МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт информационных и телекоммуникационных технологий

Кафедра кибербезопасность, обработка и хранение информации

Мухтарханов Бейбарыс Дәулетұлы

«Разработка фронтальной системы банковского обслуживания»

**ДИПЛОМНЫЙ ПРОЕКТ**

Специальность 5В070300 – «Информационные системы»

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт информационных и телекоммуникационных технологий

Кафедра кибербезопасность, обработка и хранение информации

**ДОПУЩЕН К ЗАЩИТЕ**
Заведующий кафедрой
КБОиХИ,
канд.техн.наук,
ассистент-профессора
_____ Н.А. Сейлова
"__14__" ____05____ 2019 г.

# ДИПЛОМНЫЙ ПРОЕКТ

На тему: «Разработка фронтальной системы банковского обслуживания»

по специальности 5В070300 – «Информационные системы»

Выполнил Б. Д. Мухтарханов

Рецензент,
заместитель руководителя Национальной научной лаборатории коллективного пользования информационных и космических технологий НАО КазНИТУ имени К.И.Сатпаева, PhD
_____ К. А. Бостанбеков
"__8__ мая ____ 2019 г.

Научный руководитель,
тьютор
_____ Б. Н. Оразов
"__13__" ____05____ 2019 г.

Алматы 2019

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт информационных и телекоммуникационных технологий

Кафедра кибербезопасность, обработка и хранение информации

5В070300 – «Информационные системы»

**УТВЕРЖДАЮ**
Заведующий кафедрой,
КБОиХИ,
канд.техн.наук,
ассистент-профессора
_____ Н.А. Сейлова
"14" 05 2019 г.

**ЗАДАНИЕ**
**на выполнение дипломного проекта**

Обучающемуся Мухтарханову Бейбарысу Дәулетұлы
Тема: «Разработка фронтальной системы банковского обслуживания»
Утверждена *приказом Ректором Университета* № 1162-б от 16.10.2018 г.
Срок сдачи законченной работы «15» мая 2019 г.
Исходные данные к дипломному проекту: результат литературного обзора современного состояния по данной теме, сбор теоретического материала.
Краткое содержание дипломного проекта:
а) анализ и обзор современных веб-приложений;
б) модели и службы функционирования веб-приложения;
в) информационное и программное обеспечение.
Перечень графического материала (с точным указанием обязательных чертежей): представлены 16 слайдов презентации работы
Рекомендуемая основная литература: из 12 наименований

**ГРАФИК**

подготовки дипломной работы

| Наименование разделов, перечень разрабатываемых вопросов | Сроки представления научному руководителю и консультантам | Примечание |
|---|---|---|
| Анализ и обзор современных веб-приложений | 15.01.2019 – 17.03.2019 | |
| Модели и службы функционирования веб-приложения | 10.02.2019 – 20.03.2019 | |
| Информационное и программное обеспечение | 21.03.2019 – 30.04.2019 | |

**Подписи**

консультантов и нормоконтролера на законченную дипломную работу с указанием относящихся к ним разделов работы

| Наименование разделов | Научный руководитель, консультанты, И.О.Ф. (уч. степень, звание) | Дата подписания | Подпись |
|---|---|---|---|
| Программное обеспечение | м.т.н<br>М.Б. Бауржан | 8.05.19 | |
| Нормоконтролер | Сениор-лектор, PhD<br>О. В. Киселева | 8.05.19 | |

Научный руководитель _____ тьютор Б. Н. Оразов

Задание принял к исполнению обучающийся _____ Б. Д. Мухтарханов

Дата "8" 01 20 19 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
СӘТБАЕВ УНИВЕРСИТЕТІ

# РЕЦЕНЗИЯ

на дипломный проект

**Мухтарханов Бейбарыс Дәулетұлы**

Специальности 5В070300 Информационные системы
На тему: **«Фронтальная система банковского обслуживания»**

Выполнено:
а) графическая часть на 16 листах
б) пояснительная записка на 59 страницах

## ЗАМЕЧАНИЯ К РАБОТЕ

Данная дипломная работа посвящена разработке веб приложения для обеспечения безопасности авторизации пользователей в системе банковского обслуживания. Актуальность данной работы связана со снижением затрат, сокращением времени обслуживания клиентов, быстрой и улучшенной взаимосвязью между отделами, повышением мобильности сотрудников предприятия.

В первом разделе проведен обзор и анализ веб-приложений, клиент-серверной архитектуры и MVC. Во втором разделе описана функциональная часть системы и алгоритм работы веб-приложения. В данном разделе не хватает подробной иллюстрации рабочего процесса передачи данных между отделами предприятия. В третьем разделе описываются информационное и программное обеспечение разрабатываемой системы. В целом разрабатываемая система требует дополнительных доработок в плане пользовательского интерфейса.

Оценка работы

Считаю, что данный дипломный проект выполнен на должном уровне и соответствует необходимым требованиям. Студент демонстрирует навыки программирования на языке Java, и работы с СУБД MySQL.

Учитывая выше изложенное, считаю, что дипломный проект Мухтарханова Бейбарыса на тему: «Фронтальная система банковского обслуживания» заслуживает оценки 95 баллов (Отлично), и присвоения академической степени бакалавра по специальности 5В070300 – Информационные системы.

**Рецензент**
Заместитель руководителя Национальной
научной лаборатории коллективного пользования
информационных и космических технологий
НАО КазНИТУ имени К.И. Сатпаева, PhD
_____ К. А. Бостанбеков

Кызметкерлермен жұмыс бөлімі

«13» 05 2019 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
СӘТБАЕВ УНИВЕРСИТЕТІ

# ОТЗЫВ
# НАУЧНОГО РУКОВОДИТЕЛЯ

на дипломный проект

**Мухтарханова Бейбарыса Дәулетұлы**

Специальности 5В070300 Информационные системы

Тема: **Разработка фронтальной системы банковского обслуживания**

Дипломная работы написана студентом Мухтархановым Бейбарысом Дәулетұлы на актуальную в настоящий момент тему «Фронтальная система банковского обслуживания». Данная тема является важной и актуальной по ряду причин. В настоящее время в мировом рынке каждая корпоративная компания нуждается в BPM услуге. Поставленные цели и задачи полностью соответствуют теме исследования. Дипломная работа написана на основе современных статистических данных и статей ученых, авторитетных в данной области.

Дипломная работа состоит из 3 глав, введения, заключения и списка использованной литературы. Оформление диплома соответствует стандартам.

Во введении содержится обоснование актуальности работы, цели, задачи, методы исследования, а также положения, выносимые на защиту.

В первой главе рассматриваются теоретические аспекты исследования, посвященные управления бизнес-процессами деятельности в IT индустрии. В главу также включены ключевые понятия по данной проблеме.

Во второй главе проводится анализ Фронтальной системы. Глава содержит в себе большое количество практических материалов и их анализ.

В третьей главе предложены возможные способы совершенствования фронтальной системы деятельности данного предприятия, а также представлены перспективы развития. Предложенные рекомендации весьма интересны и заслуживают внимания.

В процессе написания дипломной работы студент соблюдал сроки календарного графика и проявил хорошие навыки работы с теоретическими и статистическими материалами.

В целом студент полно и точно раскрыл тему дипломной работы. Недостатков обнаружено не было. Работа допускается к защите. Рекомендуемая оценка – «отлично».

**Научный руководитель**
Тьютор
_____ Б. Н. Оразов

«13» 05 2019 г.

**Протокол анализа Отчета подобия**

**заведующего кафедрой / начальника структурного подразделения**

Заведующий кафедрой / начальник структурного подразделения заявляет, что ознакомился(-ась) с Полным отчетом подобия, который был сгенерирован Системой выявления и предотвращения плагиата в отношении работы:

**Автор**: Мухтарханов Бейбарыс

**Название**: Фронтальная система банковского обслуживания

**Координатор:** Бауыржан Оразов

**Коэффициент подобия 1:**1

**Коэффициент подобия 2:**0

**Тревога:3**

**После анализа отчета подобия заведующий кафедрой / начальник структурного подразделения констатирует следующее:**

☑ обнаруженные в работе заимствования являются добросовестными и не обладают признаками плагиата. В связи с чем, работа признается самостоятельной и допускается к защите;
☐ обнаруженные в работе заимствования не обладают признаками плагиата, но их чрезмерное количество вызывает сомнения в отношении ценности работы по существу и отсутствием самостоятельности ее автора. В связи с чем, работа должна быть вновь отредактирована с целью ограничения заимствований;
☐ обнаруженные в работе заимствования являются недобросовестными и обладают признаками плагиата, или в ней содержатся преднамеренные искажения текста, указывающие на попытки сокрытия недобросовестных заимствований. В связи с чем, работа не допускается к защите.

