3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Описание структуры приложения

Этот раздел посвящен описанию работы и состава разрабатываемого программного продукта. В дальнейшем представлены взаимосвязи между различными классами программного обеспечения в виде диаграммы классов ГУИР.400201.307 РР.1.

Для разработки серверной части программного продукта выбраны несколько технологий, которые используются в качестве инструментов и средств для создания функциональной и надежной системы. Эти технологии являются ключевыми элементами в разработке приложения и включают в себя набор инструментов для работы с базами данных, обеспечения безопасности и аутентификации пользователей, а также для реализации бизнес-логики приложения. Каждая из выбранных технологий имеет свои особенности и преимущества, что позволяет создавать более эффективную и гибкую систему в соответствии с требованиями проекта. Данные технологии в общих чертах рассматриваются ниже.

1. Spring Boot, которая позволяет создавать веб-сервер и настраивать взаимодействие между различными классами приложения, он автоматически добавляет в проект все необходимые зависимости и настраивает их для работы вместе.

2. Maven, используется для настройки процесса сборки, упаковки и запуска приложения, иначе говоря, для автоматизации сборки проектов. Данная технология использует файлы конфигурации POM (Project Object Model), в которых содержится информация о проекте, его зависимостях, конфигурациях, плагинах и других параметрах. С помощью данного инструмента и осуществляется подключение технологий Spring, например.

3. Spring Web, для создания веб-приложений. Этот модуль фреймворка Spring предоставляет инструменты для разработки на языке Java. Предоставляет ряд абстракций и компонентов, которые позволяют создавать масштабируемые, гибкие и безопасные веб-приложения.

4. Spring Data JPA, для работы с базой данных. Предоставляет реализацию JPA, которая упрощает доступ к базе данных и сокращает объем кода, необходимого для создания репозиториев и выполнения операций с базой данных. Spring Data JPA позволяет автоматически генерировать репозитории, которые позволяют выполнять CRUD (Create, Read, Update, Delete) операции с объектами, не нужно писать много кода вручную.

5. Spring Security, для обеспечения защиты и авторизации пользователей в системе. Предоставляет инструменты для обеспечения безопасности веб- приложений на основе Java.

В данном приложении серверная часть приложения реализовала на Java. Исходя из гексагональной архитектуры, в приложении классы делятся по

сисполняемому функционалу. Вся структура пакетов и их описание приведена ниже (см. рисунок 3.1):

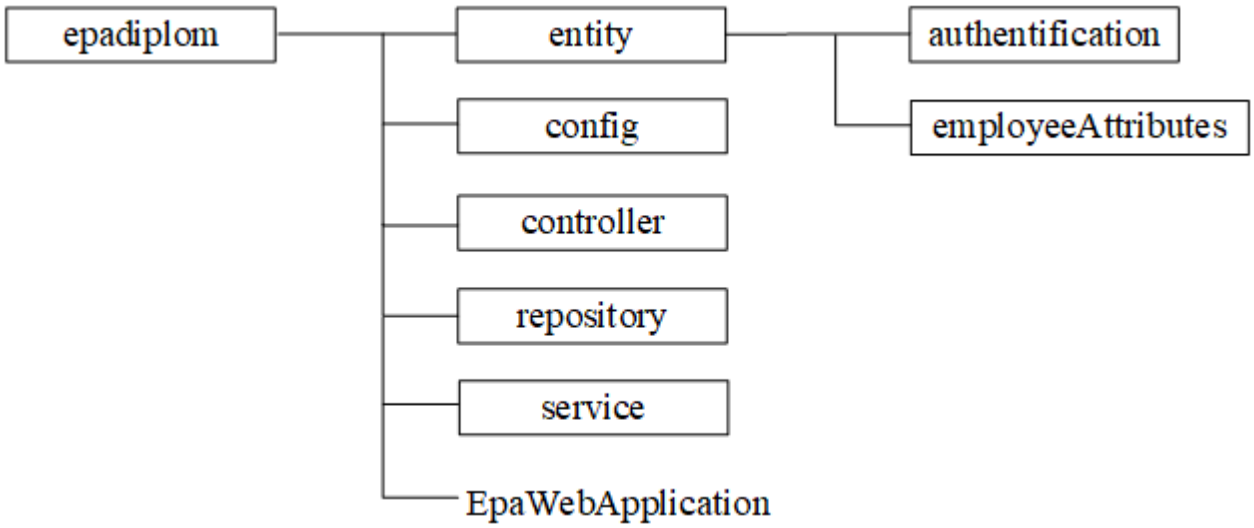

Рисунок 3.1 – Структура пакетов в веб-приложении

1. Пакет config – пакет, содержащий классы, которые отвечают за конфигурацию и настройку Spring Security, фреймворка, предназначенного для обеспечения безопасности приложений на платформе Spring. Данный пакет также отвечает за настройку JSON Web Token (JWT), механизма аутентификации и авторизации пользователей, использующего технологию передачи данных в формате JSON. Здесь происходит создание и настройка токенов, которые используются для идентификации пользователей и обеспечения безопасного доступа к ресурсам системы. Благодаря настройке JWT в данном пакете, система гарантирует безопасность передачи данных между клиентом и сервером и защищает от несанкционированного доступа. В него входят классы, перечисленные ниже:

- класс `ApplicationConfig`;
- класс `JwtAuthentificationFilter`;
- класс `SecurityConfig`.

