2. Предоставление данных списка статей
5. Функционал Категорий
   5.1. Добавление/изменение/удаление категории к статьям
   5.2. Предоставление данных категории статей
6. Функционал Реакции и подписки пользователей
   6.1. Добавление/изменение/удаление реакции («нравится»/комментарии)

6.2. Предоставление данных реакций пользователей

6.3. Добавление/изменение/удаление подписки на другого пользователя

6.4. Предоставление данных подписок пользователя

3.2. Описание алгоритма и функционирования программы

3.2.1. Описание построения серверной части

Архитектура MVC (Model-View-Controller) является одной из наиболее популярных архитектурных паттернов для разработки веб-приложений. Эта архитектура разделяет приложение на три основных компонента: Model, View и Controller. Каждый компонент выполняет свою определенную роль в приложении, что упрощает его разработку и сопровождение.

В контексте нашего приложения "Форум для размещения гайдов", архитектура MVC реализована следующим образом:

1) Model - Модель отвечает за данные, хранящиеся в приложении. В нашем случае, модель представляет базу данных, где будут храниться все пользовательские данные, такие как профили, статьи, категории, тэги и т.д. Модель также будет содержать логику работы с базой данных, такую как добавление, обновление, удаление и запросы на получение данных.
2) View - Представление отвечает за отображение данных, полученных из модели. В нашем приложении, представление будет представлять собой интерфейс пользователя, где пользователи могут просматривать статьи, редактировать свой профиль, выбирать категории и т.д. Представление будет взаимодействовать с контроллером, чтобы получить данные из модели и отобразить их на странице.
3) Controller - Контроллер является посредником между моделью и представлением. Он обрабатывает все запросы от пользователя, вызывает нужные методы в модели для получения данных и возвращает эти данные представлению для отображения. В нашем приложении, контроллер будет содержать методы для обработки запросов на регистрацию и авторизацию пользователей, редактирования профиля, добавления и редактирования статей, управления категориями, управления подписками и реакциями пользователей.

Таким образом, использование архитектуры MVC помогает разделить приложение на три логически связанных компонента, что делает его более структурированным и удобным для разработки и поддержки**.**

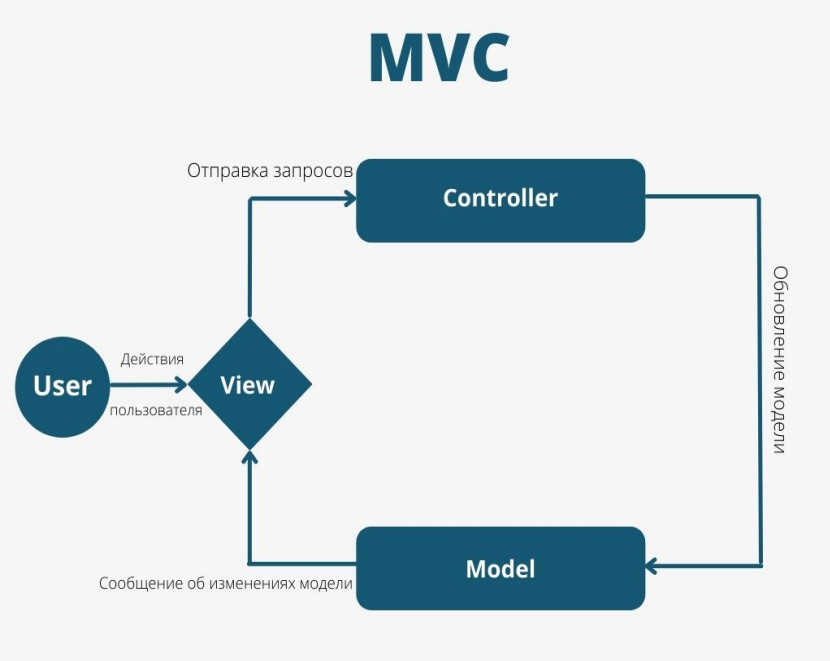

Рисунок 1 - MVC

3.2.1.1. Структура серверной части

Структура серверной части организована с помощью трех слоев: Controllers, Service и Repository.

1. Слой Controllers - управляет взаимодействием между клиентской и серверной частями приложения. Он принимает запросы от клиента и передает их в слой Service. Кроме того, он отвечает за обработку ошибок и форматирование ответов для клиента.
2. Слой Service - реализует бизнес-логику приложения и обращается к слою Repository для доступа к базе данных. Этот слой содержит логику, связанную с регистрацией и аутентификацией пользователей, управлением профилем и настройками пользователей, отображением статей, категорий, а также управлением реакциями и подписками пользователей.
3. Слой Repository - обеспечивает доступ к базе данных и выполнение запросов. Он реализует интерфейсы, определенные в слое Service, и обеспечивает выполнение операций с базой данных, таких как добавление, изменение, удаление и получение данных.

Такая структура позволяет легко масштабировать приложение и вносить изменения в отдельные модули, не затрагивая другие части приложения.

Структура серверной части представлена на рисунке 2.

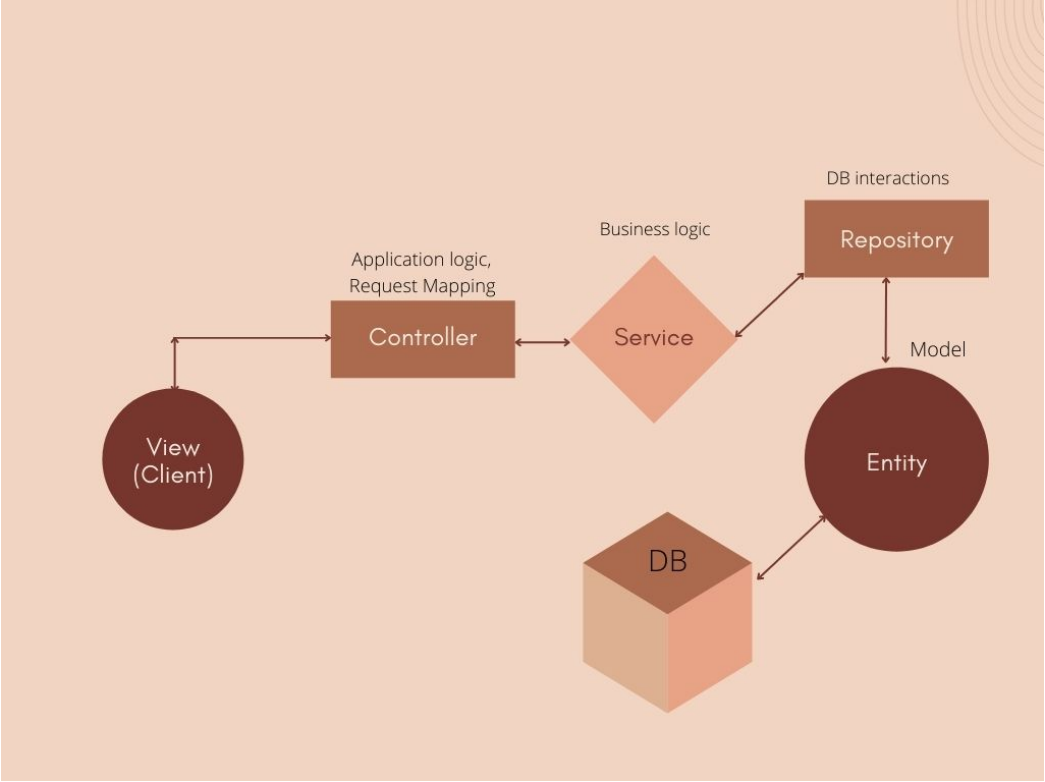

Рисунок 2 – структура серверной части

Контроллеры

В проекте имеется 3 контроллера, которые занимаются обработкой и управлением HTTP запросов и обеспечивают связь между клиентом и сервером.