Обоснование:

.....
.....
.....
.....
.....

..... .....

Дата 14.05.19г *Подпись заведующего кафедрой /*

*начальника структурного подразделения*

**Окончательное решение в отношении допуска к защите, включая обоснование:**

.....
.....
Допускается к защите
.....
.....

..... .....

Дата 14.05.19 *Подпись заведующего кафедрой*

*начальника структурного подразделения*

**Протокол анализа Отчета подобия Научным руководителем**

Заявляю, что я ознакомился(-ась) с Полным отчетом подобия, который был сгенерирован Системой выявления и предотвращения плагиата в отношении работы:

**Автор:** Мухтарханов Бейбарыс

**Название:** Фронтальная система банковского обслуживания

**Координатор:**Бауыржан Оразов

**Коэффициент подобия 1**:1

**Коэффициент подобия 2**:0

**Тревога:3**

**После анализа Отчета подобия констатирую следующее:**

☐ обнаруженные в работе заимствования являются добросовестными и не обладают признаками плагиата. В связи с чем, признаю работу самостоятельной и допускаю ее к защите;

☐ обнаруженные в работе заимствования не обладают признаками плагиата, но их чрезмерное количество вызывает сомнения в отношении ценности работы по существу и отсутствием самостоятельности ее автора. В связи с чем, работа должна быть вновь отредактирована с целью ограничения заимствований;

☐ обнаруженные в работе заимствования являются недобросовестными и обладают признаками плагиата, или в ней содержатся преднамеренные искажения текста, указывающие на попытки сокрытия недобросовестных заимствований. В связи с чем, не допускаю работу к защите.

Обоснование:

Допущен к защите

13.05.2019

*Дата*

*Подпись Научного руководителя*

## Краткий отчет

| | |
|---|---|
| **Университет:** | Satbayev University |
| **Название:** | Фронтальная система банковского обслуживания |
| **Автор:** | Мухтарханов Бейбарыс |
| **Координатор:** | Бауыржан Оразов |
| **Дата отчета:** | 2019-05-06 08:42:54 |
| **Коэффициент подобия № 1:** | **1,0%** |
| **Коэффициент подобия № 2:** | **0,0%** |
| **Длина фразы для коэффициента подобия № 2:** | 25 |
| **Количество слов:** | 5 944 |
| **Число знаков:** | 47 009 |
| **Адреса пропущенные при проверке:** | |
| **Количество завершенных проверок:** | 47 |

**К вашему сведению, некоторые слова в этом документе содержат буквы из других алфавитов. Возможно - это попытка скрыть позаимствованный текст. Документ был проверен путем замещения этих букв латинским эквивалентом. Пожалуйста, уделите особое внимание этим частям отчета. Они выделены соответственно.**
**Количество выделенных слов 3**

>> **Самые длинные фрагменты, определеные, как подобные**

| № | Название, имя автора или адрес гиперссылки *(Название базы данных)* | Автор | Количество одинаковых слов |
|---|---|---|---|
| 1 | URL_ http://spring-projects.ru/guides/maven/ | | 10 |
| 2 | URL_ http://spring-projects.ru/guides/maven/ | | 8 |
| 3 | URL_ https://e-libra.ru/read/191977-kak-perejti-na-windows-vista.-nachali.html | | 7 |
| 4 | URL_ https://github.com/codedokode/pasta/blob/master/arch/mvc.md | | 6 |
| 5 | URL_ https://github.com/codedokode/pasta/blob/master/arch/mvc.md | | 6 |
| 6 | URL_ http://spring-projects.ru/guides/maven/ | | 6 |
| 7 | URL_ https://github.com/codedokode/pasta/blob/master/arch/mvc.md | | 5 |

| | | |
|---|---|---|
| 8 | URL_ https://github.com/codedokode/pasta/blob/master/arch/mvc.md | 5 |
| 9 | URL_ https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/connection-string-syntax | 5 |

## >> Документы,содержащие подобные фрагменты: Из домашней базы данных

*Не обнаружено каких-либо заимствований*

## >> Документы,содержащие подобные фрагменты: Из внешних баз данных

*Не обнаружено каких-либо заимствований*

## >> Документы,содержащие подобные фрагменты: Из интернета

Документы, выделенные жирным шрифтом, содержат фрагменты потенциального плагиата, то есть превышающие лимит в длине коэффициента подобия № 2

| № | Источник гиперссылки | Количество одинаковых слов (количество фрагментов) |
|---|---|---|
| 1 | URL_ http://spring-projects.ru/guides/maven/ | 24 (3) |
| 2 | URL_ https://github.com/codedokode/pasta/blob/master/arch/mvc.md | 22 (4) |
| 3 | URL_ https://e-libra.ru/read/191977-kak-perejti-na-windows-vista.-nachali.html | 7 (1) |
| 4 | URL_ https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/connection-string-syntax | 5 (1) |

## АНДАТПА

Бұл дипломдық жұмыс BPM-жүйесін (business process management) әзірлеуге арналған. Бұл жүйе компания үдерістерін, қызметкерлерді жылдам және тиімді басқаруды қамтамасыз етеді, бөлімдер арасында ыңғайлы және мобильді өзара іс-қимыл жасайды.

Әзірленген жүйе MVC архитектурасы бар веб-қосымша болып табылады. Оны құру үшін қауіпсіздікті қамтамасыз ету технологиялары және рөлдерді анықтау арқылы пайдаланушылардың қол жеткізу құқықтарын шектеу қолданылды.

Жүйені ақпараттық қамтамасыз ету MySQL ДББЖ-да жүзеге асырылды. Қосымша екі бөліктен тұрады – серверлік және клиенттік, оларды іске асыру үшін Spring Boot, Bootstrap Framework технологиясы қолданылған.



## АННОТАЦИЯ

Данная дипломная работа посвящена разработке BPM-системы (business process management). Данная система обеспечивает быстрое и эффективное управление процессами компании, сотрудниками, предоставляет удобное и мобильное взаимодействие между отделами.

Разработанная система является веб-приложением с архитектурой MVC. Для ее создания использовались технологии обеспечения безопасности и разграничением прав доступа пользователей, посредством определения ролей.

Информационное обеспечение системы реализованно в СУБД MySQL. Приложение состоит из двух частей – серверной и клиентской, для реализации которых использовались технология Spring Boot, Bootstrap Framework.



## SUMMARY

This thesis is devoted to the development of BPM-system (business process management). This system provides fast and efficient management of company processes, employees, and provides convenient and mobile interaction between departments.

The developed system is a web application with MVC architecture. For its creation, security technologies and the differentiation of user access rights were used, through the definition of roles.

Information support of the system is implemented in MySQL. The application consists of two parts - server and client, for the implementation of which the technology used Spring Boot, Bootstrap Framework.

## СОДЕРЖАНИЕ


| | | |
|---|---|---|
| | Введение | 15 |
| 1 | Обзор и анализ веб-приложений | 16 |
| 1.1 | Алгоритм работы веб-приложения | 16 |
| 1.2 | Виды веб-приложений | 17 |
| 1.3 | Преимущества веб-приложений | 17 |
| 1.4 | Архитектура клиент-сервер | 18 |
| 1.5 | Архитектура MVC | 18 |
| 1.6 | Характеристика фронтальной системы | 20 |
| 1.7 | Постановка задачи дипломной работы | 21 |
| 2 | Модели и службы функционирования системы | 22 |
| 2.1 | Функциональная модель системы | 22 |
| 2.2 | Обобщенный алгоритм функционирования системы | 23 |
| 2.3 | Информационное обеспечение системы | 25 |
| 2.4 | Разработка структуры базы данных | 28 |
| 3 | Информационное и программное обеспечение | 31 |
| 3.1 | Системное программное обеспечение | 31 |
| 3.2 | Прикладное программное обеспечение | 33 |
| 3.3 | Инструментальное программное обеспечение | 38 |
| 3.4 | Функциональное назначение | 39 |
| 3.5 | Вызов и загрузка | 41 |
| 3.6 | Используемые технические средства | 41 |
| | Заключение | 42 |
| | Список использованной литературы | 43 |
| | Приложение | 44 |

# ВВЕДЕНИЕ

В современном информационном мире главным ресурсом общества является информация. Процессы связанные с ней основаны на информационных и коммуникационных технологиях. К информационным технологиям относятся разработка, обслуживание и использование компьютерного программного обеспечения, систем и сетей, для хранения, обработки, анализа и распространения данных. Данные означают информацию, факты, статистику и т. д. Данные бывают структурированные, полуструктурированные, неструктурированные. Структурированные – форма представления данных в базе. Полуструктурированные – html (Hyper Text Markup Language), xml (eXtensible Markup Language) документы. Неструктурированные – текстовые документы, аудио, видео, изображения. Управление компьютером и данными осуществляет программное обеспечение.

