Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Инженерно-экономический факультет

Кафедра экономической информатики

Дисциплина: Распределенные информационные системы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ОНЛАЙН-СИСТЕМА ПО ПОИСКУ РАБОТЫ

БГУИР КП 1-40 01 02-02 301 ПЗ 126

Студент: гр. 572301 Шевцов В. И.

Руководитель: Хомяков П. В.

Минск 2018

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc513452215)

[1 Описание онлайн-системы по поиску работы 7](#_Toc513452216)

[2 Описание основных процессов поиска работы 10](#_Toc513452217)

[3 Спецификация вариантов использования системы поиска работы 18](#_Toc513452218)

[4 Информационная модель системы по поиску работы и её описание 23](#_Toc513452219)

[5 Обоснование выбора компонентов и технологий 26](#_Toc513452220)

[5.1 Диаграмма последовательностей 29](#_Toc513452221)

[5.2 Диаграмма компонентов 30](#_Toc513452222)

[5.3 Диаграмма развёртывания 31](#_Toc513452223)

[6. Модели представления системы и их описание 33](#_Toc513452224)

[6.1 Диаграмма состояний 33](#_Toc513452225)

[6.2 Описание алгоритма регистрации пользователя 33](#_Toc513452226)

[6.3 Описание алгоритма отправления электронного сообщения HR-у от кандидата 34](#_Toc513452227)

[7 Описание применения паттернов проектирования 37](#_Toc513452228)

[7.1 Паттерн «Одиночка» (Singleton) 37](#_Toc513452229)

[7.2 Паттерн «Заместитель» (Proxy) 38](#_Toc513452230)

[7.3 Паттерн «Команда» (Command) 39](#_Toc513452231)

[8 Руководство по развертыванию системы 40](#_Toc513452232)

[9 Результаты тестирования системы 43](#_Toc513452233)

[Заключение 60](#_Toc513452234)

[Список использованных источников 61](#_Toc513452235)

[Приложение А (обязательное) Диаграммы классов 62](#_Toc513452236)

[Приложение Б (обязательное) Листинг скрипта генерации базы данных 78](#_Toc513452238)

[Приложение В (обязательное) Листинг основных элементов программы 83](#_Toc513452241)

# ВВЕДЕНИЕ

В современном обществе большинство людей работает в какой-то сфере деятельности, в то же время и многие ищут работу.

Так зачем же человеку нужна работа?

Работа нужна человеку, чтобы жить – зарабатывать деньги на своё существование: на еду, на оплату коммунальных услуг, на одежду, медицинские услуги, учёбу, на удовлетворение культурных потребностей. Но работа – это не только источник денег.

Благодаря работе, человек может удовлетворить свою потребность в самореализации, а нередко и в самовыражении. Безделье же ведёт к личностной и физической деградации, выбивает человека из системы социальных отношений, мешает ему вести образ жизни, нужный для сохранения физиологического и психологического равновесия [1].

Также работа может помочь вам:

* проявить индивидуальные качества;
* обрести статус, вы можете показать окружающим кто вы такой, что собой представляете;
* овладеть новыми знаниями и обогатить свой опыт;
* научиться решать различного рода проблемы, задачи;
* обрести новых друзей, завести полезные знакомства [2].

Поиск работы, соответствующей вашим профессиональным навыкам, интересам и темпераменту, является немаловажной частью жизни каждого человека, ведь от этого зависит его ближайшее будущее. Поэтому существует множество способов и сервисов по поиску работы, которые оказывают помощь людям по данному вопросу. Однако большинство из них требуют траты большого количества ресурсов человека, а особенно времени, объявления о старых и неактуальных вакансиях остаются, существует зависимость от каких-либо посредников, предоставляющих услуги по поиску работы. Всего этого хотелось бы избежать.

Цель данного курсового проекта – оптимизация поиска работы путем обеспечения удобства при составлении вакансий и резюме, оперативного отображения достоверной информации, её постоянного обновления и редактирования, предоставления необходимых условий и обеспечения быстродействия между менеджерами по персоналу и потенциальными работниками, затрачивая при этом минимальное количество средств и времени, что позволит быстро решить проблемы, связанные с ускорением бизнес-процессов данной системы.

Для достижения поставленной цели необходимо выполнить ряд задач:

* изучить процесс онлайн-поиска работы;
* определить набор функций, обеспечивающих эффективную работу программного продукта;
* построить функциональную модель данного процесса;
* рассмотреть и построить модели представления данной системы;
* реализовать программный продукт с удобным и интуитивно понятным интерфейсом, реализующий тематическую бизнес логику, а также взаимодействующий с базой данных;
* протестировать полученный программный продукт для проверки его корректной работы.

# 1 ОПИСАНИЕ ОНЛАЙН-СИСТЕМЫ ПО ПОИСКУ РАБОТЫ

Актуальность поиска работы в наше время не только не снижается, если сравнивать с предыдущими десятилетиями, но, наоборот, становится всё выше. Одни люди увольняются, других увольняют, одни хотят сменить место работы по какой-то причине, другие просто встают перед необходимостью перемен.

Но если раньше все использовали классические методы: прямое обращение в фирму, поиск объявлений в газетах и сми, обращение в кадровое агентство или центр занятости населения, то сегодня многие способы можно смело вычёркивать, ведь в распоряжении людей есть интернет, что позволяет искать работу онлайн. Преимуществом онлайн режима является то, что информация обновляется практически каждую минуту, а это, в свою очередь, дает вам возможность в числе первых оказаться в курсе наиболее привлекательных вакансий [3].

Поэтому онлайн-система по поиску работы является необходимой, ведь она обеспечит потенциальных работников всеми средствами для рационального и быстрого поиска интересующих вакансий, а также поможет менеджерам по персоналу находить необходимые кадры для привлечения их в компанию в более короткие сроки.

Разрабатываемая система предназначена для автоматизации процесса поиска работы.

При укрупненном рассмотрении данного процесса можно выделить двух субъектов системы, работающих с информацией – менеджер по персоналу и соискатель. Объектами системы являются хранимые в ней сущности. Для организации информационного потока в системе служат различные инструменты, например, параметры поиска, права доступа различных субъектов, спецификации для хранимой информации и т.п. Таким образом, глобальную задачу можно представить как совокупность более мелких, но взаимосвязанных задач:

* администрирование системы;
* ввод, хранение и обработка информации о вакансиях;
* ввод, хранение и обработка информации о соискателях работы;
* предоставление средств поиска и отображения информации;
* предоставление средств общения менеджеров по персоналу и соискателей.

Администрирование системы подразумевает управление всей системы в целом. Администратор системы имеет возможность получать отчетную информацию о списке текущих вакансий и резюме, а также итоговые результаты по собеседованию. Удаление вакансий, резюме, кандидатов также входят в его функционал.

Ввод, хранение и обработка информации о вакансиях рассматривается для менеджеров по персоналу. Эта часть системы предназначена для сбора и обработки информации о вакансиях, имеющихся у менеджеров по персоналу. Данные о вакансиях автоматически запрашиваются у менеджера по персоналу после инициации им запроса на ввод новой вакансии или на редактирование существующей, далее путем заполнения соответствующих форм и полей, формируется информационная запись, а далее менеджер по персоналу инициирует ее сохранение в базе данных или отменяет свои действия. Менеджер по персоналу также может отслеживать информацию о кандидате, а также фиксировать результаты собеседований с ним после отклика на существующую вакансию.

Часть системы, относящейся к вводу, хранению и обработке информации о соискателях работы предназначена для сбора и обработки информации о людях, находящихся в поиске работы. Данные автоматически запрашиваются у соискателя после инициации им запроса на ввод нового резюме или на редактирование существующего, далее путем заполнения соответствующих форм и полей, формируется информационная запись, соискатель инициирует ее сохранение в базе данных или отменяет свои действия. Соискатель может откликнуться на существующую вакансию, и менеджер по персоналу сможет оперативно увидеть всю необходимую информацию о кандидате для принятия последующих действий .

Система имеет в наличии возможность поиска информации. Имеется специальное поле для ввода текстовой строки. После инициации поиска, система выводит всю информацию содержащую данную строку.

Система предоставляет средство для возможности общения менеджеров по персоналу и соискателей посредством электронной почты. Соискатель имеет возможность написать менеджеру по персоналу на его электронную почту для получения ответа на соответствующие вопросы.

# 2 ОПИСАНИЕ ОСНОВНЫХ ПРОЦЕССОВ ПОИСКА РАБОТЫ

Ключевой процесс в данной теме – поиск работы. Для создания функциональной модели, которая дала бы нам более детальное представление о способе автоматизации данного процесса, был выбран стандарт IDEF0.

IDEF0 – методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов.

На контекстной диаграмме верхнего уровня (рисунок 2.1) представлена функциональная модель «Выполнить поиск работы», в которой определены потоки входных и выходных данных, механизмы выполнения процесса, а также управление данными.

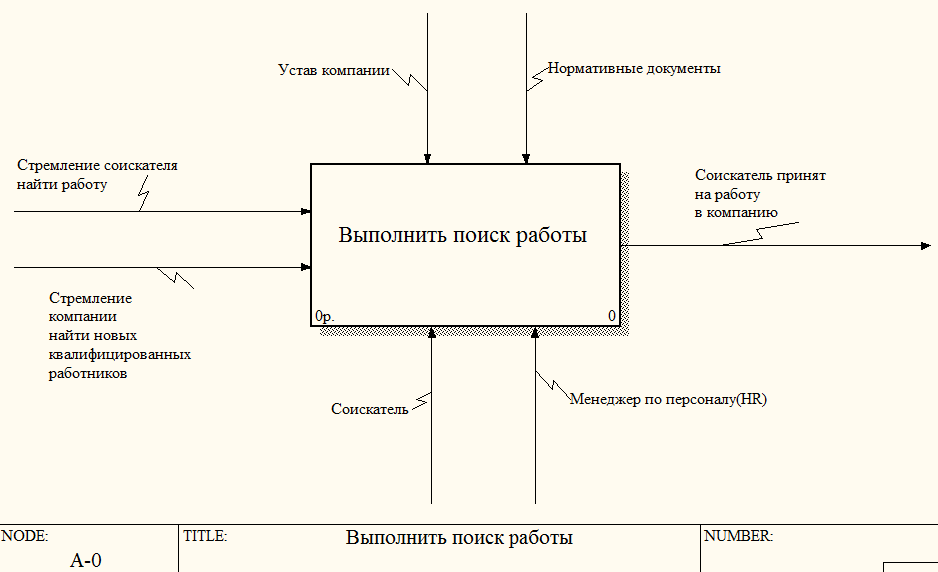


Рисунок 2.1 – Контекстная диаграмма верхнего уровня

В качестве входного параметра выступают стремление соискателя найти работу и стремление компании найти новых квалифицированных работников, в роли механизмов выступают соискатель и менеджер по персоналу(HR), внешним управлением в процессе поиска работы являются устав компании и нормативные документы. В качестве выходных данных после прохождения всего процесса получаем тот факт, что соискатель принят на работу в компанию.

Декомпозиция данной модели представляет собой разбиение процесса «Выполнить поиск работы» на 4 подпроцесса и представлена на рисунке 2.2.

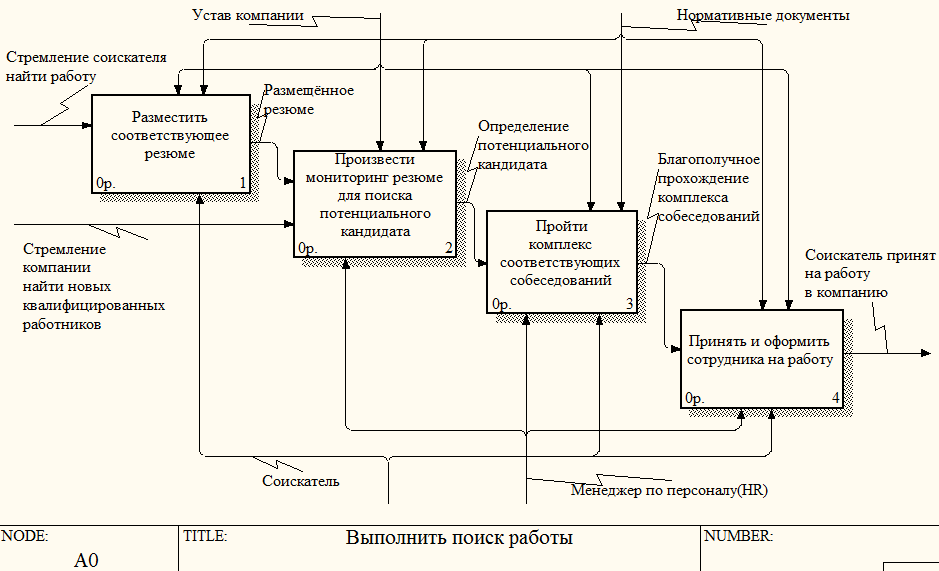


Рисунок 2.2 – Декомпозиция контекстной диаграммы верхнего уровня

Первый компонент данной декомпозиции («Разместить соответствующее резюме») подразумевает, что соискатель выберет наиболее подходящую должность, правильно составит и разместит своё резюме, отражающее его личностные и профессиональные навыки и способности, которые могут заинтересовать менеджера по персоналу какой-либо компании.

Второй компонент декомпозиции контекстной диаграммы («Произвести мониторинг резюме для поиска потенциального кандидата») включает в себя размещение вакансии, на которую необходим специалист, мониторинг существующих резюме и просмотр откликнувшихся на данную вакансию людей для последующего изучения их резюме и установлении контакта для дальнейших действий.

Третий компонент данной декомпозиции («Пройти комплекс соответствующих собеседований») подразумевает, что у менеджера по персоналу есть потенциальный кандидат на соответствующую должность, с которым необходимо назначить и провести комплекс собеседований для выявления профессиональной пригодности соискателя.

Четвертый компонент данной декомпозиции («Принять и оформить сотрудника на работу») подразумевает, что соискатель успешно прошел весь комплекс собеседований, произвёл впечатление на определённых сотрудников компании и будет принят в данную компанию.

Разберём каждый из этих компонентов.

Декомпозиция процесса «Разместить соответствующее резюме» показана на следующей странице.

Данный процесс включает в себя:

* выбор подходящей должности;
* составление соответствующего резюме;
* размещение составленного резюме.

Декомпозиция данного процесса представлена на рисунке 2.3.

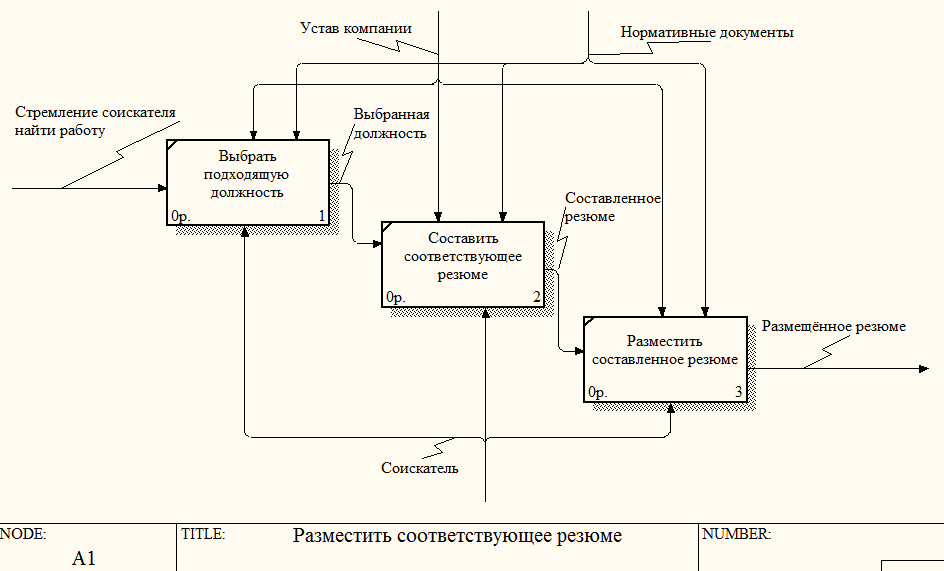


Рисунок 2.3 – Декомпозиция процесса «Разместить соответствующее резюме»

На следующем рисунке представлена декомпозиция процесса «Произвести мониторинг резюме для поиска потенциального кандидата», которая включает в себя следующие подпроцессы:

* размещение объявления о вакансии;
* получение отклика на вакансию или поиск потенциального кандидата;
* изучение резюме потенциального кандидата;
* установление связи с соискателем и обсуждение дальнейших действий.

Декомпозиция данного процесса представлена на рисунке 2.4.



Рисунок 2.4 – Декомпозиция процесса «Произвести мониторинг резюме для поиска потенциального кандидата»

Далее представлена декомпозиция процесса «Изучить резюме потенциального кандидата» вышестоящего процесса «Произвести мониторинг резюме для поиска потенциального кандидата»

Данный процесс включает в себя:

* просмотр общей информации о кандидате;
* просмотр информации о языковых навыках кандидата;
* просмотр информации о технических навыках кандидата;
* просмотр информации об опыте работы кандидата.

Декомпозиция данного процесса представлена на рисунке 2.5.

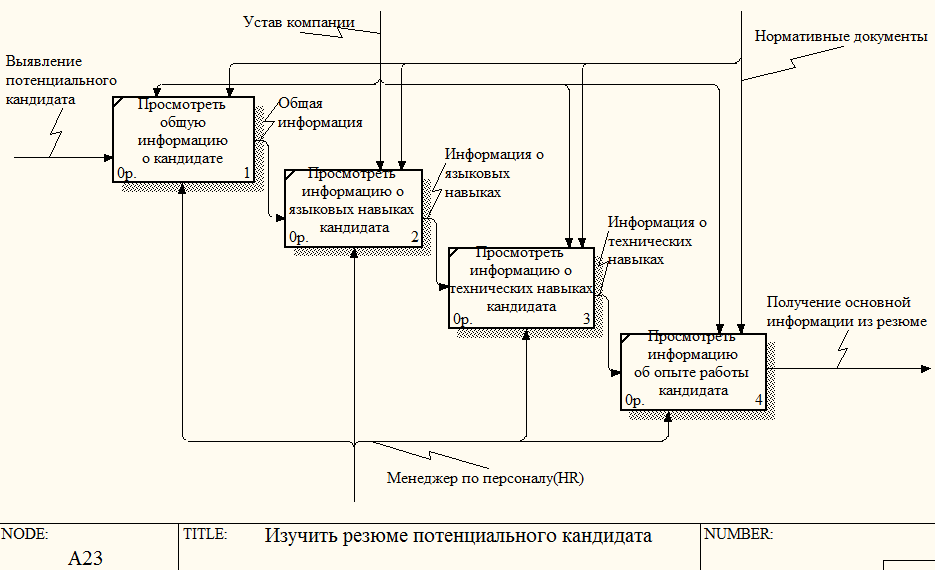


Рисунок 2.5 – Декомпозиция процесса «Изучить резюме потенциального кандидата»

Далее будет показана декомпозиция процесса «Пройти комплекс соответствующих собеседований».

Данный процесс включает в себя:

* прохождение телефонного собеседования;
* прохождение очного собеседования;
* прохождение языкового и технического собеседования.

Декомпозиция данного процесса представлена на рисунке 2.6.

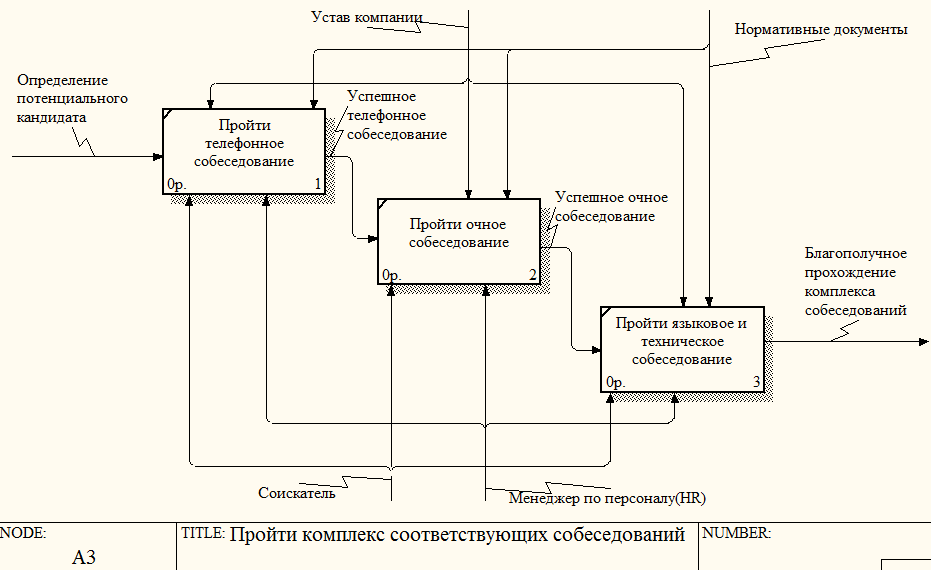


Рисунок 2.6 – Декомпозиция процесса «Пройти комплекс соответствующих собеседований»

И в заключении, декомпозиция процесса «Принять и оформить сотрудника на работу» показана далее.

Данный процесс включает в себя:

* утверждение соискателя в новой должности;
* занесение соискателя в базу сотрудников;
* оформление оставшихся бумаг.

Декомпозиция данного процесса представлена на рисунке 2.7.

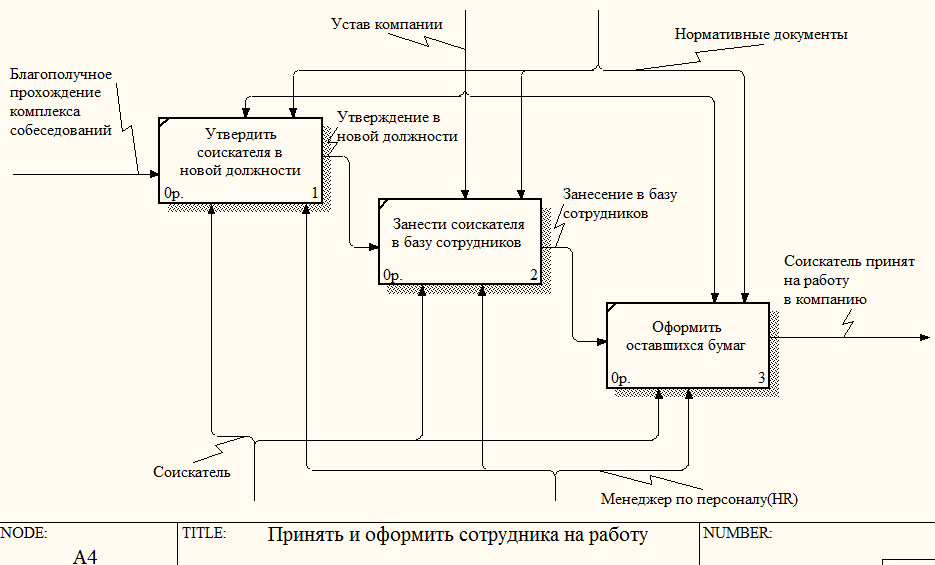


Рисунок 2.7 – Декомпозиция процесса «Принять и оформить сотрудника на работу»

После моделирования бизнес-процессов можно непосредственно начинать проектировать архитектуру будущей системы по автоматизации бизнес-процессов.

Моделирование бизнес-процессов является неотъемлемым этапом в разработке информационных систем.

# 3 СПЕЦИФИКАЦИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ ПОИСКА РАБОТЫ

Диаграмма вариантов использования – UML диаграмма, наглядно демонстрирующая взаимодействие актеров и системы.

В данной диаграмме вариантов использования в роли актёров выступают администратор (рисунок 3.1), соискатель (рисунок 3.2) и менеджер по персоналу HR(рисунок 3.3).

Рассмотрим каждую роль отдельно.

Администратор – главный управляющий системой. Он обладает следующим функционалом: авторизация (прецедент «Авторизация»); возможность выбора русского или английского языка интерфейса (прецедент «Выбрать используемый язык интерфейса»); просмотр информации о компании или людях, участвующих в разработке данного программного продукта (прецедент «Просмотреть информацию о разработчике») и просмотр различной информации, хранящейся в системе (прецедент «Просмотреть информацию»).

От прецедента «Просмотреть информацию» идут связи включения к прецедентам «Просмотреть информацию о соискателях», «Просмотреть информацию о сотрудниках HR», «Просмотреть информацию о вакансиях» и «Просмотреть информацию об интервью».Также соответственно, прецеденты «Удалить информацию о соискателях» и «Редактировать информацию о соискателях» расширяют прецедент «Просмотреть информацию о соискателях», прецеденты «Удалить информацию о сотрудниках HR» и «Редактировать информацию о сотрудниках HR» расширяют прецедент «Просмотреть информацию о сотрудниках HR», прецеденты «Удалить информацию о вакансиях» и «Редактировать информацию о вакансиях» расширяют прецедент «Просмотреть информацию о вакансиях» и прецеденты «Удалить информацию об интервью» и «Редактировать информацию об интервью» расширяют прецедент «Просмотреть информацию об интервью».

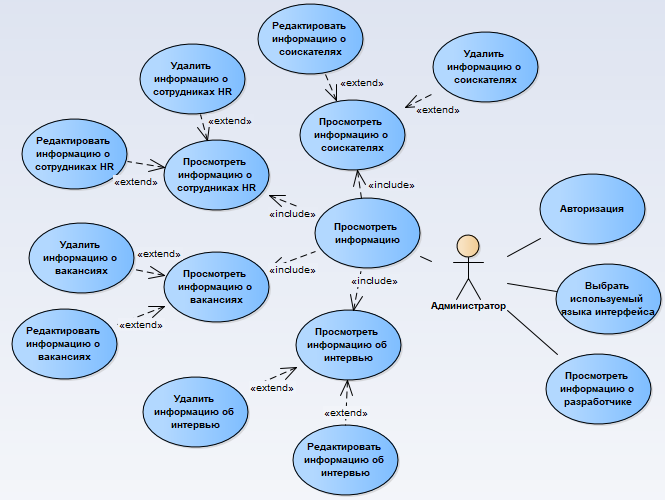


Рисунок 3.1 – Диаграмма вариантов использования онлайн-системы «Поиск работы», где актером является «Администратор»

Соискатель – это физическое или юридическое лицо, чьим стремлением является поиск и устройство на работу. Диаграмма вариантов использования для данной роли представлена на рисунке 3.2.