1. UserController — контроллер, отвечающий за информацию о пользователе.
2. ArticleController — контроллер, отвечающий за информацию о курсах.
3. AuthController - контроллер, отвечающий за управление информации о заданиях.

```
@RestController
@SecurityRequirement(name = "Authorization")
@Tag(description = "Api to manage authorization",
        name = "User Resource")
public class AuthController {

    @Autowired
    AuthService authService;

    @Autowired
    RefreshTokenService refreshTokenService;
```

```
    @Autowired
    JwtUtils jwtUtils;

    @PostMapping(value = "/auth/signin", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> authenticateUser(@Valid @RequestBody AuthRequest
loginRequest) {
        return authService.authenticateUser(loginRequest);
    }

    @PostMapping(value = "/auth/signup", produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> registerUser(@Valid @RequestBody SignUpRequest
signUpRequest) {
        return authService.registerUser(signUpRequest);
    }
```

Листинг 1 - язык Java, AuthController

Сервисы

В проекте используются 4 сервиса, в соответствии с контроллерами: UserService, AuthService, ArticleService, RefreshService, которые реализуют бизнес-логику приложения и обращается к слою Repository для доступа к базе данных. Четвертым сервисом является сервис для обработки, создания и управления Refresh Token-ом.

```
@Service
public class AuthService {
    @Autowired
    UserRepository userRepository;

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    JwtUtils jwtUtils;

    @Autowired
    RefreshTokenService refreshTokenService;

    @Autowired
    PasswordEncoder encoder;

    public ResponseEntity<?> authenticateUser(AuthRequest loginRequest) {
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(loginRequest.getLogin(),
loginRequest.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtUtils.generateJwtToken(authentication);
        UserDetailsImpl userDetails = (UserDetailsImpl)
authentication.getPrincipal();
        RefreshToken refreshToken =
refreshTokenService.createRefreshToken(userDetails.getId());
```

```
        return ResponseEntity.ok(new JwtResponse(jwt, refreshToken.getToken(),
userDetails.getId(),
                userDetails.getLogin(), userDetails.getUsername(),
userDetails.getRole()));
    }

    public ResponseEntity<?> registerUser(SignUpRequest signUpRequest) {
        if (userRepository.existsByLogin(signUpRequest.getLogin())) {
            return ResponseEntity
                    .badRequest()
                    .body(new MessageResponse("Error: Login is already taken!"));
        }
        UserEntity user = new UserEntity();
        user.setLogin(signUpRequest.getLogin());
        user.setPasswordHash(encoder.encode(signUpRequest.getPassword()));
        user.setUsername(signUpRequest.getUsername());
        user.setUserRole(signUpRequest.getUserRole());
        user.setCardDetails(signUpRequest.getCardDetails());
        user.setProfile(signUpRequest.getProfile());
        user.setStatus(true);
        user.setCreatedAt(LocalDateTime.now());
        userRepository.save(user);
        return ResponseEntity.ok(new MessageResponse("User registered
successfully!"));
    }
```

Листинг 2 - язык Java, AuthService

Также некоторая логика была вынесена в отдельные пакеты:

1) Пакет security:
    1) Пакет jwt
        1) AuthTokenFilter, AuthEntryPointJwt, JwtUtils
    2) Пакет services
        1) UserDetailsImpl, UserDetailsServiceImpl
    3) WebSecurityConfig

Для реализации аутентификации в проекте используется аутентификация с Bearer Token в формате JWT. В Spring Framework была использована библиотека Spring Security для реализации аутентификации с JWT access и refresh токенами в данном пакете.

Алгоритм работы аутентификации с JWT access и refresh токенами в проекте следующий:

1. Пользователь вводит логин и пароль на клиентской стороне и отправляет их на сервер.

2. Сервер проверяет логин и пароль и если они верны, создает пару JWT токенов - access и refresh.
3. Сервер отправляет эти токены обратно клиенту.
4. Клиент сохраняет полученные токены в local storage или cookie.
5. Клиент отправляет access токен с каждым запросом на сервер, чтобы доказать свою идентичность.
6. Сервер проверяет валидность access токена и если он прошел проверку, обрабатывает запрос и отправляет результат обратно клиенту.
7. Если access токен истек или недействителен, клиент отправляет refresh токен на сервер для получения нового access токена.
8. Сервер проверяет валидность refresh токена и если он прошел проверку, создает новую пару access и refresh токенов и отправляет их обратно клиенту.
9. Клиент обновляет access и refresh токены в local storage или cookie и продолжает отправлять запросы с новым access токеном.

Это позволяет снизить риск несанкционированного доступа к ресурсам сервера и увеличивает безопасность приложения.

1. AuthTokenFilter - это класс-фильтр, который отвечает за проверку наличия JWT-токена в запросе пользователя и его валидность. Он реализует интерфейс jakarta.servlet.FilterChain и обрабатывает каждый запрос, проходящий через сервер. Если токен валидный, то фильтр передает запрос дальше по цепочке фильтров и обработчиков. В противном случае, если токен отсутствует или недействителен, то фильтр возвращает ошибку авторизации.
2. AuthEntryPointJwt - это класс, который отвечает за обработку ошибок авторизации. Он реализует интерфейс org.springframework.security.web.AuthenticationEntryPoint и перехватывает ошибки авторизации, генерируемые AuthTokenFilter. Если пользователь не авторизован, то AuthEntryPointJwt возвращает HTTP-ответ со статусом 401 (Unauthorized) и сообщением об ошибке.

3. JwtUtils - это класс, который отвечает за генерацию и проверку JWT-токенов. Он содержит методы для создания JWT-токена на основе данных пользователя, проверки валидности токена и извлечения информации из токена. JwtUtils использует библиотеку JSON Web Token (JWT), которая позволяет создавать и проверять токены, содержащие зашифрованную информацию о пользователе и его правах в системе.
4. Класс UserDetailsImpl предназначен для представления информации о пользователе, используемой для аутентификации и авторизации в системе. Он реализовывает интерфейс org.springframework.security.core.userdetails.UserDetails, который определяет методы для доступа к данным пользователя, таким как имя пользователя, пароль, права доступа и т.д.
5. Класс UserDetailsServiceImpl предназначен для предоставления реализации сервиса, который загружает данные пользователя из базы данных и передает их в объект UserDetailsImpl. Он реализовывает интерфейс org.springframework.security.core.userdetails.UserDetailsService, который определяет метод для загрузки UserDetails по имени пользователя.
6. Класс WebSecurityConfig предназначен для конфигурации Spring Security в приложении. Он содержит настройки безопасности, такие как ограничение доступа к определенным страницам, настройку аутентификации и авторизации, использование SSL, CSRF защиты и т.д. Класс расширяет WebSecurityConfigurerAdapter, который определяет методы для настройки безопасности в приложении.

2) Пакет exception: AppException, BusinessException, InternalException, TokenRefreshException, ExceptionMapper, ErrorMessage.

В этом пакете была реализована обработка ошибок. В проекте реализован свой обработчик исключений, расширяющий класс ResponseEntityExceptionHandler. Были созданы собственные классы исключений, чтобы обработать конкретные ошибки, которые возникают в приложении.

Описание всех классов проекта указаны в Приложении 4.

3.2.1.2. Взаимодействие и хранение данных в базе данных

Для проекта "Форум для размещения гайдов" в качестве базы данных используется СУБД PostgreSQL 12. База данных состоит из 8 таблиц, которые хранят информацию о пользователях, гайдах, комментариях и других объектах проекта.

Для взаимодействия с базой данных проект использует стандарт JDBC, который позволяет работать с БД на разных платформах. Для реализации концепции ORM и удобного хранения Java-объектов в БД используется спецификация JPA. Для ее реализации используется библиотека Hibernate, которая позволяет описывать отношения между Java-объектами и записями в БД.

Hibernate предоставляет мощные средства для автоматической генерации и обновления данных, автоматизирует построение SQL-запросов и обработки полученных наборов данных, что существенно ускоряет разработку проекта. Благодаря этому проект может быстро и эффективно обрабатывать большие объемы данных и обеспечивать высокую производительность.

```
@Repository
public interface ArticleRepository extends JpaRepository<ArticleEntity, Long> {
    List<ArticleEntity> findAllByCreatedBy(UserEntity createdBy);
    List<ArticleEntity> findAllByCreatedByAndDraft(UserEntity createdBy, Boolean
draft);
    List<ArticleEntity> findAllByCategoryId(Integer categoryId);
}
```

Листинг 3 - язык Java, представлено взаимодействие с таблицей БД Article

В проекте для представления модели базы данных использовались сущности в JPA. Эти сущности представляют данные, которые могут быть сохранены в БД и описывают таблицы, хранящиеся в базе данных. Каждое поле сущности представляет собой столбец в таблице. Например, может быть создана сущность "Статья", которая будет соответствовать таблице "Статьи" в базе данных и будет содержать поля, такие как "Заголовок", "Содержание", "Дата создания" и т.д. В каждой из них описаны связи с другими таблицами/сущностями – oneToOne, oneToMany, ManyToMany.

1. ArticleEntity — информация о статье.
2. CategoryEntity — данные о категориях для статей.
3. CommentEntity — данные о комментариях пользователей к статье.
4. RefreshToken — данные refresh токенов.

5. UserEntity — данные о пользователе.

```
@Entity
@Table(name = "article", schema = "public")
public class ArticleEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(optional=false, cascade= CascadeType.MERGE)
    @JoinColumn(name="createdby")
    private UserEntity createdBy;

    @ManyToOne(optional=false, cascade=CascadeType.MERGE)
    @JoinColumn(name="categoryid")
    private CategoryEntity category;

    @Column(name = "title")
    private String title;
```

Листинг 4 - язык Java, сущность ArticleEntity, описывающая таблицу Article

Также были использованы классы DTO – Request и Response для каждой сущности.
Классы DTO (Data Transfer Objects) используются для передачи данных между слоями приложения. Они позволяют разделить представление данных от их обработки и хранения, что повышает гибкость приложения и упрощает его разработку и тестирование.