Прикладное ПО, или приложение, — программа, которая выполняет определенную функцию и рассчитанная на непосредственное взаимодействие с пользователем.

Одним из таких видов прикладных программ являются фронтальные системы. Фронтальная система - это система, через которую любой пользователь взаимодействует с банком. Это может быть интернет-банк, онлайн приложения, банкоматы, терминалы, интерфейсы для сотрудников в отделениях и другие системы.

Фронтальная система банковского обслуживания относится к BPM- системам (business process management). Бизнес - процесс представляет собой деятельность, которая будет выполнять определенную организационную цель. BPM предназначен для улучшения порядка, понимания и эффективности коллективных рабочих процессов, составляющих любой данный бизнес-процесс. Хорошо выполненный BPM может сократить потери, сократить количество ошибок, сэкономить время и повысить качество услуг и продуктов.

Фронтальная система является веб-приложением, основанным на технологии клиент-сервер. Для разработки серверной части (backend-разработка) системы использовалась технология Spring Boot, которая является фреймворком, основанным на языке программирования Java. Для разработки клиентской части (frontend) использовалась технология Bootstrap Framework. Для хранения входных и выходных данных, использовалась СУБД MySQL.

# 1 Обзор и анализ веб-приложений

Логика веб-приложения распределена между сервером и клиентом. Клиент взаимодействует с сервером при помощи браузера. хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

Миллионы предприятий используют интернет, как экономически эффективный канал связи. Это позволяет им совершать безопасные, быстрые транзакции. Однако эффективное взаимодействие возможно только тогда, когда предприятие может собирать и хранить все необходимые данные и иметь средства для обработки этой информации и представления результатов пользователю.

Веб-приложения позволяют пользователям взаимодействовать с компанией, используя браузер, мобильные приложения и др. Кроме того, они позволяют сотрудникам создавать документы, обмениваться информацией, совместно работать над проектами и работать над общими документами независимо от местоположения или устройства.

Веб-приложения обычно пишутся на языках, поддерживаемых браузером, таких как JavaScript и HTML. Некоторые из приложений являются динамическими, требующими обработки на стороне сервера. Другие полностью статичны без необходимости обработки на сервере.

Веб-приложению требуется веб-сервер для управления запросами от клиента, сервер приложений для выполнения запрошенных задач и база данных для хранения информации.

## 1.1 Алгоритм работы веб-приложения

Пользователь отправляет запрос веб-серверу через браузер, либо через пользовательский интерфейс приложения.

Веб-сервер перенаправляет этот запрос на соответствующий сервер веб-приложений.

Сервер веб-приложений выполняет запрошенную задачу - такую как запрос базы данных или обработка данных - затем генерирует результаты запрошенных данных.

Сервер веб-приложений отправляет результаты на веб-сервер с запрошенной информацией или обработанными данными.

Веб-сервер отвечает клиенту запрошенной информацией, которая затем появляется в интерфейсе пользователя.

## 1.2 Виды веб-приложений

Корпоративный портал. Многофункциональный веб-сервис, позволяющий удобно и эффективно оптимизировать бизнес процессы. Выполняет следующие цели:

- улучшает качество работы с клиентами;
- повышает производительность труда сотрудников;
- укрепляет и улучшает взаимосвязь между отделами компании;
- повышает мобильность сотрудников;
- удаленная работа с документами.

CRM (Customer Relationship Management). Инструмент для автоматизации взаимодействия с покупателями. CRM записывает контактную информацию клиентов, такую как электронная почта, телефон, профиль в социальных сетях и т. д. Также может автоматически извлекать другую информацию, такую как последние новости о деятельности компании, и может хранить такие сведения, как личные предпочтения клиента при общении.

Система CRM систематизирует эту информацию, чтобы предоставить полную информацию о лицах и компаниях.

ERP (Enterprise resource planning). ERP означает планирование ресурсов предприятия. Он относится к набору программного обеспечения, которое организации используют для управления повседневной деятельностью, такой как учет, закупки, управление проектами, управление рисками и соблюдение нормативных требований, а также операции в цепочке поставок. Полный пакет ERP также включает управление производительностью предприятия, программное обеспечение, которое помогает планировать, составлять бюджет, прогнозировать и сообщать о финансовых результатах организации.

Системы электронной коммерции. Благодаря данной системе производители и поставщики услуг/товаров могут предлагать в сети свою продукцию покупателям, осуществлять прием и обработку заказов и т.д. Выполняет следующие функции:

- получение подробной информации о запросах каждого потребителя;
- стремительный вывод нового продукта на рынок;
- уменьшение затрат на совершение сделки;
- сокращение пути товара к потребителю.

## 1.3 Преимущества веб-приложений

Веб-приложения работают на нескольких платформах независимо от ОС или устройства, если браузер совместим.

Все пользователи имеют доступ к одной и той же версии.

Они не установлены на жестком диске, таким образом устраняя ограничения пространства.

Они снижают пиратство программного обеспечения в веб-приложениях на основе подписки (т.е. SaaS).

Они снижают затраты как для бизнеса, так и для конечного пользователя, так как бизнесу требуется меньше поддержки и обслуживания и снижаются требования к компьютеру конечного пользователя.

## 1.4 Архитектура клиент-сервер

В настоящее время компьютерные транзакции, в которых сервер выполняет запрос, отправленный клиентом, очень распространены, и модель клиент-сервер стала одной из главных идей сетевых вычислений. Клиент устанавливает соединение с сервером через локальную (LAN) или глобальную сеть (WAN). Как только сервер выполнил запрос клиента, соединение разрывается. Поскольку несколько клиентских программ совместно используют службы одной и той же серверной программы, специальный сервер, называемый daemon, может быть активирован только для ожидания клиентских запросов.

Важным преимуществом модели клиент-сервер является то, что ее централизованная архитектура помогает упростить защиту данных с помощью средств управления доступом. Кроме того, не имеет значения, построены ли клиенты и сервер в одной операционной системе, поскольку данные передаются по протоколам клиент-сервер, которые не зависят от платформы.

Недостатком модели клиент-сервер является то, что, если слишком много клиентов одновременно запрашивают данные с сервера, они могут быть перегружены. В дополнение к перегрузке сети, слишком много запросов может привести к отказу в обслуживании.

## 1.5 Архитектура MVC

В качестве архитектуры для разработки данного приложения используется архитектура MVC (Model-View-Controller, «Модель-Представление-Контроллер»). MVC разделяет данные приложения, пользовательский интерфейс и логику управления на три независимых компонента: модель, представление, контроллер.

Модель – предоставляет данные и представляет бизнес-логику. Объекты модели получают и хранят состояние модели базы данных, также реагирует на команды контроллера, изменяя своё состояние.

Представление - является пользовательским интерфейсом. Отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер - обрабатывает запросы пользователей. Обычно пользователь взаимодействует с интерфейсом, и, в свою очередь, происходит запрос URL. Запрос обрабатывается контроллером. Контроллер отображает данные модели и отображает view в качестве ответа.

MVC позволяет упростить большой код, путем разделения его на части. Написание длинного кода сбивает с толку и его трудно изменить без ошибок.

MVC не зависит от определенного языка программирования и не требует объектно-ориентированного метода или других примеров.