Данный актер обладает следующим функционалом: авторизация (прецедент «Авторизация»); регистрация (прецедент «Регистрация»); возможность выбора русского или английского языка интерфейса (прецедент «Выбрать используемый язык интерфейса»); просмотр информации о компании или людях, участвующих в разработке данного программного продукта (прецедент «Просмотреть информацию о разработчике»); возможность изменения указанных ранее при регистрации в системе логина и пароля (прецедент «Изменить данные аккаунта»); возможность навсегда удалить свой текущий аккаунт из системы (прецедент «Удалить аккаунт»); просмотр информации о хранящихся в системе вакансиях (прецедент «Просмотреть вакансии») и возможность добавить свое резюме (прецедент «Добавить резюме»).

От прецедента «Регистрация» идёт связь включения к прецеденту «Ввести ReCaptcha». Это является подтверждением, что вы не являетесь роботом. Если не поставить галочку в данном поле, то вы не пройдёте регистрацию. В свою очередь, прецедент «Забыли пароль?» расширяет прецедент «Авторизация», прецеденты «Просмотреть резюме» и «Редактировать резюме» расширяют прецедент «Добавить резюме», прецеденты «Откликнуться на вакансию», «Отправить письмо сотруднику HR» и «Поиск вакансий» расширяют прецедент «Просмотреть вакансии».

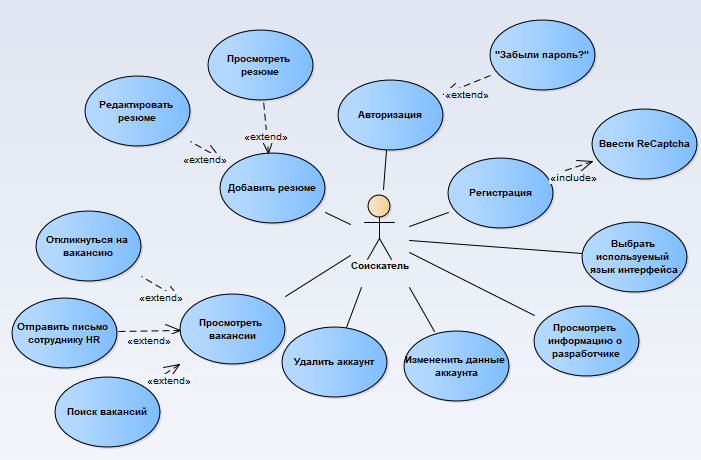


Рисунок 3.2 – Диаграмма вариантов использования онлайн-системы «Поиск работы», где актером является «Соискатель»

Сотрудник HR (HumanResource) – специалист, менеджер по персоналу, в чью специализацию входит подбор персонала, его адаптация, оценка, обучение, мотивация и прочее. Диаграмма вариантов использования для данной роли представлена на рисунке 3.3.

Данный актер обладает следующим функционалом: авторизация (прецедент «Авторизация»); регистрация (прецедент «Регистрация»); возможность выбора русского или английского языка интерфейса (прецедент «Выбрать используемый язык интерфейса»); просмотр информации о компании или людях, участвующих в разработке данного программного продукта (прецедент «Просмотреть информацию о разработчике»); возможность изменения указанных ранее при регистрации в системе логина и пароля (прецедент «Изменить данные аккаунта»); возможность навсегда удалить свой текущий аккаунт из системы (прецедент «Удалить аккаунт»); просмотр информации обо всех хранящихся в системе интервью (прецедент «Просмотреть интервью») и возможность добавить различного рода информацию (прецедент «Добавить информацию»).

От прецедента «Регистрация» идёт связь включения к прецеденту «Ввести ReCaptcha». Это является подтверждением, что вы не являетесь роботом. Если не поставить галочку в данном поле, то вы не пройдёте регистрацию.

В свою очередь, прецедент «Забыли пароль?» расширяет прецедент «Авторизация», прецеденты «Удалить интервью», «Редактировать интервью», «Просмотреть подробную информацию о соискателе» и «Просмотреть подробную информацию о вакансии» расширяют прецедент «Просмотреть интервью». Кроме того, прецедент «Отправить письмо соискателю» расширяет прецедент «Просмотреть подробную информацию о соискателе».

От прецедента «Добавить информацию» идёт связь включения к прецеденту «Добавить личную информацию», который может быть расширен за счёт прецедентов «Просмотреть личную информацию» и «Редактировать личную информацию».

В то же время, прецедент «Добавить вакансии» расширяет прецедент «Добавить информацию». А прецеденты «Просмотреть вакансии», «Редактировать вакансии» и «Удалить вакансии», соответственно, расширяет прецедент «Добавить вакансии».

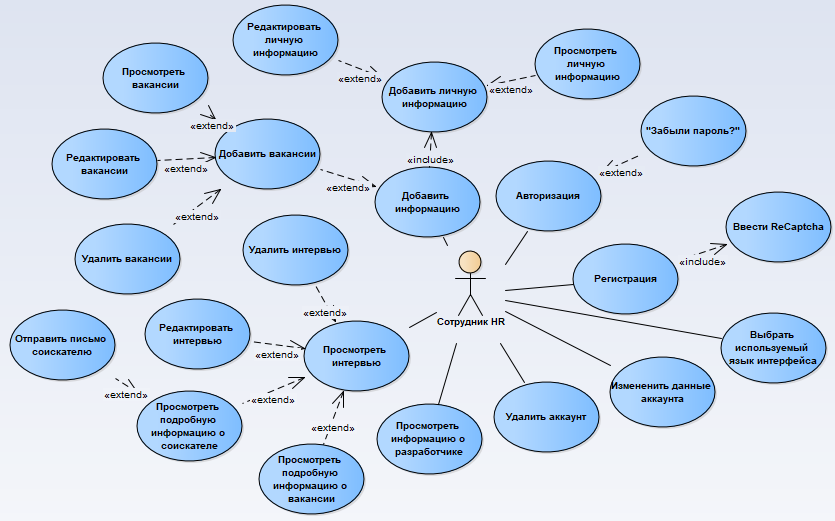


Рисунок 3.3 – Диаграмма вариантов использования онлайн-системы «Поиск работы», где актёром является «Сотрудник HR»

Все существующие роли в данной системы и их функционал были рассмотрены.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ ПО ПОИСКУ РАБОТЫ И ЕЁ ОПИСАНИЕ

Для разработки системы по поиску работы были выделены и созданы следующие сущности:

* информация об аккаунте соискателя и сотрудника HR;
* информация о конкретном соискателе;
* информация о конкретном сотруднике HR;
* информация о вакансиях;
* информация о собеседованиях.

Сущность системы «информация об аккаунте соискателя и сотрудника HR» содержит такие общие сведения об аккаунте, как синтетический идентификационный номер аккаунта, логин, пароль и принадлежность аккаунта к соискателю либо к сотруднику HR соответственно.

Сущность системы «информация о конкретном соискателе» служит для хранения данных о личностных и профессиональных качествах и навыках конкретного соискателя. Список атрибутов включает в себя: синтетический идентификационный номер соискателя, фамилия, имя и отчество соискателя, его возраст, адрес электронной почты, гражданство, номер телефона, желаемая должность, образование, опыт работы, уровень английского языка, его профессиональные навыки, а также идентификационный номер аккаунта. Связь с таблицей «информация об аккаунте соискателя и сотрудника HR» – один-ко-многим.

Сущность системы «информация о конкретном сотруднике HR» содержит личную информацию о конкретных сотрудниках HR. В списке атрибутов имеются такие поля, как синтетический идентификационный номер сотрудника HR, фамилия, имя и отчество, адрес проживания, номер телефона, адрес электронной почты, название компании, в которой работает сотрудник, а также идентификационный номер аккаунта. Связь с таблицей «информация об аккаунте соискателя и сотрудника HR» - один-ко-многим.

Сущность системы «информация о вакансиях» служит для хранения информации о имеющихся вакансиях. В списке атрибутов содержатся: синтетический идентификационный номер вакансии, должность, на которую проводится набор, компания, к которой относится данная открытая вакансия, приблизительный уровень заработной платы, который будет получать будущий работник, утверждённый в этой должности, местоположение, требуемый опыт работы, необходимый уровень английского языка, основная, более подробная текстовая информация о вакансии, состояние вакансии (открыта либо закрыта) и идентификационный номер сотрудника HR, ответственный за курирование данной вакансии. Связь с таблицей «информация о конкретном сотруднике HR» - один-ко-многим.

Сущность системы «информация о собеседованиях» используется для хранения информации, содержащей состояние продвижения соискателя на конкретную вакансию через комплекс собеседований. Список атрибутов состоит из следующих полей: синтетический идентификационный номер собеседования, идентификационный номер соискателя, идентификационный номер вакансии, состояние предварительного собеседования, состояние финального технического собеседования. Связи с таблицей «информация о вакансиях» - один-ко-многим, с таблицей «информация о конкретном соискателе» - один-ко-многим.

Данная модель системы приведена к третьей нормальной форме. Это подтверждают следующие аргументы:

1. Все атрибуты являются простыми, все используемые домены содержат только скалярные значения, вероятность повторения строк в таблице отсутствует (первая нормальная форма).
2. Каждый не ключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа (вторая нормальная форма).
3. Отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых (третья нормальная форма) [5].

В результате получаем информационную модель в графическом виде, которая представлена ниже на рисунке 4.1.

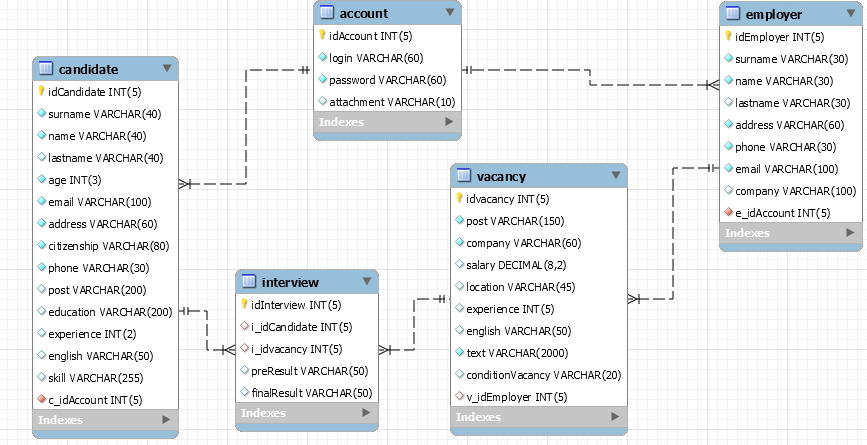


Рисунок 4.1 – Информационная модель базы данных онлайн-системы по поиску работы

В данном разделе была рассмотрена и описана информационная модель системы по поиску работы.

# 5 ОБОСНОВАНИЕ ВЫБОРА КОМПОНЕНТОВ И ТЕХНОЛОГИЙ

Задачей курсового проекта является создание клиент-серверного программного приложения (онлайн-системы), позволяющего оптимизировать процесс поиска работы.

Для достижения поставленной в курсовом проекте цели был выбран объектно-ориентированный язык программирования Java.

Объектно-ориентированный язык Java предназначен для создания переносимых на различные платформы и операционные системы программ. Язык Java нашел широкое применение в Интернет-приложениях, добавив на статические и клиентские Web-страницы динамическую графику, улучшив интерфейсы и реализовав вычислительные возможности. Но объектно-ориентированная парадигма и кроссплатформенность привели к тому, что уже буквально через несколько лет после своего создания язык практически покинул клиентские страницы и перебрался на сервера. На стороне клиента его место занял язык JavaScript.

При создании язык Java предполагался более простым, чем его синтаксический предок С++. На сегодняшний день с появлением версий J2SE 9 и J2EE 8 возможности языка Java существенно расширились и во многом перекрывают функциональность С/С++/С#.

Системная библиотека классов языка Java содержит классы и пакеты, реализующие и расширяющие базовые возможности языка, а также сетевые возможности, взаимодействие с базами данных, графические интерфейсы и многое другое. Методы классов, включенных в эти библиотеки, вызываются из JVM (Java Virtual Machine) во время интерпретации программы [6].

Язык Java делает сетевое программирование простым благодаря наличию специальных средств и классов. Сетевые классы имеют методы для установки сетевых соединений, передачи запросов и сообщений. Многопоточность позволяет обрабатывать несколько соединений. Сетевые приложения используют Internet-приложения, к которым относятся Web-браузер, e-mail, сетевые новости, передача файлов. Для создания таких приложений используются сокеты, порты, протоколы TCP/IP, UDP.

Приложения клиент-сервер используют компьютер, выполняющий специальную программу-сервер, которая обычно устанавливается на удаленном компьютере и предоставляет услуги другим программам-клиентам. Клиент − это программа, получающая услуги от сервера. Клиент устанавливает соединение с сервером и посылает серверу запрос. Сервер осуществляет прослушивание клиентов, получает и выполняет запрос после установки соединения. Результат выполнения запроса может быть возвращен сервером клиенту.

В качестве основных фреймворков были выбраны Spring и Hibernate.

Среда Spring Framework, содержащая большое количество классов и пакетов, была разработана как модульная среда, которая может быть поэтапно или частично введена в проект, т.е. использованы будут только необходимые средства (например, среда выполнения Web). Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java – на любых платформах.

Spring Framework был придуман не столько для выполнения какой-то прикладной задачи, а для обеспечения лучшей масштабируемости, возможности более простого тестирования и более простой интеграции с другими фреймворками (например Hibernate). Благодаря этому писать большие приложения становится проще – разработчики просто избегают ряда проблем, связанных с созданием enterprise-приложений вместо того, чтобы их решать. Spring была выбрана в качестве среды выполнения Web потому, что необходимы также и другие ее возможности, такие как IоС (инверсия усправления), управление транзакциями, электронная почта, планирование и т.д. Среда выполнения Web Spring MVC не имеет аналогов в области отказоустойчивости и гибкости [7].

Hibernate – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (object-relational mapping — ORM). Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных. Разработчик может использовать Hibernate как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой данных. Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода.

Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL. Hibernate обеспечивает прозрачную поддержку сохранности данных (persistence) для «POJO» (то есть для стандартных Java-объектов); единственное строгое требование для сохраняемого класса — наличие конструктора по умолчанию (без параметров). Для корректного поведения в некоторых приложениях требуется также уделить внимание методам equals() и hashCode(). Hibernate легко интегрируется с другими Java EE фреймворками, например, Spring Framework поддерживает встроенную интеграцию с Hibernate [8].

В качестве основной среды разработки в проекте используется IntelliJ IDEA 2017.2.2. В качестве СУБД в курсовом проекте используется MySQL. Для создания UML-диаграмм в работе было использовано средство Enterprise Architect, которое позволяет осуществлять создание диаграмм вариантов использования, состояний, последовательность с помощью просто моделирования. Для выполнения UML-моделей в стандарте IDEF0 использовалось CASE-средство CA AllFusion Process Modeler r7 (BPwin). Для информационного моделирования применялось средство MySQLWorkbench 6.3 CE.

Для более подробного изучения и рассмотрения системы программы необходимо рассмотреть следующие модели:

* диаграмма последовательностей;
* диаграмма компонентов;
* диаграмма развёртывания.

## 5.1 Диаграмма последовательностей

Диаграмма последовательностей (Sequence diagram) относится к диаграммам взаимодействия UML, описывающим поведенческие моменты системы, взаимодействие объектов в течение определённого отрезка времени.

На диаграмме последовательностей могут присутствовать участники процесса, компоненты системы и сообщения, которые они могут вернуть.

С помощью диаграммы последовательности рассмотрим процесс редактирования информации (информации о соискателе, сотруднике HR, вакансии, интервью) администратором системы.

В качестве объектов диаграммы выступают администратор, клиент, сервер и база данных (MySQL). Администратору необходимо выбрать поле, чья информацию впоследствии будет отредактирована. После выбора оно будет подсвечено и можно будет вводить новые данные. Как только предыдущие шаги будут выполнены, клиентское приложение проведет валидацию данных и, если она успешна, то они отправляются на сервер. Сервер соответственно проверяет корректность полученной информации, и в случае успеха формирует запрос, и отправляет данные в базу данных, где они заносятся в соответствующие поля и таблицы. В случае успеха, должный ответ посылается на сервер. Клиентскому приложению передаётся сигнал об успешном редактировании. Администратор может продолжать работу в системе.

Данная диаграмма показана на рисунке 5.1.

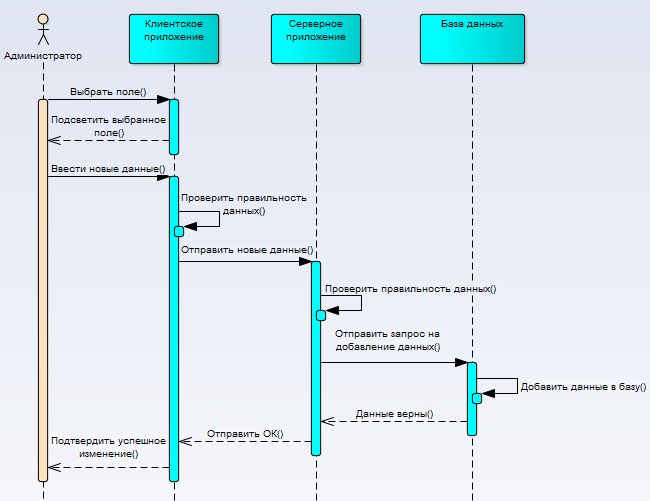


Рисунок 5.1 – Диаграмма последовательностей для вариантов использования «Редактировать информацию»

## 5.2 Диаграмма компонентов

Диаграмма компонентов (component diagram) – структурная диаграмма. Она показывает конкретный вид структуры моделируемой системы, разбивая программу на структурные компоненты и зависимости между ними.

Представление обычно включает компоненты с их интерфейсами или портами.

Полный проект разрабатываемого приложения состоит из совокупности моделей разных уровней, которые должны быть согласованы между собой. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. Соответствующая диаграмма компонентов отражает отношение подсистем веб-приложения:

* СУБД предоставляет интерфейс веб-сервису;
* используя предоставленный интерфейс, веб-сервис обращается к СУБД,
* веб-клиенту предоставляется интерфейс веб-сервисом;
* веб-клиент обращается к веб-сервису, применяя данный интерфейс;
* пользователь взаимодействует с веб-клиентом при помощи веб-браузера.

Данную диаграмму можно увидеть ниже на рисунке 5.2.



Рисунок 5.2 – Диаграмма компонентов онлайн-системы по поиску работы

## 5.3 Диаграмма развёртывания

Диаграмма развертывания (deployment diagram) необходима для представления того, какие устройства будут использовать те или иные компоненты.

Рассмотрим основные узлы диаграммы развёртывания для того, чтобы получить детальное представление о существующих на этапе исполнения элементов программы.

На диаграмме показаны следующие узлы:

* устройство пользователя (персональный компьютер, ноутбук, планшет, смартфон и т.п.), на котором размещается веб-браузер;
* сервер приложений 2, на котором размещается веб-клиент;
* сервер приложений 1, на котором размещается веб-сервис;
* сервер баз данных, на котором размещается СУБД MySQL.

Данная диаграмма продемонстрирована на рисунке 5.3.



Рисунок 5.3 – Диаграмма развертывания онлайн-системы по поиску работы

В данном разделе было приведено обоснование выбора компонентов и технологий для реализации курсового проекта.

# 6. МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ

## 6.1 Диаграмма состояний

Диаграммы состояний (Statechart) строятся для единственного класса и описывают поведение единственного объекта. Диаграмма состояний представляет собой граф состояний, в которых может находиться объект и связей между ними.

Диаграмма состояния данного проекта описывает состояния аккаунта кандидата в системе и этапы, через которые этот объект проходит. Диаграмма состояний для данного объекта представлена на рисунке 6.1.



Рисунок 6.1 – Диаграмма состояний аккаунта кандидата в системе

## 6.2 Описание алгоритма регистрации пользователя

Данный алгоритм описывает регистрацию кандидата или сотрудника HR в системе.

На главной странице регистрации пользователя просят ввести логин, пароль, повторить пароль, выбрать роль (кандидат/HR), а также подтвердить, что он не является роботом посредством прохождения рекапчи. После ввода данной информации проверяется наличие в базе данных только что введённого логина. При его наличии будет выведено сообщение об ошибке при регистрации пользователя ввиду некорректных данных. Если же такого логина нет в базе данных, то вся введённая информация проходит валидацию на стороне клиента и сервера. При неудаче валидации будет выведено сообщение о неверно введенных данных, и пользователь останется на главной странице регистрации. В случае успеха, пользователь с полученными данными будет добавлен в базу данных, затем в сессию устанавливается статус и информация клиента, а также произойдёт переход в соответствующую домашнюю страницу кандидата или HR для дальнейших действий.

Блок-схема для данного алгоритма представлена в конце раздела на рисунке 6.2.

## 6.3 Описание алгоритма отправления электронного сообщения HR-у от кандидата

Данный алгоритм описывает отправление электронного сообщения HR-у от кандидата.

При просмотре списка вакансий кандидат может нажать на «Отправить e-mail» и перейти в соответствующее меню. Далее извлекается электронный адрес HR-а той вакансии, из которой и был произведён переход на страницу отправления электронного сообщения, в поле «Кому», а в поле «Пароль вашего e-mail» заносится пароль от электронной почты кандидата текущей сессии. Следующие шаги включают в себя ввод темы письма и самого сообщения. После этого формируется объект отправляемого сообщения, которое и отсылается HR-у от кандидата. При возникновении ошибки в процессе формирования объекта отправляемого сообщения или при самой отправке, о ней будет немедленно сообщено и будет необходимо повторить отправку.

Блок-схема для данного алгоритма представлена в конце раздела на рисунке 6.3.

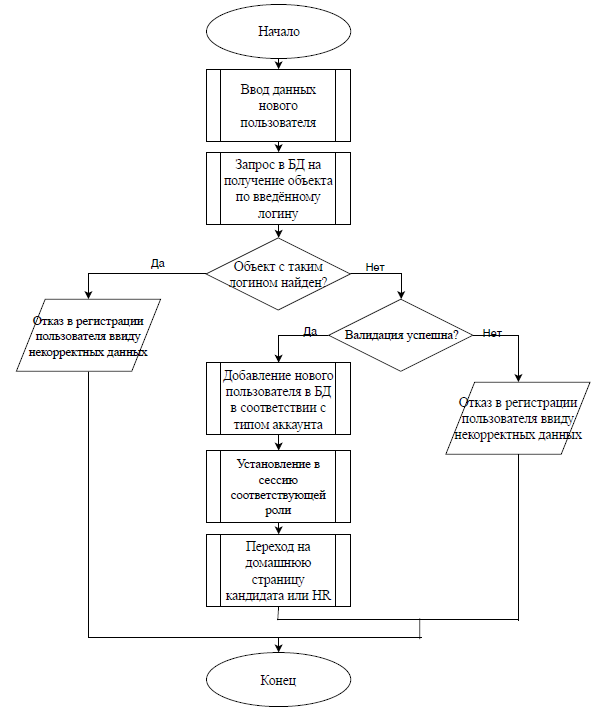


Рисунок 6.2 – Блок-схема регистрации пользователя

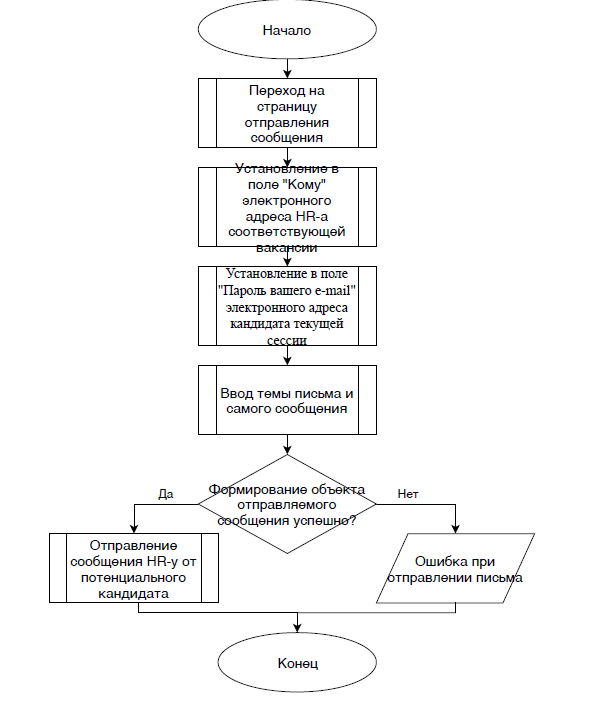


Рисунок 6.3 – Блок-схема отправления электронного сообщения HR-у от кандидата

В данном разделе были рассмотрены модели представления системы и их описание.

# 7 ОПИСАНИЕ ПРИМЕНЕНИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ

## 7.1 Паттерн «Одиночка» (Singleton)

**Одиночка (**«Singleton»**)** — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

В данном приложении паттерн используется в случае работы с пулом соединений для базы данных на стороне сервера.

При работе с базой данных данный паттерн является неотъемлемой частью, так как объект класса, работающего с пулом соединений, требуется в разных классах, и, чтобы каждый раз не происходило подключение к базе данных, необходим объект с глобальной точкой доступа.

Схема данного паттерна представлена на рисунке 7.1.