В проекте классы DTO использованы для передачи данных пользователей, статей, категорий и других объектов. Классы DTO содержат только необходимые поля объекта. (например, для статьи - заголовок, дату публикации, автора) без дополнительной информации, которая не требуется на клиентской стороне. Это используется и для получения объектов в запросе, и для ответа сервера клиентской части. Классы следующие: ArticleRequest/ArticleResponse, AuthRequest/SignUpRequest, ReactionRequest/CommentResponse, TokenRefreshRequest/TokenRefreshResponse, MessageResponse, ProfileResponse.

```
public class ArticleResponse {

    private Long id;
    private ProfileResponse createdBy;
    private CategoryEntity category;
    private String title;
    private String text;
    private String description;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private Boolean draft;
    private Boolean statusSave;
    private Integer likes;
    private Boolean statusLike;
}
```

Листинг 5 - язык Java, сущность ArticleResponse, описывающая ответ сервера

3.3. Описание и обоснование метода организации входных и выходных данных

3.3.1. Описание метода организации входных и выходных данных

Система использует современные протоколы и форматы передачи данных для обеспечения информационного обмена между подсистемами. Для обмена информацией между подсистемами используются только документированные программные интерфейсы (API), а прямой вызов функций и данных одной подсистемы из другой не допускается, чтобы обеспечить безопасность и стабильность работы системы.

Обмен информацией между подсистемами осуществляется посредством сетевого режима передачи данных, в котором данные передаются по протоколам HTTP/HTTPS. Взаимодействие между Backend-частью и клиентом происходит через запрос-ответ.

Запрос с клиентской стороны содержит URL (endpoint API), который определяет требуемый ресурс, и метод, который определяет необходимое действие с ресурсом (получить, изменить, создать, удалить ресурс). Запрос также может содержать дополнительную информацию в параметрах URL или в теле запроса (HTTP POST). В заголовках запроса могут содержаться дополнительные данные, такие как токены авторизации и аутентификации, типы контента и т.д.

Методы поддерживают четыре типа запросов: GET, POST, PUT и DELETE. Метод GET используется для получения ресурса, метод POST – для создания ресурса, метод PUT – для обновления ресурса, а метод DELETE – для удаления ресурса. Эти методы обеспечивают стандартный способ взаимодействия между клиентом и сервером, что позволяет обрабатывать запросы и отвечать на них в соответствии с установленными правилами и форматами данных.

Описание методов REST API приведено в Приложении 3.

Для обмена информацией в Системе используются документированные программные интерфейсы (API) с использованием современных протоколов и форматов передачи данных, таких как HTTP/HTTPS и JSON. Пользовательский интерфейс передает данные в формате JSON, а для передачи медиаконтента используются файлы соответствующих типов.

Выходные данные формируются на уровне бизнес-логики и БД, дополнительно обрабатываются в соответствии с бизнес-логикой приложения и преобразовываются в JSON в соответствии со стандартом REST API для унификации интерфейса клиент-серверного взаимодействия.

При обработке запросов клиентской части Backend-часть отсылает в ответе статус обработки запроса, указывающий на успех или неудачу операции, а также сообщение об ошибке в формате JSON. Коды статусов обработки запросов могут быть различными, включая 200, 201, 400, 401, 404 и др.

3.3.2. Обоснования выбора метода организации входных и выходных данных

Для обеспечения унификации интерфейса клиент-серверного взаимодействия и удобства передачи данных между серверной и клиентской частями приложения, рекомендуется использовать JSON для организации входных и выходных данных.

JSON является удобным форматом для представления структурированных данных в виде объектов и массивов, что позволяет легко передавать сложные данные в запросах и ответах. Кроме того, JSON поддерживается большинством языков программирования и является удобным форматом для чтения и записи данных в БД.

Использование JSON для организации входных и выходных данных также обеспечивает возможность автоматического преобразования данных из объектов и массивов в структуры данных языка программирования, что упрощает работу с данными на стороне сервера.

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Состав технических средств

Для работы серверной части приложения необходимо выделенное серверное оборудование или виртуальный сервер с определенными минимальными техническими характеристиками:

1. Вычислительная мощность: не менее одного процессора со спецификацией: тактовая частота 3,1 ГГц, 2 ядра, 4 потока.
2. Оперативная память: не менее 2 ГБ.
3. Жесткие диски: рабочее пространство не менее 10 ГБ.
4. Сетевой адаптер: не менее 1 порта Gigabit Ethernet.

Также, для обеспечения масштабируемости и производительности серверных технических средств, необходимы ресурсы для расширения емкости оперативной памяти, дискового пространства и числа каналов ввода-вывода.

3.4.2. Состав программных средств

1. Наличие Java SE Development Kit 17 и выше.
2. Наличие фреймворков Spring Boot, Spring Data JPA, Spring Security, Postgresql Driver, Liquidbase.
3. ОС Ubuntu 21 и выше.
4. PostgreSQL версии 12 и выше.

3.4.3. Обоснование выбора технических и программных средств

Выбор Java и фреймворка Spring Boot для реализации серверной части был обусловлен несколькими причинами.

Во-первых, Java является одним из самых распространенных языков программирования, который обладает множеством библиотек и фреймворков, позволяющих разработчикам быстро и эффективно создавать высококачественное программное обеспечение.

Во-вторых, Spring Boot является одним из наиболее популярных фреймворков для разработки серверных приложений на Java. Он предоставляет множество инструментов и библиотек, упрощающих разработку, тестирование и развертывание приложений. Spring Boot также обеспечивает интеграцию с другими технологиями и библиотеками, что делает его привлекательным выбором для создания сложных и расширяемых систем.

Дополнительно была использована Spring Data JPA, которая обеспечивает удобный и простой доступ к базе данных, используя объектно-реляционное отображение (ORM). Это позволило избежать написания большого количества SQL-кода и сосредоточиться на логике приложения.

Библиотека Spring Security обеспечивает безопасность приложения, предоставляя механизмы аутентификации и авторизации, защиту от атак и утечки данных.

Была выбрана СУБД PostgreSQL в качестве технического и программного средства для работы с базой данных.

Основные причины выбора PostgreSQL:

- Надежность и стабильность: PostgreSQL имеет долгую историю разработки и широкое использование в различных проектах. СУБД прошла множество тестов и имеет высокую степень надежности.
- Расширяемость: PostgreSQL имеет множество расширений, которые могут быть использованы для реализации различных функций приложения.
- Безопасность: PostgreSQL имеет множество функций безопасности, таких как шифрование данных и авторизация пользователей, что важно для приложений, которые работают с конфиденциальной информацией.
- Гибкость: PostgreSQL может работать на различных операционных системах и поддерживает множество языков программирования, что упрощает разработку и интеграцию с другими системами.

Liquibase обеспечило управление миграциями базы данных, что позволило управлять версиями базы данных и поддерживать ее в актуальном состоянии.

Для удобного способа создания документации API на основе аннотаций кода была использована Springdoc-api.

Библиотека jjwt предоставило удобный способ генерации и проверки JSON Web Token (JWT), который используется для аутентификации и авторизации в приложении.

Lombok предоставил аннотации для генерации кода автоматически, что сократило количество рутинной работы и повысило производительность разработки.

В качестве тестового сервиса для деплоя серверной части был выбран облачный VPS, на котором было установлено программное окружение в минимальных требованиях сервера. Это позволило не ограничиваться лимитами сервисов других платформ.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

4.2. Предполагаемая потребность

В целом, предполагаемая потребность в серверном приложении "Форум размещения гайдов" заключается в предоставлении пользователям платформы для обмена знаниями и информацией, общения с единомышленниками и создания сообщества пользователей, увлеченных изучением и обменом информацией в различных областях человеческой деятельности.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

В рамках анализа конкурентов приложения были выделены следующие основные характеристики для сравнения:

1. Функциональность - это набор возможностей и функций приложения, позволяющих пользователю выполнить определенные задачи.
2. Русская локализация - это перевод интерфейса и содержимого приложения на русский язык, чтобы обеспечить удобство использования для русскоязычных пользователей. Русская локализация может быть важна для приложений, которые ориентированы на русскоязычную аудиторию.
3. Удобство редакторов - это способность редактировать контент приложения с легкостью и эффективностью. Удобные редакторы могут ускорить процесс создания и редактирования контента, а также улучшить качество результата.
4. Функциональность редактора - это спектр возможностей, доступных для редактирования контента, таких как форматирование текста, добавление изображений и видео, создание таблиц и другие функции.
5. UI (User Interface) - это дизайн и интерфейс приложения, которые влияют на удобство и привлекательность приложения для пользователей. UI должен быть интуитивно понятным и легким в использовании, что может привлечь больше пользователей.

6. Безопасность данных - это важная характеристика, которая обеспечивает защиту данных пользователей. Система безопасности должна предотвращать несанкционированный доступ к данным и защищать их от вирусов и других угроз.
7. Объем информации - это количество информации, которое может быть добавлено в приложение.