Разделение здесь не означает, что нужны три файла (или три папки которые содержат файлы) с названиями model, view и controller. На практике модель занимает основной объем приложения, и представлена в виде большого числа разнотипных классов - сущностей, сервисов, классов работы с БД, и для каждого вида классов делают отдельные папки.

MVC может применяться ко многим различным типам приложений, включая серверные веб-приложения и настольные (клиентские) приложения. Разница между ними заключается в том, что программа веб-приложения обрабатывается в ответ на запрос пользователя, отображает результат (как правило, веб-страницу) и завершает работу. Когда поступают другие запросы, выполняется новая независимая копия программы для обработки этого. В отличие от веб-приложений, персональных компьютеров, мобильных приложений (и приложений, использующих страницы браузера, созданные на JavaScript), они имеют продолжительный срок службы. Они обрабатывают различные запросы пользователя и обновляют информацию на экране без выхода.

Взаимодействие между компонентами MVC реализовано немного по-разному в серверных и десктопных приложениях, веб-приложение обрабатывает один запрос пользователя и завершается, а десктопное приложение обрабатывает много запросов без перезапуска.

Серверные приложения обычно используют «пассивную» модель, а десктопные приложения - «активную» модель. В отличие от активных моделей, пассивные модели позволяют подписываться и получать уведомления об изменениях. Серверные приложения в этом не нуждаются.

В схеме с активной моделью Вид подписывается на изменения в Модели. Затем, когда происходит событие (например, пользователь нажимает кнопку), вызывается контроллер. Он предоставляет команды изменения данных для модели. Модель сообщает своим подписчикам (в том числе Виду), что данные изменились, и Вид обновляет интерфейс программы.

Схемы с активными моделями рассчитаны на продолжительную работу. Напротив, схемы пассивной модели обычно используются в короткоживущих приложениях.

Серверные приложения используют пассивные модельные системы. Предположим, что пользователь заходит на страницу форума. В своем браузере он отправляет HTTP-запрос на поиск страницы, содержащей список сообщений. В это время контроллер запускается, анализирует запрос пользователя и запрашивает список сообщений у Model. Получив его, он вызывает View, отправляет список и отображает его в виде веб-страницы. Чтобы добавить сообщение, пользователь

заполняет форму, далее вызывается контроллер, отвечающий за обработку данных этой формы, который примет данные, и задействует Модель, чтобы проверить и вставить в базу данных новое сообщение, и затем вернет HTTP ответ с редиректом на страницу просмотра сообщений.

На рисунке 1 приведена схема, изображающая взаимодействие компонентов.

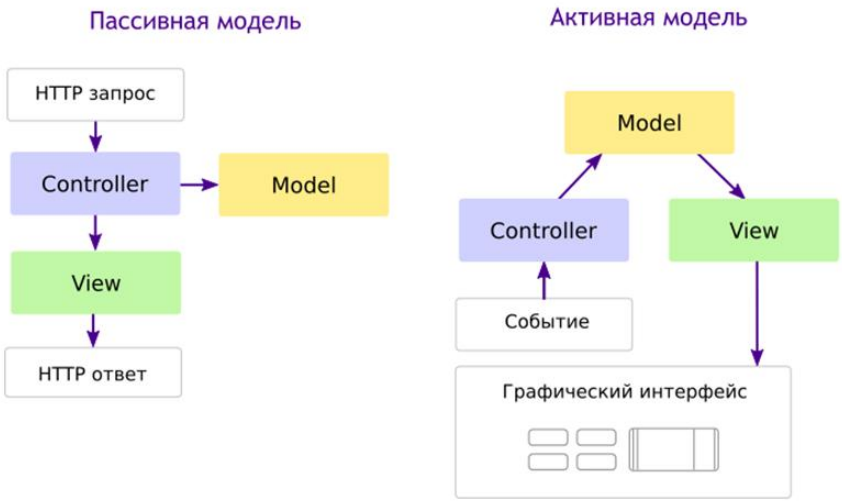


Рисунок 1- Взаимодействие компонентов MVC

1.6 Характеристика фронтальной системы

Фронтальная система относится к BPM-системам (business process management). Система управления бизнес-процессами (BPM) позволяет выполнять и обслуживать бизнес-операции и сотрудников, которые ими управляют, в разных отделах. Эти программные решения призваны помочь предприятиям оптимизировать свои повседневные бизнес-процессы(БП) для максимальной эффективности и продуктивности. хорошая система управления БП имеет инструменты и технологии для включения ориентированных на человека задач наряду с приложениями для машинной обработки, что позволяет компании или организации гибко управлять своей работой. Системы управления БП также включают возможности составления отчетов, позволяющие конечным пользователям и менеджерам понимать пропускную способность и время отклика бизнес-процессов.

Повседневная практика работы каждой организации, от продавца до генерального директора, требует управления процессами. Система управления бизнес-процессами обеспечивает:

- быстрый, гибкий процесс, в котором ИТ и бизнес работают вместе, используя инструменты, которые позволяют им находить решения;

- множество различных средств, с помощью которых бизнес-аналитик или менеджер может постоянно отслеживать и оценивать улучшения одного, нескольких бизнес-процессов;

- автоматизация рутинных транзакций и динамического взаимодействия людей;

- повышение операционной эффективности, сокращая время обслуживания;

- улучшение соответствия нормативным требованиям, полученное за счет улучшения контроля, регулирования и мониторинга процессов;

- повышение удержания клиентов, что достигается за счет лучшей, быстрой и глубокой интеграции систем управления клиентами.

## 1.7 Постановка задачи дипломной работы

Целью дипломной работы является разработка информационной системы для банковского обслуживания с использованием технологий обеспечения безопасности и разграничением прав доступа пользователей посредством определения ролей.

Актуальность данной дипломной работы связана со снижением затрат, сокращением времени обслуживания клиентов, быстрой и улучшенной взаимосвязью между отделами, повышением мобильности сотрудников, масштабируемостью и гибкостью архитектуры информационных технологий.

Требования к разрабатываемой информационной системе:

- программа должна обладать удобным, интуитивно понятным интерфейсом;
- программа должна предусматривать контроль вводимой информации;
- программа должна осуществлять блокировку, при некорректных действиях пользователя, во время работы с приложением, с выдачей сообщений об ошибках;
- программа должна включать справочный материал о работе и подсказки пользователю в ходе его работы;
- программа должна включать хранение и обработку информации о сотрудниках, клиентах, кредитах;
- возможность создания и редактирования информации о сотрудниках, разграничении их прав доступа, клиентах банка;
- возможность создания и редактирования информации о клиентах банка;
- взаимосвязь между отделами для проверки информации;
- создание и редактирование текстовых оповещений;

## 2 Модели и службы функционирования системы

### 2.1 Функциональная модель системы

Функциональная модель системы представляет собой взаимодействие клиентского интерфейса и серверной части приложения, отвечающего за обработку получаемой информации и отображение результата.

В качестве архитектуры для разработки приложения использована архитектура MVC (Model-View-Controller). Клиентский интерфейс представлен компонентом View(Представление), который отвечает за показ данных модели пользователю, реагируя на изменения модели. Модели представлены классами Java. Каждая модель имеет свои индивидуальные параметры и определенный функционал. Реализованы пять моделей, необходимых для работы приложения: модель клиента, модель сотрудника, модель роли сотрудника, модель кредита, модель должности сотрудника.

Распределение ролей осуществляется при создании нового объекта модели сотрудника, т.е. при добавлении нового сотрудника в базу данных, ему присваивается определенная роль, связанная с должностью сотрудника.

Созданы следующие роли: кредитор, менеджер, администратор, бухгалтер, СБ (служба безопасности), директор.

Кредитор – отвечает за добавление, редактирование информации о клиенте(заемщике), определении вида кредита и отправке ее для последующей проверки. Добавление информации о клиенте происходит посредством заполнения полей модели клиента и сохранением ее в базу данных.

Менеджер – отвечает за проверку информации о клиенте, полученной от кредитора, и отправку ее на следующий шаг проверки клиента. Также он имеет возможность отправки информации на доработку кредитору, добавления новых видов кредита и отправку запроса на добавление нового сотрудника администратору.

Администратор – отвечает за добавление нового сотрудника в базу данных, определение его должности и роли, создание и редактирование учетных записей сотрудников с последующей аутентификацией сотрудника, для входа в систему, добавление новых видов должностей.

