Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Кафедра «школа бакалавриата (школа)»

Оценка работы_____
Руководитель от УрФУ Корнякова Е. М.

Тема задания на практику:
«**Проектирование и разработка ПО**»

ОТЧЕТ
Вид практики Производственная практика
Тип практики Производственная практика, преддипломная

Руководитель практики от предприятия (организации) Неклюдов Д. А.
ФИО руководителя Подпись

Студент Шутов А. И.
ФИО студента Подпись

Специальность (направление подготовки) 09.03.03 Прикладная информатика

Группа РИ-400019

Екатеринбург 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Процесс создания и оценки проектов по треку Gamedev в рамках проектного практикума (ПП) в институте ИРИТ-РТФ имеет несколько проблем. Эксперты сталкиваются с ограниченными возможностями для предварительного просмотра и оценки проектов заранее, полагаясь исключительно на презентации команд во время защиты. Также студентам тяжело искать подходящих кандидатов в команду для ПП на треке Gamedev, так как нет возможности посмотреть портфолио, в котором отображается информация о том, в каких проектах кандидат работал.

Создание витрины игровых проектов с функцией портфолио для студентов в рамках ПП значительно поможет и экспертам, и обучающимся.

Актуальность этой работы обусловлена отсутствием комплексного решения, специально адаптированного к требованиям трека Gamedev в рамках ПП. Уже есть системы, работающие в рамках ПП, такие как Teamproject и Прокомпетенции, однако они предлагают ограниченную функциональность, закрыты для внешних пользователей и не позволяют напрямую просматривать и оценивать игровые проекты в браузере. Потому, необходимо спроектировать и разработать такой веб-сервис, в котором студент должен иметь возможность публиковать игры, а любой другой пользователь (в том числе, и эксперт) мог бы их посмотреть.

Объектом данного исследования является бизнес-процесс ПП на треке Gamedev. Предметом исследования является оптимизация и совершенствование этого процесса путем внедрения витрины проектов для публикации и архивирования интерактивных проектов.

В рамках преддипломной практики будет проводиться работа над разработкой серверной части веб-сервиса и написанием его API. Серверное приложение будет получать запросы от внешних пользователей, создавать записи в базе данных, передавать загружаемые файлы в объектное хранилище и возвращать ответ пользователям. Также приложение должно поддерживать

разграничение доступа пользователей, для этого следует реализовать ролевую модель доступа [1].

Основной целью работы является проектирование и разработка серверной части веб-сервиса. Для достижения этой цели были определены следующие ключевые задачи:

– Разработать архитектуру базы данных для веб-сервиса;

– Написать API серверного приложения.

1 Описание работы

1.1 Используемые инструменты

Разработка API серверного приложения и его отладка требуют тщательно подобранного набора инструментов разработки программного обеспечения (ПО). Так, для разработки API нужно выбрать фреймворк, язык, систему управления базой данных (СУБД). Потому, сразу во время разработки архитектуры был подобраны инструменты для разработки серверной части. Список инструментов представлен на рисунке 1.

Рисунок 1 – Используемые инструменты

Для создания схем архитектуры базы данных использовался сервис Miro, так как он предоставляет удобный и простой пользовательский интерфейс и обладает большим количеством шаблонов для проектирования.

Для серверной части сервиса в качестве основной среды выполнения и фреймворка разработки был выбран Nest.js. «Nest.js – фреймворк для создания эффективных, масштабируемых Node.js серверных приложений. Он использует прогрессивный JavaScript, построен на TypeScript и полностью поддерживает его и сочетает в себе элементы объектно-ориентированного программирования (ООП), функционального программирования (ФП) и функционально-реактивного программирования (ФРП)» [2]. Nest.js прямо из

коробки предоставляет модульную архитектуру на базе фреймворка Express, что дает возможность создавать хорошо тестируемые, масштабируемые, слабо связанные и легко обслуживаемые серверные приложения.

Для абстрагирования от конкретной СУБД в серверном приложении была внедрена технология Object-Relational Mapping (ORM). Данная технология связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» [3]. В качестве реализации ORM была выбрана TypeORM, а в качестве СУБД - PostgreSQL, так как эти технологии легко интегрируются и поддерживаются в Nest.js.

Для хранения проектов студентов был развернут отдельный Docker контейнер и на нем установлено объектное хранилище данных MinIO, предоставляющее быстрый доступ к файлам и асинхронное скачивание данных.

1.2 Календарный план

На срок проведения преддипломной практики был составлен календарный план. В нем отображены сроки начала и окончания задач, связанных с проектированием и разработкой. Календарный план приведен в таблице 1.

Таблица 1 – Календарный план

Задача	Срок
Разработать архитектуру базы данных для веб-сервиса	15.02.2024 – 04.03.2024
Написать API серверного приложения	04.03.2024 – 05.04.2024

Как видно из календарного плана, самой трудоемкой является задача создания API серверной части, так как API является ключевой частью всего сервиса в целом.