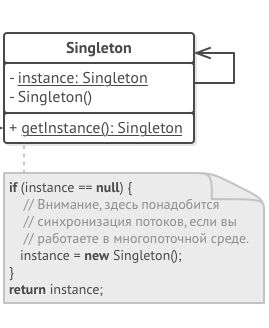


Рисунок 7.1 – Схема представления паттерна «Одиночка»

## 7.2 Паттерн «Заместитель» (Proxy)

Заместитель (Proxy) — структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

Идея паттерна «Заместитель» заключается в предоставлении клиенту другого объекта (заместителя), взамен объекту с контролируемым доступом. При этом, объект-заместитель, реализует тот же интерфейс, что и оригинальный объект, в результате чего, поведение клиента не требует изменений. Иначе, клиент взаимодействует с заместителем ровно как с оригинальным объектом посредством единого интерфейса. Клиент, так же, не делает предположений о том работает ли он с реальным объектом или его заместителем. Контролирование доступа к объекту, при этом, достигается за счет использования ссылки на него в заместителе, благодаря которой заместитель переадресовывает внешние вызовы контролируемому объекту, возможно сопровождая их дополнительными операциями. В общем случае реализует технологию обертывания(wrapping) класса с целью повышения безопасности или оптимизации взаимодействия.

Схема данного паттерна представлена на рисунке 7.2.

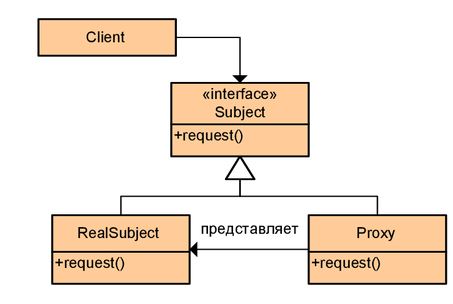


Рисунок 7.2 – Схема представления паттерна «Заместитель»

## 7.3 Паттерн «Команда» (Command)

Команда (Command) — поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.

Паттерн Команда (Command) преобразовывает запрос на выполнение действия в отдельный объект-команду. Такая инкапсуляция позволяет передавать эти действия другим функциям и объектам в качестве параметра, приказывая им выполнить запрошенную операцию. Команда – это объект, поэтому над ней допустимы любые операции, что и над объектом.

Объект-источник запроса отделяется от команды, но от его типа зависит, какая из команд будет выполнена.

В данном случае паттерн используется для разделения источника запроса и его исполнителя, а также для построения последовательности команд, порядок которых определяет сам пользователь в зависимости от своего желания или результата выполнения предыдущей команды.

Схема данного паттерна представлена на рисунке 7.3.

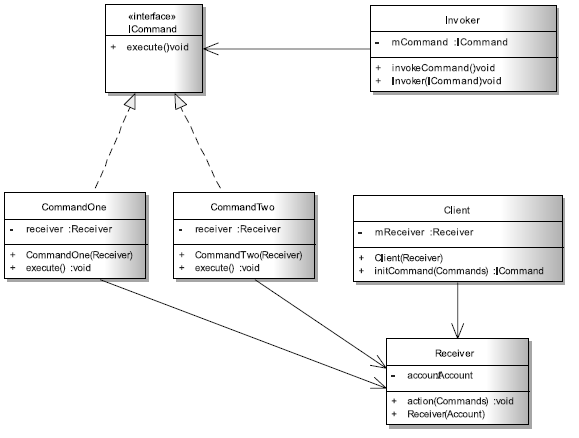


Рисунок 7.3 – Схема представления паттерна «Команда»

# 8 РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ СИСТЕМЫ

Процесс по развертыванию данного проекта по поиску работы подразделяется на несколько шагов.

Сперва для функционирования веб-приложения требуется СУБД MySQL. Рекомендуется версия 6.3 CE или выше.

С помощью MySQL необходимо создать новую базу данных.Для этого необходимо открыть файл SQL-скрипта script.sql из поставки и запустить его посредством нажатия на кнопку «Execute the selection portion of the script or everything, if there is no selection» (рисунок 8.1).

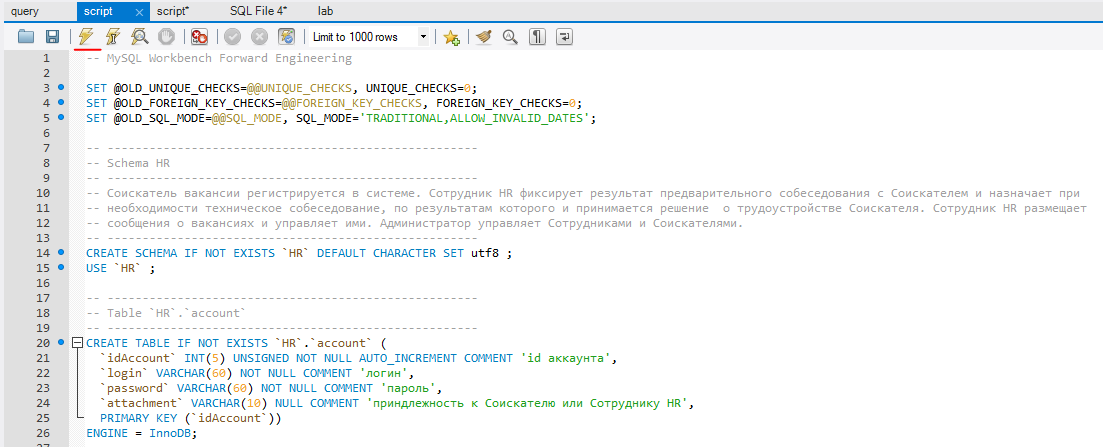


Рисунок 8.1 – Выполнение sql-скрипта

В результате успешного выполнения данного скрипта в разделе «SCHEMAS» должна появиться полностью готовая к эксплуатации база данных (рисунок 8.2). Такой вариант развертывания позволяет не привязываться к конкретному имени базы данных.

Далее необходимо настроить права доступа к базе данных согласно применяемой схеме авторизации согласно общим инструкциям.

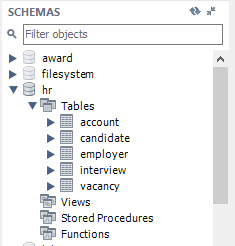


Рисунок 8.2 – Результат успешного выполнения sql-скрипта

Следующим шагом по развертыванию системы будет загрузка сервера на хостинг. Для его правильного выполнения нужно скачать и установить контейнер сервлетов либо сервер приложений. Рекомендуются известные реализации такие как ApacheTomcat, Jetty, WildFly (бывший JBoss), GlassFish, IBMWebSphere. В плане установки все просто и для каждого из перечисленных контейнеров и серверов установка – это просто распаковка архива.

Настройка сервера описывается для ApacheTomcat и начинается с настройки порта HTTP, на котором он будет работать. Однако следует предупредить, что для каждого из серверов есть свои особенности настройки.  
Для Tomcat нужно зайти в папку с распакованным tomcat, далее папка conf, файл server.xml. Нужно найти в этом файле число 8080 (http порт по умолчанию) и поменять его на что захотите.

Теперь нужно запустить tomcat. Для этого требуется перейти в папку с распакованным tomcat, далее папка bin и запустить файл startup.bat. Затем перейти в браузере по адресу http://localhost:8080/ (порт по умолчанию). Tomcatуспешно начал свою работу, если в браузере откроется страничка как на рисунке 8.3.

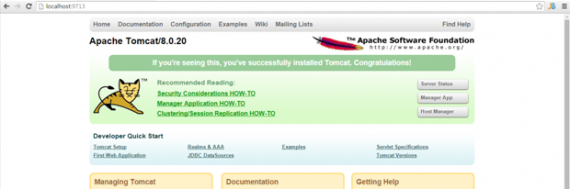


Рисунок 8.3 – Успешный старт ApacheTomcat

Кроме того, можно прописать себя как администратора сервера для увеличения функционала, например, для деплоя приложения и через графический интерфейс, и через удаленное подключение.

Для этого нужно перейти в папку с распакованным Tomcat, затем в папку conf, открыть файл tomcat-users.xml. В нем перед закрывающим тегом </tomcat-users> прописать следующий тег:

<user username="egarmin" password="1" roles="manager-gui,manager-script,manager-status,manager-jmx"/>

После чего необходимо скопировать war-файл из поставки в папку webapps и запустить Tomcat c помощью файла startup.bat.

Последним шагом развертывания будет загрузка клиента на хостинг, которая выполняется в точно такой же последовательности, как и второй шаг.

# 9 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СИСТЕМЫ

После запуска рассматриваемого программного продукта нас встречает страница авторизации (рисунок 9.1).

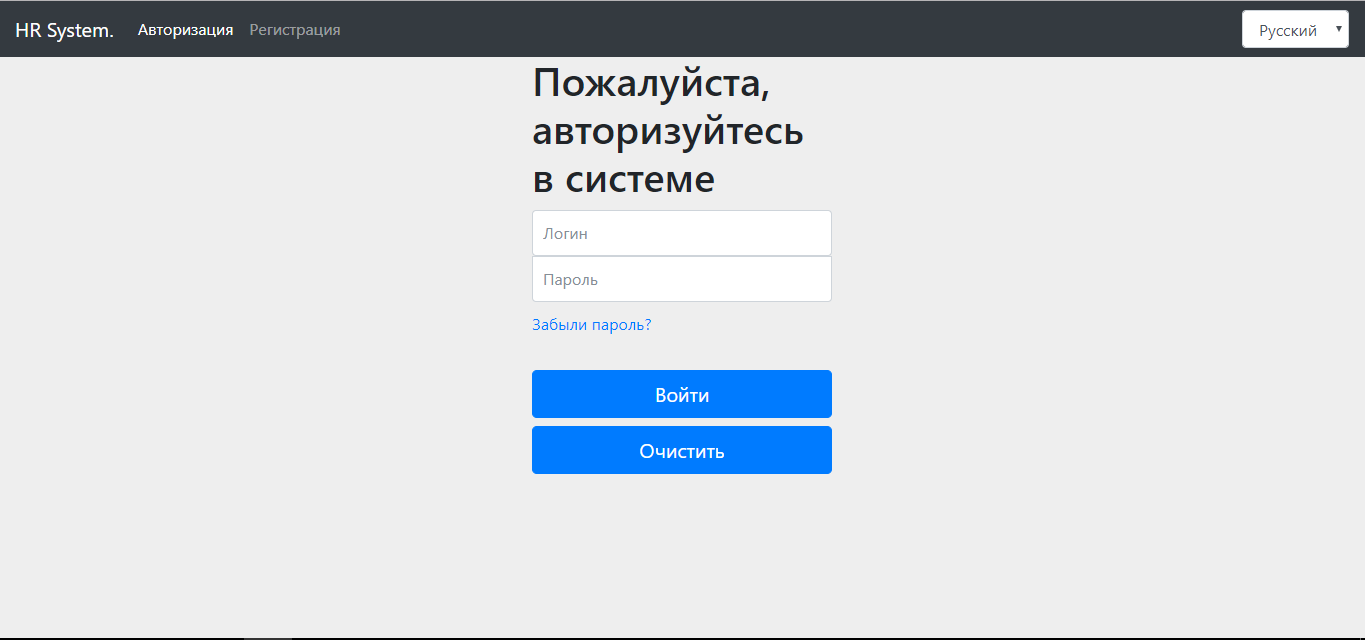


Рисунок 9.1 – Страница авторизации

После ввода данных, проверяется их корректность и в зависимости от этих данных, проверяется совпадение в базе данных. Если они были найдены, пользователя перекинет на соответствующую страницу – кандидата, сотрудника HR или же администратора. В обратном случае, будет выведено сообщение о некорректно введенных данных.

Также имеется кнопка «Забыли пароль?» (рисунок 9.2).

Также на данном этапе можно перейти на страницу регистрации (рисунок 9.3).

Здесь требуется ввести свой логин, пароль, подтвердить его, а также нажать на галочку, что вы не робот. Все поля также наделены проверкой корректности как на клиентской, так и на серверной стороне.

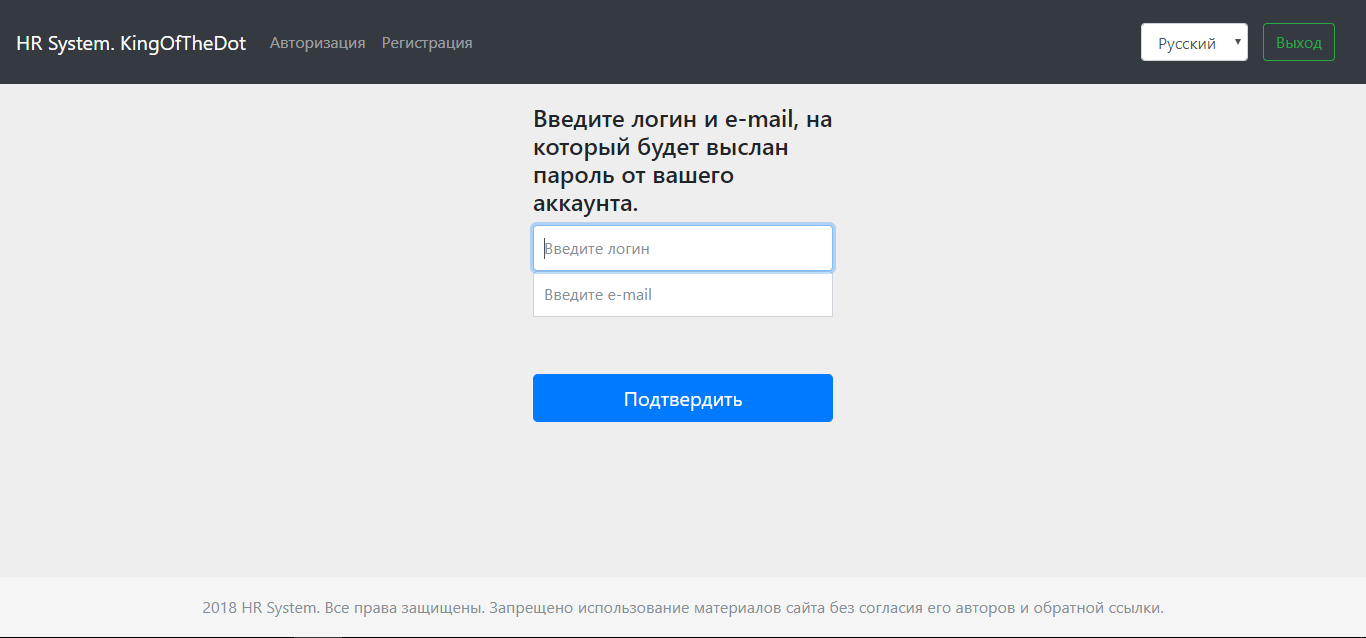


Рисунок 9.2 – Страница «Забыли пароль?»

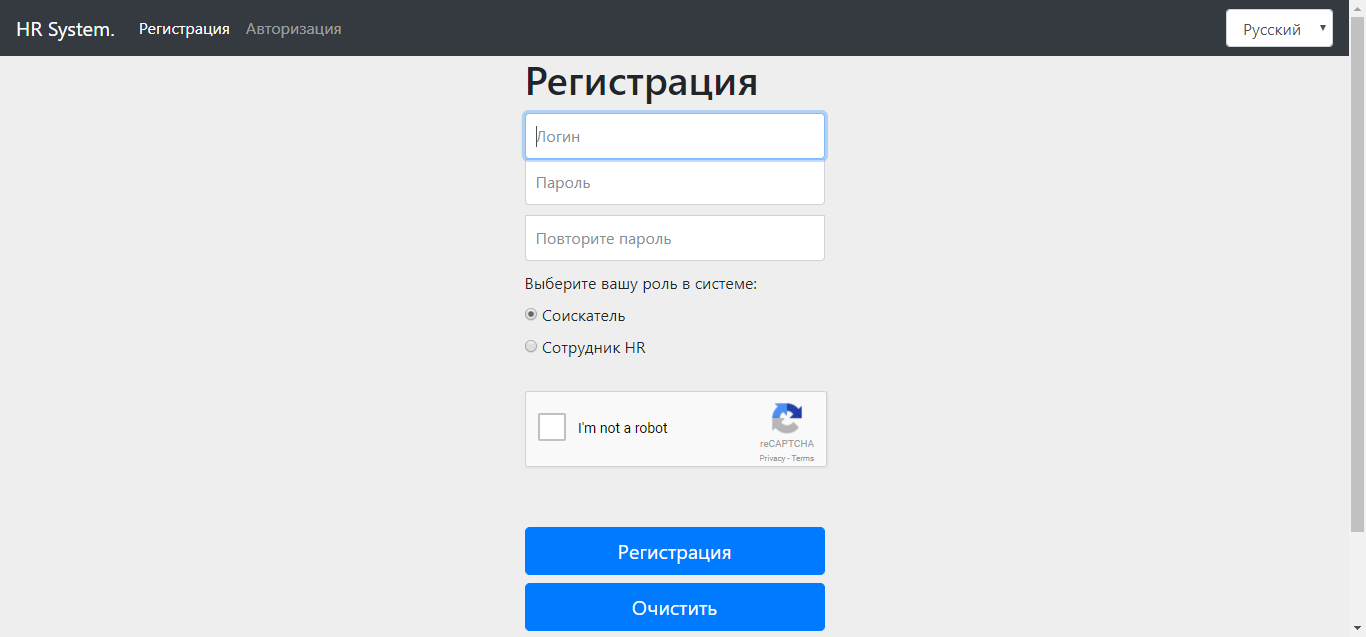


Рисунок 9.3 – Страница регистрации

Кроме того, на каждой странице есть выбор используемого языка интерфейса (рисунок 9.4).

Для показа работы программного продукта создадим аккаунт кандидата на странице регистрации. После успешного прохождения данного этапа нас перекинет на его домашнюю страницу (рисунок 9.5).

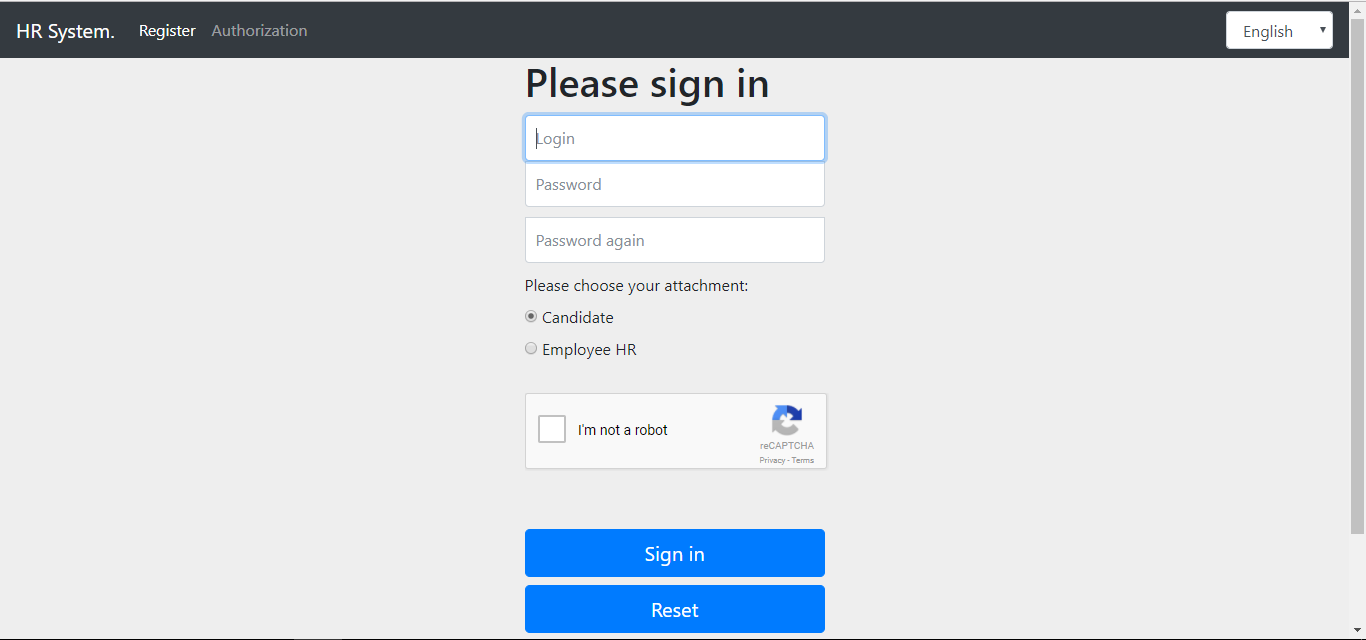


Рисунок 9.4 – Демонстрация английского языка интерфейса

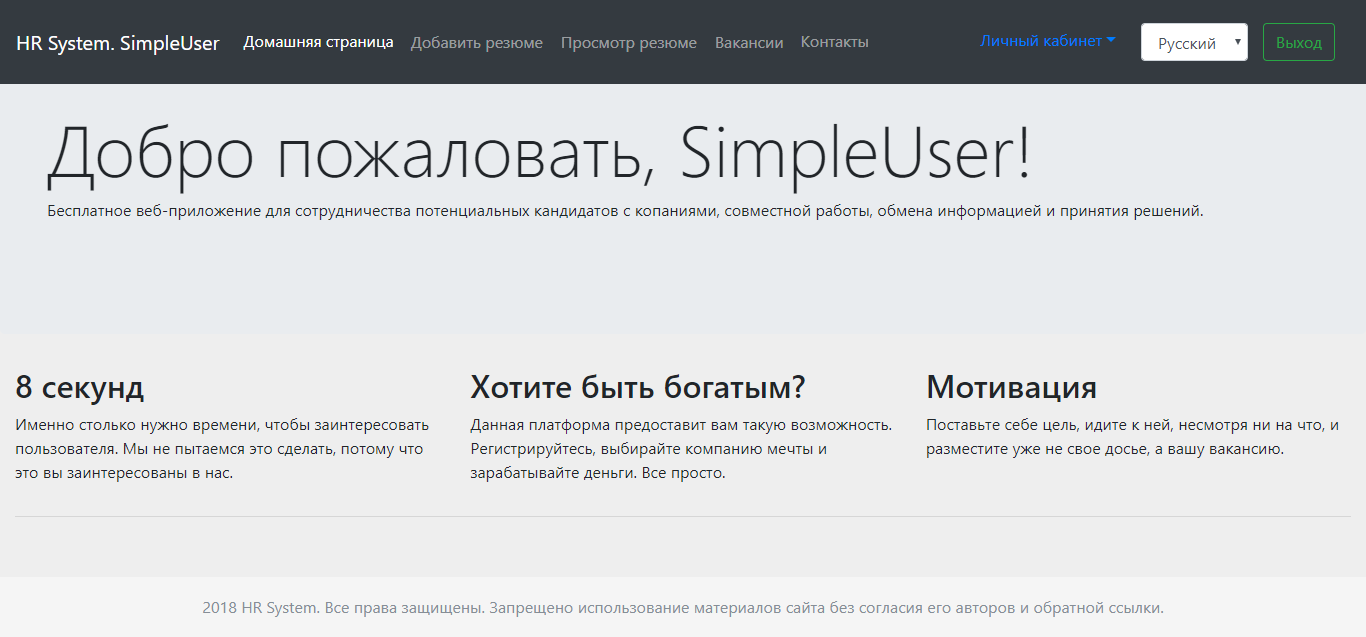


Рисунок 9.5 – Домашняя страница кандидата

Как кандидат, так и сотрудник HR имеют возможность редактировать данные своего аккаунта (рисунок 9.6), а также удалить его навсегда (рисунок 9.7).

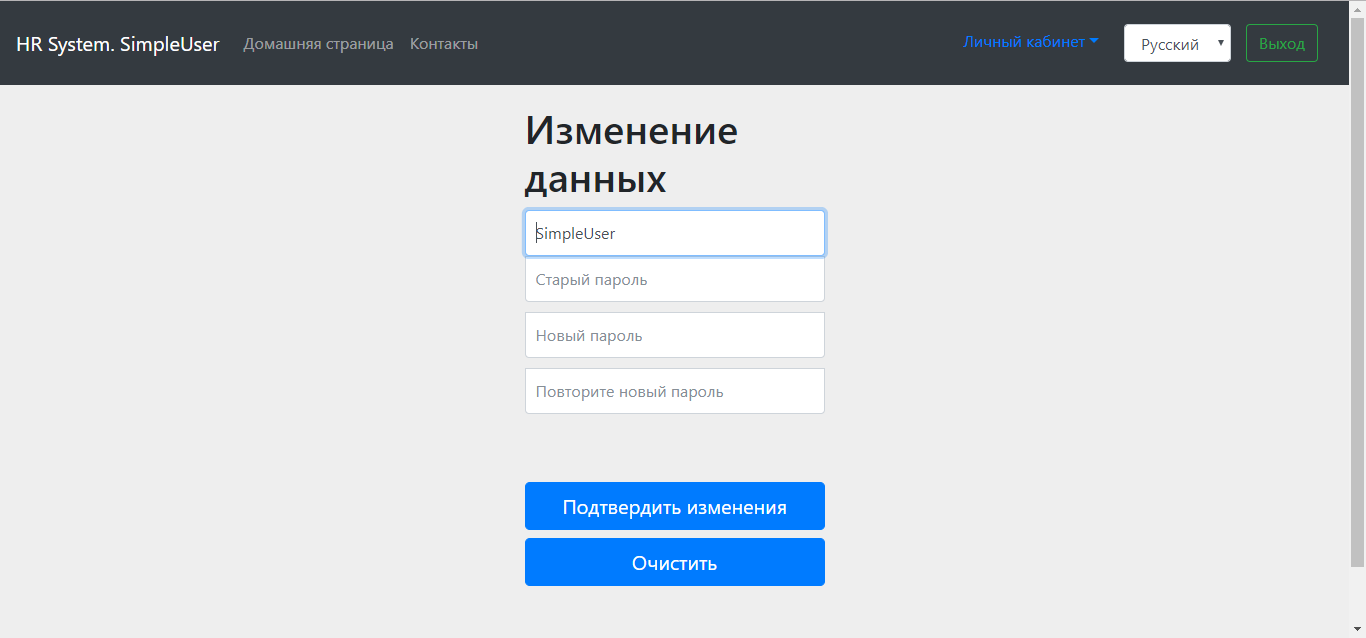


Рисунок 9.6 – Редактирование данных аккаунта

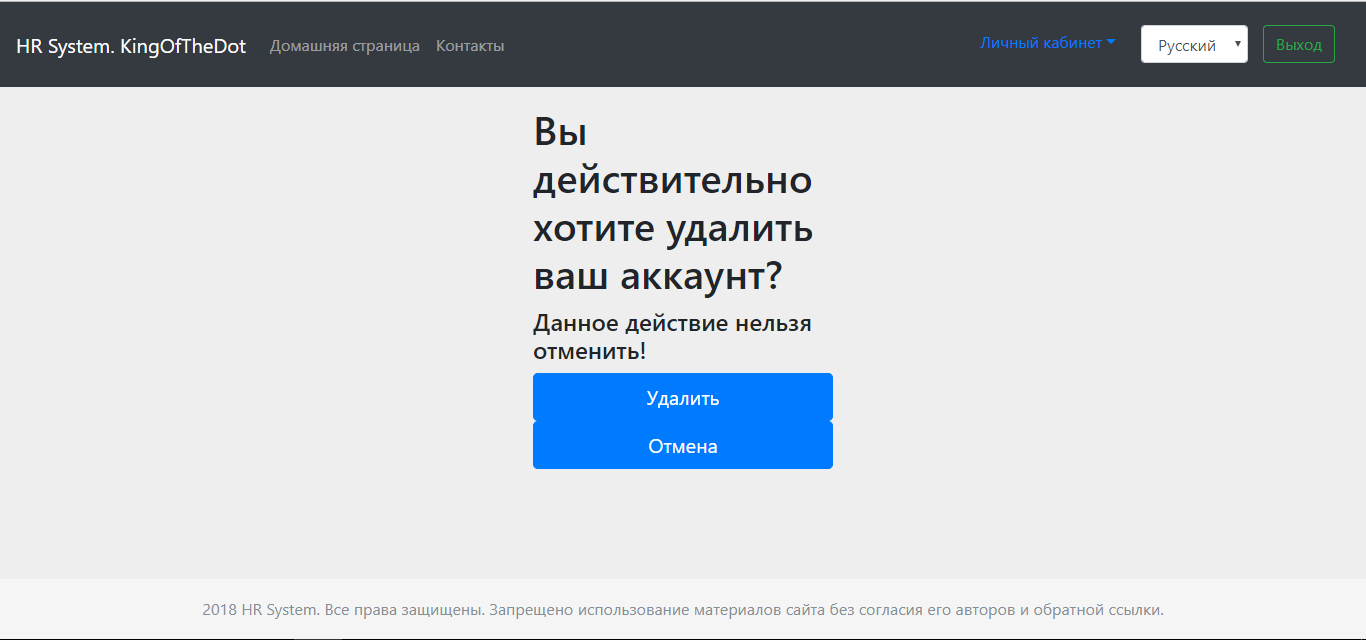


Рисунок 9.7 – Удаление аккаунта

Перейдем к рассмотрению основных функций кандидата.