Бухгалтер – отвечает за проверку информации о клиенте и отправку ее на следующий шаг проверки. Имеется возможность отправить информацию о клиенте на доработку, или отказать в предоставлении кредита, в случае необходимости.

Служба безопасности – отвечает за проверку и подтверждение информации о клиенте и последующую отправку ее кредитору, для добавления клиента в базу данных.

Взаимодействие и передачу информации между сотрудниками осуществляют контроллеры.

## 2.2 Обобщенный алгоритм функционирования системы

Информационная система состоит из двух составляющих:

- клиентская часть;
- серверная часть.

Клиентская часть реализует пользовательский интерфейс, создает запросы к серверу и обрабатывает ответы от него.

Серверная часть принимает запрос от клиента, производит обработку, после, формирует и отправляет страницу клиенту с использованием протокола HTTP.

Приложение включает в себя следующие подсистемы:

- аутентификация сотрудника – проверка подлинности сотрудника по авторизованной учетной записи, не позволяющая осуществить несанкционированный доступ;
- новый сотрудник – процедура создания новых сотрудников системы, с распределением ролей;
- новый клиент – добавление нового клиента, для оформления кредита;
- оформление кредита – присвоение кредита пользователю;
- второй шаг – проверка информации о клиенте менеджером;
- третий шаг - проверка информации о клиенте бухгалтерией;
- четвертый шаг– проверка информации о клиенте службой безопасности;
- виды кредита – добавление новых видов кредита;
- виды должностей – добавление новых должностей сотрудников;
- поиск сотрудника – поиск необходимого сотрудника по введенным критериям, как фамилия, имя, ИИН, должность;
- поиск клиента - поиск необходимого клиента по введенным критериям, как фамилия, имя, ИИН, должность;
- редактирование сотрудника – изменение данных о сотруднике или удаление его учетной записи из базы данных;
- редактирование клиента - изменение данных о клиенте или удаление его из базы данных;
- доработка – отправка информации о клиенте на доработку в ходе проверки;
- отказ – отказ в одобрении кредита клиенту, в случае неудовлетворения требованиям.
- выход из системы – осуществляется выход сотрудника из системы, после чего необходима повторная аутентификация для продолжения работы.

В работе имеются алгоритмы, реализующие функционирование системы, учитывая степень развития современных информационных технологий, были разработаны и предложены новые алгоритмы, которые представлены ниже.

Рассмотрим работу алгоритмов приложения более подробно.

При аутентификации сотрудник, посредством ввода необходимых данных (логин, пароль) формирует и отправляет запрос на сервер. Серверная часть

обрабатывает запрос, и с помощью специальных методов производит сверку введенных логина и пароля с данными, хранящимися в базе. Они должны совпадать. В случае успешной проверки, формируется токен пользователя. Токен (JWT – JSON Web Token) – это набор символов, полученный шифрованием некой информации о пользователе, прошедшем аутентификацию в приложении. Любой, кто прошел аутентификацию получает токен и этот же токен запоминается еще и системой выдавшей его, чтобы при последующих запросах от этого пользователя сверять переданный им токен с тем, что система запомнила в момент успешной аутентификации. Если токен не совпадает, либо истек срок его годности (срок годности выбирается разработчиком на свое усмотрение), то запрос считается не прошедшим авторизацию и отклоняется системой. При успешной аутентификации сотрудник переходит на главную страницу системы, содержащую в себе главное меню функционала системы, для дальнейшей работы.

При оформлении кредита осуществляется переход на необходимую веб-страницу. При переходе производится проверка роли сотрудника, на наличие прав доступа. Если роль сотрудника не соответствует, то запрос отклоняется системой с кодом http 403(отсутствие прав доступа). Если роль соответствует, то сотруднику необходимо заполнить поля, следующими данными клиента: ФИО, дата рождения, ИИН, сумма кредита, вид кредита, место работы, должность, заработная плата, место прописки. Далее эти данные отправляются второй шаг проверки.

Доступ на второй шаг имеют только сотрудники с ролью Manager. При переходе на второй шаг производится проверка роли сотрудника, на наличие прав доступа. На втором шаге имеется возможность отказать или отправить на доработку, в случае необходимости. Далее данные отправляются на третий шаг проверки.

Доступ на третий шаг имеют только сотрудники с ролью Accountant. При переходе на третий шаг производится проверка роли сотрудника, на наличие прав доступа. На третем шаге имеется возможность отказать или отправить на доработку, в случае необходимости. Далее данные отправляются на четвертый шаг проверки.

Доступ на четвертый шаг имеют только сотрудники с ролью Security. При переходе на четвертый шаг производится проверка роли сотрудника, на наличие прав доступа. На четвертем шаге имеется возможность отказать или отправить на доработку, в случае необходимости. Далее данные утверждаются и отправляются кредитору. После эти данные добавляются в базу.

При добавлении нового сотрудника необходимо перейти на страницу добавления сотрудника. Для этого нужно обладать соответствующими привилегиями, чтобы система разрешила доступ к добавлению. Если роль сотрудника не соответствует, то запрос отклоняется системой. Далее, необходимо заполнить поля, следующими данными сотрудника: ФИО, дата рождения, ИИН, номер удостоверения личности, должность, роль.

При редактировании информации о сотруднике или клиенте осуществляется переход на необходимую веб-страницу, где также производится проверка на наличие прав доступа. После успешной проверки откроется веб-страница, где находится форма с изменяемыми полями, в которых прописаны данные клиента или сотрудника, которые необходимо изменить или удалить.

## 2.3 Информационное обеспечение системы

### 2.3.1 СУБД MySQL

СУБД позволяет пользователям создавать, считывать, обновлять и удалять данные из баз данных. СУБД выступает в качестве интерфейса между базой данных и конечным пользователем или приложением, обеспечивая согласованную организацию данных и легкий доступ.

SQL (язык структурированных запросов) - это стандартизированный язык программирования для управления реляционными базами данных и выполнения различных задач обработки данных. Использование SQL включает изменение таблицы базы данных и структуры индекса; добавление, обновление и удаление строк данных; и извлечение подмножеств информации из базы данных для приложений обработки транзакций и аналитики. Запросы и другие операции SQL принимают форму команд, написанных как операторы. Обычно используемые операторы SQL включают выбор, добавление, вставку, обновление, удаление, создание, изменение и усечение. в SQL существует специальный набор общих команд — таких, как SELECT, INSERT, UPDATE и DELETE.

MySQL - СУБД с открытым исходным кодом. MySQL работает практически на всех платформах, включая Linux, UNIX и Windows. Его можно использовать со многими другими приложениями.

MySQL является важным элементом корпоративного стека с открытым исходным кодом, который называется LAMP. LAMP - это платформа веб-разработки, которая использует операционную систему Linux и использует Apache в качестве веб-сервера, MySQL в качестве СУБД и в качестве объектно-ориентированного языка программирования, например PHP. (В некоторых случаях используются Perl, Python, PHP.)

Основан на модели клиент-сервер. Ядром MySQL является сервер MySQL, который управляет всеми командами базы данных. Сервер спроектирован как каталог, который может быть собран из другой программы в отдельное приложение (клиент-сервер для использования в сетевой среде).

MySQL работает с несколькими утилитами, которые поддерживают управление базами данных MySQL. Команды устанавливаются на компьютер и отправляются в MySQL Server через клиента MySQL.

MySQL предназначен для быстрой обработки исходной большой базы данных. Обычно устанавливается только на одном компьютере, но можно обращаться к базе данных через различные клиентские интерфейсы MySQL, поэтому можно расположить базу данных в нескольких местах.

Основные особенности MySQL.

MySQL позволяет хранить и получать доступ к данным через несколько механизмов хранения, включая InnoDB, CSV и NDB. MySQL также может реплицировать данные и таблицы разделов для повышения производительности и долговечности. Пользователям MySQL не нужно изучать новые команды. Доступ к данным можно получить с помощью стандартных команд SQL.

MySQL имеет формат C и C ++ и может использоваться на более чем 20 платформах, включая Mac, Windows, Linux и Unix. СУБД поддерживает большое количество баз данных с миллионами записей и поддерживает множество типов данных, включая целые числа со знаком, без знака, 1, 2, 3, 4 и 8 байтов. FLOAT; Double; CHAR; VARCHAR; VARBINARY; STRING; BLOB; DATE. Подсчет и OpenGIS пространственных типов. Он также поддерживает строковые типы фиксированной и переменной длины.