2 Результаты практики

2.1 Разработка архитектуры базы данных

Для начала работы в рамках практики, было предпринято проектирование схем работы модулей серверной части и набор сущностей в базе данных. Для более наглядного визуального представления, был выбран инструмент Miro, в котором были созданы подробные модели необходимых объектов и их взаимосвязей (рисунок А.1 приложения А).

Так, в модуле «Пользователи» находиться вся бизнес-логика по работе с пользователями, например создание, чтение, редактирование и удаление (CRUD). Также в этом модуле мы должны получать информацию о портфолио студента – проектах, в которых он принимал непосредственное участие.

В модуле «Авторизация» ведется работа с регистрацией и авторизацией пользователей. Для реализации такой функциональности на серверной части была применена технология JWT токен, что позволило передавать данные между клиентом и сервером и идентифицировать пользователя.

В модуле «События» происходят CRUD-операции над сущностями событий. События – это интерпретация семестра в рамках учебного года. Событие всегда имеет название, состоящее из двух слов – время года (весна, осень) и год соответственно.

В модуле «Проекты» инкапсулирована вся логика по работе с проектами студентов, а именно: CRUD-операции с сущностями проектов, хранение проектных заявок для дальнейшего модерирования администратором, оценивание проектов другими студентами, получение истории проекта по предыдущим событиям, получение команд, которые работали над проектом.

Вся логика по работе с командами находится в модуле «Команды». В этом модуле реализованы CRUD-операции для сущностей команд, а также хранение проектных ролей, благодаря которым пользователи разграничивают зону ответственности в команде.

Серверной части также необходимо хранилище данных, где можно хранить игровые проекты пользователей. Игра загружается в хранилище в формате ZIP-архива при помощи запроса из модуля «Проекты».

Так, необходимый функционал веб-сервиса был выражен в модулях серверного приложения, которое будет обрабатывать ответы от клиентской части.

После окончания работ над проектированием началась разработка схемы базы данных. Для реализации в проекте использовалась TypeORM, популярная библиотека ORM для TypeScript и Node.js. Использование TypeORM существенно упростило управление данными и выполнение запросов, так как отсутствовала необходимость писать SQL код. Все низкоуровневые операции выполняет TypeORM. Также при разработке схемы существенную роль сыграла технология миграций, настроенных в проекте. При помощи миграций появилась возможность создавать необходимые таблицы базы данных и настраивать взаимосвязи данных прямо из среды разработки, не отвлекаясь на написание SQL кода.

Для представления пользователей была создана сущность в коде серверного приложения под названием UserEntity (рисунок Б.1 приложения Б). В этой сущности отображается информация, относящаяся к пользователю, такая как адрес электронной почты, имя, фамилия, зашифрованный пароль и дополнительные поля для программы обучения пользователя и его курс. Поля «группа» и «контакты» также были добавлены для сбора более полных пользовательских данных и для удобства коммуникации студентов.

Подробную информацию о том, в каких проектах был задействован пользователь можно узнать при помощи сущности ProjectEntity (рисунок Б.2 приложения Б). В этой сущности отображается информация об игровых проектах, включая название, описание, инструкции по игре, ссылку на репозиторий Git, данные скриншотов, рейтинг игры. Также проект хранит информацию о текущем событии, в котором он был создан, и о команде, которая его выполняет, при помощи связи «один ко многим».

Сущность TeamEntity (рисунок Б.3 приложения Б) представляет собой информацию о названии команды и список участников, которые находятся в команде. Данная сущность нужна для корректного отображения портфолио.

Сущность EventEntity (рисунок Б.4 приложения Б) хранит в себе данные о текущем и прошедших семестрах обучения, а также предоставляет информацию о проектах, выполненных во время действия события.

Чтобы отслеживать роли, выбранные пользователями в рамках конкретного проекта, а также для сохранения полной информации в портфолио была создана сущность ProjectRoles (рисунок Б.5 приложения Б). Данная таблица связывает пользователей с проектами при помощи связи «многие ко многим».

Результирующая структура базы данных соответствует целям, которые должен выполнять сервис в целом, а также API в частности (рисунок В.1 приложения В). Так, у каждого проекта может быть несколько команд в зависимости от события, а у пользователя – несколько проектных ролей.

После создания архитектуры базы данных была начата разработка API серверной части сервиса.

2.2 Написание API

Опираясь на архитектуру базы данных, следующим шагом была разработка серверного API, которое служит посредником между клиентским приложением и базой данных, предоставляя четко определенный набор конечных точек для управления проектами, пользователями, командами и другими сущностями.