Был произведен сравнительный анализ с прямыми и косвенными аналогами:

1. Дзен (Яндекс.Дзен) - это платформа для чтения и публикации контента, использующая индивидуальные рекомендации и алгоритмы, чтобы персонализировать поток контента для каждого пользователя.
2. Хабр - это платформа, которая сосредоточена на технологическом контенте, включая новости, статьи и блоги о программировании, информационных технологиях и связанных темах.
3. Tproger - это ресурс, специализирующийся на обзорах, новостях и статьях об информационных технологиях, программировании, веб-разработке и связанных темах.
4. Stack Overflow - это платформа, предназначенная для обмена знаниями в области программирования и разработки. Пользователи могут задавать вопросы и получать ответы от других пользователей, которые обладают соответствующими знаниями.
5. Reddit - это платформа, которая позволяет пользователям создавать, делиться и обсуждать контент в широком спектре тематик.
6. HashTap - это социальная сеть для обмена и обсуждения идей, проектов и опыта в сфере информационных технологий.

Приведена таблица сравнительного анализа аналогов:

Критерии / Название приложения	Дзен [15]	Хабр [16]	Tproger [17]	Stack Overflow [18]	Reddit [19]	HashTap [20]	Guidepedia
Функциональность	3	5	5	4	5	3	3
Локализация	5	5	5	0	2	5	5
Удобство редактора	4	4	5	5	5	4	4
Функциональность редактора	4	5	3	3	4	3	3
UI	4	3	4	3	4	4	4
Безопасность	2	4	4	4	4	3	4
Количество данных	3	4	4	5	5	2	2
Оценка	**3.57**	**4.28**	**4.28**	**3.42**	**4.14**	**3.42**	**3.57**

Указанные выше характеристики ранжированы. Наивысший балл - 5, наименьший - 1.

5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5) ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8) ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

9) ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10) Системные требования Java [Электронный ресурс]// URL https://www.java.com/ru/download/help/sysreq.html (Дата обращения: 20.04.2022, режим доступа: свободный).

11) Системные требования PostgreSQL [Электронный ресурс]// URL https://postgrespro.ru/docs/postgresql/13/install-requirements (Дата обращения: 20.04.2022, режим доступа: свободный).

12) Требования к системе для Spring [Электронный ресурс]// URL: https://java-ru-blog.blogspot.com/2020/02/spring-boot-features.html (Дата обращения: 20.04.2022, режим доступа: свободный).

13) Документация SpringBoot [Электронный ресурс]// URL: https://spring.io/projects/spring-boot (Дата обращения: 20.04.2022, режим доступа: свободный).

14) Документация LiquidBase [Электронный ресурс]// URL: https://www.liquibase.org (Дата обращения: 20.04.2022, режим доступа: свободный).

15) «Дзен» [Электронный ресурс]. Режим доступа: https://dzen.ru/ (Дата обращения 03.04.2023)

16) «Хабр» [Электронный ресурс]. Режим доступа: https://habr.com/ru/all/ (Дата обращения 03.04.2023)

17) «Tproger» [Электронный ресурс]. Режим доступа: https://tproger.ru/ (Дата обращения 03.04.2023)

18) «Stack Overflow» [Электронный ресурс]. Режим доступа: https://stackoverflow.com/ (Дата обращения 03.04.2023)

19) «Reddit» [Электронный ресурс]. Режим доступа: https://www.reddit.com/ (Дата обращения 03.04.2023)

20) «Hashtap» [Электронный ресурс]. Режим доступа: https://www.hashtap.com/ (Дата обращения 03.04.2023)

ПРИЛОЖЕНИЕ 1
ТЕРМИНОЛОГИЯ

1. **RESTful API** — это API, которое соответствует принципам архитектурного стиля REST. Оно использует запросы HTTP для выполнения стандартных функций базы данных, таких как создание, чтение, обновление и удаление записей.

2. **Система управления базами данных (СУБД)** - это набор программных и лингвистических средств, которые обеспечивают управление созданием и использованием баз данных.

3. **HTTP** — это протокол прикладного уровня, используемый для передачи данных, который изначально был разработан для гипертекстовых документов в формате HTML, но сейчас используется для передачи произвольных данных.

4. **JDBC –** это платформенно-независимый промышленный стандарт, который позволяет Java-приложениям взаимодействовать с различными СУБД через пакет java.sql, входящий в состав Java SE.

5. **JPA –** это спецификация API Java EE, которая позволяет сохранять Java-объекты в базе данных в удобном виде.

6. **JSON (**англ. *JavaScript Object Notation*) — это текстовый формат обмена данными, основанный на языке JavaScript.

7. **JWT** (англ. *JSON Web Token*) — это JSON-объект, который определен в открытом стандарте RFC 7519.

8. **Spring Boot**— это открытая среда на основе Java, используемая для создания микросервисов.

9. **База данных** (БД) — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде на компьютерной системе

10. **API** - это набор правил, определяющих способ взаимодействия между приложениями или устройствами.

ПРИЛОЖЕНИЕ 2
СТРУКТУРА БАЗЫ ДАННЫХ

Таблица CATEGORY

Таблица категорий

Поле	**Описание**	**Тип значения**	**Значение по умолчанию**	**Свойство поля**
id	Уникальный идентификатор записи в таблице	bigint		pk, serial
categoryName	Наименование категории	varchar(50)		

Таблица USERS

Таблица пользователей

Поле	**Описание**	**Тип значения**	**Значение по умолчанию**	**Свойство поля**
id	Уникальный идентификатор записи в таблице	bigint		pk, serial
login	Логин пользователя	varchar(50)		
passwordHash	Хэш пароля пользователя	varchar(50)		index
username	ФИО пользователя	varchar(50)		
avatar	Ссылка на аватар пользователя	varchar(255)		
profile	Описание профиля пользователя	varchar(255)		
cardDetails	Реквизиты карты пользователя	varchar(50)		
status	Статус профиля (удален/не удален)	Boolean		
userRole	Роль пользователя	varchar(50)		
createdAt	Дата и время создания записи	Timestamp without timezone(0)		

Таблица SUBSCRIBTION

Таблица связей пользователей в БД (подписки/подписчики)

Поле	Описание	Тип значения	Значение по умолчанию	Свойство поля
subscriberd	ID пользователя внешний ключ из таблицы User, поле id	bigint		fk, index
publisherid	ID пользователя внешний ключ из таблицы User, поле id	bigint		fk, index

Таблица ARTICLE

Таблица статей

Поле	Описание	Тип значения	Значение по умолчанию	Свойство поля
id	Уникальный идентификатор записи в таблице	bigint		pk, serial
createdBy	ID пользователя внешний ключ из таблицы Users, поле id	bigint		fk, index
categoryId	ID категории внешний ключ из таблицы category, поле id	int		
title	Наименование статьи	varchar(50)		
text	Текст статьи	text		
description	Описание статьи	text		nullable
created_at	Дата и время создания записи	Timestamp without timezone(0)		
updated_at	Дата и время изменения записи	t Timestamp without timezone(0)		nullable
draft	Статус статьи (черновик/опубликовано)	boolean		

Таблица SAVED_ARTICLE

Таблица сохраненных статей у пользователя

Поле	Описание	Тип значения	Значение по умолчанию	Свойство поля
userid	ID пользователя внешний ключ из таблицы User, поле id	bigint		fk, index
articleid	ID статьие внешний ключ из таблицы Article, поле id	bigint		fk, index

Таблица COMMENT

Таблица комментариев к статье

Поле	Описание	Тип значения	Значение по умолчанию	Свойство поля
id	Уникальный идентификатор записи в таблице	bigint		pk, serial
articleid	ID статьи внешний ключ из таблицы Article, поле id	bigint		fk, index
userid	ID пользователя внешний ключ из таблицы Users, поле id	bigint		fk, index
comment	Текст комментария	text		

Таблица REFRESH

Таблица refresh токенов

Поле	Описание	Тип значения	Значение по умолчанию	Свойство поля
id	Уникальный идентификатор записи в таблице	bigint		pk, serial
userid	ID пользователя внешний ключ из таблицы Users, поле id	bigint		fk, index
token	Refresh токен	varchar (200)		

expiryDate	Дата и время, когда рефреш токен перестанет быть действительным	varchar (200)		

Таблица REACTION

Таблица реакций пользователей на статью

Поле	**Описание**	**Тип значения**	**Значение по умолчанию**	**Свойство поля**
articleid	ID статьи внешний ключ из таблицы Article, поле id	bigint		fk, index
userid	ID пользователя внешний ключ из таблицы User, поле id	bigint		fk, index

ПРИЛОЖЕНИЕ 3
ОПИСАНИЕ REST API

Header: Authorization: "access_token"

Регистрация

POST /auth/signup

Запрос регистрации пользователя

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
login	string	обязательный	Логин пользователя
password	string	обязательный	Пароль пользователя
username	string	обязательный	ФИО пользователя
userRole	string	обязательный	Роль пользователя
cardDetails	string	обязательный	Реквизиты пользователя
profile	string	обязательный	Описание профиля пользователя

Пример: {

"login": "string",

"password": "string",

"username": "string",

"userRole": "string",

"cardDetails": "string",

"profile": "string"