Из соображений безопасности MySQL использует зашифрованный пароль и систему доступа, которая обеспечивает контроль на основе хоста;

Клиент MySQL может использовать несколько протоколов для сервера MySQL, включая соединители TCP / IP для всех платформ. MySQL также поддерживает различные клиентские и служебные программы, программы командной строки и инструменты администрирования, такие как MySQL Workbench.

Кроме того, MySQL позволяет программам использовать несколько механизмов хранения для каждой таблицы, позволяя им выбирать наиболее подходящий механизм хранения для указанных таблиц. Одним из двигателей MySQL является InnoDB. InnoDB разработан для высокой доступности. Из-за этого он не так быстр, как другие двигатели. SQL использует свое собственное пространство хранения, но поддерживает ряд мер для предотвращения потери данных. Он может работать в кластере для обеспечения высокой доступности обеих систем.

### 2.3.2 Язык программирования Java

Java является параллельным, основанным на классах, объектно-ориентированным языком программирования общего назначения, специально разработанным для максимально возможного уменьшения зависимостей реализации. Данная функция необходима для того, чтобы дать возможность разработчикам приложений писать, компилировать код Java и запускать его на всех

платформах, которые поддерживают Java, без перекомпиляции (WORA- Write once, run anywhere).

Например, можно создать Java-программу в UNIX, скомпилировать ее и запустить на компьютере под управлением Microsoft Windows, Macintosh или UNIX без изменения исходного кода. WORA достигается путем компиляции Java-программы на промежуточный язык, называемый байт-кодом. Формат байт-кода зависит от платформы. Виртуальные машины, называемые Java Virtual Machines (JVM), используются для выполнения байт-кодов на каждой платформе.

Реализация Oracle упакована в два разных дистрибутива.

Java Runtime Environment (JRE): предназначается для конечных пользователей, включает в себя платформы Java SE, которые необходимы для запуска программ Java.

Для разработчиков ПО, Разработан Java Development Kit (JDK), дистрибутив, в который входят такие инструменты для разработки, как компилятор Java, Javadoc, Jar и отладчик.

Java использует автоматическую сборку мусора для управления памятью жизненного цикла объекта. Программист решает, когда создавать объект, а среда выполнения Java восстанавливает память, когда объект не используется. Если не осталось ссылок на объекты, вы сможете использовать память, которая не может достичь автоматического сброса сборщиком мусора.

Объекты, которые не нужны для кода программиста, хранятся в контейнере, который все еще используется, и могут возникнуть проблемы, такие как утечки памяти, если код программиста содержит ссылки на объекты, которые больше не нужны. Когда метод вызывается для объекта, который не существует, возникает «NullPointerException».

Сбор мусора может произойти в любое время. В идеале это происходит, когда программа простаивает. Он гарантированно работает, если в куче недостаточно памяти для размещения новых объектов. Это может немедленно прервать программу. Java не позволяет явно управлять памятью.

Параллельная обработка - это функция, которая может выполнять несколько программ и несколько программных частей параллельно. Параллелизм позволяет программам достигать высокой производительности и пропускной способности, используя неиспользуемые функции операционной системы по умолчанию и аппаратного обеспечения машины.

Объектно-ориентированное программирование - это вид программирования, основанный на представлении программы с набором объектов, причем каждый объект является экземпляром определенного класса, а классы образуют иерархию наследования. Существуют четыре основных принципа ООП.

Абстракция. Цель абстракции - скрыть сложность от пользователей и показать им только соответствующую информацию. Обладает следующими свойствами:

- скрывает сложность данных;
- помогает избежать повторяющегося кода;
- представлена только подпись внутренней функциональности;
- предоставляет программистам гибкость для изменения реализации; абстрактного поведения.

Инкапсуляция. Инкапсуляция позволяет защитить данные, хранящиеся в классе, от общесистемного доступа. Обладает следующими свойствами:
- ограничивает прямой доступ к данным-членам (полям) класса;
- поля настроены как приватные;
- каждое поле имеет метод получения и установки;
- методы геттера возвращают поле;
- методы установки позволяют нам изменять значение поля.

Полиморфизм. Полиморфизм - способность к изменению или расширению унаследованного функционала и параметров. В Java полиморфизм может принимать две формы: перегрузка метода и переопределение метода. Перегрузка методов происходит, когда в классе присутствуют различные методы с одинаковыми именами. Когда они вызываются, они различаются по количеству, порядку и типам своих параметров.

Наследование. Наследование позволяет создать дочерний класс, который наследует поля и методы родительского класса. Дочерний класс может переопределять значения и методы родительского класса, однако это необязательно. Он также может добавлять новые данные и функциональность к своему родителю. Родительские классы также называются суперклассами или базовыми классами, в то время как дочерние классы также называются подклассами или производными классами.

### 2.3.3 Технология Spring Boot

Spring Boot - это среда на основе Java с открытым исходным кодом, используемая для создания микро-сервисов. С помощью Spring Boot легко создавать автономные и готовые к использованию приложения. Spring Boot содержит всестороннюю инфраструктурную поддержку для разработки микро-сервиса и позволяет разрабатывать готовые к работе приложения

Микро-сервис - это структура, которая позволяет разработчикам самостоятельно разрабатывать и развертывать сервисы. Каждая служба имеет свой собственный процесс, который обеспечивает облегченную модель для поддержки бизнес-приложений.

Микро-сервис предлагает разработчикам следующие преимущества:
- простое открытие;
- простая масштабируемость;
- совместим с контейнерами;

- минимальная конфигурация;
- меньше времени для производства.

Spring Boot предоставляет отличную платформу для разработчиков Java для разработки автономных приложений Spring. Можно начать минимальную настройку без необходимости полностью конфигурировать Spring.

Spring Boot дает разработчикам следующие преимущества:

- легко понять и разработать Spring приложения;
- повышенная производительность;
- сокращенное время разработки;

Spring Boot предназначен для следующих целей:

- избежать сложной конфигурации XML в Spring;
- сократить время разработки и запускать приложения напрямую.

Java Beans предоставляет гибкий способ определения конфигурации XML и транзакций базы данных. Обеспечивает мощную пакетную обработку и управляет конечными точками REST. В Spring Boot все настраивается автоматически. Облегчено управление зависимостями. Включен встроенный контейнер сервлетов.

Spring Boot автоматически настраивает приложение на основе зависимостей, используя аннотацию @EnableAutoConfiguration . Начало работы приложения Spring - класс, содержащий аннотацию @SpringBootApplication и метод main. Spring Boot автоматически сканирует все компоненты, включенные в проект, используя аннотацию @ComponentScan .

Spring Boot Starters. Управление зависимостями - сложная задача для больших проектов. Spring Boot решает эту проблему, предоставляя набор зависимостей для удобства разработчиков.

Например, для использования Spring и JPA для доступа к базе данных, достаточно включить в проект зависимость spring-boot-starter-data-jpa. Все стартеры Spring Boot следуют одному и тому же шаблону именования spring-boot-starter- *, где * указывает, что это тип приложения.

Зависимость Spring Boot Starter Actuator используется для мониторинга и управления приложением. Его код показан на рисунке 2.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Рисунок 2 - Зависимость Spring Boot Starter Actuator

Зависимость Spring Boot Starter Security используется для Spring Security. Его код показан рисунке 3.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Рисунок 3 - Зависимость Spring Boot Starter Security

Веб-зависимость Spring Boot Starter-Web используется для записи конечных точек. Его код показан рисунке 4.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Рисунок 4 - Зависимость Spring Boot Starter Web

Сначала нужно настроить Java-проект для сборки Maven. Проекты Maven определяются с помощью XML-файла с именем pom.xml . Среди прочего, этот файл содержит имя проекта, версию и зависимости, которые он имеет от внешних библиотек.

За исключением необязательного <packaging> элемента. Он включает в себя следующие детали конфигурации проекта:

- <modelVersion> - версия модели POM (всегда 4.0.0);
- <groupId> - группа или организация, к которой принадлежит проект. Часто выражается как доменное имя;
- <artifactId> - имя, которое будет присвоено артефакту библиотеки проекта (например, имя его файла JAR или WAR);
- <version> - версия проекта, который строится;
- <packaging> - как проект должен быть упакован. По умолчанию используется "jar" для упаковки файлов JAR;
- <dependencies> - набор необходимых библиотек для работы проекта;
- <plugin> - программный модуль, подключаемый к основной программе и предназначенный для расширения или использования её возможностей.
- <build> - сборка.

Ниже показан код файла pom.xml с необходимыми зависимостями для проекта.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.4.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.myspring</groupId>
<artifactId>bpm-system</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>bpm-system</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <!-- Use MySQL Connector-J -->

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
    <version>5.1.47</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mustache</artifactId>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

Spring Boot Maven plugin предоставляет множество удобных функций.

Он собирает все файлы jar на пути к классам и создает единый работающий «uber-jar», который делает его более удобным для выполнения и транспортировки сервиса.

Он ищет public static void main() метод для отметки как исполняемый класс.

Он предоставляет встроенный распознаватель зависимостей, который устанавливает номер версии в соответствии с зависимостями Spring Boot. Можно переопределить любую версию, но по умолчанию будет выбран выбранный Boot набор версий.

Автоматическая настройка Spring Boot настраивает приложение Spring на основе зависимостей JAR, добавленных в проект.

JAR-файл — это Java-архив. Представляет собой ZIP-архив, в котором содержится часть программы на языке Java.

MySQL Connector Java - является драйвером JDBC.

Spring boot starter mustache - зависимость для создания веб-приложений с использованием технологии Mustache.

BpmSystemApplication.class – является основным классом, который отвечает за запуск проекта. Его код показан ниже.

```
@SpringBootApplication
public class BpmSystemApplication {

  public static void main(String[] args) {
    SpringApplication.run(BpmSystemApplication.class, args);
  }
}
```

@SpringBootApplication это удобная аннотация, которая добавляет все необходимые конфигурации перечисленные ниже.

@Configuration помечает класс как источник определений компонента для контекста приложения.

@EnableAutoConfiguration сообщает Spring Boot, что нужно начинать добавлять bean-компоненты на основе настроек пути к классам, других bean-компонентов и различных настроек свойств.

Обычно вы добавляете @EnableWebMvcприложение Spring MVC, но Spring Boot добавляет его автоматически, когда видит spring-webmvc в пути к классам. Это помечает приложение как веб-приложение и активирует ключевые поведения, такие как настройка a DispatcherServlet.

@ComponentScan говорит Spring искать другие компоненты, конфигурации и службы в пакете, позволяя ему найти контроллеры.

Public static void main() метод использует SpringApplication.run(), метод для запуска приложения.

Существует также CommandLineRunner метод, помеченный как, @Bean и он запускается при запуске. Он извлекает все компоненты, которые были созданы приложением или были автоматически добавлены благодаря Spring Boot.

### 2.3.4 Java Persistence API

Стандарт Java ORM для хранения, доступа и управления объектами Java в реляционных базах данных. Идея состоит в том, чтобы сопоставить Java-классы с реляционными таблицами, а свойства этих классов - со строками в таблице. Это меняет значение общего опыта Java-кодирования благодаря плавному взаимодействию двух технологий в одной и той же парадигме программирования.

Для работы с реляционными базами данных ANSI стандартизировал язык (язык структурированных запросов SQL). Данные оператора, написанные на этом языке, могут быть определены и обработаны. Тем не менее, проблема SQL в работе с Java заключается в том, что они имеют несоответствующую синтаксическую структуру и сильно различаются по своей сути. То есть SQL является процедурным, Java - объектно-ориентированным. Решение заключается в том, чтобы реляционные базы данных понимали язык Java. JPA отвечает на эти вызовы и предоставляет механизм, который может решить эти проблемы.

Java-программы используют API Java Database Connectivity (JDBC) для работы с реляционными базами данных. Драйвер JDBC является ключом соединения, позволяя программам Java манипулировать базой данных с помощью API JDBC. Как только соединение установлено, Java-программа выполняет SQL-запрос в формате String для отправки операций создания, вставки, обновления и удаления. Структура реляционной таблицы представляет собой табличное

логическое представление данных. Таблица состоит из столбцов, которые описывают характеристики объекта и набора объектов.

JPA описывает управление реляционными данными в Java-приложении. Это спецификация, и существует ряд ее реализаций. Некоторые популярные реализации - это Hibernate, EclipseLink и Apache OpenJPA.

Использование JPA в проекте показано на примере кода файла EmployeeRepository.java.

```
@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Integer> {
   Employee findByUsername(String username);
}
```

@Repository - это аннотация Spring, которая указывает, что данный класс является хранилищем. Репозиторий - это механизм для инкапсуляции хранения, извлечения и поиска, который эмулирует коллекцию объектов.

CrudRepository <T, ID> - интерфейс для общих операций CRUD(create, read, update, delete) в хранилище для определенного типа.

Метод findByUsername - находит объект по его имени учетной записи из базы данных.

Также для реализации методов JPA создан класс-сервис EmployeeService.java.

```
@Service
public class EmployeeService implements UserDetailsService {

   @Autowired
   private EmployeeRepository employeeRepository;

   @Override
   public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      return employeeRepository.findByUsername(username);
   }
   public List<Employee > read(){
      return (List<Employee>) employeeRepository.findAll();
   }
   public Employee create(Employee employee){
      return employeeRepository.save(employee);
   }
   public Employee update(Employee employee){
      return employeeRepository.save(employee);
   }
   public void delete(Integer id){
      employeeRepository.deleteById(id);
   }
}
```

Аннотация @Service может применяться только к классам. Она используется для обозначения класса в качестве сервиса, исполняющего определенную бизнес-логику.

Метод findAll() - возвращает все экземпляры из базы данных.

Save(S entity) - сохраняет данную сущность в базу данных.

DeleteById(ID id) - удаляет объект с указанным идентификатором из базы данных.

### 2.3.5 Hibernate

Hibernate - это структура на основе Java, которая упрощает разработку приложений Java, взаимодействующих с базами данных. Это простой инструмент реляционного отображения объектов с открытым исходным кодом (ORM). Hibernation выполняет спецификацию Java Persistence API (JPA) для хранения данных. Hibernate предоставляет инструменты для запроса и извлечения данных, а также для отображения классов Java из таблиц базы данных (типы данных Java для типов данных SQL).

ORM - это программа, которая преобразует данные между реляционной базой данных и объектно-ориентированным языком программирования.

Реализация Hibernate показана на примере кода класса Employee.java, которая является сущностью сотрудника.

```
@Entity
@Table(name = "employees")
public class Employee implements UserDetails {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String username;
   private String password;
   private String employee_name;
   private String surname;
   private String patronymic;
   private String birthDate;

   @ManyToOne(fetch = FetchType.EAGER)
   @JoinColumn(name = "position_id")
   private Position position;
   private String iin;
   private boolean active;

   @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
   @CollectionTable(name = "employee_role", joinColumns = @JoinColumn(name =
"employee_id"))
```

```
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;
    public Employee() {
    }
```

@Entity - аннотация, регистрирующая класс как сущность БД.

@Table (name = "employees") – связываем сущность с конкретной таблицей по названию.

@Column (name = "username") – название поля таблицы в БД.

@Id - указывает на то, что следующее поле является идентификатором (id), также первичным ключом данного объекта.

Класс с аннотацией @Entity, указывает , что он является объектом JPA. аннотация @Table предполагает, что этот объект будет отображен в таблицу с именем customers.

@GeneratedValue(strategy = GenerationType.AUTO) – настройка авто инкремента, автоматического увеличения значения идентификатора при последующем добавлении нового объекта в таблицу.

@OneToMany и @ManyToOne определяет отношения «один ко многим» и «многие к одному» между двумя объектами. @JoinColumn указывает на то, что сущность является владельцем отношения, а соответствующая таблица имеет столбец с внешним ключом к указанной таблице.

### 2.3.6 Spring Security

Spring Security – это инфраструктура, которая фокусируется на аутентификации и авторизации приложений Java.