Первое – это добавление резюме о себе (рисунок 9.8).

Каждое поле имеет защиту от неправильного ввода как на клиентской, так и на серверной стороне.

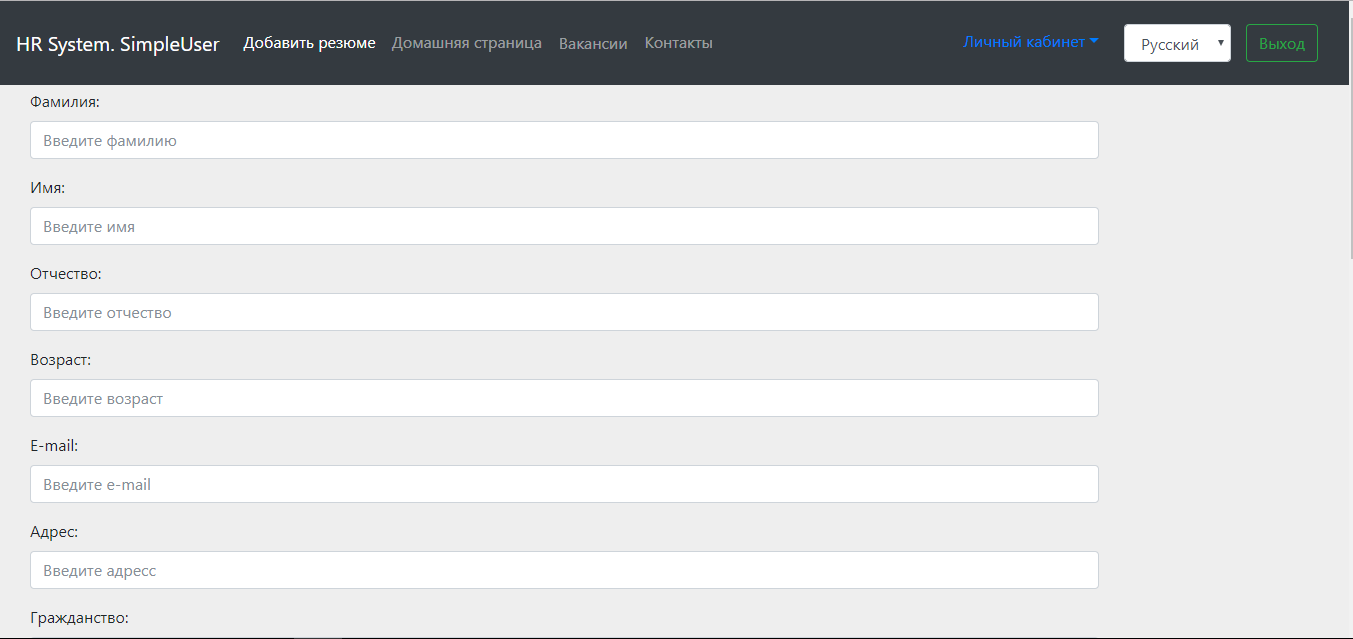


Рисунок 9.8 – Добавление резюме кандидатом

Кроме того, если кандидат попытается создать резюме, если оно уже имеется, то ему будет высвечено соответствующее предупреждающее сообщение (рисунок 9.9).

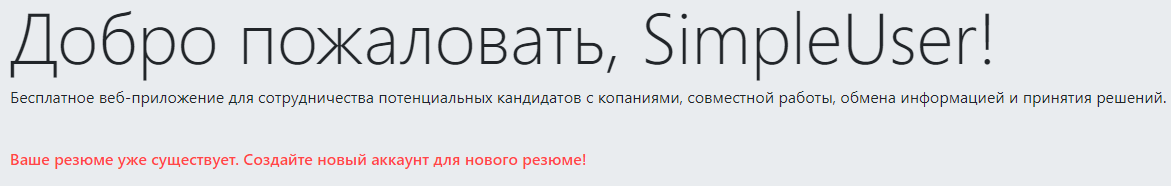


Рисунок 9.9 – Предупреждение о уже имеющемся резюме

Далее в списке основных функций – это просмотр и редактирование своего резюме (рисунок 9.10).

По умолчанию все поля заблокированы и предназначены для просмотра, однако если нажать соответствующею каждому полю кнопку «Редактировать», то поле станет открытым для ввода и изменения данных.

Все поля на данной странице также обладают защитой от неправильного ввода.

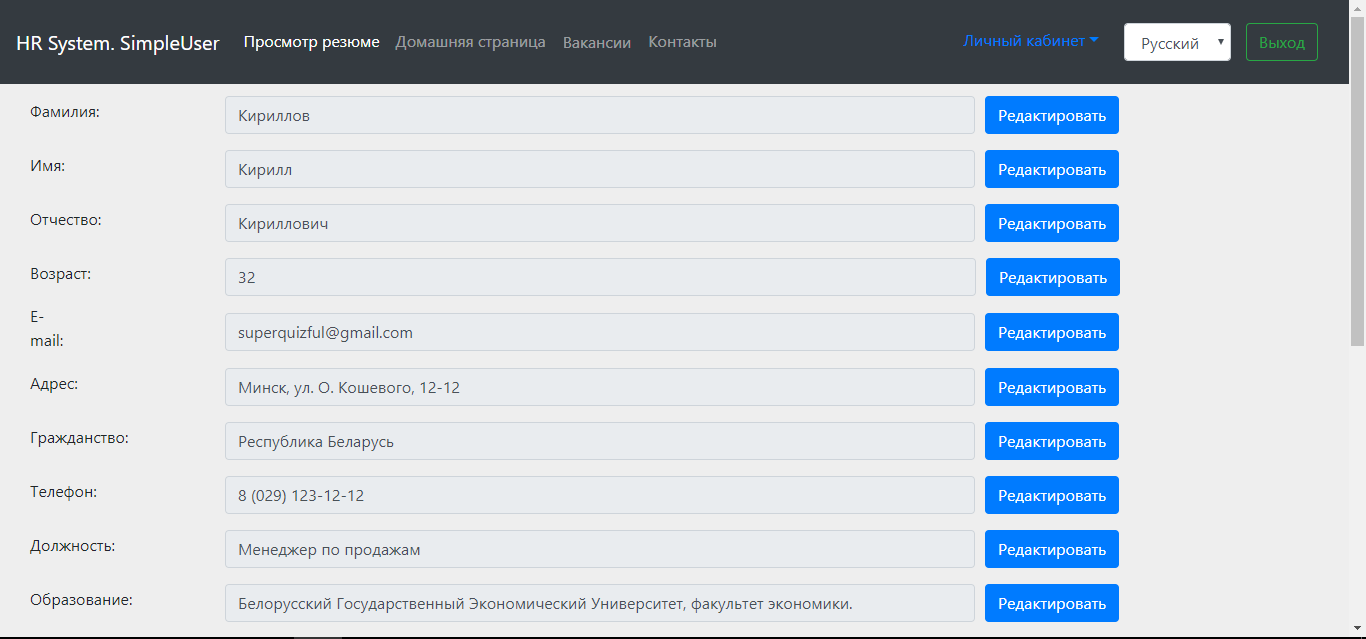


Рисунок 9.10 – Просмотр и редактирование резюме

Далее – это возможность кандидату просматривать список всех вакансий (рисунок 9.11).

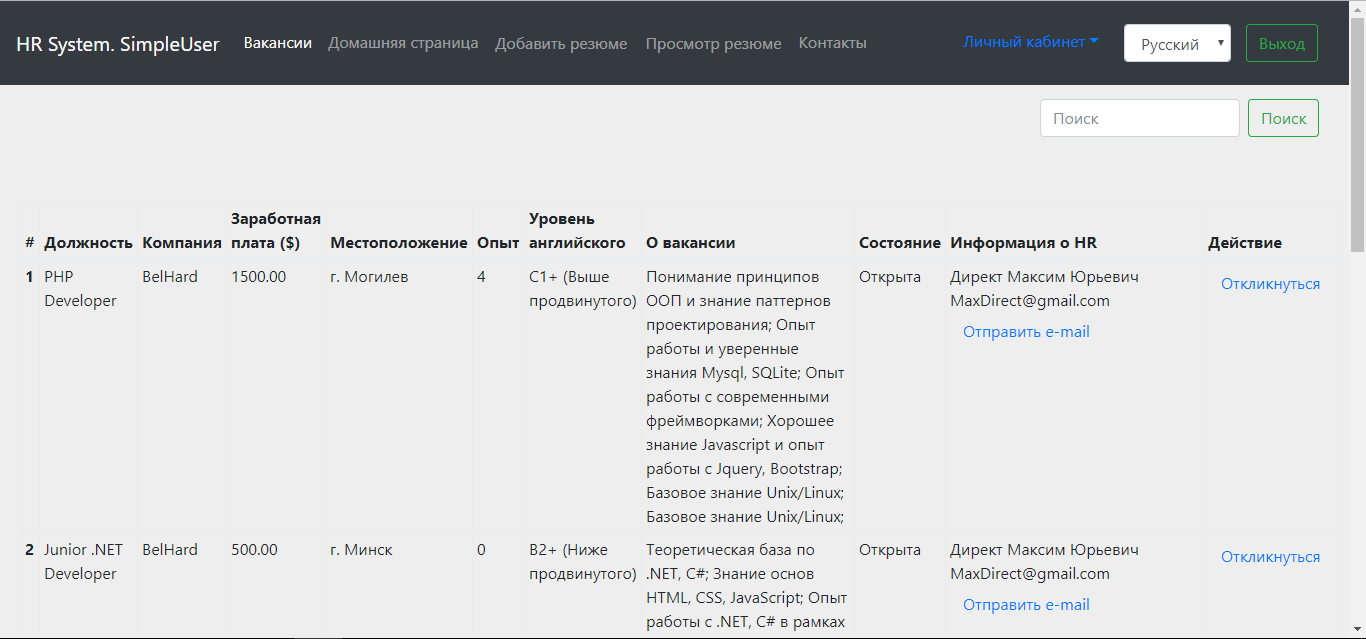


Рисунок 9.11 – Просмотр всех вакансий

Информация, представленная на данной странице, является понятной сама по собе, следовательно перейдем к рассмотрению кнопки «Отправить e-mail» в колонке «Информация о HR».

По названию колонки следует, что в ней содержатся краткие данные о HR’e, который представлен за данной вакансией и имеется возможность отправить ему письмо на указанный электронный адрес (рисунок 9.12).

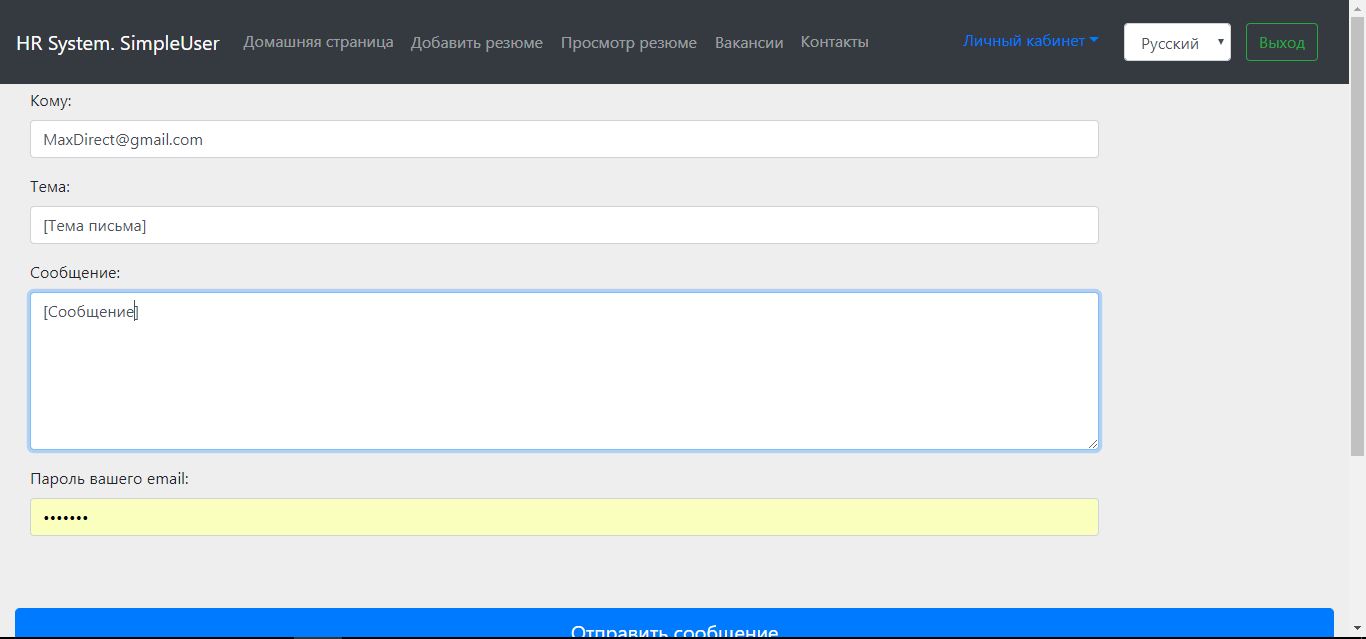


Рисунок 9.12 – Отправка электронного письма

Также в колонке «Действие» имеется кнопка «Откликнуться».

При нажатии на нее в базу идет запрос на формирование нового интервью. Результат нажатия на нее представлен на рисунке 9.13.

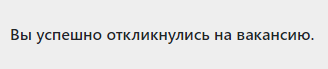


Рисунок 9.13 – Результат отклика на вакансию

Но, если кандидат попытается откликнуться на вакансию, на которую он уже откликался, то появится соответствующее сообщение (рисунок 9.14).

D:\УНИВЕР\Курсачи\Курсач СИТАИРИС\screenshots\14_Кандидат_уже_откликнулся.PNG

Рисунок 9.14 – Попытка отклика на вакансию, на которую уже откликались ранее

Также почти на каждой странице в блоке навигации можно заметить ссылку на вкладку «Контакты». Эта страничка содержит краткие сведения о компании и людях, которые разрабатывали данный программный продукт (рисунок 9.15).

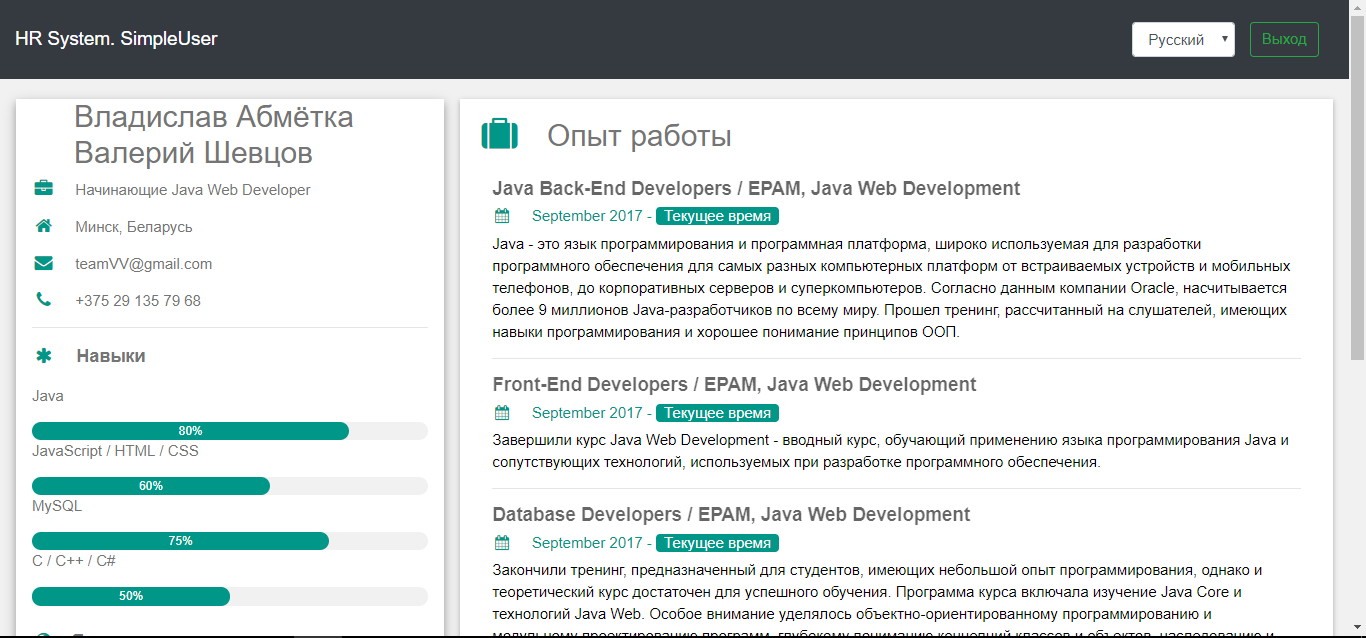


Рисунок 9.15 – Страница «Контакты»

Теперь перейдем к рассмотрению функций сотрудника HR.

Для него также создадим показательный аккаунт и после этого мы попадаем на его главную страницу (рисунок 9.16).

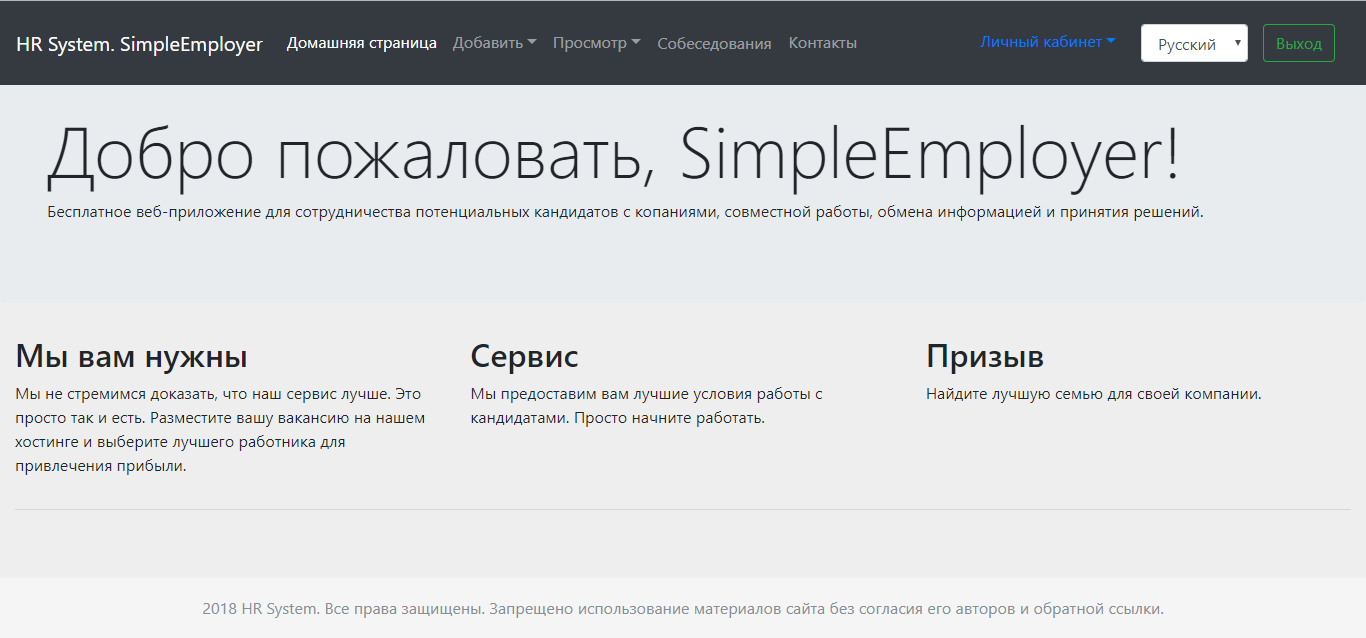


Рисунок 9.16 – Главная страница сотрудника HR

Первое, что должен сделать сотрудник – это добавить личную информацию о себе. Без этого он не сможет осуществлять все остальные функции, такие как создание резюме (рисунок 9.17), просмотр личной информации (рисунок 9.18), просмотр вакансий (рисунок 9.19) и просмотр собеседований (рисунок 9.20).

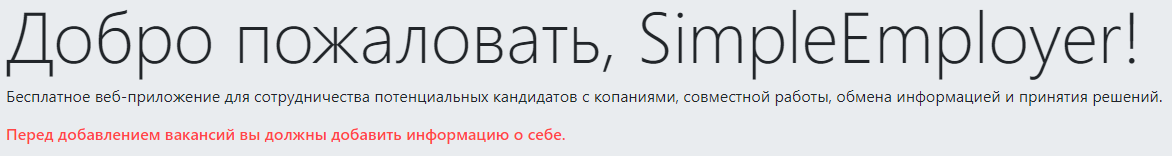


Рисунок 9.17 – Запрет на создание вакансий

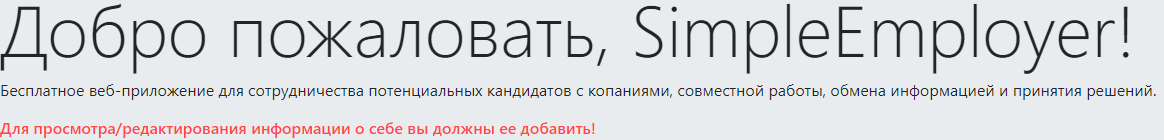


Рисунок 9.18 – Запрет на просмотр и редактирование личной информации

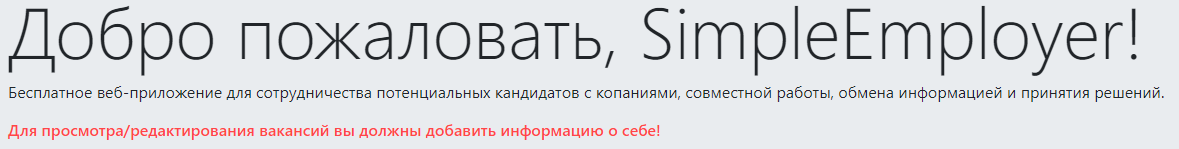


Рисунок 9.19 – Запрет на просмотр и редактирование вакансий

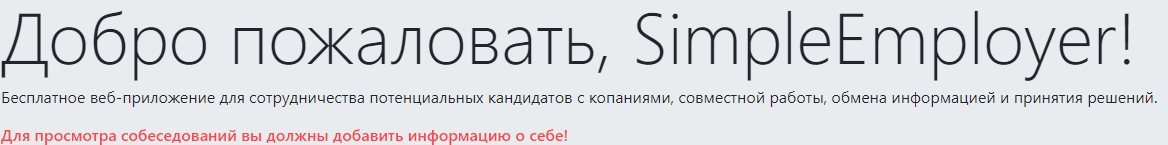


Рисунок 9.20 – Запрет на просмотр собеседований

Учитывая вышестоящее, добавим личную информацию о сотруднике (рисунок 9.21).

Все схоже, каждое поле имеет защиту от неправильного ввода как на клиента, так и на сервере, и при попытке ввести некорректные данные будет высвечено соответствующая информация.

После успешного добавления личной информации ее можно только просматривать и редактировать, опять создать ее уже будет невозможно. Попытка это сделать представлена на рисунке 9.22, где показано сообщение об ошибке.