}

Ответ на успешный запрос:

В теле ответа передается объект MessageResponse в JSON формате

Content-Type response header: application/json

Пример ответа:

Content-Type: application/json

```
{
  "message": "User registered successfully!"
}
```

Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
401	Invalid Code	Неверные данные в запросе
403	Access Denied	Для данного аккаунта доступ запрещен

Авторизация

POST /auth/signin

Запрос авторизации пользователя

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
login	string	обязательный	Логин пользователя
password	string	обязательный	Пароль пользователя

Пример: {

"login": "string",

"password": "string"

}

Ответ на успешный запрос:

В теле ответа передается объект AuthResponse в JSON формате

Content-Type response header: application/json

Пример ответа:

Content-Type: application/json


```
{
 "id": 1,
 "login": "test",
 "role": "USER",
 "accessToken": "1234567sdfghjk",
 "tokenType": "bearer"
}
```


Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
401	Invalid Code	Неверные данные в запросе
403	Access Denied	Для данного аккаунта доступ запрещен

Получение нового access токена

POST /auth/refresh

Запрос на получение обновленного access токена

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
refreshToken	string	обязательный	Токен

Пример: {

"refreshToken": "string",

}

Ответ на успешный запрос:

В теле ответа передается объект AuthResponse в JSON формате

Content-Type response header: application/json

Пример ответа:

Content-Type: application/json


```
{
  "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJsb2 ",
  "refreshToken": "36b5ff77-e0ab-4325-879d-742b0d44f655",
  "tokenType": "Bearer"
}
```


Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
401	Invalid Code	Неверные данные в запросе
403	Access Denied	Для данного аккаунта доступ запрещен

Статьи

GET /article

Получить все статьи

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов с параметрами статьи и автора:

Пример ответа:

Content-Type: application/json

```
[{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "AnnaKara",
    "username": "AnnaPutty",
    "avatar": "string",
    "profile": "хороший человек",
    "cardDetails": "234234234",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "Биология"
  },
  "title": "Исследования генома человека",
  "description": "string",
  "text": "Исследования генома человека очень интересная тема",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
},
{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "Andrey123",
    "username": "AndreyKirikov",
    "avatar": "string",
    "profile": "специалист по фронтенд разработке",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
```

```
    "categoryId": "string",
    "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
 ]}
```


GET /user/article

Получить статьи текущего пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов с параметрами статьи и автора:

Пример ответа:

Content-Type: application/json


```
 [{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "AnnaKara",
    "username": "AnnaPutty",
    "avatar": "string",
    "profile": "хороший человек",
    "cardDetails": "234234234",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "Биология"
  },
```

```
  "title": "Исследования генома человека",
  "description": "string",
  "text": "Исследования генома человека очень интересная тема",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
 },
 {
  "articleId": 0,
  "users": {
   "userId": 0,
   "login": "Andrey123",
   "username": "AndreyKirikov",
   "avatar": "string",
   "profile": "специалист по фронтенд разработке",
   "cardDetails": "string",
   "status": true,
   "userRole": "string",
   "createdAt": "string",
   "statusSubscribtion": true
  },
  "category": {
   "categoryId": "string",
   "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
 ]}
```


GET /user/article/{articleId}

Получить статью по идентификатору статьи {articleID}

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
articleD	long	обязательный	ID статьи

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект Article с параметрами User

Пример ответа:

Content-Type: application/json


```
{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "Andrey123",
    "username": "AndreyKirikov",
    "avatar": "string",
    "profile": "специалист по фронтенд разработке",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
 }
```

GET /user/article/catgory/{categoryId}

Получить массив обьектов Article, сгруппированных по идентификатору категории {categoryID}

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
categoryID	long	обязательный	ID категории

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Article с параметрами User

Пример ответа:

Content-Type: application/json


```
[{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
     "avatar": "string",
     "profile": "специалист по фронтенд разработке",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "category": {
     "categoryId": "string",
     "categoryName": "DOM"
   },
   "title": "статья о DOM",
   "description": "string",
   "text": "Рассмотрим элементы DOM",
   "likes": 0,
   "statusLike": true,
   "createdAt": "string",
   "updatedAt": "string",
   "status": true
```

```
 }]
```

GET /user/subscribtion/article

Получить статью по подпискам пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Article с параметрами User

Пример ответа:

Content-Type: application/json

```
[{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
     "avatar": "string",
     "profile": "специалист по фронтенд разработке",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "category": {
     "categoryId": "string",
     "categoryName": "DOM"
   },
   "title": "статья о DOM",
   "description": "string",
   "text": "Рассмотрим элементы DOM",
   "likes": 0,
   "statusLike": true,
   "createdAt": "string",
   "updatedAt": "string",
   "status": true
 }]
```

GET /user/saved/articles

Получить сохраненные статьи пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Article с параметрами User

Пример ответа:

Content-Type: application/json


```
[{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
     "avatar": "string",
     "profile": "специалист по фронтенд разработке",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "category": {
     "categoryId": "string",
     "categoryName": "DOM"
   },
   "title": "статья о DOM",
   "description": "string",
   "text": "Рассмотрим элементы DOM",
   "likes": 0,
   "statusLike": true,
   "createdAt": "string",
   "updatedAt": "string",
   "status": true
 }]
```


GET /user/article/draft

Получить черновики статей пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Article с параметрами User

Пример ответа:

Content-Type: application/json


```
[{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
     "avatar": "string",
     "profile": "специалист по фронтенд разработке",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "category": {
     "categoryId": "string",
     "categoryName": "DOM"
   },
   "title": "статья о DOM",
   "description": "string",
   "text": "Рассмотрим элементы DOM",
   "likes": 0,
   "statusLike": true,
   "createdAt": "string",
   "updatedAt": "string",
   "status": true
 }]
```


POST /user/article

Создать статью

Данные для создания статьи передаются в теле запроса в формате JSON

Content-Type request header: application/json

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
categoryID	integer	обязательный	ID категории
title	string	обязательный	Заголовок статьи
description	string	обязательный	Описание статьи
text	string	обязательный	Текст статьи
status	boolean	обязательный	Статус статьи

Пример: {

"categoryId": "string",

"title": "string",

"description": "string",

"text": "string",

"status": true}

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект Article

Пример ответа:

Content-Type: application/json


```
{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "Andrey123",
    "username": "AndreyKirikov",
    "avatar": "string",
    "profile": "специалист по фронтенд разработке",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
```

```
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
}
```


POST /user/save/article/{articleID}

Сохранить/удалить статус статьи в закладках

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
articleID	integer	обязательный	ID статьи

Параметры Query:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
status	boolean		**статус статьи в закладках**

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект Article

Пример ответа:

Content-Type: application/json


```
{
```

```
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "Andrey123",
    "username": "AndreyKirikov",
    "avatar": "string",
    "profile": "специалист по фронтенд разработке",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
}
```


GET /article/search

Получить массив обьектов Article, в которых содержится строка, передаваемая в Query параметрах

Параметры Query:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
line	string	обязательный	Подстрока для поиска

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Article с параметрами User

Пример ответа:

Content-Type: application/json


```
[{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
     "avatar": "string",
     "profile": "специалист по фронтенд разработке",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "category": {
     "categoryId": "string",
     "categoryName": "DOM"
   },
   "title": "статья о DOM",
   "description": "string",
   "text": "Рассмотрим элементы DOM",
   "likes": 0,
   "statusLike": true,
   "createdAt": "string",
   "updatedAt": "string",
   "status": true
 }]
```


Реакции

GET /user/article/reaction/count/{articleID}

Получить количество реакций на статье с идентификатором articleID

Параметры Path:

Параметр запроса	**Тип параметра**	**Обязательный параметр**	**Описание**
articleID	long	обязательный	ID статьи

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается число реакций на статью с идентификатором articleID

Пример ответа:

Content-Type: application/json

```
{
 “reaction”: 25
}
```

POST /user/article/{articleID}/reaction

Поставить/удалить реакцию на статью

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
articleID	integer	обязательный	ID статьи

Параметры Query:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
status	boolean		**Статус реакции**

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект Article

Пример ответа:

Content-Type: application/json

```
{
   "articleId": 0,
   "users": {
     "userId": 0,
     "login": "Andrey123",
     "username": "AndreyKirikov",
```

```
    "avatar": "string",
    "profile": "специалист по фронтенд разработке",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "category": {
    "categoryId": "string",
    "categoryName": "DOM"
  },
  "title": "статья о DOM",
  "description": "string",
  "text": "Рассмотрим элементы DOM",
  "likes": 0,
  "statusLike": true,
  "createdAt": "string",
  "updatedAt": "string",
  "status": true
 }
```


Категории

GET /article/category

Получить список существующих объектов Category для статей.

Каждый объект содержит свойства категории:

1. id
2. categoryName

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Category.

Пример ответа:

Content-Type: application/json


```
[ {
"categoryId": 1,
```

```
  "name": "Биология"
 },
 {
  "categoryId": 2,
  "name": "Математика"
} ]
```


Пользователи

GET /user/profile

Получить профиль текущего пользователя.