2. Пакет controller – пакет, содержащий классы-контроллеры, которые являются частью паттерна проектирования Model-View-Controller (MVC). Контроллеры представляют собой классы, которые обрабатывают запросы от клиента и выполняют соответствующие действия в системе. Внутри каждого контроллера находятся методы-обработчики, которые реагируют на определенный тип запросов и возвращают клиенту соответствующий ответ. Данный пакет играет важную роль в обработке запросов и представляет собой основной механизм, с помощью которого клиент взаимодействует с системой. Содержит в себе классы:

- класс `AuthenticationController`;
- класс `MainPageConroller`.

3. Пакет service – пакет, содержащий классы, связанные с сервисом сущностей таблицы, а также касаемо классов сервиса для обеспечения аутентификации, настройки токена. С помощью этих классов можно реализовать сложные запросы к таблицам, которые не могут быть выполнены с помощью JpaRepository. В сочетании с пакетами repositories и entities, они позволяют реализовать бизнес-логику приложения.

4. Пакет repository – пакет, содержащий интерфейсы, которые нужны для взаимодействия с базой данных с помощью репозиториев, таких как JpaRepository. Это упрощает создание запросов к таблицам и в сочетании с пакетами service и entities позволяет реализовывать бизнес-логику приложения.

5. Пакет entities – пакет, содержащий сущности базы данных, которые используются вместе с пакетами service и repository для реализации бизнес-логики приложения. Он включает в себя 18 классов, из которых шесть являются представлениями, которые уже были упомянуты в модели данных и не нуждаются в дополнительном описании. Сущности, находящиеся в этом пакете, служат основой для работы с базой данных и представляют структуру данных, которые хранятся в ней. Вместе с классами из пакетов service и repositories, этот пакет обеспечивает полную реализацию бизнес-логики приложения. Также в пакете с сущностями содержится два подпакета – authentification и amployeeAttributes.

5.1 Подпакет authentication – пакет содержащий классы-сущности, которые отвечают за реализацию процесса аутентификации и авторизации пользователей в системе. В него входят классы, перечисленные ниже:

- класс `AuthenticationRequest`;
- класс `AuthenticationResponse`;
- класс `RegisterRequest`.

5.2 Подпакет employeeAttribures содержит список именованных констант – `role`, в котором находятся роли пользователей. Этот список используется для управления уровнем доступа пользователей при аутентификации и авторизации в системе. Различные роли дают пользователям различные уровни доступа.

Отдельно от всех этих моделей и пакетов находится класс `EpaWebApplication` в общей папке со всем вышеперечисленным с названием `epadiplom`.

Подобная организация пакетов в приложении позволяет удобно добавлять новый функционал и вносить изменения в существующий код. При создании нового компонента можно просто создать новый пакет и добавить в него нужные классы, не затрагивая другие компоненты приложения. Такая организация обеспечивает изоляцию функционала и позволяет легко поддерживать код приложения. Кроме того, такая структура приложения упрощает работу команды разработчиков и ускоряет процесс разработки.

3.2 Описание модели данных

В данном разделе рассматривается база данных, которая работает с помощью СУБД PostrgeSQL. Этот блок включает в себя данные, которые использует разрабатываемая система.

Для удобства модель данных можно условно разбить на несколько логических блоков, каждый из которых будет выполнять свою функцию в создаваемом веб-приложении (см. рисунок 3.2).

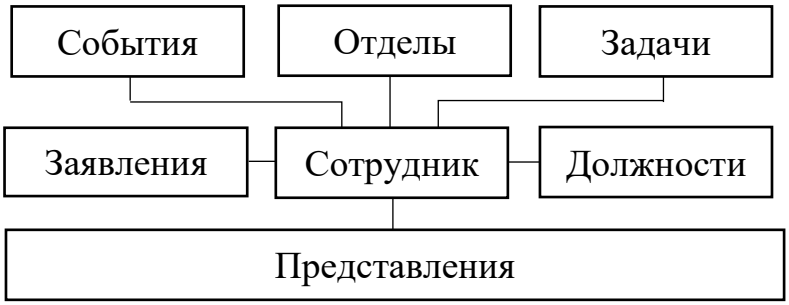

Рисунок 3.2 – Условное логическое разделение блоков БД

Данные сотрудника будут использоваться как для основной информации о сотруднике, так и хранить данные о пользователе, которые помогут получать информацию об определенном сотруднике.

Данные по событиям, задачам, отделам, заявлениям и должностям помогут получать информацию, исходя из их названий.

Все вышеперечисленные блоки представляют собой таблицы реляционной базы данных, которыми можно всячески манипулировать, но, в представленном логическом разделении, также присутствует блок представлений.

Представления являют собой виртуальные таблицы, которые будут помогать облегчать доступ к данным, не прибегая к созданию сложных запросов. Они нужны, если необходимо получить информацию из нескольких таблиц одновременно. Такие «таблицы» можно только просматривать без изменения данных. Чтобы их изменить придется обращаться к исходным таблицам.

3.2.1 Таблица `Employee`

Данная таблица предназначена для хранения основной информации о пользователе, которая можно указать в общем доступе, и которая будет отображаться в глобальном поиске сотрудников.

Поля таблицы:

- `id` – первичный ключ, bigint;
- `first_name` – имя сотрудника, varchar (128);
- `middle_name` – отчество сотрудника, varchar (128);

- last_name – фамилия сотрудника, varchar (128);
- work_number – рабочий номер сотрудника, numeric (16);
- location_street – зашехированный пароль, который был выслан пользователю для восстановления аккаунта, varchar (128);
- cabinet_office – время и дата отправления пароля для восстановления аккаунта, varchar(8);
- id_dep – внешний ключ для связи с таблицей department, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем id используется специальный атрибут «null: false», это означает, что поле не может быть нулевым. Для остальных колонок это значение не выставлено, чтобы при создании аккаунта для сотрудника данные заполнялись отделом кадров и не было ошибок в системе и документах.