Для каждого из модулей серверного приложения был разработан интерфейс CRUD – операций. Так, в самом начале создания контроллера мы определяем его название – это будет первой частью нашей строки запроса к определенному методу контроллера. Второй частью является конкретный метод. Спецификация CRUD – операций представлена в таблице 2.

Таблица 2 – Спецификация CRUD – операций

Первая часть запроса	Метод	Вторая часть запроса	Назначение
“/Название контроллера”	Post	“/create”	Создание сущности
	Get	“/:id”	Получение сущности по id
	Patch	“/:id”	Редактирование сущности по id
	Delete	“/:id”	Удаление сущности по id

Перейдем к рассмотрению каждого модуля, которые реализованы в серверном приложении.

Основа аутентификации пользователя, модуль авторизации, включает в себя методы регистрации пользователя и входа в систему. Контроллер модуля имеет название «auth» и управляет этими операциями, вызывая метод signUp при запросе на /auth/signUp. Перед выполнением запроса входные данные, которые приходят в теле запроса, проходят валидацию при помощи технологии Pipes. В документации Nest.js описана необходимость данной технологии: «Validation: evaluate input data and if valid, simply pass it through unchanged; otherwise, throw an exception» [4]. При успешной валидации генерируется JWT токен доступа при помощи метода sign библиотеки jsonwebtoken, который возвращает JSON Web токен как строку. Согласно документации, метод sign предлагает механизм генерации токенов, гарантирующий соблюдение надежных мер безопасности на основе алгоритма HS256 [5].

Чтобы проверить действительность токенов JWT, отправленных в заголовках запросов, модуль Авторизации использует middleware AuthMiddleware. Данная технология позволяет нам вместо написания и

дублирования логики в контроллерах добавить методы в отдельную функцию, которая будет вызываться перед выполнением метода и применяться на все или заранее определенные методы [6]. Поэтому в функцию AuthMiddleware был добавлен метод verify из библиотеки jsonwebtoken для проверки JWT токена [7]. Использование данного подхода гарантирует целостность механизма аутентификации пользователей.

После создания модуля авторизации был написан модуль пользователей. В контроллере с названием «users» обрабатываются операции, связанные с учетными записями и профилями пользователей. Здесь находятся методы по созданию нового, получению сведений, изменению информации о пользователях, а также удаление учетных записей. Эти CRUD-операции написаны по спецификации (см. таблицу 2) и обеспечивают применение бизнес-логики. Кроме того, в этом же контроллере реализован метод для получения портфолио проектов пользователя, то есть списка проектов, в которых пользователь принимал участие.

Аналогично пользовательскому модулю, модуль проектов поддерживает набор CRUD-операций, написанных по спецификации (см. таблицу 2), которые находятся в контроллере с названием «projects». В дополнение к этому, контроллер имеет метод по оцениванию проектов учащимися, представленный на рисунке 2. В данном методе изменяется поле rating по id проекта, который приходит в запросе с клиентского приложения

Рисунок 2 – Метод для оценивания проектов

Кроме того, модуль проектов включает в себя методы по манипулированию статусом конкретного проекта, однако данный функционал доступен только администраторам сервиса. Данная функциональность

реализуется при помощи технологии Guards в Nest.js. В документации сказано, что Guard должен реализовывать интерфейс CanActivate и возвращать Boolean тип обозначающий разрешен ли текущий запрос или нет [8]. В данной реализации мы проверяем роль у текущего пользователя, которого мы получаем по токену при помощи метода «findByToken». Если роль пользователя не совпадает с ролью «Адмнин», данному пользователю не удастся выполнить запрос, вернется ошибка 401 Unauthorized. Реализация AdminGuard для проверки роли администратора представлена на рисунке 3.