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект User

Пример ответа:
Content-Type: application/json


```
 {
 "userId": 0,
 "login": "Kate123",
 "username": "KaravaevaKate",
 "avatar": "string",
 "profile": "Студент",
 "cardDetails": "23423424",
 "status": true,
 "userRole": "string",
 "createdAt": "string",
 "statusSubscribtion": true
}
```


GET /user/profile/{userID}

Получение профиль пользователя по ID пользователя.

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
userID	integer	обязательный	ID пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект User

Пример ответа:

Content-Type: application/json


```
{
 "userId": 0,
 "login": "Kate123",
 "username": "KaravaevaKate",
 "avatar": "string",
 "profile": "Студент",
 "cardDetails": "23423424",
 "status": true,
 "userRole": "string",
 "createdAt": "string",
 "statusSubscribtion": true
}
```


PUT /user/profile

Изменение данных текущего объекта User

Данные для изменения профиля пользователя передаются в теле запроса в формате JSON

Content-Type request header: application/json

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
username	string		ФИО пользователя
avatar	string		Ссылка на аватар пользователя
profile	string		Описание профиля пользователя
cardDetails	string		Реквизиты карты пользователя

Пример: {

"username": "string",

"avatar": "string",

"profile": "string",

"cardDetails": "string"}

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается обновленный объект ProfileResponse

Пример ответа:

Content-Type: application/json


```
 {
 "userId": 0,
 "login": "Kate23",
 "username": "AnnaKaravaeva",
 "avatar": "string",
 "profile": "школьник",
 "cardDetails": "32432423423432324",
 "status": true,
 "userRole": "string",
 "createdAt": "string",
 "statusSubscribtion": true
}
```

Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
400	Missing Required Information	Неверные данные в запросе
404	User Not Found	Акаунт не найден

Подписки/подписчики пользователя

GET /user/subscribtion

Получение подписок текущего пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив обьектов User

Пример ответа:

Content-Type: application/json


```
[
 {
  "userId": 0,
  "login": "string",
  "username": "string",
  "avatar": "string",
  "profile": "string",
  "cardDetails": "string",
  "status": true,
  "userRole": "string",
  "createdAt": "string",
  "statusSubscribtion": true
 },
 {
  "userId": 0,
  "login": "string",
  "username": "string",
  "avatar": "string",
  "profile": "string",
  "cardDetails": "string",
```

```
  "status": true,
  "userRole": "string",
  "createdAt": "string",
  "statusSubscribtion": true
]}
```

Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
401	Unauthorized	Для запроса требуется авторизация

GET /user/subscribtion

Получение подписчиков текущего пользователя

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив обьектов User

Пример ответа:

Content-Type: application/json

```
 [
 {
  "userId": 0,
  "login": "string",
  "username": "string",
  "avatar": "string",
  "profile": "string",
  "cardDetails": "string",
  "status": true,
  "userRole": "string",
  "createdAt": "string",
  "statusSubscribtion": true
 },
 {
  "userId": 0,
  "login": "string",
```

```
    "username": "string",
    "avatar": "string",
    "profile": "string",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
]}
```


Коды ответа на запрос:

Код ответа	Сообщение	Описание
200	OK	Запрос обработан успешно
401	Unauthorized	Для запроса требуется авторизация

POST /user/subscribtion/{userID}

Изменить статус подписки на пользователя

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
userID	integer	обязательный	ID пользователя

Параметры Query:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
status	boolean		**Статус подписки**

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект User

Пример ответа:

Content-Type: application/json

```
 {
 "userId": 0,
 "login": "Kate23",
 "username": "AnnaKaravaeva",
 "avatar": "string",
 "profile": "школьник",
 "cardDetails": "32432423423432324",
 "status": true,
 "userRole": "string",
 "createdAt": "string",
 "statusSubscribtion": true
}
```


Комментарии

POST /user/article/{articleID}/comment

Создать комментарий Comment к статье с идентификатором articleID. Для создания обьекта параметры комментария передаются в теле запроса в формате JSON :

1. Текст комментария: comment

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
articleID	integer	обязательный	ID статьи

Данные для создания комментария передаются в теле запроса в формате JSON

Content-Type request header: application/json

Request body

Параметры тела запроса формата JSON

Параметр запроса	Тип параметра	Обязательный параметр	Описание
comment	string	обязательный	Текст комментария

Пример: { "comment": "string"}

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается объект Comment

Пример ответа:

Content-Type: application/json


```
  {
 "articleId": 0,
   "users": {
     "userId": 0,
     "login": "string",
     "username": "string",
     "avatar": "string",
     "profile": "string",
     "cardDetails": "string",
     "status": true,
     "userRole": "string",
     "createdAt": "string",
     "statusSubscribtion": true
   },
   "comment": "comment1"
 }
```


GET /user/article/{articleID}/comment

Получить комментарии к статье с идентификатором articleID

Параметры Path:

Параметр запроса	Тип параметра	Обязательный параметр	Описание
articleID	long	обязательный	ID статьи

Ответ на успешный запрос:

Content-Type response header: application/json

В теле ответа в формате JSON передается массив объектов Comment

Пример ответа:

Content-Type: application/json


```
[{
 "articleId": 0,
   "users": {
     "userId": 0,
```

```
    "login": "string",
    "username": "string",
    "avatar": "string",
    "profile": "string",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "comment": "comment1"
},
{
  "articleId": 0,
  "users": {
    "userId": 0,
    "login": "string",
    "username": "string",
    "avatar": "string",
    "profile": "string",
    "cardDetails": "string",
    "status": true,
    "userRole": "string",
    "createdAt": "string",
    "statusSubscribtion": true
  },
  "comment": "comment2"
}]
```

ПРИЛОЖЕНИЕ 4
ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 4.1

Описание и функциональное назначение классов серверной части

Класс	Назначение
ArticleController	Контроллер для управления информации о статье.
UserController	Контроллер для управления аккаунтом пользователя.
AuthController	Контроллер для управления аутентификацией/авторизацией пользователя.
ArticleService	Сервис для работы со статьей.
AuthService	Сервис для работы с аутентификацией/авторизацией пользователя.
UserService	Сервис для работы с аккаунтом пользователя.
RefreshTokenService	Сервис для работы с refresh-токеном.
ArticleRepository	Интерфейс позволяет взаимодействовать с моделями статей.
CategoryRepository	Интерфейс позволяет взаимодействовать с моделями категорий.
CommentRepository	Интерфейс позволяет взаимодействовать с моделями комментариев.
RefreshRepository	Интерфейс позволяет взаимодействовать с моделями refresh-токенов.
UserRepository	Интерфейс позволяет взаимодействовать с моделями пользователей.
ArticleEntity	Модель статьи в БД
UserEntity	Модель пользователя в БД
CategoryEntity	Модель категории в БД
CommentEntity	Модель комментария в БД
RefreshToken	Модель refresh-токена в БД
ArticleRequest	Модель тела запроса статьи для контроллера
AuthRequest	Модель тела запроса авторизации для контроллера
ReactionRequest	Модель тела запроса реакции для контроллера
SignUpRequest	Модель тела запроса регистрации для контроллера
TokenRefreshRequest	Модель тела запроса refresh токена для контроллера
ArticleResponse	Модель тела ответа на запрос статьи для контроллера
JwtResponse	Модель тела ответа на запрос jwt токенов для контроллера
CommentResponse	Модель тела ответа на запрос комментария для контроллера
MessageResponse	Модель тела ответа на запрос сообщения для контроллера
ProfileResponse	Модель тела ответа на запрос профиля для контроллера
TokenRefreshResponse	Модель тела ответа на запрос refresh токена для контроллера
AppException	Главный класс exception приложения.
BusinessException	Класс ошибок бизнес-логики.
TechnicalException	Класс технических ошибок.
TokenRefreshException	Класс ошибок refresh токена.
ExceptionMapper	Класс, обрабатывающий ошибки.
ErrorMessage	Класс, представляющий модель сообщения об ошибке.
DocumentationConfig	Класс конфигурации для документации API.

Продолжение таблицы 4.1

AuthTokenFilter	Класс-фильтр, который отвечает за проверку наличия JWT-токена в запросе пользователя и его валидность.
WebSecurityConfig	Класс для конфигурации Spring Security в приложении
AuthEntryPointJwt	Класс, который отвечает за обработку ошибок авторизации
JwtUtils	Класс, который отвечает за генерацию и проверку JWT-токенов
UserDetailsImpl	Класс для представления информации о пользователе, используемой для аутентификации и авторизации в системе
UserDetailsServiceImpl	Класс для предоставления реализации сервиса, который загружает данные пользователя из базы данных и передает их в объект UserDetailsImpl
GuidepediaApplication	Главный запускаемый класс.