3.2.2 Таблица Login

Данная таблица предназначена для хранения информации о пользователе, которая связана с аккаунтом сотрудника, а именно содержит данные необходимые для авторизации и создания аккаунта.

Поля таблицы:

- id_login – первичный ключ, а также внешний ключ для таблицы employee, bigint;
- login_user – логин сотрудника, varchar (319);
- password_user – захешированный пароль, varchar (128);
- mail_user – мейл сотрудника, varchar (319);
- role – роль пользователя (касаемо аккаунта, а не работы), varchar (6).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем id_login, login_user, password_user, mail_user, role используется специальный атрибут «null:false», это означает, что поле не может быть нулевым. Все эти поля заполняются при создании профиля сотрудника.

3.2.3 Таблица Personal

Данная таблица предназначена для хранения личной информации о пользователе, которую можно будет увидеть только при наличии определенных прав, и которая требуется, в основном, отделу кадров.

Поля таблицы:

- id_personal – первичный ключ, а также внешний ключ для таблицы employee, bigint;
- birth_d – дата рождения сотрудника, date;
- entry_d – дата устройства на работу (в этот же день должен быть и создан аккаунт date;
- personal_number – личный номер сотрудника, numeric (16).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полем id_personal используется специальный атрибут «null: false», это означает, что поле не может быть нулевым. Для остальных колонок это значение не выставлено, чтобы при создании аккаунта для сотрудника данные заполнялись отделом кадров и не было ошибок в системе и документах. Поле entry_d будет заполняться автоматически при создании аккаунта.

3.2.4 Таблица Department

Данная таблица нужна для содержания списка отделов университета, чтобы было удобнее распределять и сортировать сотрудников.

Поля таблицы:

- id – первичный ключ, bigint;
- name_dep – название отдела университета, varchar (128).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. С полями id и name_dep используется специальный атрибут «null: false». Эти поля не могут быть нулевыми.

3.2.5 Таблица Log_statement

Данная таблица предназначена для информации о заявлениях. Они могут быть разных типов: от отпусков, до увольнения за свой счет и тому подобные.

Поля таблицы:

- id – первичный ключ, bigint;
- days_sum – сумма дней, integer;
- date_leave – дата, когда работник уходит по заявлению (дата действия заявления), date;
- date_of_ls – дата, когда работник составляет заявление date;
- id_approver – номер сотрудника, который должен подтвердить заявление, bigint;
- comment_ls – комментарий сотрудника к заявлению, varchar (300);
- type_leave – тип заявления (типы будут прописаны в логике), smallint;
- status – статус подтверждения, numeric;
- id_employee – внешний ключ для таблицы employee (того работника, что составляет заявление), bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля за исключением comment_ls используется специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении и сотрудник не может оставить их пустыми, за исключением комментария и самого скана документа. Поле date_of_ls будет заполняться автоматически при создании аккаунта.

3.2.6 Таблица `Document`

Данная таблица предназначена для хранения сканов оригинальных заявлений. Они не являются обязательными, поэтому таблица `log_statement` может существовать без привязки к данной таблице.

Поля таблицы:

- `id` – первичный ключ, bigint;
- `id_ls` – внешний ключ для таблицы `log_statement`, bigint;
- `body_doc` – ссылка на скан оригинального заявления, varchar (300).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.7 Таблица `Job_title`

Данная таблица предназначена для хранения списка должностей сотрудников БГУИР.

Поля таблицы:

- `id` – первичный ключ, bigint;
- `job_title_name` – название должности, varchar (128).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.8 Таблица `Job_employee`

Данная таблица является реализацией связи многие-ко-многим между таблицами `job_title` и `employee`.

Поля таблицы:

- `id` – первичный ключ, bigint;
- `id_job_title` – внешний ключ для таблицы `job_title`, bigint;
- `id_employee` – внешний ключ для таблицы `employee`, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.9 Таблица `Task`

Данная таблица предназначена для хранения информации о заданиях или же действиях, которые нужно сделать работнику.

Поля таблицы:

- `id` – первичный ключ, bigint;

- date_task – дата, когда работник составляет заявление, date;
- name_of_task – название задания или его суть, varchar (128);
- id_executor – номер сотрудника, который будет исполнять задание, bigint;
- comment_te – комментарий сотрудника к заданию, по сути, описание, если таковое требуется, varchar (300).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля, кроме comment_te, используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении. Поле comment_te можно пропустить, потому что некоторые задания могут быть ясны без уточнений.

3.2.10 Таблица Emp_task

Данная таблица является реализацией связи многие-ко-многим между таблицами task и employee.

Поля таблицы:

- id – первичный ключ, bigint;
- id_job_title – внешний ключ для таблицы job_title, bigint;
- id_employee – внешний ключ для таблицы employee, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.11 Таблица Event

Данная таблица предназначена для хранения информации о событиях, созданных сотрудниками, а также назначения, для кого они предназначены.

Поля таблицы:

- id – первичный ключ, bigint;
- date_of_event – дата события, date;
- type_of_event – название события, varchar (40);
- comment_fe – комментарий сотрудника к событию, по сути, описание, если таковое требуется, varchar (300).

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля, кроме comment_fe, используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении. Поле comment_fe можно пропустить, потому что некоторые задания могут быть ясны без уточнений. Формат данных timestamp without time zone означает, что нет привязки к часовому поясу. Это сделано, чтобы не было разногласий во времени.

3.2.12 Таблица `Notice_event`

Данная таблица является реализацией связи многие-ко-многим между таблицами `event` и `employee`.

Поля данной таблицы:

- `id` – первичный ключ, bigint;
- `id_event` – внешний ключ для таблицы `event`, bigint;
- `id_recipient` – номер сотрудника, которому предназначается отправить событие, bigint;
- `id_employee` – внешний ключ для таблицы `employee`, bigint.

В таблице для колонок существуют определенные атрибуты и условия, которые задавались при их создании. Все поля используют специальный атрибут «null: false», это означает, что поле не может быть нулевым, потому что все эти поля имеют важное значение при заполнении.

3.2.13 Представление `Employee_full_info_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых общей информации о сотруднике, которая представляет собой данные по аккаунту, персональные данные и общие сведения, которые будут в общем доступе.

Таблицы, которые объединены в данном представлении: `personal`, `employee`, `department`, `job_employee`, `job_title`, `login`.

Поля представления:

- `job_employee.id`;
- `login.id_login`;
- `employee.first_name`;
- `employee.middle_name`;
- `employee.last_name`;
- `personal.birth_d`;
- `personal.entry_d`;
- `login.login_user`;
- `login.password_user`;
- `login.mail_user`;
- `login.role`;
- `employee.work_number`;
- `personal.personal_number`;
- `employee.location_street`;
- `employee.cabinet_office`;
- `department.name_dep`;
- `job_title.job_title_name`.

3.2.14 Представление `Employees_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых общей информации о сотруднике, которые могут быть в общем доступе и видимы для других сотрудников.

Таблицы, которые объединены в данном представлении: `employee`, `department`, `job_employee`, `job_title`.

Поля представления:

- `job_employee.id`;
- `job_employee.id_employee`;
- `employee.first_name`;
- `employee.middle_name`;
- `employee.last_name`;
- `employee.work_number`;
- `employee.location_street`;
- `employee.cabinet_office`;
- `department.name_dep`;
- `job_title.job_title_name`.

3.2.15 Представление `Ls_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых заявлений и их заполнения.

Таблицы, которые объединены в данном представлении: `employee`, `log_statement`, `document`, `login`.

Поля представления:

- `document.id`;
- `log_statement.type_leave`;
- `log_statement.date_leave`;
- `log_statement.date_of_ls`;
- `log_statement.days_sum`;
- `log_statement.id_approver`;
- `log_statement.status`;
- `log_statement.comment_ls`;
- `log_statement.id_employee`;
- `document.body_doc`;
- `login.role`.

3.2.16 Представление `Events_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых событий сотрудников университета.

Таблицы, которые объединены в данном представлении: `event`, `notice_event`. Поля представления:

- `notice_event.id`;
- `event.type_of_event`;
- `event.date_of_event`;
- `event.comment_fe`;
- `notice_event.id_recipient`;
- `notice_event.id_employee`.

3.2.17 Представление `Job_title_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых должности сотрудников.

Таблицы, которые объединены в данном представлении: `job_title`, `job_employee`.

Поля представления:

- `job_employee.id`;
- `job_title.job_title_name`;
- `job_employee.id_employee`.

3.2.18 Представление `Tasks_view`

Данная таблица является представлением, виртуальной или же логической таблицей, которая представляет собой поименованный запрос. Это представление нужно для облегченной работы с блоком данных, касаемых заданий, которые сотрудники могут назначать друг на друга.

Таблицы, которые объединены в данном представлении: `task`, `emp_task`.

Поля представления:

- `emp_task.id`;
- `emp_task.id_task`;
- `task.name_of_task`;
- `task.date_task`;
- `task.comment_te`;
- `task.id_executor`;
- `emp_task.id_employee`.

3.3 Описание структуры и взаимодействия между классами

При создании приложения использовался паттерн Model-View-Controller (MVC), который определяет его структуру, состоящую из трех основных компонентов: контроллера, сервиса и репозитория. Кроме того, следует отметить, что все созданные сервисы разработаны в соответствии с правилами и стандартами REST-архитектуры. Это касаемо серверной части приложения.

Контроллеры отвечают за обработку входящих HTTP-запросов и вызывают соответствующие методы сервисов, которые обрабатывают эти запросы и возвращают результаты. Сервисы представляют собой прослойку между контроллером и репозиторием и отвечают за бизнес-логику приложения, такую как проверка прав доступа, обработка данных и т.д. Репозитории служат для связи с базой данных и содержат методы для выполнения CRUD-операций (создание, чтение, обновление и удаление данных).

3.3.1 Класс `EpaWebApplication`

Точка входа для запуска веб-приложения на основе фреймворка Spring Boot. Аннотация `@SpringBootApplication` указывает, что это главный класс приложения и сообщает Spring, что нужно выполнить все необходимые конфигурации и инициализации для запуска веб-приложения.

Метод `main()` вызывает метод `run()` класса `SpringApplication`, который запускает приложение. В качестве аргументов метод `run()` принимает класс `EpaWebApplication` и аргументы командной строки `args`.

Таким образом, этот класс и его метод `main()` запускают Spring Boot приложение и начинают обработку входящих HTTP запросов.

3.3.2 Класс `ApplicationConfig`

Класс представляет собой конфигурационный класс Spring, который содержит конфигурацию для аутентификации пользователей в системе. В нем объявлены такие аннотации, как `@Component` указывает, что класс является компонентом Spring и должен быть автоматически сканирован и зарегистрирован в контексте Spring, классы, отмеченные этой аннотацией, могут быть внедрены (injected) в другие классы в качестве зависимостей, и аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В классе определены следующие поля, описанные ниже:

1. Класс `userDetailsService` возвращает сервис для поиска пользователей по имени пользователя, используя репозиторий `UserRepo`.

2. Класс `authenticationProvider` создает провайдера аутентификации `DaoAuthenticationProvider`, который использует `userDetailsService` для поиска пользователя в базе данных и `passwordEncoder` для проверки пароля пользователя.

3. Класс `authenticationManager` создает и возвращает менеджер аутентификации `AuthenticationManager`, используя конфигурацию аутентификации.

4. Класс `passwordEncoder` возвращает объект `BCryptPasswordEncoder`, который используется для хэширования пароля пользователя.

3.3.3 Класс `JwtAuthenticationFilter`

Данный класс представляет собой фильтр аутентификации, который будет вызван один раз для каждого запроса, прошедшего через контроллер в приложении. Фильтр проверяет наличие токена авторизации в заголовке запроса и, если он присутствует, использует сервис JWT для проверки его валидности и получения имени пользователя из токена. Затем фильтр проверяет, что пользователь существует в базе данных и, если это так, создает аутентификационный токен Spring Security и устанавливает его в контекст безопасности. Если токен авторизации не найден или недействителен, фильтр пропускает запрос и передает его дальше по цепочке фильтров.

В классе объявлены такие аннотации, как `@Component` указывает, что класс является компонентом Spring и должен быть автоматически сканирован и зарегистрирован в контексте Spring, классы, отмеченные этой аннотацией, могут быть внедрены (injected) в другие классы в качестве зависимостей, и аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В нем всего один метод, который выполняет все вышеперечисленное:

- `protected void doFilterInternal`;

Также в коде есть два поля, которые получает конструктор:

- `private final JwtService jwtService` – класс, который реализует логику работы с JWT токенами;
- `private final UserDetailsService userDetailsService` – сервис, который будет использоваться для загрузки информации о пользователе по логину.

3.3.4 Класс `SecurityConfig`

Этот класс содержит конфигурацию Spring Security для веб-приложения. Он использует аннотации Spring `@Configuration`, которая указывает, что этот класс является конфигурационным файлом Spring, `@EnableWebSecurity`, которая включает использование Spring Security в приложении и

`@RequiredArgsConstructor`, которая генерирует конструктор, использующий поля с аннотацией `@NonNull` как параметры конструктора.

Данный класс содержит методы:

- `securityFilterChain` – метод, который создает цепочку фильтров безопасности;
- `corsConfigurationSource` – метод, который создает и настраивает бин, который определяет конфигурацию Cross-Origin Resource Sharing (CORS);
- методы `sessionManagement()` и `sessionCreationPolicy()` устанавливают стратегию управления сеансами;
- метод `authenticationProvider()` конфигурирует провайдер аутентификации для приложения;
- метод `addFilterBefore()` добавляет фильтры перед указанным фильтром.

В данном классе еще есть два поля:

- `jwtAuthFilter` – это объект фильтра, который будет использоваться для проверки JWT-токенов и аутентификации пользователей;
- `authenticationProvider` – это объект, который будет использоваться для проверки учетных данных пользователей.

3.3.5 Класс `AuthenticationController`

Данный класс представляет контроллер для обработки запросов, связанных с аутентификацией и авторизацией пользователей.

Аннотация `@RestController` указывает на то, что класс предназначен для обработки HTTP-запросов, а возвращаемые им методы должны быть преобразованы в тело ответа HTTP.

Аннотация `@RequestMapping("/auth")` указывает на корневой путь, который будет использоваться для обработки запросов, обрабатываемых этим контроллером.

Аннотация `@RequiredArgsConstructor` от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией `@NonNull` или `final`, также автоматически создает методы `hashCode()`, `equals()`, и `toString()`, используя поля класса.

В классе есть несколько методов, которые рассмотрены ниже:

1. Метод `register` – он обрабатывает POST-запросы на `/auth/register`. Он принимает в теле запроса объект `RegisterRequest`, содержащий данные, необходимые для регистрации нового пользователя, и передает их в сервис `AuthenticationService` для выполнения регистрации. Затем он возвращает объект `AuthenticationResponse`, содержащий информацию об успешности регистрации и авторизации нового пользователя.

2. Метод `authenticate` – он обрабатывает POST-запросы на `/auth/authenticate`. Он принимает в теле запроса объект

AuthenticationRequest, содержащий учетные данные пользователя (имя пользователя и пароль), и передает их в сервис AuthenticationService для выполнения аутентификации. Затем он возвращает объект AuthenticationResponse, содержащий JWT-токен, который пользователь может использовать для авторизации на защищенных ресурсах.

3.3.6 Класс MainPageController

Этот класс является контроллером Spring Boot и содержит обработчики HTTP-запросов. Он предназначен для работы с главной страницей приложения. Класс MainPageController использует несколько репозиториев для доступа к данным в базе данных, которые хранят информацию о сотрудниках, логах, событиях и других объектах. Каждый метод возвращает список объектов, который сериализуется в JSON и отправляется обратно клиенту в ответ на запрос. Также в этом классе есть методы, которые используют Spring Security для аутентификации пользователей и контроля доступа к данным.