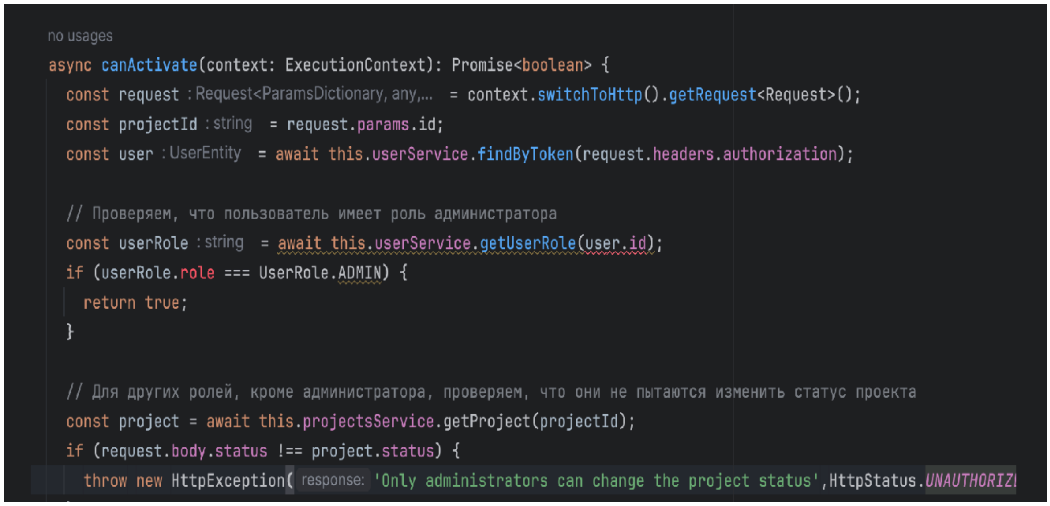

Рисунок 3 – Реализация AdminGuard

Далее был реализован модуль команд и контроллер с названием «teams». Данный контроллер инкапсулирует функции управления командой, при этом пользователям предоставляется возможность выбирать различные роли в проектных командах. При создании команды также применяется технология Guard, так как создать команду может только пользователь с проектной ролью «Team Leader». Реализация данного Guard представлена на рисунке 4.

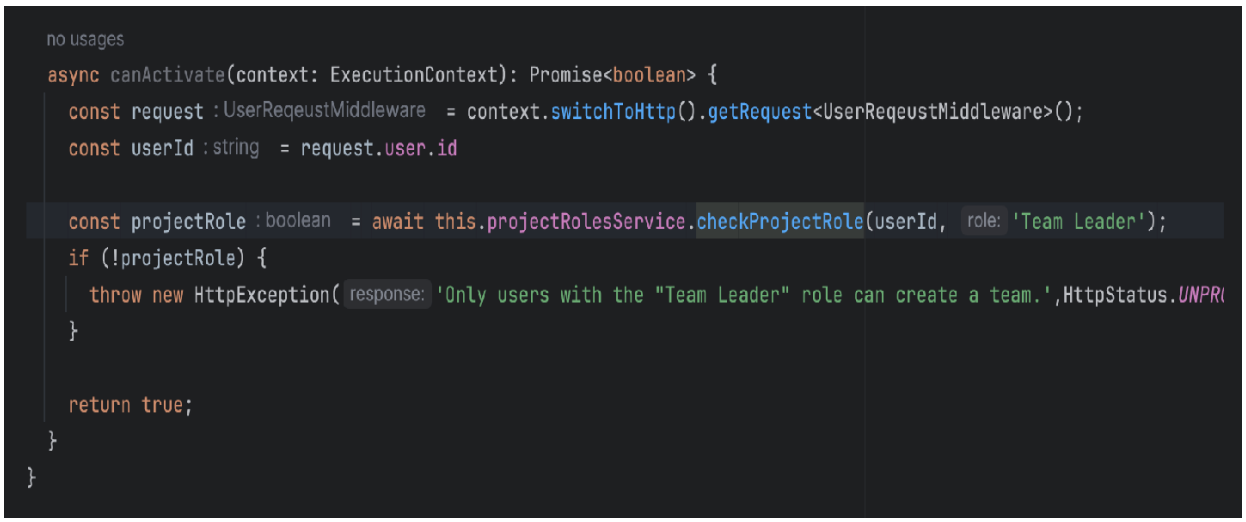

Рисунок 4 – Реализация TeamLeadGuard

Затем были реализованы следующие 2 модуля – модуль ролей и модуль событий. В них реализованы CRUD-операции, реализованные по спецификации (см. таблицу 2), которые доступны только пользователям с ролью админа. Для валидации этого условия применяется Guard под названием AdminGuard.

Наконец, веб-сервис интегрируется с отдельным файловым хранилищем, запущенным на отдельном Docker контейнере, представленный локальным хранилищем MinIO, что позволяет хранить и извлекать файлы и ресурсы игровых проектов. MinIO позиционирует себя как полноценный аналог хранилищу S3, при этом являясь полностью совместимым с ним [9]. При загрузке пользователем проекта в MinIO создается bucket – объект схожий с папкой или директорией в файловой системе [10]. Данному bucket присваивается имя – строковое представление id созданного проекта. Именно в этот bucket будет загружаться сборка с файлами игрового проекта. Данное хранилище открыто для чтения всем пользователям, чтобы даже неавторизированный пользователь мог скачать файлы игры или запустить игру в браузере. Однако, загружать файлы с играми могут только авторизированные пользователи. Так как хранилище поддерживает управление доступом с помощью политик [11], были добавлены следующие настройки (рисунок Г.1 Приложения Г).

ЗАКЛЮЧЕНИЕ

Все задачи, поставленные в календарном плане, были выполнены. В рамках преддипломной практики была проведена работа над проектированием функционала веб-сервиса, который представляет собой витрину игровых проектов студентов на треке ПП GameDev. Была составлена схема взаимодействия модулей серверного приложения, а также определена функциональность каждого из них. Также были изучены необходимые инструменты и технологии для проектирования серверной части и написания API. Было реализовано построение архитектуры базы данных, при помощи ORM. И наконец, было написано API и произведена интеграция файлового хранилища и API серверной части.

В рамках проектирования архитектуры базы данных сначала был определен функционал каждого из модулей. Затем, на основе функционала были определены сущности, которые будут взаимодействовать в системе. Наконец, при помощи технологии ORM была создана схема базы данных, а механизм миграций предоставил удобный инструмент для поддержания базы данных в консистентном состоянии.

При подборе инструментов были выбраны средства для разработки серверной части, для создания и изменения базы данных, и для настройки файлового хранилища. При выборе инструментов ориентация была на скорость и современность разработки, а также на обеспечение удобного процесса работы.

На основании подобранных инструментов был реализован API и серверная часть приложения. Для этого были созданы запроектированные модули, для каждого были написаны соответствующие контроллеры и сервисы. Также был создан отдельный Docker контейнер, на котором работало файловое хранилище.

В результате, все поставленные задачи были выполнены в полном объеме. Проектирование и разработка серверной части были осуществлены

тщательно, и в последующей работе не ожидается возникновения значительных трудностей. Дальнейшая разработка сервиса будет продолжена в рамках работы над выпускной квалификационной работой (ВКР). На этапе ВКР будут усовершенствованы и доработаны уже разработанные решения, а также создан окончательный рабочий продукт на основе полученных результатов текущего этапа разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Плетнев А. В. Организация разграничения доступа пользователей к функционалу информационной системы / А. В. Плетнев, С. В. Поляков // журнал «Восточно-европейский научный журнал». – 2022.

2. Nest.js. Introduction : офиц. сайт. – URL: https://docs.nestjs.com (дата обращения: 25.02.24).

3. Усеинов Ш.М. Introduction to the ORM library RedBeanPHP / Венкова И.В., Решитов А.А., Сеитмеметов А.А., Бакиев Э.Э // журнал «Информационно-компьютерные технологии в экономике, образовании и социальной сфере». – 2018. –№1(19). – ISSN: 2658-5944

4. Nest.js. Pipes : офиц. сайт. – URL: https://docs.nestjs.com/pipes (дата обращения: 04.03.24).

5. Jsonwebtoken : сайт. – URL: https://www.npmjs.com/package /jsonwebtoken (дата обращения: 05.03.24).

6. Exploring NestJS middleware: Benefits, use cases, and more : сайт. – URL: https://blog.logrocket.com/exploring-nestjs-middleware-benefits-use-cases (дата обращения: 06.03.24).

7. Using the jsonwebtoken Node Package to Verify JSON Web Tokens : сайт. – URL: https://medium.com/swlh/using-the-jsonwebtoken-node-package-to-verify-json-web-tokens-497ecdaba830 (дата обращения: 06.03.24).

8. Nest.js. Guards : офиц. сайт. – URL: https://docs.nestjs.com/faq/request-lifecycle#guards (дата обращения: 07.03.24).

9. The World's Most Advanced Object Store : офиц. сайт. – URL: https://min.io/product/enterpriseoverview (дата обращения: 10.03.24).

10. Core Administration Concepts: сайт. – URL: https://min.io/docs/minio/container/administration/concepts.html (дата обращения: 12.03.24).

11. Предоставляем ограниченный доступ к бакету AWS S3 и MinIO : сайт. – URL: https://sysops.host/75-predostavljaem-ogranichennyj-dostup-k-hranilischu-aws-s3-i-minio.html (дата обращения: 20.03.24).

ПРИЛОЖЕНИЕ А
(справочное)

Схема модулей и методов контроллеров

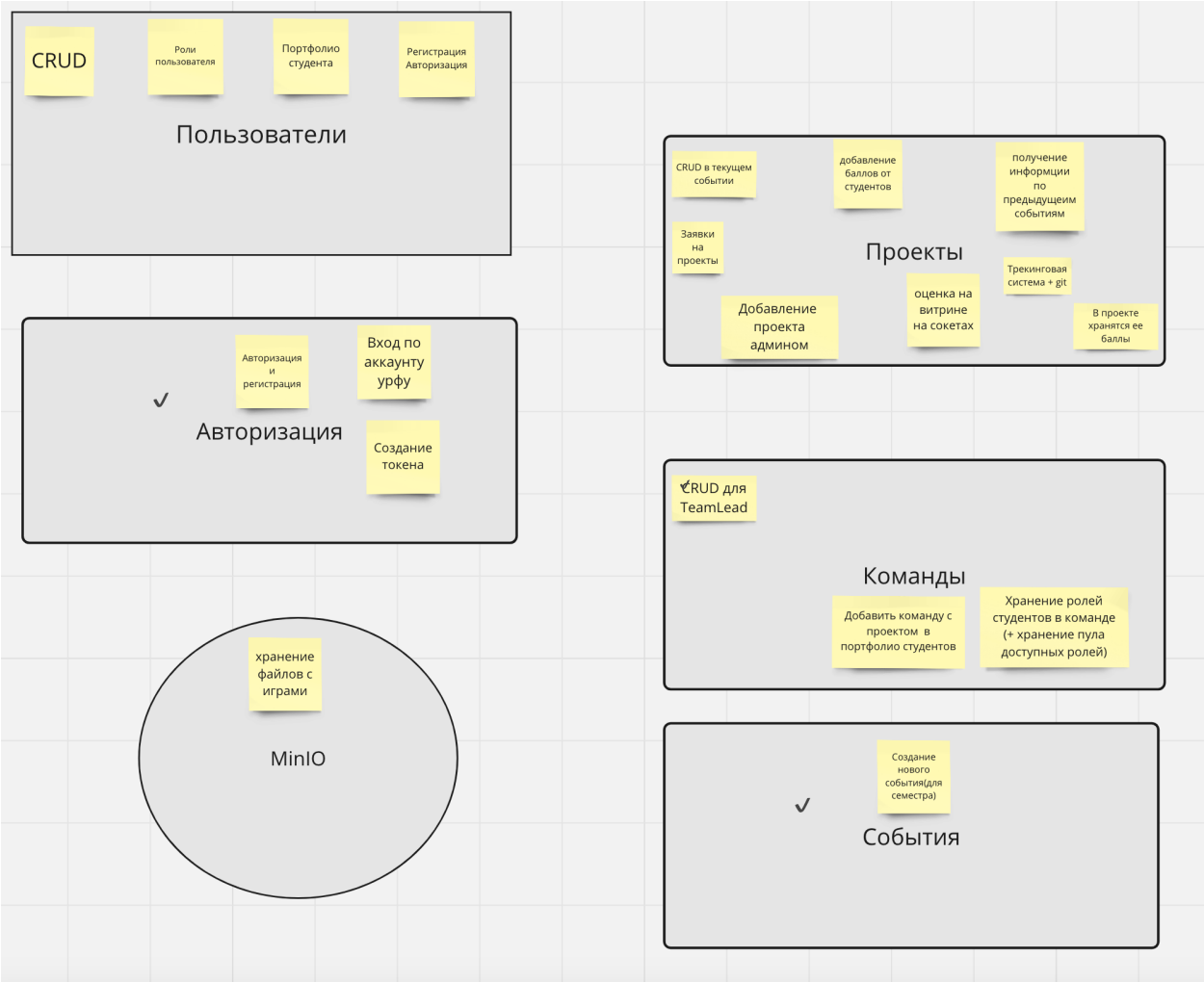

Рисунок А.1 – Схема модулей и методов контроллеров

ПРИЛОЖЕНИЕ Б
(справочное)

Сущности, используемые в приложении

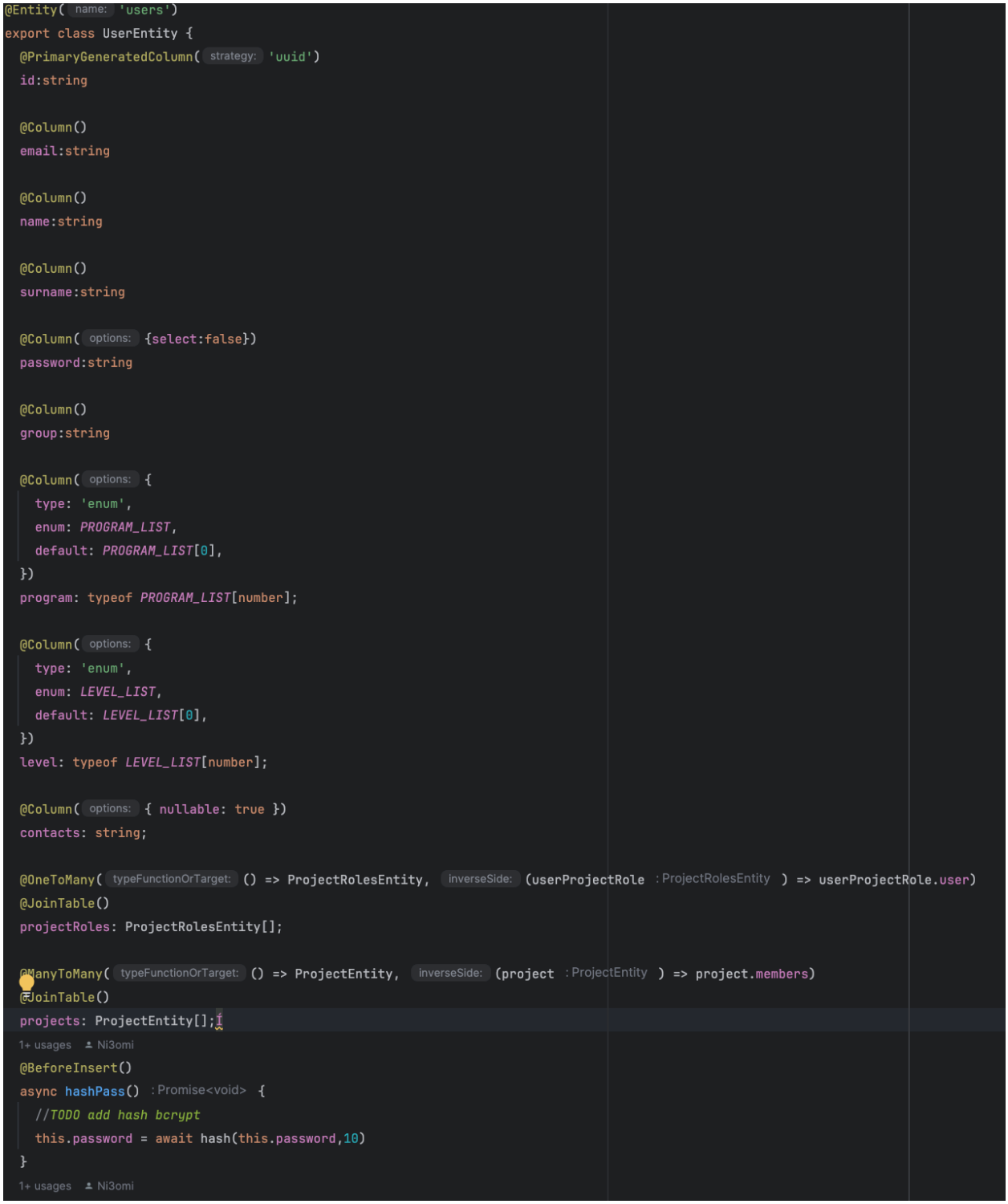

Рисунок Б.1 – Сущность UserEntity

```
@Entity( name: 'projects')
export class ProjectEntity {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @Column()
  name: string;

  @Column()
  description: string;

  @Column()
  howToPlay: string;

  @Column()
  gitLink: string;

  @Column( options: 'simple-array')
  screenshots: string[];

  @Column()
  rating: number;

  @ManyToOne( typeFunctionOrTarget: ()=>EventEntity, inverseSide: (event : EventEntity )=>event.projects)
  event:EventEntity

  @ManyToOne( typeFunctionOrTarget: () => TeamEntity,  inverseSide: (team : TeamEntity ) => team.projects)
  team: TeamEntity;

  @ManyToMany( typeFunctionOrTarget: () => UserEntity,  inverseSide: (user : UserEntity ) => user.projects)
  @JoinTable()
  members: UserEntity[];



  @OneToMany( typeFunctionOrTarget: () => ProjectRolesEntity,  inverseSide: (userProjectRole : ProjectRolesEntity ) => userProjectRole.project)
  userRoles: ProjectRolesEntity[];
}
```

Рисунок Б.2 – Сущность ProjectEntity

```
1+ usages
@Entity( name: 'teams')
export class TeamEntity {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @Column()
  name: string;

  @ManyToMany( typeFunctionOrTarget: () => ProjectEntity,  inverseSide: (project : ProjectEntity ) => project.members)
  projects: ProjectEntity[];
}
```

Рисунок Б.3 – Сущность TeamEntity

```
@Entity( name: 'events')
export class EventEntity {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @Column()
  name: string;

  @OneToMany( typeFunctionOrTarget: ()=>ProjectEntity, inverseSide: (project : ProjectEntity )=>project.event)
  projects:ProjectEntity[]
}
```

Рисунок Б.4 – Сущность EventEntity

```
@Entity( name: 'project_roles')
export class ProjectRolesEntity {
  @PrimaryGeneratedColumn( strategy: 'uuid')
  id: string;

  @Column()
  role: string;

  @ManyToOne( typeFunctionOrTarget: () => UserEntity, inverseSide: (user : UserEntity ) => user.projectRoles)
  user: UserEntity;

  @ManyToOne( typeFunctionOrTarget: () => ProjectEntity, inverseSide: (project : ProjectEntity ) => project.userRoles)
  project: ProjectEntity;
}
```

Рисунок Б.5 – Сущность ProjectRoles

ПРИЛОЖЕНИЕ В
(обязательное)

Схема базы данных

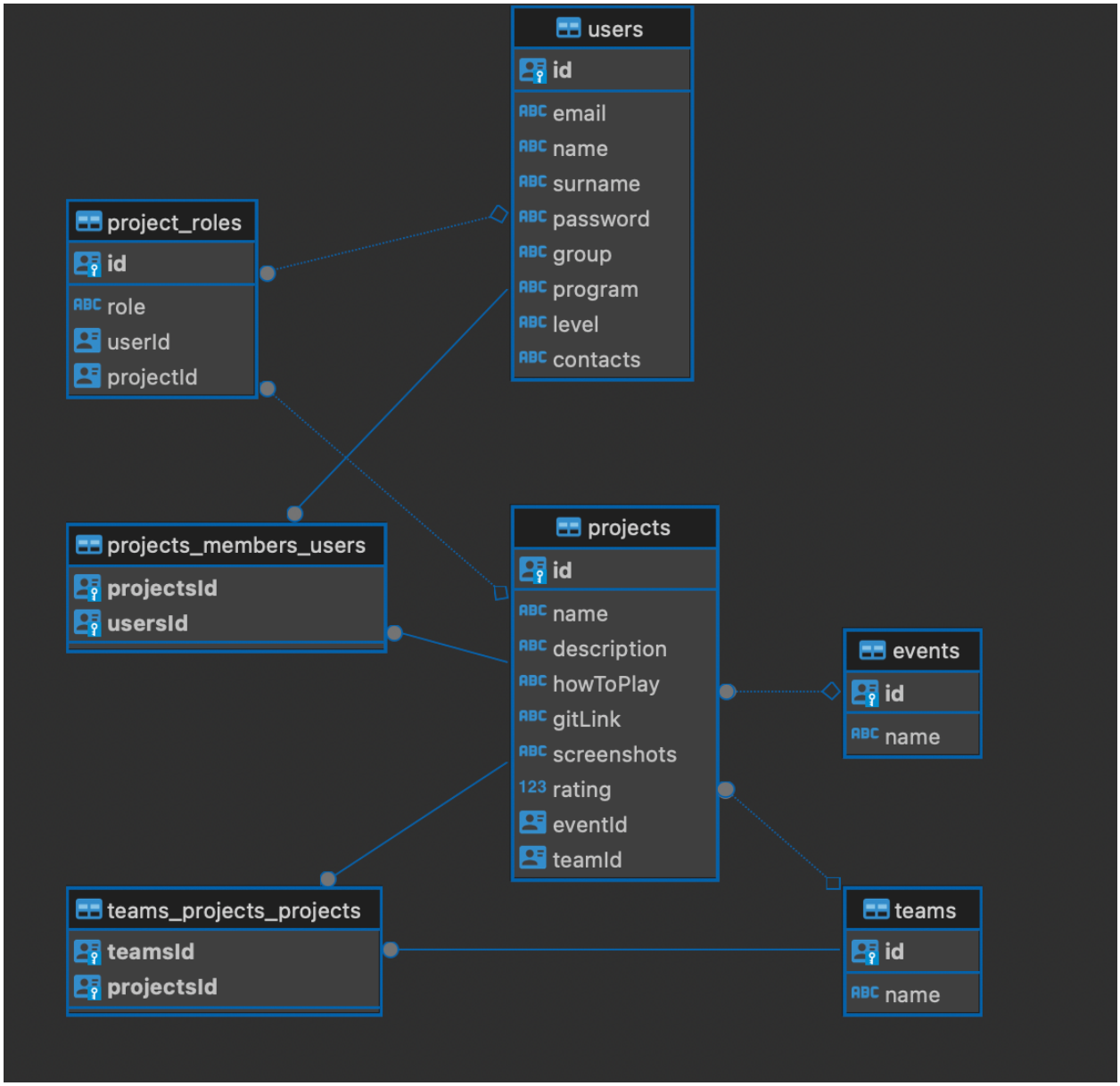

Рисунок В.1 – Схема базы данных

ПРИЛОЖЕНИЕ Г
(обязательное)

Настройка политик MinIO для работы хранилища

```
private getPolicy(bucketName:string) :{Statement: {Effect: string, P... {
  return  {
    Statement: [
      {
        Effect: 'Allow',
        Principal: {
          AWS: ['*'],
        },
        Action: [
          's3:ListBucketMultipartUploads',
          's3:GetBucketLocation',
          's3:ListBucket',
        ],
        Resource: [`arn:aws:s3:::${bucketName}`],
      },
      {
        Effect: 'Allow',
        Principal: {
          AWS: ['*'],
        },
        Action: [
          // 's3:PutObject',
          's3:AbortMultipartUpload',
          's3:GetObject',
          's3:ListMultipartUploadParts',
        ],
        Resource: [`arn:aws:s3:::${bucketName}/*`],
      },
    ],
  };

}
```

Рисунок Г.1 – Настройка политик MinIO для работы хранилища