ПРИЛОЖЕНИЕ 5
ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ МЕТОДОВ И СВОЙСТВ

Таблица 5.1

Описание методов и свойств класса UserController.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
getProfile	public	Метод	User: userDetailsImpl	Метод для получения информации о профиле пользователя.
updateProfile	public	Метод	profileResponse: ProfileResponse, User: userDetailsImpl	Метод для обновления информации о профиле пользователя.
changeSaveArticle	public	Метод	articleID: Long, status: Boolean User: userDetailsImpl	Метод для изменения статуса закладки статьи.
getSavedArticles	public	Метод	User: userDetailsImpl	Метод для получения сохраненных статей пользователя.
changeStatusSubscribtion	public	Метод	User: userDetailsImpl, userID: Long, status: Boolean	Метод для изменения статуса подписки пользователя на автора.
getSubscribers	public	Метод	User: userDetailsImpl	Метод для получения подписчиков пользователя.
getSubscribtions	public	Метод	profileEntity: ProfileEntity User: userDetailsImpl	Метод для получения подписок пользователя.
getUserSubscribtions	public	Метод	User: userDetailsImpl , userID: Long	Метод для получения подписчиков пользователя по ID.
getUserSubscribers	public	Метод	User: userDetailsImpl, userID: Long	Метод для получения подписок пользователя по ID.

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userService	private	UserService	get, set	Сервис для работы с пользователями.
userRepository	private	userReporitory	get, set	Репозиторий пользователей.

Таблица 5.2

Описание методов и свойств класса UserService

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
getProfile	public	Метод	User: userDetailsImpl	Метод для получения информации о профиле пользователя.
updateProfile	public	Метод	profileResponse: ProfileResponse, User: userDetailsImpl	Метод для обновления информации о профиле пользователя.
changeSaveArticle	public	Метод	articleID: Long, status: Boolean User: userDetailsImpl	Метод для изменения статуса закладки статьи.
getSavedArticles	public	Метод	User: userDetailsImpl	Метод для получения сохраненных статей пользователя.
changeStatusSubscribtion	public	Метод	User: userDetailsImpl, userID: Long, status: Boolean	Метод для изменения статуса подписки пользователя на автора.
getSubscribers	public	Метод	User: userDetailsImpl	Метод для получения подписчиков пользователя.
getSubscribtions	public	Метод	profileEntity: ProfileEntity User: userDetailsImpl	Метод для получения подписок пользователя.
getUserSubscribtions	public	Метод	User: userDetailsImpl , userID: Long	Метод для получения

				подписчиков пользователя по ID.
getUserSubscribers	public	Метод	User: userDetailsImpl, userID: Long	Метод для получения подписок пользователя по ID.
createUserCourse	public	Метод	courseID, roleID: Integer userID: Long	Метод для создания добавления пользователя в таблицу UserCourse.
registerUser	public	Метод		Метод для отправки авторизованного пользователя.
updateProfile	public	Метод	profileEntity: ProfileEntity	Метод для обновления профиля пользователя.
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
userRepository	private	UserRepository	get, set	Репозиторий пользователей.
articleRepository	private	ArticleRepository	get, set	Репозиторий статей.

Таблица 5.3

Описание методов и свойств интерфейса UserRepository.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findByLogin	public	Метод	login: String	Метод для получения пользователя по его login из БД.
existsByLogin	public	Метод	login: String	Метод для проверки нахождения пользователя по его Login из БД.

Таблица 5.4

Описание методов и свойств класса UserEntity.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID пользователя
login	private	String	get, set	Логин пользователя
passwordHash	private	String	get, set	Хэш пароля пользователя
username	private	String	get, set	ФИО пользователя
avatar	private	String	get, set	Аватар пользователя
profile	private	String	get, set	Описание профиля пользователя
cardDetails	private	String	get, set	Реквизиты карты пользователя
status	private	Boolean	get, set	Статус аккаунта пользователя
userRole	private	String	get, set	Роль пользователя
createdAt	private	LocalDateTime	get, set	Дата создания записи

Таблица 5.5

Описание методов и свойств класса AuthController.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
authenticateUser	public	Метод	authRequest: AuthRequest	Метод для авторизации пользователя
registerUser	public	Метод	signUpRequest: SignUpRequest	Метод для регистрации пользователя
refreshtoken	public	Метод	tokenRefreshRequest: TokenRefreshRequest	Метод для получения обновленного access токена
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
authService	private	AuthService	get, set	Сервис для работы с авторизацией
refreshTokenService	private	refreshTokenService	get, set	Сервис для работы с refresh токенами
jwtUtils	private	JwtUtils	get, set	Сервис для работы с jwt токенами

Таблица 5.6

Описание методов и свойств класса AuthService.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
authenticateUser	public	Метод	authRequest: AuthRequest	Метод для авторизации пользователя
registerUser	public	Метод	signUpRequest: SignUpRequest	Метод для регистрации пользователя
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userRepository	private	UserRepository	get, set	Репозиторий пользователей
refreshTokenService	private	refreshTokenService	get, set	Сервис для работы с refresh токенами

Таблица 5.7

Описание методов и свойств класса AuthRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
login	private	String	get, set	Логин пользователя
password	private	String	get, set	Пароль пользователя

Таблица 5.8

Описание методов и свойств класса SignUpRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
login	private	String	get, set	Логин пользователя
password	private	String	get, set	Пароль пользователя
username	private	String	get, set	ФИО пользователя
profile	private	String	get, set	Описание профиля пользователя
cardDetails	private	String	get, set	Реквизиты карты пользователя
userRole	private	String	get, set	Роль пользователя

Таблица 5.9

Описание методов и свойств класса RefreshTokenService.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findByToken	public	Метод	token: String	Метод для нахождения обьекта RefreshToken
createRefreshToken	public	Метод	userId: Long	Метод для создания refresh токена
verifyExpiration	public	Метод	refreshToken: RefreshToken	Метод для проверки времени токена
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userRepository	private	UserRepository	get, set	Репозиторий пользователей
refreshTokenRepository	private	RefreshTokenrepository	get, set	Репозиторий refresh токенов

Таблица 5.10

Описание методов и свойств интерфейса RefreshRepository.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findByToken	public	Метод	token: String	Метод для получения обьекта по его токену из БД.
deleteByUser	public	Метод	user: UserEntity	Метод для удаления обьекта по пользователю из БД.

Таблица 5.11

Описание методов и свойств класса TokenRefreshRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
token	private	String	get, set	Токен пользователя

Таблица 5.12

Описание методов и свойств класса SignUpRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
login	private	String	get, set	Логин пользователя
password	private	String	get, set	Пароль пользователя
username	private	String	get, set	ФИО пользователя
profile	private	String	get, set	Описание профиля пользователя
cardDetails	private	String	get, set	Реквизиты карты пользователя
userRole	private	String	get, set	Роль пользователя

Таблица 5.12

Описание методов и свойств класса TokenRefreshResponse.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
accessToken	private	String	get, set	Access Токен пользователя
refreshToken	private	String	get, set	Refresh Токен пользователя
type	private	String	get, set	Тип токена пользователя

Таблица 5.13

Описание методов и свойств класса JwtResponse.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	Логин пользователя
login	private	String	get, set	Логин пользователя
username	private	String	get, set	ФИО пользователя
token	private	String	get, set	Access Токен пользователя
type	private	String	get, set	Тип токена пользователя
refreshToken	private	String	get, set	Refresh Токен пользователя
userRole	private	String	get, set	Роль пользователя

Таблица 5.14

Описание методов и свойств класса ArticleController.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
createArticle	public	Метод	User: userDetailsImpl, articleRequest: ArticleRequest	Метод для создания статьи
getArticleById	public	Метод	User: userDetailsImpl, articleID: Long	Метод для получения статьи по ее ID
getArticleByCategoryId	public	Метод	User: userDetailsImpl, categoryID: Long	Метод для получения статьи по ID категории
getArticle	public	Метод	User: userDetailsImpl	Метод для получения статей пользователей
getArticleDrafts	private	Метод	User: userDetailsImpl	Метод для получения черновиков статей пользователей
getAllArticle	private	Метод	User: userDetailsImpl	Метод для получения всех статей
getAllCategories	private	Метод		Метод для получения всех категорий статей
createReaction	private	Метод	User: userDetailsImpl	Метод для создания реакции на статью
getSubscribtionArticles	private	Метод	User: userDetailsImpl	Метод для получения всех статей по подписке пользователя
createComment	private	Метод	User: userDetailsImpl, articleID: Long	Метод для создания комментария на статью
getAllComments	private	Метод	User: userDetailsImpl, articleID: Long	Метод для получения всех комментариев статьи по ID
getSearchArticle	private	Метод	User: userDetailsImpl, line: String	Метод для получения всех статей по поиску подстроки в заголовке статьи

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
logger	private	Logger	get, set	Логгер.
articleService	private	ArticleService	get, set	Сервис для работы со статьями