Содержит такие аннотации, как @RestController, которая указывает на то, что класс предназначен для обработки HTTP-запросов, а возвращаемые им методы должны быть преобразованы в тело ответа HTTP, @EnableMethodSecurity, которая включает использование Spring Security в приложении, @EnableWebMvc, которая разрешает проекту использовать MVC, и @RequiredArgsConstructor от библиотеки Lombok генерирует конструктор для всех полей класса, отмеченных аннотацией @NonNull или final, также автоматически создает методы hashCode(), equals(), и toString(), используя поля класса.

Содержит ряд методов, которые выполняют различные операции, вроде получения информации о сотрудниках, заявках, событиях и заявлениях, утверждение полученных заявок, а также создание событий, заявок и заявлений, смена пароля, получение информации о себе. Ниже будет список мотодов:

1. Метод getEmployeeInfo() – в этом методе контроллера происходит получение данных о авторизированном пользователе, которые содержатся в таблице employee_full_view. Метод findAllByIdLogin() выполняет выборку всех записей из этого представления.

2. Метод getEmployees()– этот метод возвращает список всех пользователей, зарегистрированных в системе. Запрос к базе данных выполняется с использованием метода findAll() из сервиса employeesViewService.

3. Метод getLsRequests() – этот метод получает список запросов на изменение данных (log statements), которые ожидают подтверждения со стороны пользователя. Выборка выполняется с использованием метода findAllByIdApproverAndStatus() из сервиса logStatementViewService.

Параметр `idApprover` указывает на идентификатор пользователя, которому требуется подтверждение изменений, а `status` задает статус запроса (1 - подтвержден, 2 - отклонен, 3 - требуется подтверждение).

4. Метод `getEvents()` – этот метод возвращает список всех событий, связанных с пользователем. Запрос выполняется с использованием метода `findAllByIdRecipient()` из сервиса `eventsViewService`. Параметр `idRecipient` указывает на идентификатор пользователя, для которого запрашиваются события.

5. Метод `setLsApprove()` – этот метод позволяет подтвердить или отклонить заявление сотрудника, обрабатывает POST-запрос в котором содержится объект `LogStatementRequest`, который содержит новый статус.

6. Метод `getTasks()` – этот метод обрабатывает GET-запрос на получение списка задач. Использует `id` для вызова метода `findAllByIdExecutor` у `tasksViewService`.

7. Метод `createEvent()` – этот метод создает новое событие, используя объект `EventRequest`, который содержит данные, необходимые для создания события.

8. Метод `createNoticeEvent()` – этот метод создает связь между событием и получателем, используя объект `NoticeEventRequest`, который содержит данные, необходимые для создания подобной таблицы.

9. Метод `createTask()` – этот метод создает новое задание, используя объект `TaskRequest`, который содержит данные, необходимые для создания задания.

10. Метод `changePassword()` – этот метод позволяет пользователю сменить его пароль, используя метод `setPassword()` и кодирует новый пароль с помощью `passwordEncoder`. Он использует объект типа `LoginRequest` в теле запроса, который содержит новый пароль.

10. Метод `changeLS()` – этот метод создает новую заявление, используя объект типа `LogStatementCreateRequest`, который содержит данные о заявлении и, если таковой имеется, то и о документе.

11. Метод `edit()` – этот метод позволяет изменять данные сотрудника, доступен только пользователям с соответствующим уровнем доступа.

Все методы принимают и\или отдают данные в формате JSON.

3.3.7 Классы сущностей

Далее идут классы сущностей, описывающих таблицы в базу данных. Они схожи по структуре и содержанию.

Аннотации `@Entity` и `@Table` указывают, что класс является сущностью JPA и что он будет сопоставлен с соответствующей таблицей в базе данных.

Аннотации Lombok `@Builder`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@Getter` и `@Setter` добавляют готовые методы для

создания объектов, конструкторов без аргументов и с аргументами, геттеров и сеттеров для полей класса.

Аннотация `@JsonIgnoreProperties(ignoreUnknown = true)` указывает на то, что при десериализации JSON-объекта, все неизвестные свойства должны быть проигнорированы, а не вызывать ошибку десериализации. Это означает, что если в JSON-объекте есть дополнительные свойства, которые отсутствуют в Java-объекте, то они будут проигнорированы при десериализации и не будут помещены в Java-объект.

Аннотация `@JsonSerialize` указывает на то, что при сериализации Java-объекта в JSON-объект, должен использоваться определенный способ сериализации по умолчанию.

Аннотация `@Transactional` применяется тогда, когда метод, помеченный аннотацией, вызывается, Spring открывает транзакцию, выполняет метод, и, если метод завершается успешно, фиксирует транзакцию. Если же метод завершается неудачно (например, возникает исключение), транзакция откатывается, и все изменения в базе данных, сделанные в рамках этой транзакции, отменяются.

Также в классах описываются связи между таблицами с помощью аннотаций `@OneToMany`, `@ManyToOne` и `@OneToOne`. Ниже перечислены классы-сущности:

- класс `Department`;
- класс `Document`;
- класс `Employee`;
- класс `EmployeeFullView`;
- класс `EmployeesView`;
- класс `EmployeeTask`;
- класс `Event`;
- класс `EventsView`;
- класс `JobEmployee`;
- класс `JobTitle`;
- класс `LogStatement`;
- класс `LogStatementsView`;
- класс `TasksView`;
- класс `JobTitleView`;
- класс `NoticeEvent`;
- класс `Personal`;
- класс `Task`;
- класс `User`.

Подробнее рассмотрим сущность `User`. Она представляет собой таблицу `Login` из базы данных. Данная сущность нужна для реализации авторизации пользователя в системе, потому что позволяет манипулировать данными сотрудника и в принципе позволяет осуществлять связь конкретного пользователя с базой данных.

Класс `User` также реализует интерфейс `UserDetails` из `Spring Security`, который содержит методы для получения информации о пользователе, используемой при аутентификации и авторизации пользователей в приложении.

В частности, методы `getAuthorities()`, `getPassword()`, `getUsername()` используются для проверки прав доступа пользователей в системе, а методы `isAccountNonExpired()`, `isAccountNonLocked()`, `isCredentialsNonExpired()`, `isEnabled()` позволяют проверять статусы учетной записи пользователя.

Также, этот класс определяет соответствующие поля и методы для работы с базой данных, используя аннотации JPA.

3.3.8 Интерфейсы репозиториев

Теперь рассмотрим интерфейсы, являющиеся репозиториями для сущностей, описанных ранее. Они наследуются от `JpaRepository<Repo, Long>`, что позволяет ему использовать стандартные методы доступа к данным (CRUD), такие как сохранение, обновление, удаление, поиск и так далее.

Ниже перечислены все интерфейсы для классов-сущностей: Данные интерфейсы предоставляют методы для выполнения операций чтения/записи данных. Классы сервиса используют эти методы репозитория для выполнения бизнес-логики и обработки запросов от контроллера, а затем возвращают результат контроллеру для отображения в пользовательском интерфейсе.

Таким образом, классы сервисов и репозиториев взаимодействуют друг с другом, обеспечивая разделение ответственностей между слоями приложения и упрощая его тестирование и сопровождение.

Аннотация `@Repository` указывает, что интерфейс является репозиторием, который может управлять базой данных и извлекать из нее данные.

- интерфейс `DepartmentRepo`;
- интерфейс `DocumentRepo`;
- интерфейс `EmployeeFullViewRepo`;
- интерфейс `EmployeeRepo`;
- интерфейс `EmployeesViewRepo`;
- интерфейс `EmployeeTaskRepo`;
- интерфейс `EventRepo`;
- интерфейс `EventsViewRepo`;
- интерфейс `JobEmployeeRepo`;
- интерфейс `JobTitleRepo`;
- интерфейс `LogStatementRepo`;
- интерфейс `LogStatementsViewRepo`;
- интерфейс `TasksViewRepo`;

- интерфейс `JobTitleViewRepo`;
- интерфейс `NoticeEventRepo`;
- интерфейс `PersonalRepo`;
- интерфейс `TaskRepo`;
- интерфейс `UserRepo`.

Благодаря подобным репозиториям создаются запросы к сущностям прямо из имени метода, используя специальные префиксы типа `find…By`, `read…By`, `query…By`, `count…By`, и `get…By` и тому подобные. Кроме них можно использовать дополнительные выражения, которые усложняют запросы и уточняют выборку необходимых данных.

3.3.9 Классы сервиса

Теперь рассмотрим классы сервиса. Данные классы представляют сервисные компоненты, которые предназначены для работы с сущностями в системе. Они использует интерфейсы репозиториев для доступа к базе данных и предоставляют методы для выполнения операций с сущностями, такие как сохранение, обновление, удаление и получение данных.

Далее будут представлен список всех классов, используемых для сервиса, которые связаны с сущностями:

- класс `DepartmentService`;
- класс `DocumentService`;
- класс `EmployeeService`;
- класс `EmployeeFullViewService`;
- класс `EmployeesService`;
- класс `EmployeeTaskService`;
- класс `EventService`;
- класс `EventsViewService`;
- класс `JobEmployeeService`;
- класс `JobTitleService`;
- класс `JobTitleViewService`;
- класс `LogStatementService`;
- класс `LogStatementsViewService`;
- класс `NoticeEventService`;
- класс `PersonalService`;
- класс `TaskService`;
- класс `TasksViewService`;
- класс `UserService`.

Кроме вышеперечисленных классов имеется два специальных класса сервиса `JwtService` и `AuthenticationService`. Классы рассмотрим отдельными пунктами.

3.3.10 Класс `JwtService`

Этот класс предоставляет функционал для генерации и проверки JSON Web Tokens (JWT), которые используются для аутентификации пользователей в приложениях.

Данный класс содержит следующие методы:

- `extractUsername(jwToken)` – извлекает имя пользователя из JWT;
- `extractClaim(jwToken, claimsResolver)` – извлекает любое утверждение из JWT, используя переданный функциональный интерфейс `claimsResolver`;
- `generateToken(userDetails)` – генерирует JWT для пользователя `userDetails`;
- `isTokenValid(jwToken, userDetails)` – проверяет, действителен ли JWT для пользователя `userDetails`;
- `isTokenExpired(jwToken)` – проверяет, истекло ли время жизни JWT;
- `extractExpiration(jwToken)` – извлекает дату истечения срока действия JWT;
- `extractAllClaims(jwToken)` – извлекает все утверждения из JWT;
- `generateToken(extraClaims, userDetails)` – генерирует JWT с переданными дополнительными утверждениями `extraClaims` для пользователя `userDetails`.

Для работы с JWT используется библиотека JSON Web Token (io.jsonwebtoken) и алгоритм подписи HS256 (используется ключ, заданный в поле `SECRET_KEY`).

3.3.11 Класс `AuthenticationService`

Этот класс представляет собой сервис, который предоставляет функциональность регистрации и аутентификации пользователей в системе.

Аннотация `@Service` указывает, что этот класс является сервисом и должен быть управляемым Spring контейнером.

Класс имеет четыре поля, приведенных ниже:

- `private final UserRepo userRepo`;
- `private final PasswordEncoder passwordEncoder`;
- `private final JwtService jwtService`;
- `private final AuthenticationManager authenticationManager`.

Методы класса:

- `register()` выполняет регистрацию нового пользователя в системе;
- `authenticate()` выполняет аутентификацию пользователя в системе.

Таким образом, этот класс предоставляет функциональность регистрации и аутентификации пользователей в системе, используя Spring Security и JSON Web Token (JWT).

3.3.12 Перечисление `Role`

Данное перечисление представляет собой список возможных ролей пользователей системы, которые могут быть назначены сотрудникам. Константы ADMIN и USER определяют две роли: администратор и обычный пользователь.

3.3.13 Класс `AuthenticationRequest`

Этот класс представляет собой модель данных для запроса аутентификации пользователя в системе.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Класс определяет структуру запроса на аутентификацию пользователя в системе и используется для передачи данных между клиентским и серверным приложениями. Имеет поля `private String login` и `String password`.

Эти поля нужны для представления данных, передаваемых для аутентификации пользователя.

3.3.14 Класс `AuthenticationResponce`

Этот класс представляет собой модель данных для ответа на запрос аутентификации пользователя в системе.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Класс, как и класс, приведенный выше, также определяет структуру ответа на запрос аутентификации пользователя в системе и используется для передачи данных между серверным и клиентским приложениями.

Имеет всего одно поле:

- `private String token`.

Токен является строкой, которая используется для идентификации пользователя на сервере и доступа к защищенным ресурсам.

3.3.15 Класс `RegisterRequest`

Класс используется как часть процесса регистрации нового пользователя в системе. По сути, этот класс является моделью данных (data model), представляющей структуру запроса на регистрацию нового пользователя. В данном классе используются аннотации фреймворка Lombok

– `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private String firstName;`
- `private String middleName;`
- `private String lastName;`
- `private String login;`
- `private String password;`
- `private String mail.`

Поля этого класса описывают ту информацию, которую можно внести в запрос при регистрации пользователя: пароль, логин и мейл, а также фамилию, имя и отчество. Чтобы зарегистрировать пользователя сначала необходимо заполнить всю информацию в базе данных через отдел кадров.

3.3.16 Класс `EditRequest`

Этот класс представляет объект запроса на изменение данных пользователя. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером в процессе обновления профиля пользователя.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private String firstName;`
- `private String middleName;`
- `private String lastName;`
- `private Date birthD;`
- `private String role;`
- `private long workNumber;`
- `private long personalNumber;`
- `private String locationStreet;`
- `private String cabinetOffice.`

Поля этого класса описывают ту информацию, которую можно внести в запрос при изменении данных сотрудника: фамилию, имя и отчество, день рождения, роль, рабочий и персональный номера, рабочий адрес, и рабочий кабинет и корпус.

3.3.17 Класс `LoginRequest`

Этот класс представляет объект запроса для смены пароля пользователя в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

Содержит всего одно поле – `private String password`, которое содержит новый пароль.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.18 Класс `EventRequest`

Этот класс представляет объект запроса для создания нового события в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private String typeOfEvent`;
- `private String commentFe`;
- `private Date dateOfEvent`.

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании события: тип события, комментарий к нему и дата самого события.

3.3.19 Класс `LogStatementCreateRequest`

Этот класс представляет объект запроса для создания нового заявления для пользователя в системе. Содержит поля, которые могут быть изменены, включая поле `private String bodyDoc`, предназначенное для документа, если таковой имеется. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private long idApprover;`
- `private String commentLs;`
- `private int daysSum;`
- `private int typeLeave;`
- `private Date dateLeave;`
- `private Date dateOfLs;`
- `private String bodyDoc.`

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании заявления: сотрудника, который сможет подтвердить или опровергнуть заявление, комментарий, количество дней, тип заявления, дату начала действия заявления, дату создания заявления, и возможность прикрепить скан физической копии документа, если таковая имеется или если она необходима.

3.3.20 Класс `LogStatementRequest`

Этот класс представляет объект запроса для изменения статуса заявления в системе. Содержит одно поле, которое может быть изменено – `private int status`, предназначенное для указания статуса заявления. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.21 Класс `NoticeEventRequest`

Этот класс представляет объект запроса для создания связи события с тем сотрудником, которого нужно оповестить. Содержит одно поле, которое может быть изменено – `private long recipientId`, предназначенное для указания статуса заявления. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

3.3.22 Класс `TaskRequest`

Этот класс представляет объект запроса для создания нового задания в системе. Содержит поля, которые могут быть изменены. Используется для передачи данных между клиентской частью и сервером.

В данном классе используются аннотации фреймворка Lombok – `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`. Они позволяют автоматически генерировать стандартные методы JavaBean, такие как `toString()`, `equals()`, `hashCode()` и геттеры/сеттеры для всех полей класса, что сокращает количество необходимого для написания кода.

Имеет поля, приведенные ниже:

- `private Date dateTask`;
- `private String nameOfTask`;
- `private String commentTe`;
- `private long idExecutor`.

Поля этого класса описывают ту информацию, которую можно внести в запрос при создании задания: дату, суть задания, комментарий и пользователя, на которого будет назначено само задание.