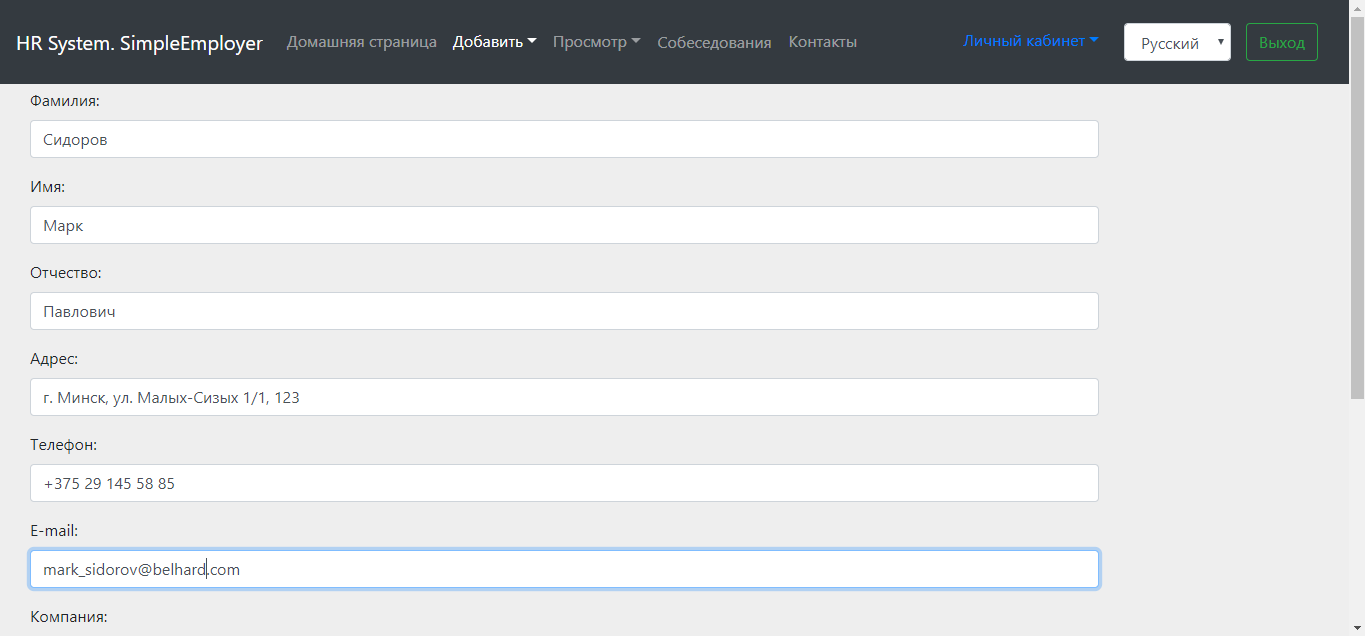


Рисунок 9.21 – Добавление личной информации о сотруднике HR

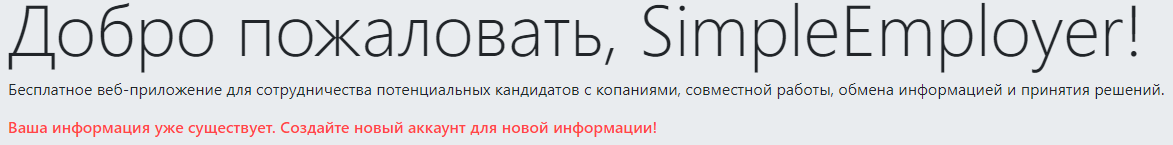


Рисунок 9.22 – Запрет на повторное добавление личной информации

Далее после добавления личной информации уже можно выполнять и все оставшиеся функции. Создадим новую вакансию (рисунок 9.23).

Опять же, все представленные на рисунке поля имеют защиту от некорректного ввода на клиенте и сервере.

Функция просмотра и редактирования личной информации о сотруднике HR представлена на рисунке 9.24.

Логика действий точно такая же, как и в случае с кандидатом.

По умолчанию все поля заблокированы и предназначены для просмотра, однако если нажать соответствующею каждому полю кнопку «Редактировать», то поле станет открытым для ввода и изменения данных.

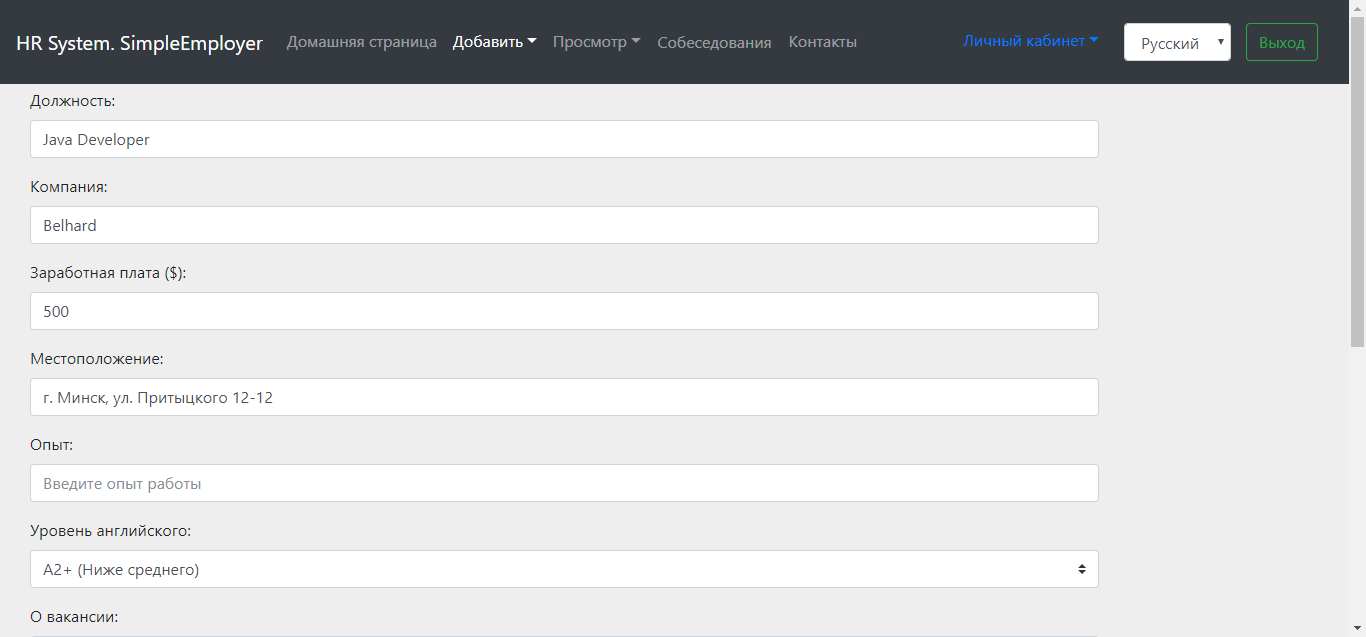


Рисунок 9.23 – Добавление вакансии

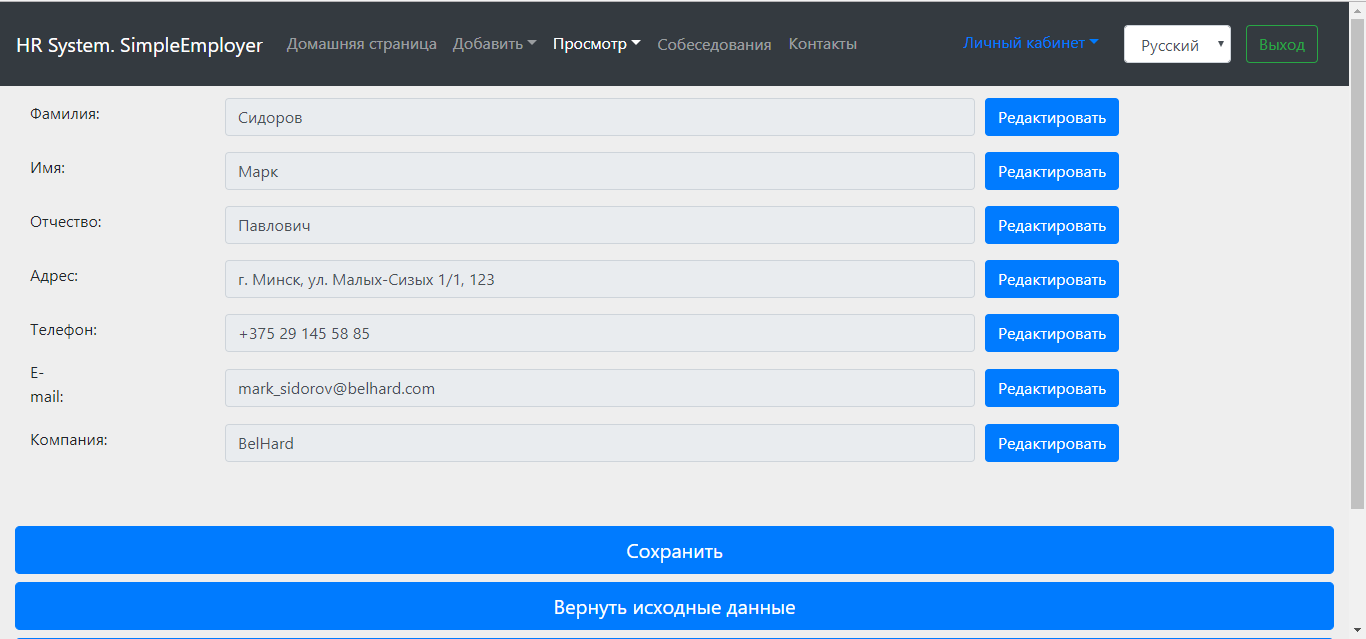


Рисунок 9.24 – Просмотр и редактирование личной информации сотрудника HR

На рисунке 9.25 представлена возможность просмотра всех вакансий.

На этой странице данному сотруднику доступны только те вакансии, которые он сам же и создал. Соответственно функции удаления и редактирования доступны только для этих вакансий.

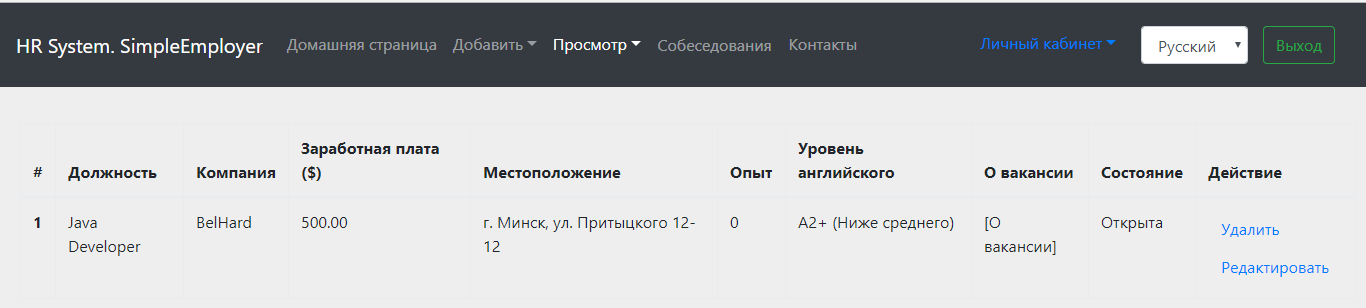


Рисунок 9.25 – Просмотр всех вакансий

Далее перейдем к рассмотрению просмотра и управления собеседованиями. Однако, если на опубликованные в системе вакансии данного сотрудника HR никто не откликался, при попытке открыть страницу просмотра интервью, HR получит сообщение, представленное на рисунке 9.26.

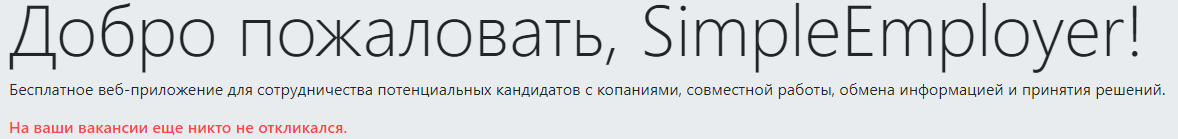


Рисунок 9.26 – На вакансии сотрудника HR никто не откликался

Но если хотя бы кто-то откликнулся на вакансию, то просмотр собеседований будет доступен (рисунок 9.27).

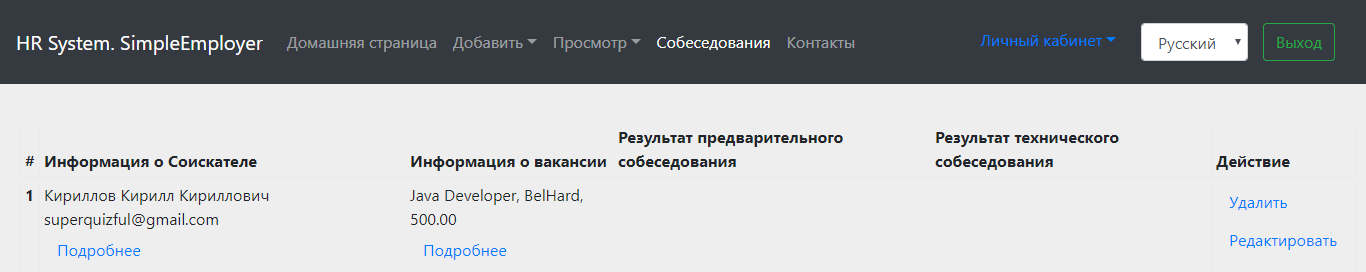


Рисунок 9.27 – Просмотр собеседований

В колонках «Информация о Соискателе» и «Информация о вакансии» находится лишь краткая информация о них, но посредством нажатия на кнопку «Подробнее» имеется возможность просмотреть расширенные данные.

Также имеется возможность удалить собеседование из системы (кнопка «Удалить» в колонке «Действий») и редактировать результаты собеседований (кнопка «Редактировать» в колонке «Действий»), что и представлено на рисунке 9.28.

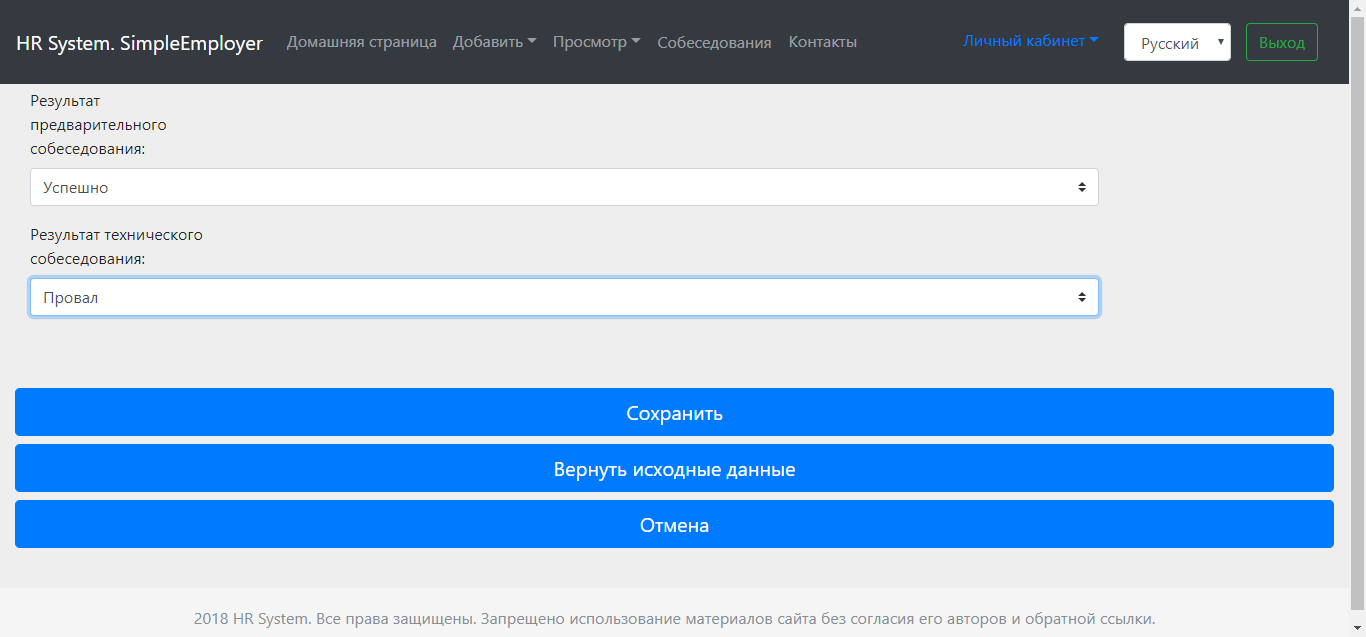


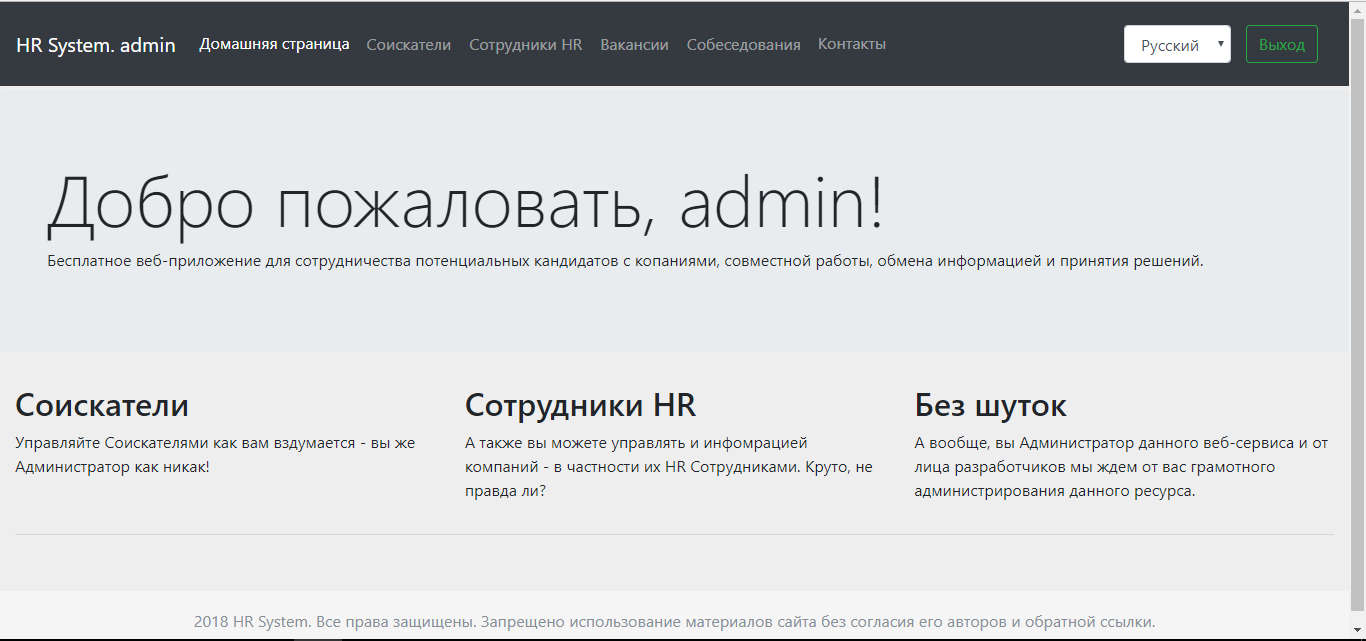
Рисунок 9.28 – Редактирование результатов собеседования

Далее перейдем к рассмотрению возможностей администратора.

Если кратко, то он имеет все полномочия как кандидата, так и сотрудника HR, описанные выше.

Введем логин и пароль администратора на странице авторизации и попадем на его главную страницу (рисунок 9.29).

Одними из всех полномочий администратора являются возможность просмотра списка всех кандидатов (рисунок 9.30), всех сотрудников HR (рисунок 9.31), всех вакансий (рисунок 9.32), а также всех собеседований (рисунок 9.33).



Рисунок

Рисунок 9.29 – Главная страница администратора

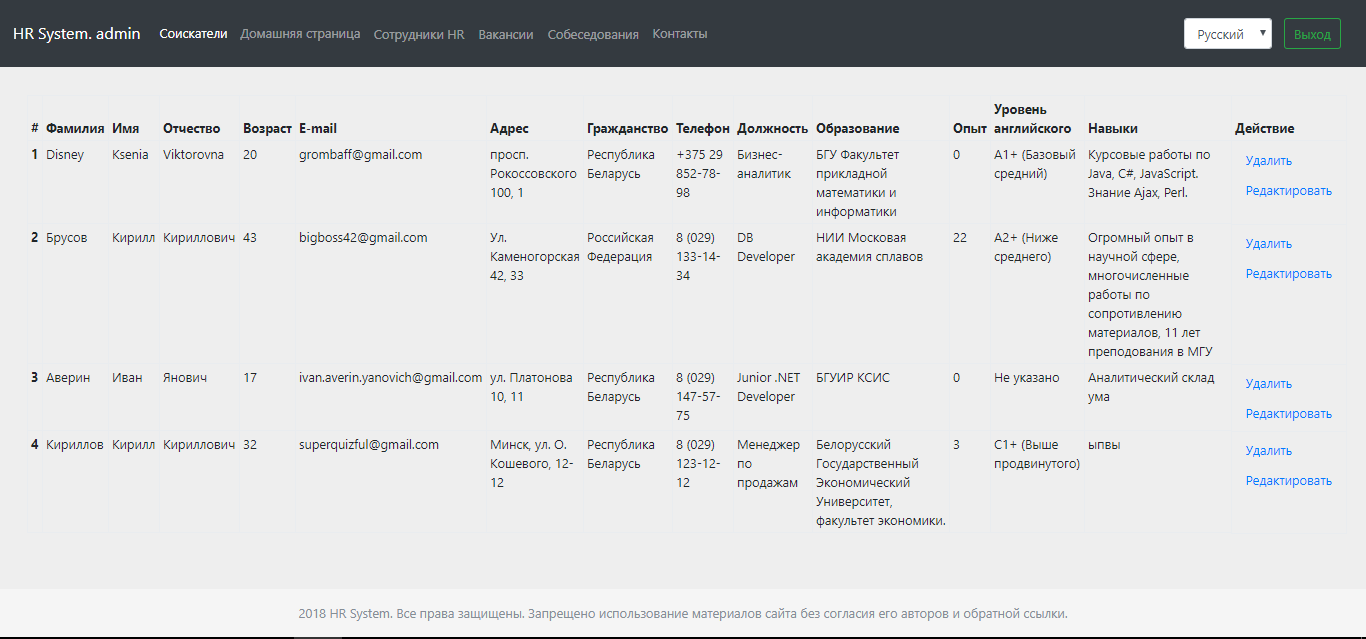


Рисунок 9.30 – Просмотр всех кандидатов администратором

В колонке «Действие» напротив каждого кандидата находятся кнопки «Удалить» и «Редактировать», которые выполняют соответствующие функции для соответствующего кандидата.

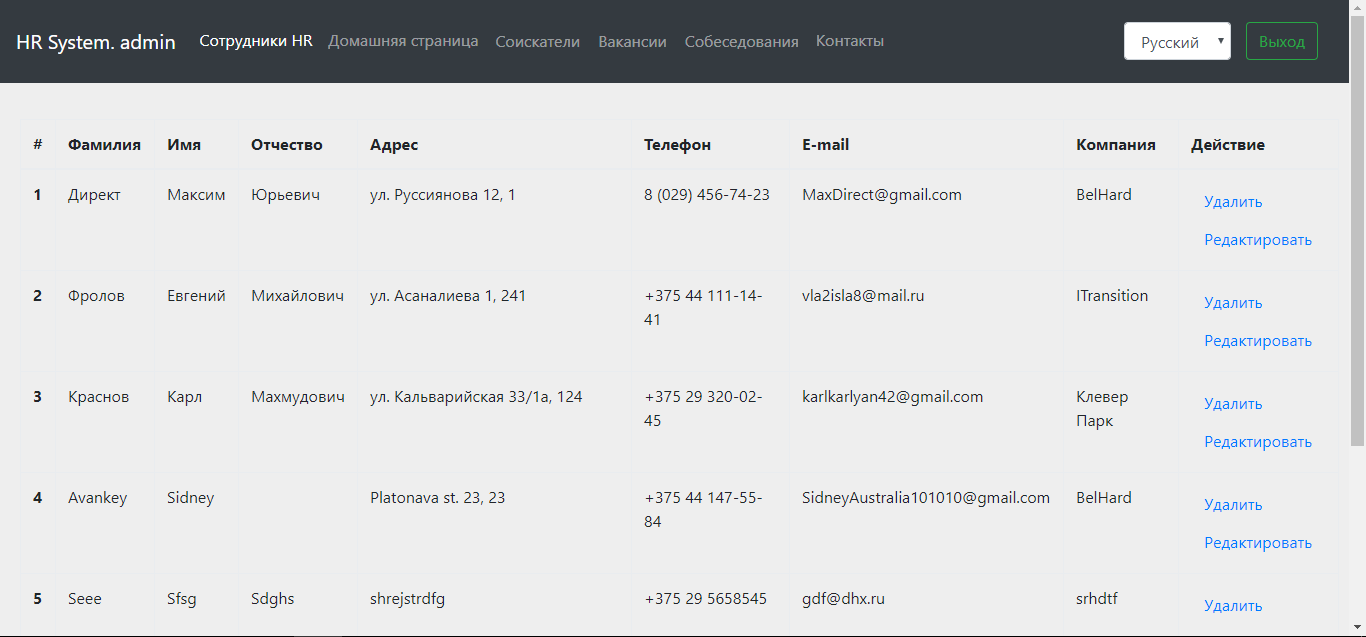


Рисунок 9.31 – Просмотр всех сотрудников HR администратором

В колонке «Действие» напротив каждого сотрудника находятся кнопки «Удалить» и «Редактировать», которые выполняют соответствующие функции для соответствующего сотрудника HR. Это же относится и к вакансиям (рисунок 9.32) и к собеседованиям (рисунок 9.33).

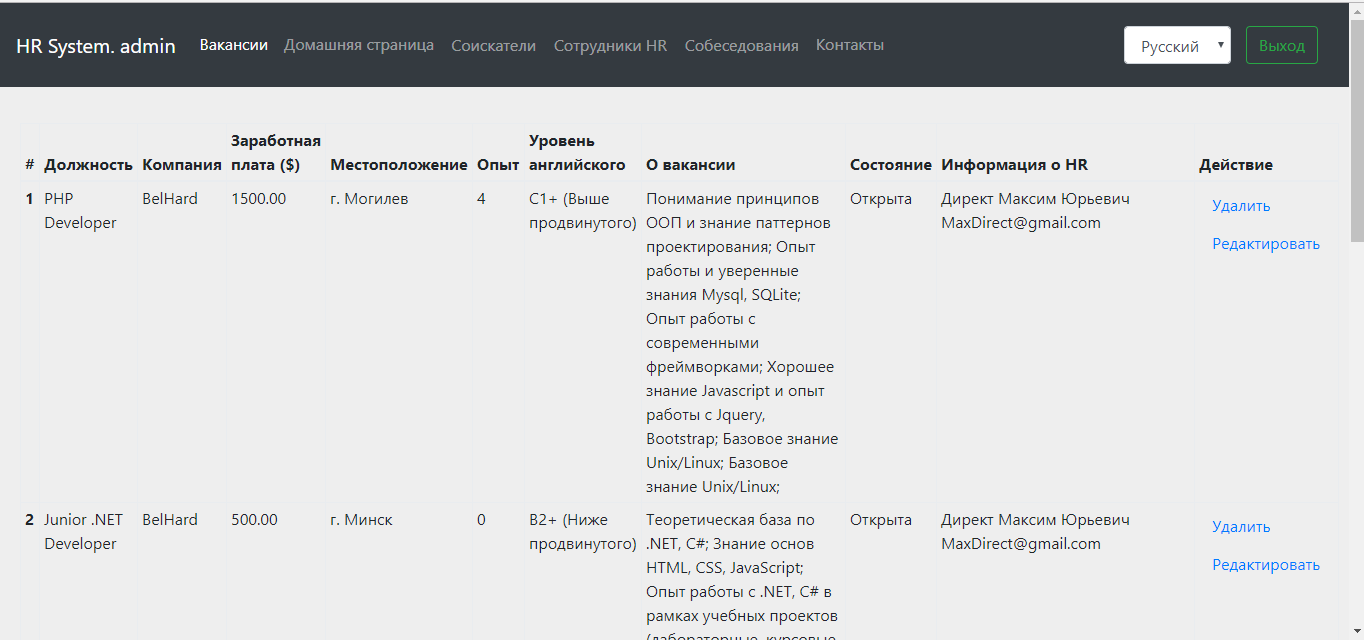


Рисунок 9.32 – Просмотр всех вакансий администратором



Рисунок 9.33 – Просмотр всех собеседований администратором

В данном разделе было проведено тестирование программы на наличие каких-либо ошибок.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта было разработано приложение, позволяющее в онлайн режиме автоматизировать процесс поиска работы, а также поиска подходящих сотрудников для компаний. Приложение имеет простой и понятный интерфейс, что делает нетрудным процесс взаимодействия с ним. Программа выполняет поставленные задачи и цели. Является устойчивой к возможным ошибкам, которые могут возникнуть при неправильной эксплуатации данного программного продукта.

Была реализована клиентская сторона приложения в качестве интерфейса системы. Это позволяет пользователям обращаться к серверу, который является связующим звеном между клиентом и базой данных, за необходимой информацией. Это взаимодействие осуществляется через запросы и ответы.

Необходимые данные хранятся в базе данных, что обеспечивает надёжность их сохранения и использования впоследствии, а также придаёт им структурность.

При разработке данной программы были использованы паттерны проектирования, что в свою очередь даёт возможность при необходимости расширить функционал приложения, а также изменить уже существующий интерфейс программы под новые конкретные задачи.

В процессе выполнения курсовой работы была изучена предметная область и рассмотрены теоретические аспекты в области поиска работы. Также были выполнены как функциональное моделирование поиска работы на основе методологии IDEF0, так и информационное моделирование.

Результатом проделанной работы является приложение, которое полностью удовлетворяет установленным требованиям.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Для чего нужна работа: [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://www.topauthor.ru/dlya_chego_nugna_rabota_4f76.html>
2. Зачем нужна работа человеку: [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://nachalo-peremen.ru/zachem-nuzhna-rabota-cheloveku/>
3. Преимущества поиска работы онлайн: [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://utro.ru/news/2013/02/21/1102671.shtml>
4. Предметная область: [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://studwood.ru/1705962/informatika/opisanie_predmetnoy_oblasti>
5. Нормализация отношений. Шесть нормальных форм: [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habr.com/post/254773/>
6. Б. Эккель. Философия Java. // СПб: Питер, 2001. – 880 с.
7. Spring Framework: [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.kv.by/archive/index2009291108.htm>
8. Hibernate (библиотека): [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/Hibernate_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)>

# ПРИЛОЖЕНИЕ А

# (обязательное)

**Диаграммы классов**

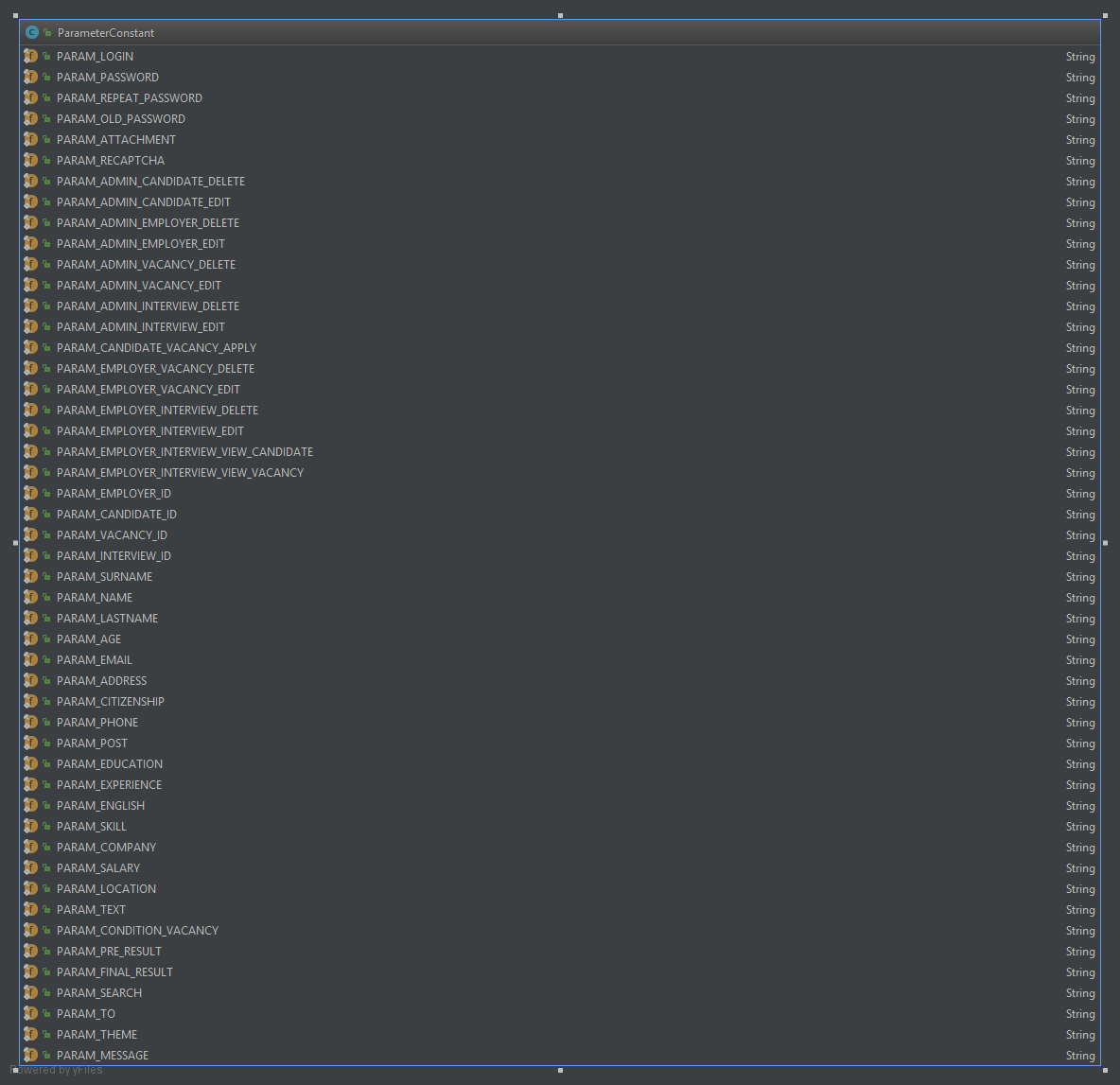


Рисунок А.1 – Диаграмма классов пакета constant

Продолжение приложения А

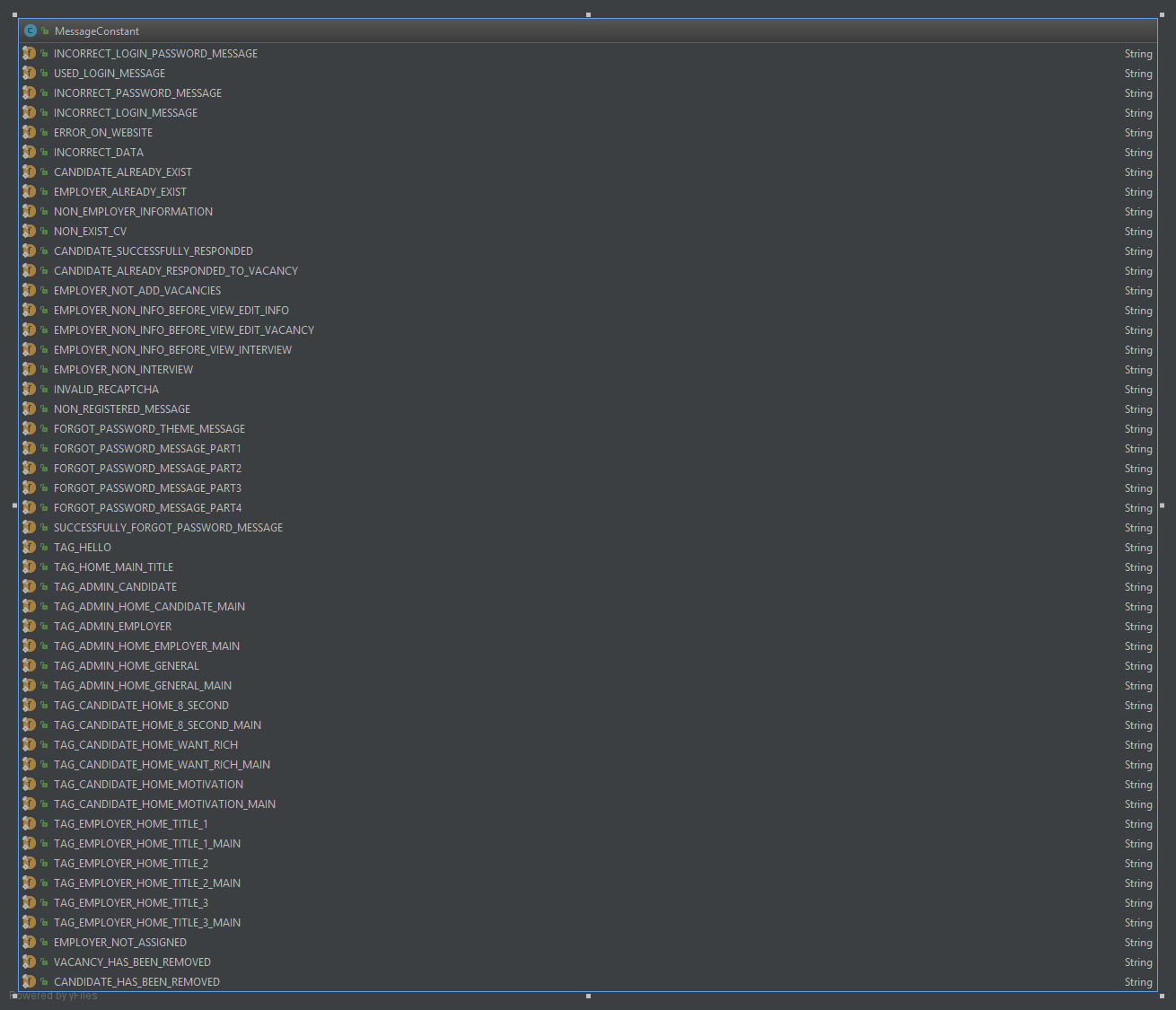


Рисунок А.2 – Диаграмма классов пакета constant

Продолжение приложения А

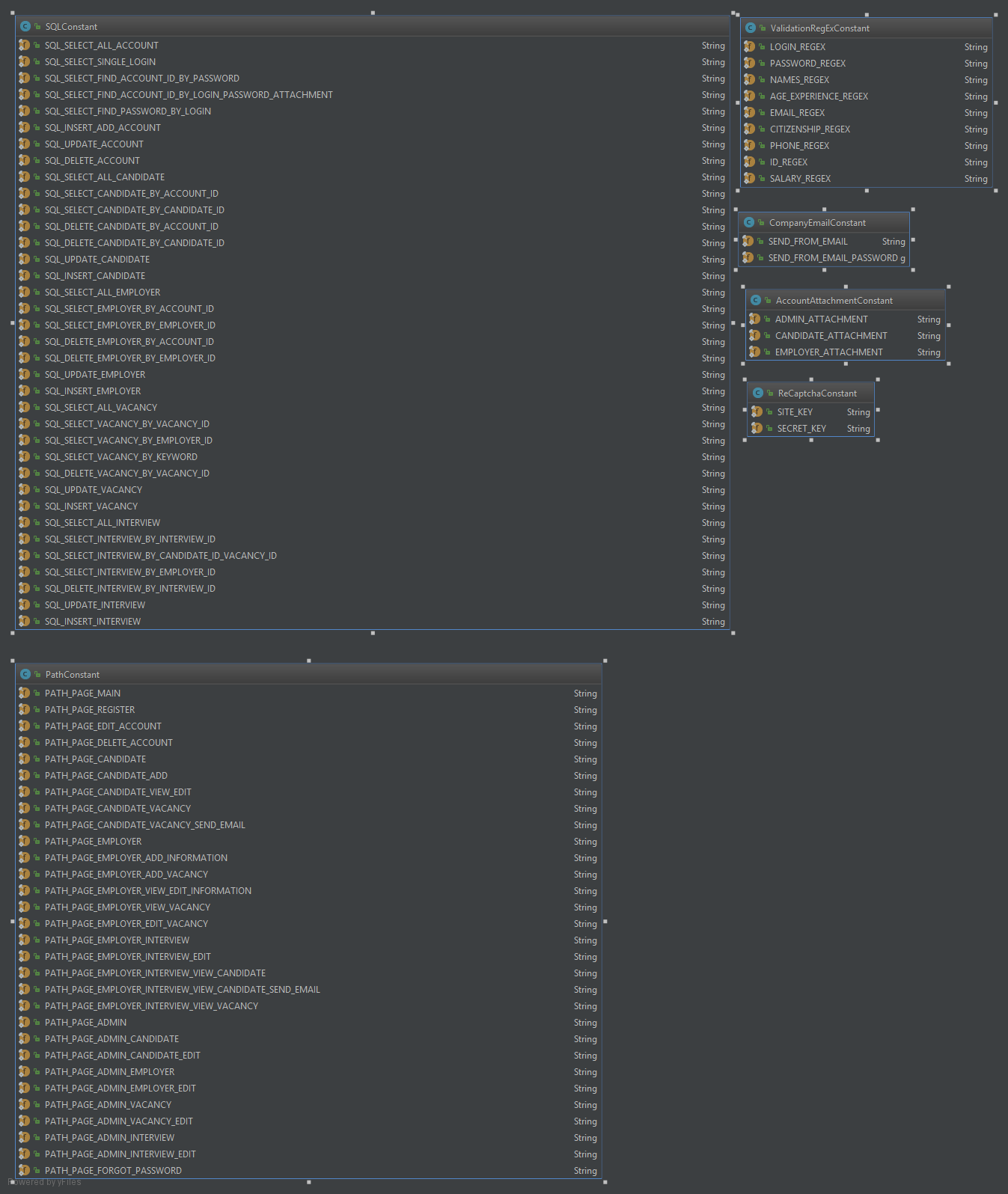


Рисунок А.3 – Диаграмма классов пакета constant

Продолжение приложения А

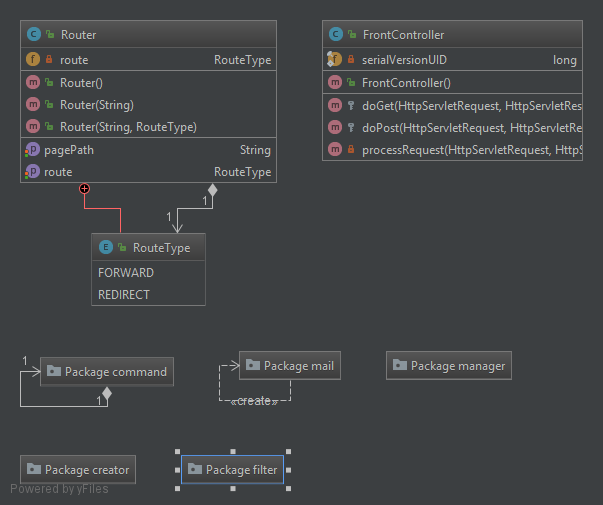


Рисунок А.4 – Диаграмма классов пакета controller

Продолжение приложения А

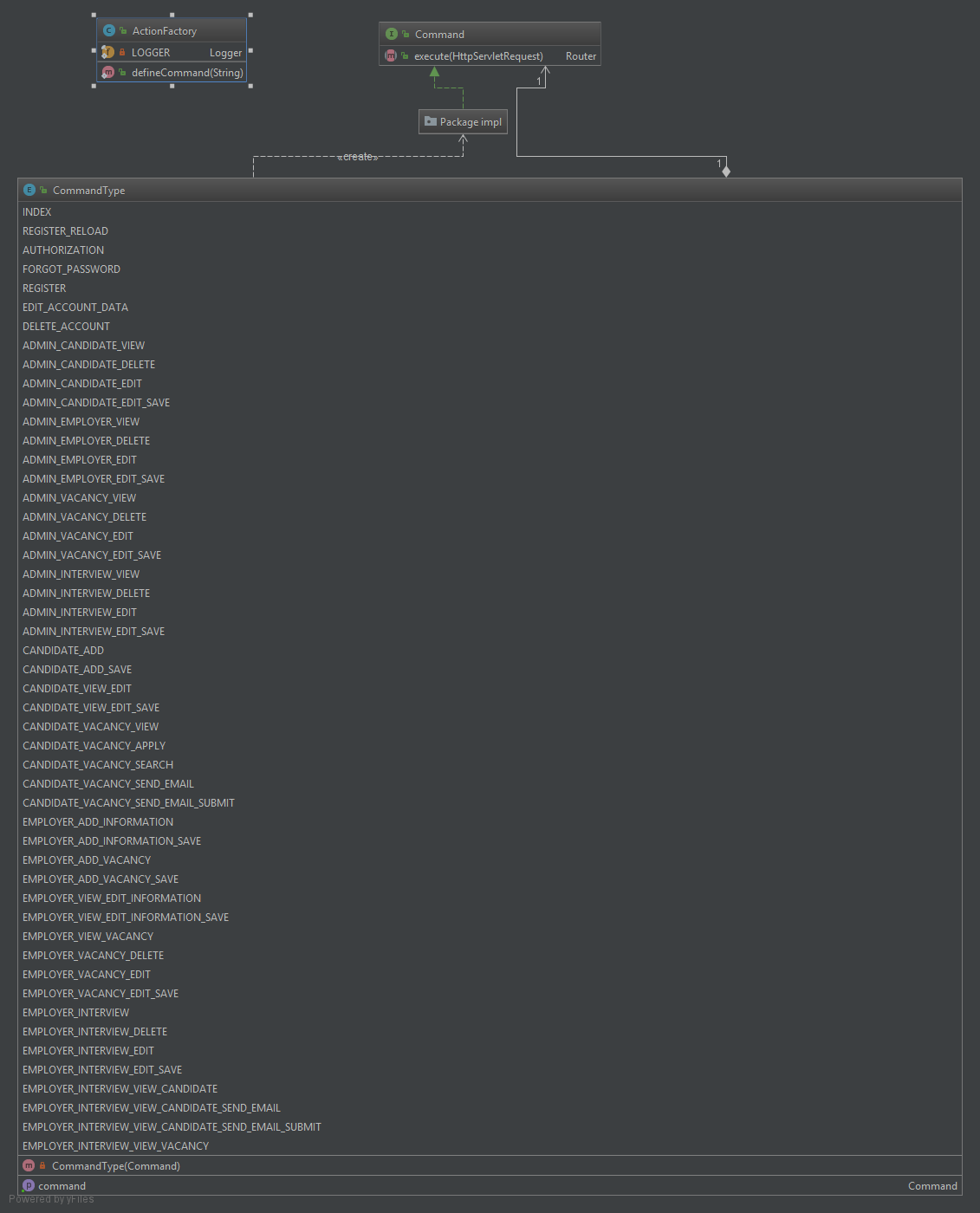


Рисунок А.5 – Диаграмма классов пакета command

Продолжение приложения А

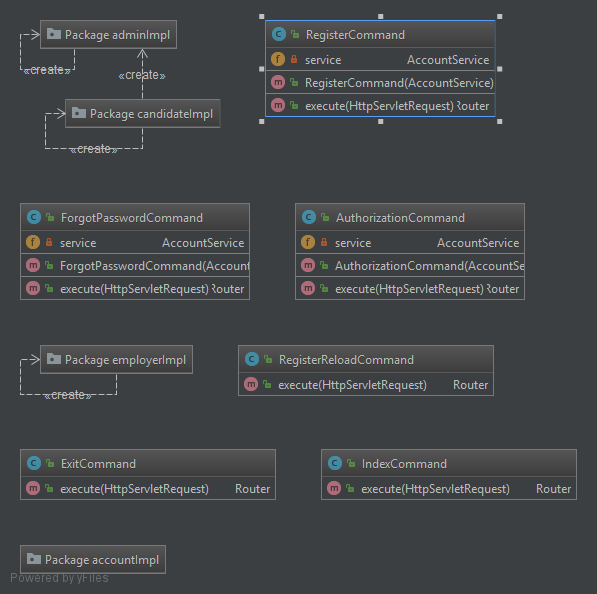


Рисунок А.6 – Диаграмма классов пакета impl

Продолжение приложения А

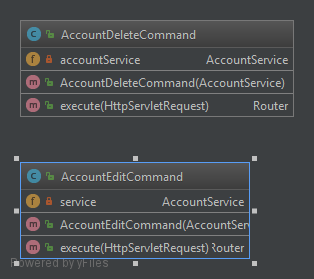


Рисунок А.7 – Диаграмма классов пакета accountimpl

Продолжение приложения А

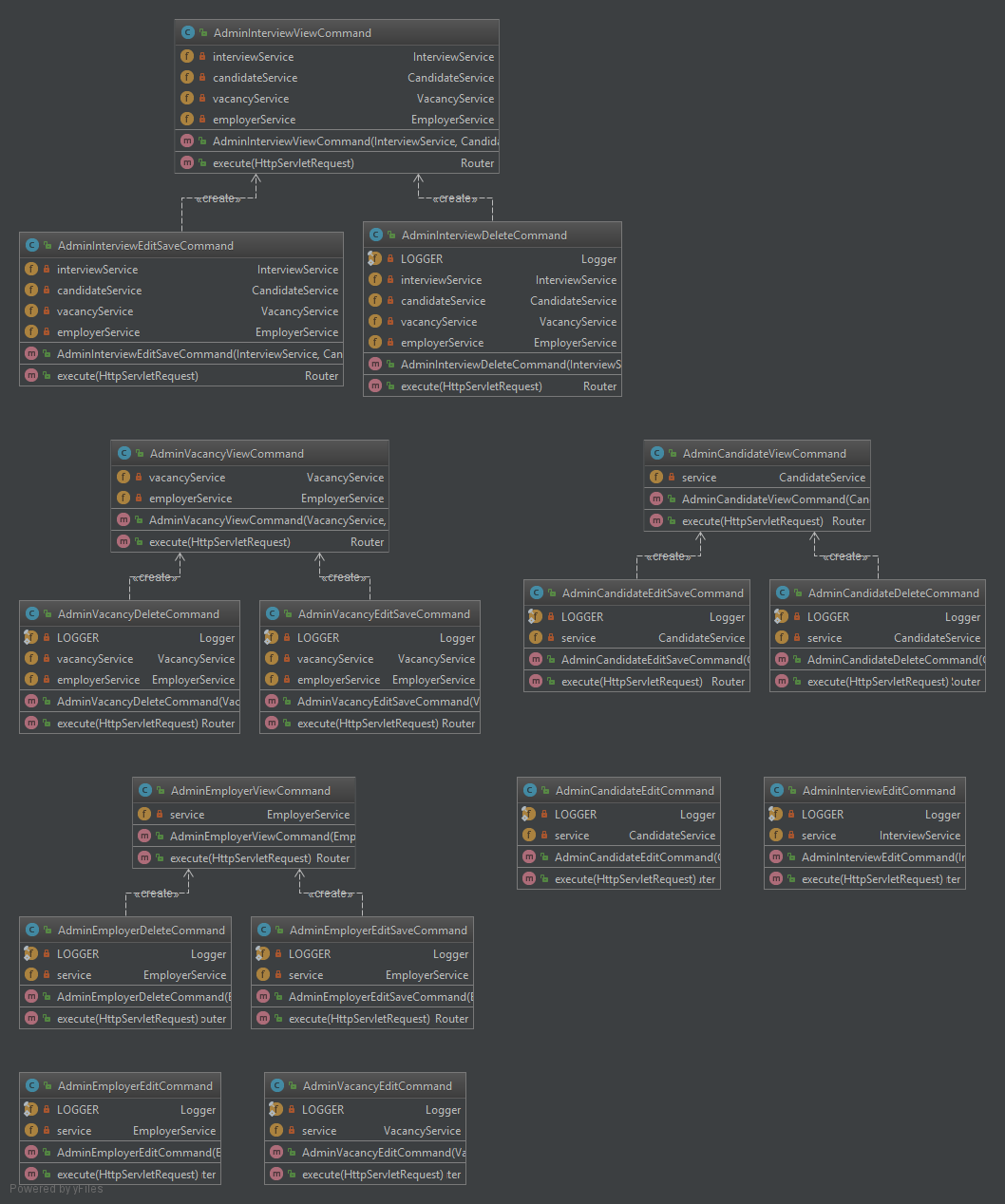


Рисунок А.8 – Диаграмма классов пакета adminimpl

Продолжение приложения А

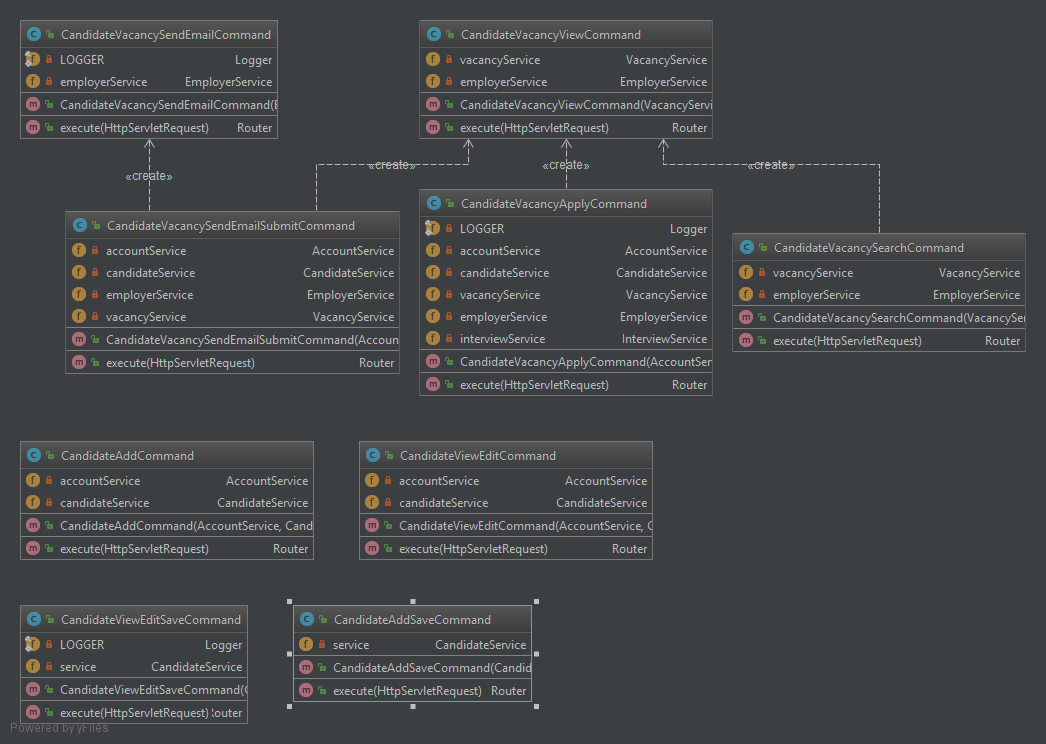


Рисунок А.9 – Диаграмма классов пакета candidateimpl

Продолжение приложения А

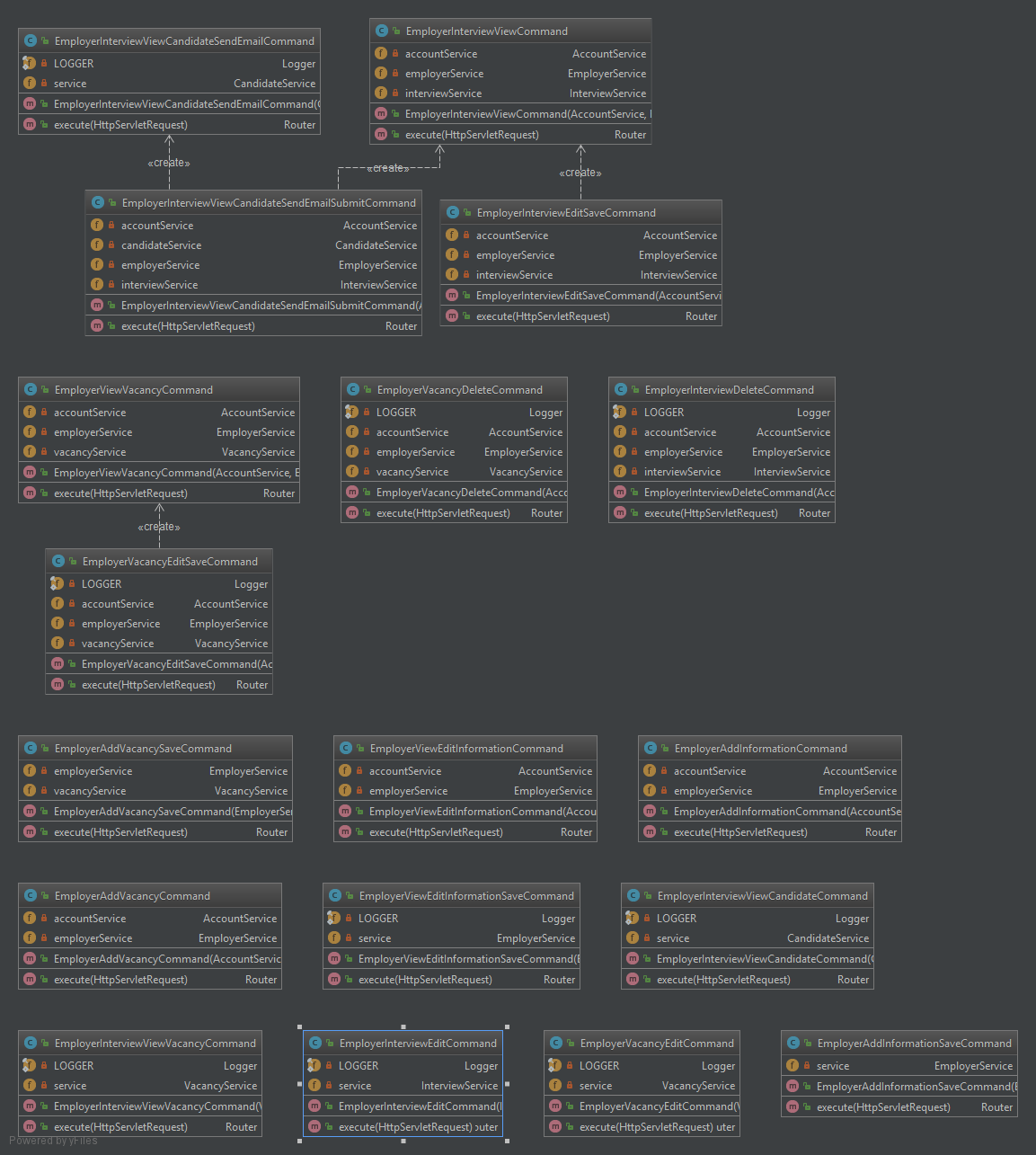


Рисунок А.10 – Диаграмма классов пакета employerimpl

Продолжение приложения А

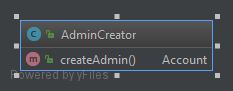


Рисунок А.11 – Диаграмма классов пакета creator

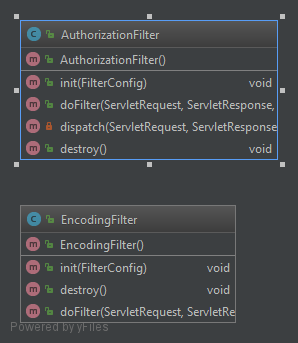


Рисунок А.12 – Диаграмма классов пакета filter

Продолжение приложения А

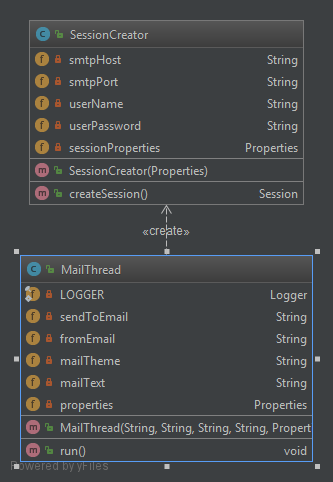


Рисунок А.13 – Диаграмма классов пакета mail

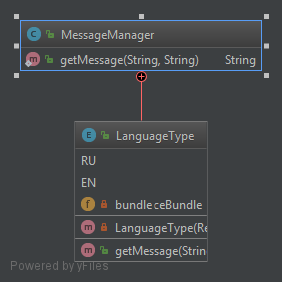


Рисунок А.14 – Диаграмма классов пакета manager

Продолжение приложения А

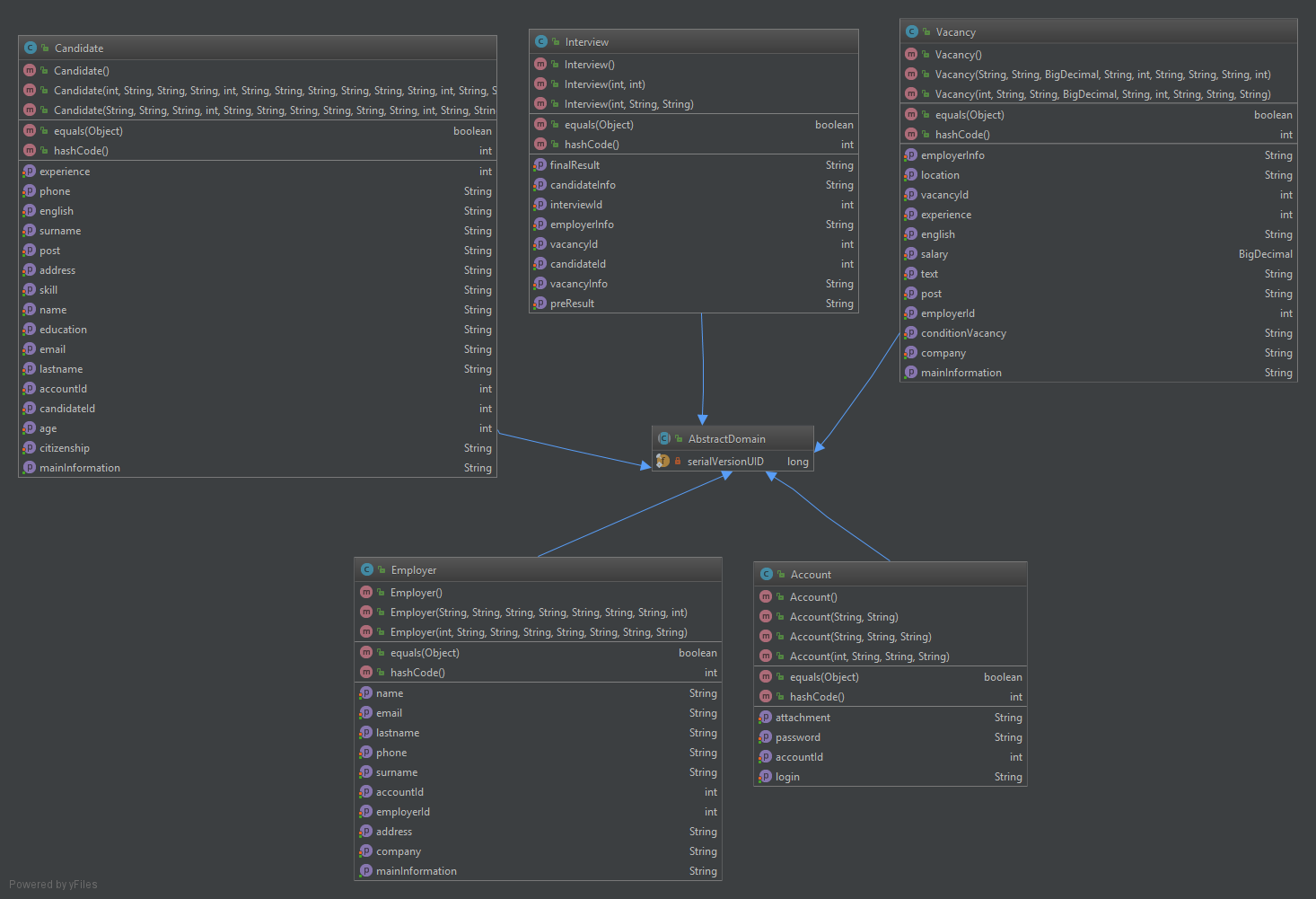


Рисунок А.15 – Диаграмма классов пакета domain

Продолжение приложения А

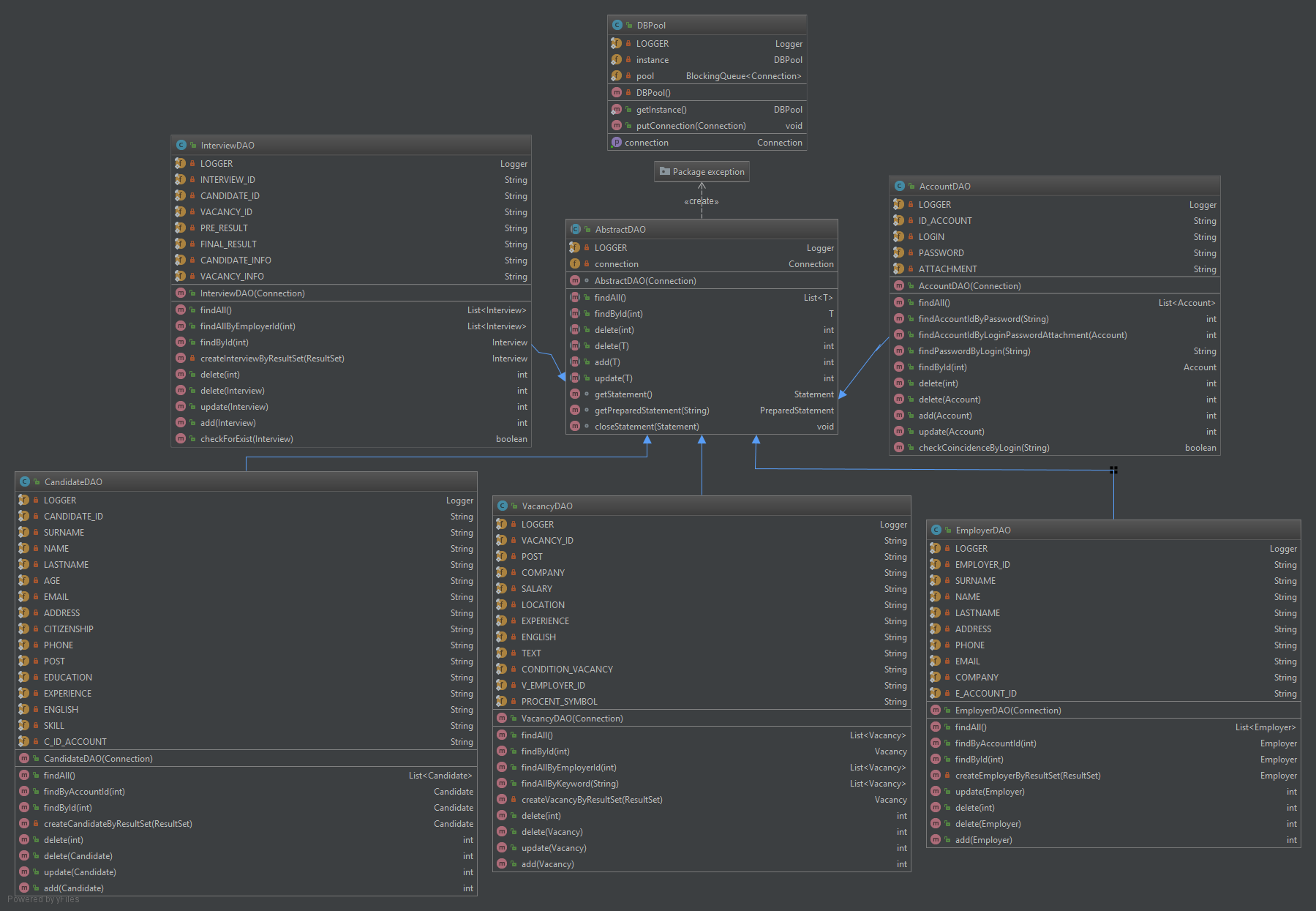


Рисунок А.16 – Диаграмма классовпакета repository

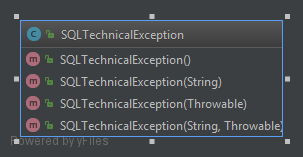


Рисунок А.17 – Диаграмма классов пакета exception

Продолжение приложения А

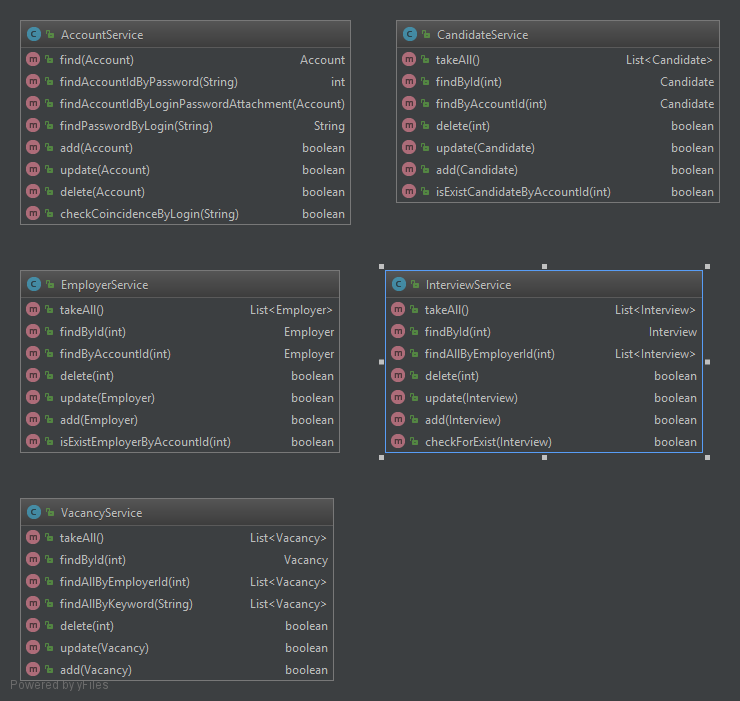


Рисунок А.18 – Диаграмма классов пакета service

Продолжение приложения А

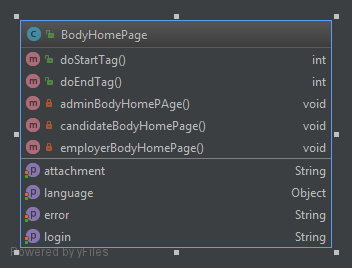


Рисунок А.19 – Диаграмма классов пакета tag

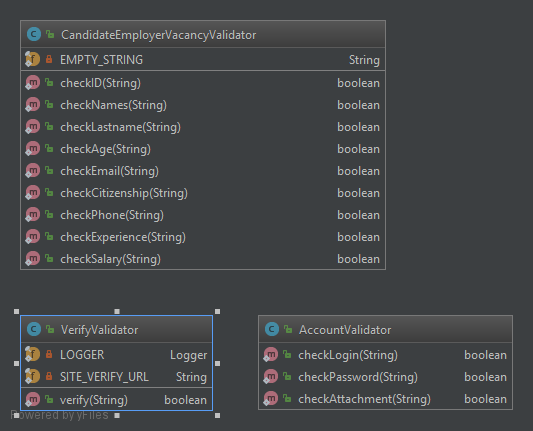


Рисунок А.20 – Диаграмма классов пакета validator

# ПРИЛОЖЕНИЕ Б

# (обязательное)

# Листинг скрипта генерации базы данных

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

-- -----------------------------------------------------

-- Schema HR

-- -----------------------------------------------------

-- Соискатель вакансии регистрируется в системе. Сотрудник HR фиксирует результат предварительного собеседования с Соискателем и назначает при

-- необходимости техническое собеседование, по результатам которого и принимается решение о трудоустройстве Соискателя. Сотрудник HR размещает

-- сообщения о вакансиях и управляет ими. Администратор управляет Сотрудниками и Соискателями.

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `HR` DEFAULT CHARACTER SET utf8 ;

USE `HR` ;

-- -----------------------------------------------------

-- Table `HR`.`account`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `HR`.`account` (

`idAccount` INT(5) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'id аккаунта',

`login` VARCHAR(60) NOT NULL COMMENT 'логин',

`password` VARCHAR(60) NOT NULL COMMENT 'пароль',

`attachment` VARCHAR(10) NULL COMMENT 'приндлежность к Соискателю или Сотруднику HR',

PRIMARY KEY (`idAccount`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `HR`.`candidate`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `HR`.`candidate` (

`idCandidate` INT(5) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'id соискателя',

`surname` VARCHAR(40) NOT NULL COMMENT 'фамилия',

`name` VARCHAR(40) NOT NULL COMMENT 'имя',

`lastname` VARCHAR(40) NULL COMMENT 'отчество',

`age` INT(3) NOT NULL COMMENT 'возраст',

`email` VARCHAR(100) NOT NULL COMMENT 'e-mail',

`address` VARCHAR(60) NOT NULL COMMENT 'адреспроживания',

`citizenship` VARCHAR(80) NOT NULL COMMENT 'гражданство',

`phone` VARCHAR(30) NOT NULL COMMENT 'мобильныйтелефон',

`post` VARCHAR(200) NULL COMMENT 'должность',

`education` VARCHAR(200) NULL COMMENT 'образование',

`experience` INT(2) DEFAULT 0 NULL COMMENT 'опытработы',

`english` VARCHAR(50) NULL COMMENT 'уровеньанглийскогоязыка',

`skill` VARCHAR(255) NULL COMMENT 'основные (ключевые) навыкиСоискателя',

`c\_idAccount` INT(5) UNSIGNED NOT NULL COMMENT 'id аккаунта',

INDEX `idAccount\_idx` (`c\_idAccount` ASC),

PRIMARY KEY (`idCandidate`),

CONSTRAINT `c\_idAccount`

FOREIGN KEY (`c\_idAccount`)

REFERENCES `HR`.`account` (`idAccount`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `HR`.`employer`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `HR`.`employer` (

`idEmployer` INT(5) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'id Сотрудника HR',

`surname` VARCHAR(30) NOT NULL COMMENT 'фамилияСотрудника HR',

`name` VARCHAR(30) NOT NULL COMMENT 'имяСотрудника HR',

`lastname` VARCHAR(30) NULL COMMENT 'отчество',

`address` VARCHAR(60) NOT NULL COMMENT 'адреспроживания',

`phone` VARCHAR(30) NOT NULL COMMENT 'мобильныйтелефон',

`email` VARCHAR(100) NOT NULL COMMENT 'E-mail',

`company` VARCHAR(100) NULL,

`e\_idAccount` INT(5) UNSIGNED NOT NULL COMMENT 'id аккаунта',

PRIMARY KEY (`idEmployer`),

INDEX `idAccount\_idx` (`e\_idAccount` ASC),

CONSTRAINT `e\_idAccount`

FOREIGN KEY (`e\_idAccount`)

REFERENCES `HR`.`account` (`idAccount`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `HR`.`vacancy`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `HR`.`vacancy` (

`idvacancy` INT(5) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'id вакансии',

`post` VARCHAR(150) NOT NULL COMMENT 'должностьнавакансии',

`company` VARCHAR(60) NOT NULL COMMENT 'названиякомпании',

`salary` DECIMAL(8, 2) NULL COMMENT 'заработнаяплата',

`location` VARCHAR(45) NULL COMMENT 'местоположение вакансии (например Минск)',

`experience` INT(5) DEFAULT 0 NULL COMMENT 'необходимыйопытработы',

`english` VARCHAR(50) NULL COMMENT 'необходимый уровень английского языка',

`text` VARCHAR(2000) NOT NULL COMMENT 'сам текст вакансии, необходимые навыки, условия работы и т.д.',

`conditionVacancy` VARCHAR(20) NULL COMMENT 'состояниевакансии',

`v\_idEmployer` INT(5) UNSIGNED NULL COMMENT 'id Сотрудника HR',

PRIMARY KEY (`idvacancy`),

INDEX `idEmployer\_idx` (`v\_idEmployer` ASC),

CONSTRAINT `v\_idEmployer`

FOREIGN KEY (`v\_idEmployer`)

REFERENCES `HR`.`employer` (`idEmployer`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `HR`.`interview`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `HR`.`interview` (

`idInterview` INT(5) UNSIGNED NOT NULL AUTO\_INCREMENT COMMENT 'id собеседования',

`i\_idCandidate` INT(5) UNSIGNED NULL COMMENT 'id Соискателя',

`i\_idvacancy` INT(5) UNSIGNED NULL COMMENT 'id Вакансии',

`preResult` VARCHAR(50) NULL COMMENT 'результат предварительного собеседования',

`finalResult` VARCHAR(50) NULL COMMENT 'результат технического (итогового) собеседования',

PRIMARY KEY (`idInterview`),

INDEX `idvacancy\_idx` (`i\_idvacancy` ASC),

CONSTRAINT `i\_idCandidate`

FOREIGN KEY (`i\_idCandidate`)

REFERENCES `HR`.`candidate` (`idCandidate`)

ON DELETE SET NULL

ON UPDATE CASCADE,

CONSTRAINT `i\_idvacancy`

FOREIGN KEY (`i\_idvacancy`)

REFERENCES `HR`.`vacancy` (`idvacancy`)

ON DELETE SET NULL

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- заполнение данными таблицы account

-- добавление аккаунтов для пяти кандидатов и четырех Сотрудников HR

INSERT INTO account (login, password, attachment) VALUES ('wolfKing1978', 'wolfCandidate1978', 'c');

INSERT INTO account (login, password, attachment) VALUES ('BigBoomTheory', 'BigBoomTheory1', 'c');

INSERT INTO account (login, password, attachment) VALUES ('KingOfTheDot', 'qwerty1234', 'c');

INSERT INTO account (login, password, attachment) VALUES ('IvanSlayer', 'IvanAverin2000', 'c');

INSERT INTO account (login, password, attachment) VALUES ('Western Cinema', 'NormKino1', 'c');

INSERT INTO account (login, password, attachment) VALUES ('Max Direct', 'TheBestHRInThisCompany1', 'e');

INSERT INTO account (login, password, attachment) VALUES ('Eugen Frolov', 'FrolovEugen1', 'e');

INSERT INTO account (login, password, attachment) VALUES ('Krasnov Karl', 'A1234567890', 'e');

INSERT INTO account (login, password, attachment) VALUES ('Avankey Sidney', 'MetroIsLove101010', 'e');

-- -----------------------------------------------------------------

-- заполнение данными таблицы candidate

-- добавление нескольких кандидатов

INSERT INTO candidate (surname, name, lastname, age, email, address, citizenship, phone, post, education, experience, english, skill, c\_idAccount)

VALUES ('Ivanov', 'Ivan', 'Ivanovich', 30, 'ivan.ivanov@gmail.com', 'ул. Притыцкого 12/1, 33', 'Республика Беларусь', '8 (029) 123-12-12',

'PHP Developer', 'БГУ Механико-математический факультет', 4, 'A0 (Абсолютный новичек)', 'Коммуникабелен, легко обучаем', 1);

INSERT INTO candidate (surname, name, lastname, age, email, address, citizenship, phone, post, education, experience, english, skill, c\_idAccount)

VALUES ('Disney', 'Ksenia', 'Viktorovna', 20, 'SmileWether@mail.ru', 'просп. Рокоссовского 100, 1', 'Республика Беларусь',

'+375 29 852-78-98', 'Бизнес-аналитик', 'БГУ Факультет прикладной математики и информатики', 0, 'A1+ (Базовый средний)', 'Курсовые работы по Java, C#, JavaScript. Знание Ajax, Perl.',

2);

INSERT INTO candidate (surname, name, lastname, age, email, address, citizenship, phone, post, education, experience, english, skill, c\_idAccount)

VALUES ('Брусов', 'Кирилл', 'Кириллович', 43, 'bigboss42@gmail.com', 'Ул. Каменогорская 42, 33', 'Российская Федерация', '8 (029) 133-14-34',

'DB Developer', 'НИИ Московая академия сплавов', 22, 'A2+ (Ниже среднего)', 'Огромный опыт в научной сфере, многочисленные работы по сопротивлению материалов, 11 лет преподования в МГУ',

3);

INSERT INTO candidate (surname, name, lastname, age, email, address, citizenship, phone, post, education, experience, english, skill, c\_idAccount)

VALUES ('Аверин', 'Иван', 'Янович', 17, 'ivan.averin.yanovich@gmail.com', 'ул. Платонова 10, 11', 'Республика Беларусь', '8 (029) 147-57-75',

'Junior .NET Developer', 'БГУИР КСИС', 0, 'Не указано', 'Аналитический склад ума', 4);

INSERT INTO candidate (surname, name, age, email, address, citizenship, phone, post, experience, english, skill, c\_idAccount) VALUES ('Сидоров',

'Марк', 23, 'MarkCreator@gmail.com', 'г. Витебск, ул. Маяковского 12/1б, 1', 'РБ', '+375 44 121-02-77', 'PHP developer', 1,

'C1 (Продвинутый)', 'Хакер по призванию', 5);

-- -----------------------------------------------------------------

-- заполнение данными таблицы employer

-- добавление нескольких Сотрудников HR

INSERT INTO employer (surname, name, lastname, address, phone, email, company, e\_idAccount) VALUES ('Директ', 'Максим', 'Юрьевич',

'ул. Руссиянова 12, 1', '8 (029) 456-74-23', 'MaxDirect@gmail.com', 'BelHard', 6);

INSERT INTO employer (surname, name, lastname, address, phone, email, company, e\_idAccount) VALUES ('Фролов', 'Евгений', 'Михайлович',

'ул. Асаналиева 1, 241', '+375 44 111-14-41', 'EugenFrolov@mail.ru', 'ITransition', 7);

INSERT INTO employer (surname, name, lastname, address, phone, email, company, e\_idAccount) VALUES ('Краснов', 'Карл', 'Махмудович',

'ул. Кальварийская 33/1а, 124', '+375 29 320-02-45', 'karlkarlyan42@gmail.com', 'Клевер Парк', 8);

INSERT INTO employer (surname, name, address, phone, email, company, e\_idAccount) VALUES ('Avankey', 'Sidney', 'Platonava st. 23, 23',

'+375 44 147-55-84', 'SidneyAustralia101010@gmail.com', 'BelHard', 9);

-- -----------------------------------------------------------------

-- заполнение данными таблицы vacancy

-- добавление нескольких вакансий

INSERT INTO vacancy (post, company, salary, location, experience, english, text, conditionVacancy, v\_idEmployer) VALUES ('PHP Developer',

'BelHard', 1500, 'г. Могилев', 4, 'C1+ (Выше продвинутого)', 'Понимание принципов ООП и знание паттернов проектирования; Опыт работы и уверенные знания Mysql, SQLite; Опыт работы с современными фреймворками; Хорошее знание Javascript и опыт работы с Jquery, Bootstrap; Базовое знание Unix/Linux; Базовое знание Unix/Linux;',

'Открыта', 1);

INSERT INTO vacancy (post, company, salary, location, english, text, conditionVacancy, v\_idEmployer) VALUES ('Junior .NET Developer', 'BelHard',

500, 'г. Минск', 'B2+ (Ниже продвинутого)', 'Теоретическая база по .NET, C#; Знание основ HTML, CSS, JavaScript; Опыт работы c .NET, C# в рамках учебных проектов (лабораторные, курсовые, диплом) приветствуется; Опыт коммерческой разработки приветствуется;',

'Открыта', 1);

INSERT INTO vacancy (post, company, salary, location, experience, english, text, conditionVacancy, v\_idEmployer) VALUES ('Бизнес-аналитик',

'ITransition', 450, 'г. Витебск', 1, 'B1+ (Выше среднего)', 'Работа с веб-проектами, составление технического задания, постановка задач инженерам программистам, представление результатов работ заказчику, сдача-приемка работ',

'Открыта', 2);

INSERT INTO vacancy (post, company, salary, location, english, text, conditionVacancy, v\_idEmployer) VALUES ('Бизнес-аналитик', 'КлеверПарк',

380, 'г. Минск', 'A2+ (Ниже среднего)', 'Умение работать в команде, Знание достаточно большого количества ПО, Знание методологии разработки программных продуктов, Активность и желание развиваться',

'Закрыта', 3);

INSERT INTO vacancy (post, company, salary, location, experience, english, text, conditionVacancy, v\_idEmployer) VALUES ('Database developer',

'BelHard', 720, 'г. Гомель', 2, 'A1+ (Базовый средний)', 'Знание языка запросов SQL (например, Transact-SQL); Знание прикладного языка программирования (например, С#); Знание технологий, обеспечивающих взаимодействие, связь приложения с базой данных (ADO.NET, Entity Framework, LINQ)',

'Открыта', 4);

-- -----------------------------------------------------------------

-- заполнение данными таблицы interview

-- добавление интервью между кандидатами на вакансию и сотрудниками HR

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (1, 1, 'Неуказано', 'Неуказано');

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (5, 1, 'Провал', 'Провал');

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (3, 5, 'Успешно', 'Успешно');

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (2, 3, 'Естьвопросы', 'Неуказано');

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (2, 4, 'Успешно', 'Сомнительно');

INSERT INTO interview (i\_idCandidate, i\_idvacancy, preResult, finalResult) VALUES (4, 2, 'Успешно', 'Естьвопросы');

-- -----------------------------------------------------------------

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

# ПРИЛОЖЕНИЕ В

# (обязательное)

**Листинг основных элементов программы**

Файл ActionFactory.java

**package** by.bsuir.abmyotkashevtsov.controller.command;  
  
**import** org.apache.logging.log4j.Level;  
**import** org.apache.logging.log4j.LogManager;  
**import** org.apache.logging.log4j.Logger;  
  
**import** java.util.Optional;  
  
*/\*\*  
 \* Class, to select a command from the enum 'CommandType' combing by the string value of the command that came with  
 \* jsp.  
 \*/***public class** ActionFactory {  
 **private final static** Logger ***LOGGER*** = LogManager.*getLogger*(ActionFactory.**class**);  
  
 **public static** Optional<Command> defineCommand(String commandName) {  
 Optional<Command> current = Optional.*empty*();  
 **if** (commandName == **null**) {  
 **return** current;  
 }  
 **try** {  
 CommandType type = CommandType.*valueOf*(commandName.toUpperCase());  
 current = Optional.*of*(type.getCommand());  
 } **catch** (IllegalArgumentException e) {  
 ***LOGGER***.log(Level.***ERROR***, **"Illegal argument type of \"Command\"! Detail: "** + e.getMessage());  
 }  
 **return** current;  
 }  
}

Файл RegisterCommand.java

**package** by.bsuir.abmyotkashevtsov.controller.command.impl;  
  
**import** by.bsuir.abmyotkashevtsov.constant.AccountAttachmentConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.MessageConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.ParameterConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.PathConstant;  
**import** by.bsuir.abmyotkashevtsov.controller.Router;  
**import** by.bsuir.abmyotkashevtsov.controller.command.Command;  
**import** by.bsuir.abmyotkashevtsov.controller.manager.MessageManager;  
**import** by.bsuir.abmyotkashevtsov.domain.Account;  
**import** by.bsuir.abmyotkashevtsov.service.AccountService;  
**import** by.bsuir.abmyotkashevtsov.validator.AccountValidator;  
**import** by.bsuir.abmyotkashevtsov.validator.VerifyValidator;

Продолжение приложения В

**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpSession;  
  
*/\*\*  
 \* Implementation of the "register" command.  
 \*  
 \* <p>  
 \* Accepts from the page login, password, re-password, role and value of the ReCaptcha for registration of a new  
 \* user of the system. The validation of the above fields for validity occurs, in case of failure, the user  
 \* receives a corresponding message on the page. Also, registration will fail for the user if the login is  
 \* already use.  
 \* </p>  
 \*  
 \** ***@see*** *Command#execute(HttpServletRequest)  
 \*/***public class** RegisterCommand **implements** Command {  
 **private** AccountService **service**;  
  
  
 */\*\*  
 \* Constructs and initialize commands type of 'register'.  
 \*  
 \** ***@param service*** *- instance of the service type of "Account" to access the database table "account".  
 \*/* **public** RegisterCommand(AccountService service) {  
 **this**.**service** = service;  
 }  
  
 @Override  
 **public** Router execute(HttpServletRequest request) {  
 Router router = **new** Router(PathConstant.***PATH\_PAGE\_REGISTER***, Router.RouteType.***FORWARD***);  
 String login = request.getParameter(ParameterConstant.***PARAM\_LOGIN***);  
 String password = request.getParameter(ParameterConstant.***PARAM\_PASSWORD***);  
 String repeatPassword = request.getParameter(ParameterConstant.***PARAM\_REPEAT\_PASSWORD***);  
 String attachment = request.getParameter(ParameterConstant.***PARAM\_ATTACHMENT***);  
 String gRecaptchaResponse = request.getParameter(ParameterConstant.***PARAM\_RECAPTCHA***);  
  
 HttpSession session = request.getSession(**true**);  
  
 **if**(AccountValidator.*checkLogin*(login) && AccountValidator.*checkPassword*(password) &&  
 AccountValidator.*checkAttachment*(attachment) && password.equals(repeatPassword)) {  
 **if** (VerifyValidator.*verify*(gRecaptchaResponse)) {  
 Account user = **new** Account(login, password, attachment);  
 **if** (!**service**.checkCoincidenceByLogin(login)) {  
 **if** (**service**.add(user)) {  
 session.setAttribute(**"role"**, user);  
 **if** (attachment.equals(AccountAttachmentConstant.***CANDIDATE\_ATTACHMENT***)) {

Продолжение приложения В

router.setRoute(Router.RouteType.***REDIRECT***);  
 router.setPagePath(PathConstant.***PATH\_PAGE\_CANDIDATE***);  
 } **else** {  
 router.setRoute(Router.RouteType.***REDIRECT***);  
 router.setPagePath(PathConstant.***PATH\_PAGE\_EMPLOYER***);  
 }  
 } **else** {  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(language.toString(),  
 MessageConstant.***ERROR\_ON\_WEBSITE***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
 } **else** {  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(language.toString(), MessageConstant.***USED\_LOGIN\_MESSAGE***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
 } **else** {  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(language.toString(),  
 MessageConstant.***INVALID\_RECAPTCHA***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
 } **else** {  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(language.toString(),  
 MessageConstant.***INCORRECT\_LOGIN\_PASSWORD\_MESSAGE***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
  
 **return** router;  
 }  
}

Файл CandidateVacancySendEmailSubmitCommand.java

**package** by.bsuir.abmyotkashevtsov.controller.command.impl.candidateImpl;  
  
**import** by.bsuir.abmyotkashevtsov.constant.MessageConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.ParameterConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.PathConstant;  
**import** by.bsuir.abmyotkashevtsov.controller.Router;  
**import** by.bsuir.abmyotkashevtsov.controller.command.Command;  
**import** by.bsuir.abmyotkashevtsov.controller.mail.MailThread;  
**import** by.bsuir.abmyotkashevtsov.controller.manager.MessageManager;  
**import** by.bsuir.abmyotkashevtsov.domain.Account;  
**import** by.bsuir.abmyotkashevtsov.domain.Candidate;  
**import** by.bsuir.abmyotkashevtsov.service.AccountService;

Продолжение приложения В

**import** by.bsuir.abmyotkashevtsov.service.CandidateService;  
**import** by.bsuir.abmyotkashevtsov.service.EmployerService;  
**import** by.bsuir.abmyotkashevtsov.service.VacancyService;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpSession;  
**import** java.util.Objects;  
**import** java.util.Properties;  
  
*/\*\*  
 \* Implementation of the "candidate\_vacancy\_send\_email\_submit" command.  
 \*  
 \* <p>  
 \* Accepts the data: to whom to send the letter, the subject of the letter, the message of the letter and  
 \* the password from the e-mail of the candidate. Based on these data, it forms an object of the MailTread  
 \* class to send a message.  
 \* </p>  
 \*  
 \** ***@see*** *Command#execute(HttpServletRequest)  
 \*/***public class** CandidateVacancySendEmailSubmitCommand **implements** Command {  
 **private** AccountService **accountService**;  
 **private** CandidateService **candidateService**;  
 **private** EmployerService **employerService**;  
 **private** VacancyService **vacancyService**;  
  
 */\*\*  
 \* Constructs and initialize commands type of 'candidate\_vacancy\_send\_email\_submit'  
 \*  
 \** ***@param accountService*** *- instance of the service type of "Account" to access the database table "account"  
 \** ***@param candidateService*** *- instance of the service type of "Candidate" to access the database table "candidate"  
 \** ***@param employerService*** *- instance of the service type of "Employer" to access the database table "employer"  
 \** ***@param vacancyService*** *- instance of the service type of "Vacancy" to access the database table "vacancy"  
 \*/* **public** CandidateVacancySendEmailSubmitCommand(AccountService accountService, CandidateService candidateService,  
 EmployerService employerService, VacancyService vacancyService) {  
 **this**.**accountService** = accountService;  
 **this**.**candidateService** = candidateService;  
 **this**.**employerService** = employerService;  
 **this**.**vacancyService** = vacancyService;  
 }  
  
 @Override  
 **public** Router execute(HttpServletRequest request) {  
 Router router = **new** Router(PathConstant.***PATH\_PAGE\_CANDIDATE\_VACANCY\_SEND\_EMAIL***, Router.RouteType.***FORWARD***);  
  
 String to = request.getParameter(ParameterConstant.***PARAM\_TO***);  
 String theme = request.getParameter(ParameterConstant.***PARAM\_THEME***);

Продолжение приложения В

String message = request.getParameter(ParameterConstant.***PARAM\_MESSAGE***);  
 String password = request.getParameter(ParameterConstant.***PARAM\_PASSWORD***);  
  
 **if** (!Objects.*equals*(to, **""**) & !Objects.*equals*(theme, **""**) & !Objects.*equals*(message, **""**)) {  
 HttpSession session = request.getSession(**true**);  
 Account candidateAccount = (Account) session.getAttribute(**"role"**);  
 **int** accountId = **accountService**.findAccountIdByLoginPasswordAttachment(candidateAccount);  
  
 **if** (accountId != 0) {  
 Candidate candidate = **candidateService**.findByAccountId(accountId);  
 Properties properties = **new** Properties();  
 properties.setProperty(**"mail.smtp.host"**, **"smtp.gmail.com"**);  
 properties.setProperty(**"mail.smtp.port"**, **"465"**);  
 properties.setProperty(**"mail.user.name"**, candidate.getEmail());  
 properties.setProperty(**"mail.user.password"**, password);  
  
 MailThread mailOperator = **new** MailThread(to, candidate.getEmail(), theme, message, properties);  
 mailOperator.start();  
  
 Command command = **new** CandidateVacancyViewCommand(**vacancyService**, **employerService**);  
 router = command.execute(request);  
 } **else** {  
 router.setPagePath(PathConstant.***PATH\_PAGE\_CANDIDATE***);  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String errorMessage = MessageManager.*getMessage*(language.toString(), MessageConstant.***ERROR\_ON\_WEBSITE***);  
 request.setAttribute(**"errorMessage"**, errorMessage);  
 }  
 } **else** {  
 Command command = **new** CandidateVacancySendEmailCommand(**employerService**);  
 router = command.execute(request);  
 Object language = request.getSession(**true**).getAttribute(**"language"**);  
 String errorMessage = MessageManager.*getMessage*(language.toString(), MessageConstant.***INCORRECT\_DATA***);  
 request.setAttribute(**"errorMessage"**, errorMessage);  
 }  
  
 **return** router;  
 }  
}

Продолжение приложения В

Файл CandidateVacancyApplyCommand.java

**package** by.bsuir.abmyotkashevtsov.controller.command.impl.candidateImpl;  
  
**import** by.bsuir.abmyotkashevtsov.constant.MessageConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.ParameterConstant;  
**import** by.bsuir.abmyotkashevtsov.constant.PathConstant;  
**import** by.bsuir.abmyotkashevtsov.controller.Router;  
**import** by.bsuir.abmyotkashevtsov.controller.command.Command;  
**import** by.bsuir.abmyotkashevtsov.controller.manager.MessageManager;  
**import** by.bsuir.abmyotkashevtsov.domain.Account;  
**import** by.bsuir.abmyotkashevtsov.domain.Candidate;  
**import** by.bsuir.abmyotkashevtsov.domain.Interview;  
**import** by.bsuir.abmyotkashevtsov.service.\*;  
**import** org.apache.logging.log4j.Level;  
**import** org.apache.logging.log4j.LogManager;  
**import** org.apache.logging.log4j.Logger;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpSession;  
  
*/\*\*  
 \* Implementation of the "candidate\_vacancy\_apply" command.  
 \*  
 \* <p>  
 \* Represents the implementation of clicking on the "Apply" button to the vacancy for the candidate.  
 \* After clicking on the button, in the database searching information about this candidate, retrieved his id  
 \* and created a new interview based on the candidate's id and the id of the vacancy that was apply.  
 \* </p>  
 \*  
 \** ***@see*** *Command#execute(HttpServletRequest)  
 \*/***public class** CandidateVacancyApplyCommand **implements** Command {  
 **private final static** Logger ***LOGGER*** = LogManager.*getLogger*(CandidateVacancyApplyCommand.**class**);  
  
 **private** AccountService **accountService**;  
 **private** CandidateService **candidateService**;  
 **private** VacancyService **vacancyService**;  
 **private** EmployerService **employerService**;  
 **private** InterviewService **interviewService**;  
  
 */\*\*  
 \* Constructs and initialize commands type of 'candidate\_vacancy\_apply'  
 \*  
 \** ***@param accountService*** *- instance of the service type of "Account" to access the database table "account"  
 \** ***@param candidateService*** *- instance of the service type of "Candidate" to access the database table "candidate"  
 \** ***@param vacancyService*** *- instance of the service type of "Vacancy" to access the database table "vacancy"  
 \** ***@param employerService*** *- instance of the service type of "Employer" to access the database table "employer"  
 \** ***@param interviewService*** *- instance of the service type of "Interview" to access the database table "interview"  
 \*/*

Продолжение приложения В

**public** CandidateVacancyApplyCommand(AccountService accountService, CandidateService candidateService,  
 VacancyService vacancyService, EmployerService employerService,  
 InterviewService interviewService) {  
 **this**.**accountService** = accountService;  
 **this**.**candidateService** = candidateService;  
 **this**.**vacancyService** = vacancyService;  
 **this**.**employerService** = employerService;  
 **this**.**interviewService** = interviewService;  
 }  
  
 @Override  
 **public** Router execute(HttpServletRequest request) {  
 Router router = **new** Router(PathConstant.***PATH\_PAGE\_CANDIDATE\_VACANCY***, Router.RouteType.***FORWARD***);  
  
 HttpSession session = request.getSession(**true**);  
 Account candidateAccount = (Account)session.getAttribute(**"role"**);  
 **int** accountId = **accountService**.findAccountIdByLoginPasswordAttachment(candidateAccount);  
  
 **if**(accountId != 0) {  
 Candidate candidate = **candidateService**.findByAccountId(accountId);  
 **if** (candidate != **null**) {  
 String stringVacancyId = request.getParameter(ParameterConstant.***PARAM\_CANDIDATE\_VACANCY\_APPLY***);  
 **int** vacancyId = 0;  
 **try** {  
 vacancyId = Integer.*parseInt*(stringVacancyId);  
 } **catch** (NumberFormatException e) {  
 ***LOGGER***.log(Level.***ERROR***, **"Error while parsing string value vacancyId to integer! Detail: "** +  
 e.getMessage());  
 }  
  
 Interview interview = **new** Interview(candidate.getCandidateId(), vacancyId);  
 **if** (!**interviewService**.checkForExist(interview)) {  
 **if** (**interviewService**.add(interview)) {  
 Command command = **new** CandidateVacancyViewCommand(**vacancyService**, **employerService**);  
 router = command.execute(request);  
 Object object = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(object.toString(),  
 MessageConstant.***CANDIDATE\_SUCCESSFULLY\_RESPONDED***);  
 request.setAttribute(**"notificationMessage"**, message);  
 } **else** {  
 router.setPagePath(PathConstant.***PATH\_PAGE\_CANDIDATE***);  
 Object object = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(object.toString(),  
 MessageConstant.***ERROR\_ON\_WEBSITE***);  
 request.setAttribute(**"errorMessage"**, message);

Продолжение приложения В

}  
 } **else** {  
 Object object = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(object.toString(),  
 MessageConstant.***CANDIDATE\_ALREADY\_RESPONDED\_TO\_VACANCY***);  
 request.setAttribute(**"errorMessage"**, message);  
 Command command = **new** CandidateVacancyViewCommand(**vacancyService**, **employerService**);  
 router = command.execute(request);  
 }  
 } **else** {  
 router.setPagePath(PathConstant.***PATH\_PAGE\_CANDIDATE***);  
 Object object = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(object.toString(),  
 MessageConstant.***NON\_EXIST\_CV***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
 } **else** {  
 router.setPagePath(PathConstant.***PATH\_PAGE\_CANDIDATE***);  
 Object object = request.getSession(**true**).getAttribute(**"language"**);  
 String message = MessageManager.*getMessage*(object.toString(),  
 MessageConstant.***ERROR\_ON\_WEBSITE***);  
 request.setAttribute(**"errorMessage"**, message);  
 }  
  
 **return** router;  
 }  
}

Файл Controller.java

**package** by.bsuir.abmyotkashevtsov.controller;  
  
**import** by.bsuir.abmyotkashevtsov.constant.PathConstant;  
**import** by.bsuir.abmyotkashevtsov.controller.command.ActionFactory;  
**import** by.bsuir.abmyotkashevtsov.controller.command.Command;  
**import** by.bsuir.abmyotkashevtsov.controller.command.impl.ExitCommand;  
  
**import** javax.servlet.RequestDispatcher;  
**import** javax.servlet.ServletException;  
**import** javax.servlet.annotation.WebServlet;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.util.Optional;  
  
@WebServlet(urlPatterns = { **"/"**})  
**public class** FrontController **extends** HttpServlet{  
 **private static final long *serialVersionUID*** = 1L;  
  
 **public** FrontController() {  
 **super**();  
 }

Продолжение приложения В

@Override  
 **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 processRequest(request, response);  
 }  
  
 @Override  
 **protected void** doPost(HttpServletRequest request, HttpServletResponse response)  
 **throws** ServletException, IOException {  
 processRequest(request, response);  
 }  
  
 **private void** processRequest(HttpServletRequest request, HttpServletResponse response)  
 **throws** ServletException, IOException {  
 request.setCharacterEncoding(**"UTF-8"**);  
 response.setCharacterEncoding(**"UTF-8"**);  
 response.setContentType(**"text/html; charset=UTF-8"**);  
  
 Optional<Command> commandOptional = ActionFactory.*defineCommand*(request.getParameter(**"command"**));  
 Command command = commandOptional.orElse(**new** ExitCommand());  
 Router router = command.execute(request);  
  
 **if** (router != **null**) {  
 String page = router.getPagePath();  
 **if** (router.getRoute() == Router.RouteType.***FORWARD***) {

RequestDispatcher dispatcher = request.getRequestDispatcher(page);  
 dispatcher.forward(request, response);  
 } **else** {  
 response.sendRedirect(request.getContextPath() + page);  
 }  
 } **else** {  
 request.getSession().invalidate();  
 response.sendRedirect(request.getContextPath() + PathConstant.***PATH\_PAGE\_MAIN***);  
 }  
 }  
}