Таблица 5.15

Описание методов и свойств класса ArticleService.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
createArticle	public	Метод	User: userDetailsImpl, articleRequest: ArticleRequest	Метод для создания статьи и сохранения ее в БД
getArticleById	public	Метод	User: userDetailsImpl, articleID: Long	Метод для получения статьи по ее ID
getArticleByCategoryId	public	Метод	User: userDetailsImpl, categoryID: Long	Метод для получения статьи по ID категории
getArticle	public	Метод	User: userDetailsImpl	Метод для получения статей пользователей
getArticleDrafts	private	Метод	User: userDetailsImpl	Метод для получения черновиков статей пользователей
getAllArticle	private	Метод	User: userDetailsImpl	Метод для получения всех статей
getAllCategories	private	Метод		Метод для получения всех категорий статей
createReaction	private	Метод	User: userDetailsImpl	Метод для создания реакции на статью и сохранения ее в БД
getSubscribtionArticles	private	Метод	User: userDetailsImpl	Метод для получения всех статей по подписке пользователя
createComment	private	Метод	User: userDetailsImpl, articleID: Long	Метод для создания комментария на статью и сохранения его в БД

getAllComments	private	Метод	User: userDetailsImpl, articleID: Long	Метод для получения всех комментариев статьи по ID
getSearchArticle	private	Метод	User: userDetailsImpl, line: String	Метод для получения всех статей по поиску подстроки в заголовке статьи

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userRepository	private	UserRepository	get, set	Репозиторий для работы с пользователями
articleRepository	private	ArticleRepository	get, set	Репозиторий для работы со статьями

Таблица 5.16

Описание методов и свойств интерфейса ArticleRepository.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findAllByCreatedBy	public	Метод	createdBy: UserEntity	Метод для нахождения всех статей по ID пользователя
findAllByCreatedByAndDraft	public	Метод	createdBy: UserEntity, draft: Boolean	Метод для нахождения всех статей по ID пользователя и статуса статьи
findAllByCategoryId	public	Метод	categoryID: Integer	Метод для нахождения всех статей по ID категории
findByTitleContainingIgnoreCase	public	Метод	title: String	Метод для нахождения всех статей по поиску подстроки в заголовке

Таблица 5.17

Описание методов и свойств класса ArticleEntity.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID статьи
createdBy	private	Long	get, set	ID автора
category	private	CategoryEntity	get, set	Объект категории статьи
title	private	String	get, set	Заголовок статьи
text	private	String	get, set	Текст статьи
description	private	String	get, set	Описание статьи
draft	private	Boolean	get, set	Статус статьи
updatedAt	private	LocalDateTime	get, set	Дата обновления записи
createdAt	private	LocalDateTime	get, set	Дата создания записи

Таблица 5.18

Описание методов и свойств класса ArticleRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
categoryId	private	Long	get, set	ID категории
title	private	String	get, set	Заголовок статьи
text	private	String	get, set	Текст статьи
description	private	String	get, set	Описание статьи
draft	private	Boolean	get, set	Статус статьи

Таблица 5.19

Описание методов и свойств класса ArticleResponse.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID статьи
createdBy	private	ProfileResponse	get, set	Объект профиля пользователя
category	private	CategoryEntity	get, set	Объект категории статьи
title	private	String	get, set	Заголовок статьи
text	private	String	get, set	Текст статьи
description	private	String	get, set	Описание статьи
draft	private	Boolean	get, set	Статус статьи
updatedAt	private	LocalDateTime	get, set	Дата обновления записи

createdAt	private	LocalDateTime	get, set	Дата создания записи
statusSave	private	Boolean	get, set	Статус закладки статьи
likes	private	Boolean	get, set	Количество реакций на статью
statusLike	private	Boolean	get, set	Статус лайка текущего пользователя на статью

В классе CategoryRepository.java нет вручную написанных методов и свойств.

Таблица 5.20

Описание методов и свойств интерфейса CommentRepository.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
findByArticle	public	Метод	article: ArticleEntity	Метод для нахождения всех комментариев по статье

Таблица 5.21

Описание методов и свойств класса CategoryEntity.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID категории
categoryName	private	String	get, set	Название категории

Таблица 5.22

Описание методов и свойств класса CommentEntity.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID комментария
article	private	ArticleEntity	get, set	Обьект статьи
user	private	UserEntity	get, set	Обьект пользователя
comment	private	String	get, set	Текст комментария

Таблица 5.23

Описание методов и свойств класса CommentResponse.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
id	private	Long	get, set	ID комментария
article	private	ArticleResponse	get, set	Обьект статьи
user	private	ProfileResponse	get, set	Обьект пользователя
comment	private	String	get, set	Текст комментария

Таблица 5.24

Описание методов и свойств класса ReactionRequest.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
articleId	private	Long	get, set	ID статьи
reaction	private	Boolean	get, set	Статус реакции

Таблица 5.25

Описание методов и свойств класса MessageResponse.java

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
message	private	String	get, set	Текст сообщения

Таблица 5.26

Описание методов и свойств класса AppException.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
AppException	public	Конструктор	message: String	Главный класс exception в приложении. Наследуется от RuntimeException

Таблица 5.27

Описание методов и свойств класса BusinessException.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
BusinessException	public	Конструктор	message: String	Класс-наследник от appException для ошибок бизнес-логики

Таблица 5.28

Описание методов и свойств класса TechnicalException.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
TecnicalException	public	Конструктор	message: String	Класс-наследник от appException для ошибок сервера(технических)

Таблица 5.29

Описание методов и свойств класса TokenRefreshException.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
TokenRefreshException	public	Конструктор	message: String	Класс-наследник от appException для ошибок токена

Таблица 5.30

Описание методов и свойств класса ErrorMessage.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
ErrorMessage	public	Конструктор	String error	Класс для хранения сообщения об ошибке
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
message	private	String	get, set	Сообщение ошибки
description	private	String	get, set	Описание ошибки
statusCode	private	Integer	get, set	Код статуса ошибки

Таблица 5.31

Описание методов и свойств класса ExceptionMapper.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
handleBusinessException	protected	Метод	e: BusinessException	Метод для обработки BusinessException
handleTechnicalException	protected	Метод	e: TechnicalException	Метод для обработки TechnicalException
handleTokenRefreshException	protected	Метод	e: TokenRefreshException	Метод для обработки TokenRefreshException
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
log	private	Logger	get, set	Логгер

Таблица 5.32

Описание методов и свойств класса DocumentConfig.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
openApiDoc	public	Метод		Метод для определения настройки openApi documentation

Таблица 5.33

Описание методов и свойств класса AuthTokenFilter.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
doFilterInternal	protected	Метод	HttpServletRequest :request, HttpServletResponse :response, FilterChain : filterChain	Метод для декодирования токена и получения данных о пользователе
parseJwt	private	Метод	HttpServletRequest :request	Метод для парсинга Jwt токена

Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userDetailsServiceImpl	private	UserDetailsServiceImpl	get, set	Сервис для работы с пользователями UserDetails
jwtUtils	private	JwtUtils	get, set	Сервис для работы с токенами

Таблица 5.34

Описание методов и свойств класса WebSecurityConfig.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
filterChain	public	Метод	http: HttpSecurity	Метод для настройки конфигурации spring Security
passwordEncoder	public	Метод		Метод для шифрования пароля
authenticationProvider	public	Метод		Метод для установки данных пользователя
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userDetailsService	private	UserDetailsServiceImpl	get, set	Сервис для работы с пользователем UserDetailsImpl

Таблица 5.35

Описание методов и свойств класса AuthEntryPointJwt.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
commence	public	Метод	HttpServletRequest :request, HttpServletResponse :response, AuthenticationException authException	Метод для обработки ошибок авторизации

Таблица 5.36

Описание методов и свойств класса JwtUtils.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
generateJwtToken	public	Метод	Authentication: authentication	Метод для генерации jwt токена
generateTokenFromLogin	public	Метод	Login: String	Метод для генерации токена из логина пользователя
getLoginFromJwtToken	public	Метод	Token: String	Метод для получения логина из jwt токена
validateJwtToken	public	Метод	authToken: String	Метод для валидации jwt токена
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
jwtSecret	private	String	get, set	Секрет для создания токена
jwtExpirationMs	private	int	get, set	Время существования токена

Таблица 5.37

Описание методов и свойств класса UserDetailsServiceImpl.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
loadUserByUsername	public	Метод	Login: String	Метод для получения пользователя по его логину
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
userRepository	private	UserRepository	get, set	Репозиторий пользователей

Таблица 5.38

Описание методов и свойств класса UserDetailsImpl.java

Методы				
Имя	Модификатор доступа	Тип	Аргументы	Назначение
UserDetailsImpl	public	Конструктор	username, login, password, role: String id: Long	Класс для описания информации о пользователе и его ролях
build	public	Метод	User: UserEntity	Метод для создания объекта UserDetailsImpl
Свойства				
Имя	Модификатор доступа	Тип	Доступ	Назначение
username	private	String	get, set	ФИО пользователя
password	private	String	get, set	Пароль пользователя
role	private	String	get, set	Роль пользователя
login	private	String	get, set	Логин пользователя
id	private	Long	get, set	ID пользователя

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулирован					