

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

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.

#### 3.1 Описание структуры приложения

Для разработки серверной части программного продукта были 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 репозиторий 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 реализована на Java. Исходя из гексагональной архитектуры, в приложении классы делятся по

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

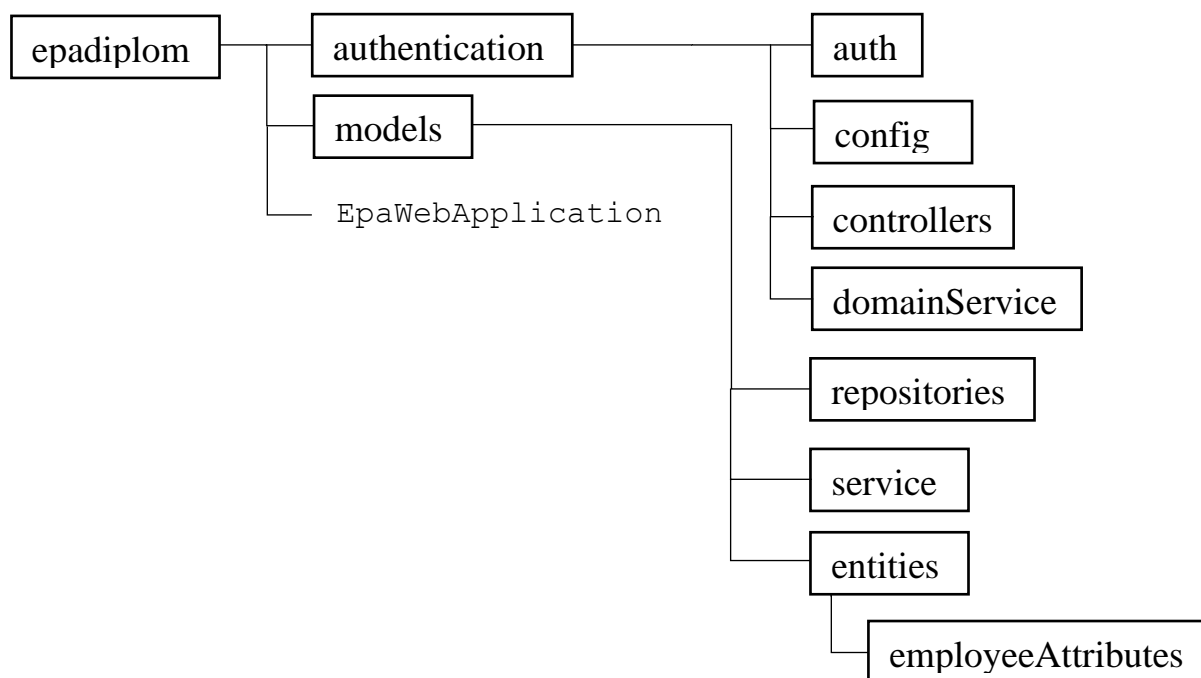


Рисунок 3.1 – Структура пакетов в веб-приложении

1. Пакет `authentication` – пакет предназначенный для реализации функционала, связанного с аутентификацией пользователей, включая создание аккаунтов, вход в систему и другие смежные задачи. Он содержит несколько подпакетов, которые обеспечивают логическую группировку функционала по его назначению и упрощают работу с ним. Эти подпакеты рассматриваются ниже:

1.1 Подпакет `auth` – пакет содержащий классы, которые отвечают за реализацию процесса аутентификации и авторизации пользователей в системе. Здесь находятся реализации функций создания аккаунтов, входа в систему и других задач, связанных с проверкой личности пользователя. Таким образом, данный пакет является важным компонентом системы безопасности и гарантирует правильный доступ к ресурсам системы только авторизованным пользователям. В него входят классы, перечисленные ниже:

- `authenticationRequest`;
- `authenticationResponse`;
- `authenticationService`;
- `registerRequest`.

1.2 Подпакет `config` – подпакет 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:

- applicationConfig;
- jwtAuthenticationFilter;
- securityConfig.

1.3 Подпакет controllers – подпакет 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:

- authenticationController;
- mainPageController.

1.4 Подпакет domainsService – подпакет, с помощью которого происходит управление JSON Web Token (JWT), реализуемом в данном приложении, для создания своеобразных ключей, которые помогают при взаимодействии клиента и сервера. Содержит в себе класс:

- jwtService.

2. Пакет models – пакет, представляющий основную модель данных системы, и содержит все компоненты, связанные с управлением сущностями и взаимодействием с базой данных. В нем находятся несколько подпакетов, отвечающих за доступ к данным, их обработку и сохранение в базе данных. Этот пакет можно назвать прослойкой между моделью данных и базой данных, так как он обеспечивает связь между ними. Кроме того, данный пакет является ключевым компонентом системы, так как представляет основную модель данных и определяет структуру и взаимосвязи между сущностями.

2.1 Подпакет entities – подпакет содержащий сущности базы данных, которые используются вместе с подпакетами service и repositories для реализации бизнес-логики приложения. Он включает в себя 16 классов, из которых четыре 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.

2.1.1. Пакет в подпакете 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.

2.2 Подпакет `repositories` – подпакет содержащий интерфейсы, которые используются для взаимодействия с базой данных с помощью репозитория, таких как `JpaRepository`. Это упрощает создание запросов к таблицам и в сочетании с пакетами `service` и `entities` позволяет реализовывать бизнес-логику приложения.

2.3 Подпакет `service` – подпакет содержащий классы, связанные с сервисом сущностей таблицы. 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.

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

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 PostgreSQL. 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).

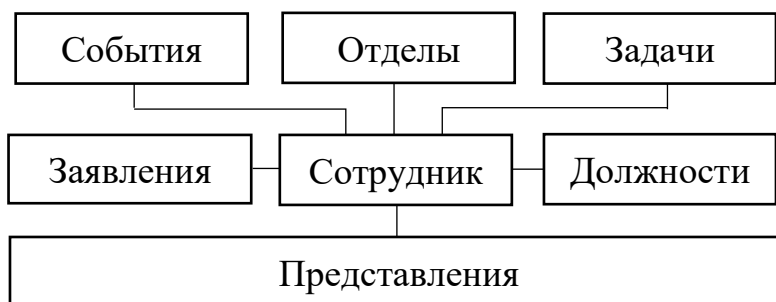


Рисунок 3.2 – Условное логическое разделение блоков БД

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 – зашифрованный 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 (касается 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, `timestamp without time zone`;
- `entry_d` – дата устройства на работу (в этот же день должен быть и создан аккаунт), `timestamp without time zone`;
- `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. Формат данных `timestamp without time zone` означает, что нет привязки к часовому поясу. Это сделано, чтобы не было разногласий во времени.

### 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), `timestamp without time zone`;
- `date_of_ls` – дата, когда работник составляет заявление, `timestamp without time zone`;
- `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`;
- `approve` – 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. Формат данных `timestamp without time zone` означает, что нет привязки к часовому поясу. Это сделано, чтобы не было разногласий во времени.

### 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, `timestamp without time zone`;
- `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. Формат данных `timestamp without time zone` означает, что нет привязки к часовому поясу. Это сделано, чтобы не было разногласий во времени.

### 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, `timestamp without time zone`;
- `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.approve;
- 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, касаемых 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`.

## 3.3 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 касается 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 `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.

### **3.3.3 Класс `AuthenticationResponse`**

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.

### **3.3.4 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).

### 3.3.5 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 password;`
- `private String mail.`

### 3.3.6 Класс `ApplicationConfig`

Класс представляет собой конфигурационный класс Spring, который содержит конфигурацию для аутентификации пользователей в системе. В классе определены следующие методы, 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.

### 3.3.7 Класс `JwtAuthenticationFilter`

Данный код 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 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.

### 3.3.8 Класс `SecurityConfig`

Этот класс содержит конфигурацию Spring Security для веб-приложения. Он использует аннотации Spring `@Configuration` и `@EnableWebSecurity`, чтобы сообщить Spring, что этот класс содержит конфигурацию безопасности для веб-приложения. Данный класс содержит метод:

- `securityFilterChain` – метод, который создает цепочку фильтров безопасности.

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.

### 3.3.9 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("/api/v1/auth")` указывает на корневой путь, который будет использоваться для обработки запросов, обрабатываемых этим контроллером.

В классе есть несколько методов, которые рассмотрены ниже:

1. Метод `register` – он обрабатывает POST-запросы на `/api/v1/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 `/api/v1/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. Метод `sayHello` – он обрабатывает GET-запросы на `/api/v1/auth/authorization`. Он возвращает строку «Hello from secured endpoint», что означает успешное прохождение аутентификации и авторизации пользователем. Он используется для проверки работоспособности механизма аутентификации и авторизации.

### 3.3.10 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 репозиторий 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 ряд методов, которые будут рассмотрены ниже:

1. Метод `getEmployeeInfo()` – этот метод контроллера происходит получение данных о залогиненном пользователе, которые содержатся в таблице `employee_full_view`. Метод `findAllByIdLogin()` выполняет выборку всех записей из этой таблицы для залогиненного пользователя.

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



3. Метод `getLsRequests()` – этот метод получает список запросов на изменение данных (`log statements`), которые ожидают подтверждения со стороны пользователя. Выборка выполняется с использованием метода `findAllByIdApproverAndApprove()` из репозитория `logStatementViewRepo`. Параметр `idApprover` указывает на идентификатор пользователя, которому требуется подтверждение изменений, а `approve` 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 репозитория `eventsViewRepo`. Параметр `idRecipient` указывает на идентификатор пользователя, для которого запрашиваются события.

### 3.3.11 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`).

### 3.3.12 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 Классы сущностей

Далее идут классы сущностей, описывающих таблицы в базу данных. Они схожи по структуре и содержанию. Аннотация @Entity сообщает JPA, что данный класс является сущностью, которая будет отображаться на таблицу в базе данных. Аннотация @Table используется для указания имени таблицы, аннотация @NoArgsConstructor генерирует конструктор без параметров, а аннотация @Getter генерирует геттеры для всех полей класса. 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 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.

### 3.3.14 Интерфейсы репозитория

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. Далее перечислены все интерфейсы для классов-сущностей:

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

### 3.3.15 Классы сервиса

Теперь рассмотрим классы сервиса. Эти классы являются слоем сервиса и отвечает за бизнес-логику, связанную с сущностями из подраздела 3.3.13. В них содержатся методы, которые могут вызываться из контроллеров для обработки запросов, связанных с их сущностями, такие как создание, чтение, обновление и удаление, а также другие методы, связанные с бизнес-логикой. Сервисный слой использует репозиторий для доступа к данным и предоставляет абстракцию и контроль над данными, что позволяет легче модифицировать бизнес-логику и масштабировать приложение. Далее будут представлен список всех классов, используемых для сервиса:

`DepartmentService,`                      `DocumentService,`                      `EmployeeService,`  
`EmployeeTaskService,`                      `EventService,`                      `JobEmployeeService,`  
`JobTitleService,`                      `LogStatementService,`                      `NoticeEventService,`  
`PersonalService,` `TaskService,` `UserService.`