Реализация Spring Security показана на примере кода класса WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class   WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Autowired
   private EmployeeService employeeService;

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
            .authorizeRequests()
            .antMatchers("/", "/registration").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
```

```
            .loginPage("/login")
            .permitAll()
            .and()
            .logout()
            .permitAll();
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(employeeService)
            .passwordEncoder(NoOpPasswordEncoder.getInstance());
    }
}
```

Класс помечается @EnableWebSecurity, чтобы включить поддержку веб - безопасности Spring Security. Он также расширяет WebSecurityConfigurerAdapter и переопределяет несколько своих методов, чтобы установить некоторые особенности конфигурации веб-безопасности.

Configure(HttpSecurity) метод определяет, какие URL пути должны быть защищены и которые не должны. В частности, пути "/" и "/ registration" настроены так, что не требуют аутентификации. Все остальные пути должны быть аутентифицированы.

Класс AuthenticationManagerBuilder, в методе configure(AuthenticationManagerBuilder auth), позволяет легко встроить аутентификацию в памяти, аутентификацию на основе JDBC.

UserDetailsService - Основной интерфейс, который загружает пользовательские данные.

loadUserByUsername (String username) - Находит пользователя на основе имени пользователя.

UserDetails - Предоставляет основную информацию о пользователе.

Реализации не используются непосредственно Spring Security в целях безопасности. Они просто хранят пользовательскую информацию, которая позже инкапсулируется в аутентифицированные объекты. Это позволяет хранить пользовательскую информацию, не связанную с безопасностью (например, адреса электронной почты, номера телефонов и т. Д.), в удобном месте. Реализация интерфейса UserDetails показана ниже.

```
@Override
public boolean isAccountNonExpired() {
    return true;
}
@Override
public boolean isAccountNonLocked() {
    return true;
}
@Override
public boolean isCredentialsNonExpired() {
```

```
    return true;
}
@Override
public boolean isEnabled() {
    return isActive();
}
public void setUsername(String username) {
    this.username = username;
}
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return (Collection<? extends GrantedAuthority>) getRoles();
}
```

## 2.4 Разработка структуры базы данных

База данных разработана в СУБД MySQL и имеет название bpm, в ней имеются 5 таблиц.

Ниже представлены разработанные структуры таблиц базы данных.

- в таблице 1 – данные о сотрудниках;
- в таблице 2 – данные о клиентах;
- в таблице 3 – роли учетных записей сотрудников;
- в таблице 4 – виды кредитов;
- в таблице 5 – должности сотрудников;
- в таблице 6 – кредиты клиентов.

Таблица 1 – Employees

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(11) | Идентификатор сотрудника |
| User_name | Varchar(200) | Логин |
| password | Varchar(200) | Пароль |
| employee_name | Varchar(200) | Имя |
| surname | Varchar(200) | Фамилия |
| patronymic | Varchar(200) | Отчество |
| birth_date | Date | Дата рождения |
| position_id | int(3) | Должность |
| iin | Varchar(200) | ИИН |
| active | byte | Активность |
| roles | int(11) | Роль |

Таблица 2 – Customers

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(11) | Идентификатор клиента |
| name | Varchar(200) | Имя |
| surname | Varchar(200) | Фамилия |

| Имя поля | Имя поля | Описание |
|---|---|---|
| patronymic | Varchar(200) | Отчество |
| birth_date | Date | Дата рождения |
| iin | Varchar(200) | ИИН |
| job | Varchar(200) | Место работы |
| position | Varchar(200) | Должность |
| salary | Varchar(200) | Заработная плата |
| sum | Varchar(200) | Сумма кредита |
| phone | Varchar(20) | Номер телефона |
| credit_id | Int(3) | Идентификатор кредита |
| employee_id | Int(11) | Идентификатор сотрудника |
| | | |

Таблица 3 – Roles

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(11) | Идентификатор роли |
| name | Varchar(200) | Название роли |

Таблица 4 – Credits

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(3) | Идентификатор кредита |
| name | Varchar(200) | Название кредита |

Таблица 5 – Positions

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(3) | Идентификатор должности |
| name | Varchar(200) | Название должности |

Таблица 6 – Customer_credits

| Имя поля | Тип поля | Описание |
|---|---|---|
| id | Int(3) | Идентификатор должности |
| name | Varchar(200) | Название должности |
| sum | Varchar(200) | Сумма кредита |
| customer_id | Int(3) | Идентификатор клиента |

На рисунке 5 представлена схема разработанной базы данных

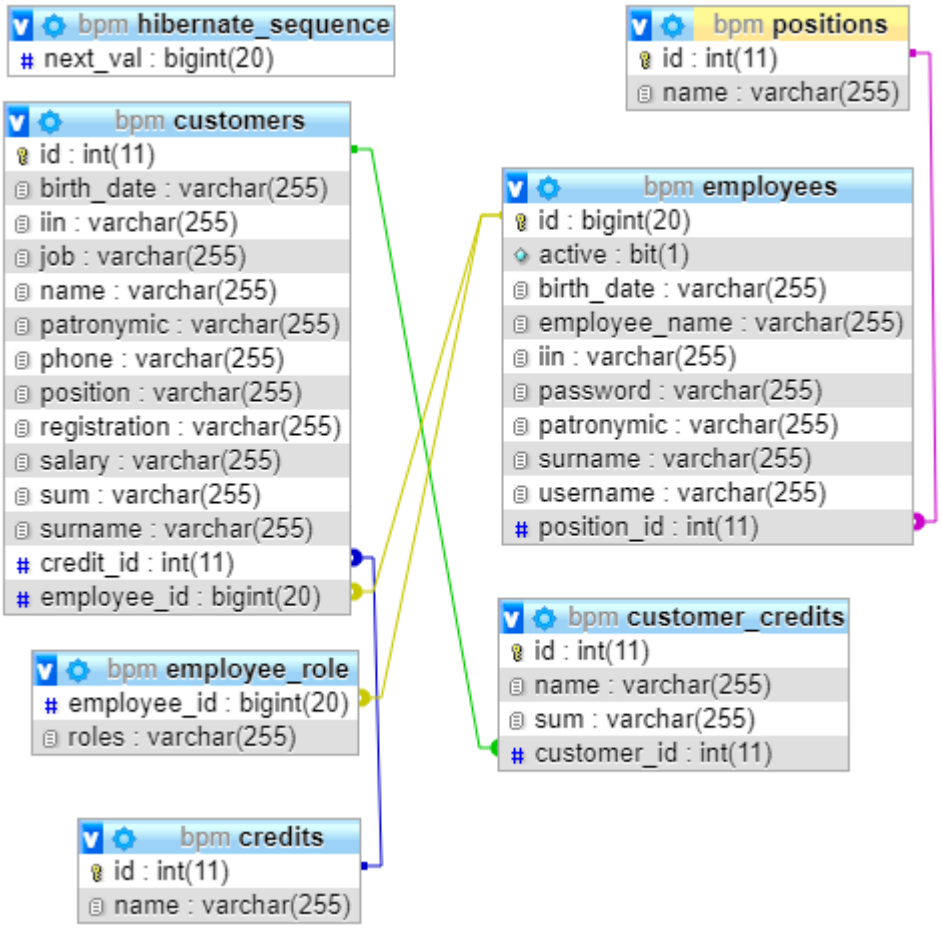


Рисунок 5 – Схема разработанной базы данных

# 3 Программное обеспечение

## 3.1 Системное программное обеспечение

### 3.1.1 Операционная система Microsoft Windows 7

Windows 7 является операционной системой для продуктов Microsoft. Существует несколько версий Windows. Windows 7 является самой популярной операционной системой в мире с 2017 года.

Эта операционная система совместима с Unicode 5.1. Теги теперь знают больше языков.

Операционная система поддерживает несколько сенсорных операций.

Все версии операционной системы содержат 50 новых источников. Доступные шрифты оптимизированы для правильного отображения всех символов. Windows 7 - это первая версия Windows с добавленными латинскими и нелатинскими символами. Панель управления источником была улучшена по умолчанию. В системе установлен только источник, показанный на рисунке.

Еще одна особенность Windows 7 - интеграция с производителями панелей. Большинство устройств регистрируются автоматически, но 90% совместимы с драйверами Windows Vista.

Windows 7 может улучшить совместимость со старыми программами, но может быть недоступна в Windows Vista. Windows XP В режиме Windows XP вы можете использовать Windows XP для запуска старых программ, которые поддерживают практически любое существующее приложение на виртуальной машине.

Во-первых, новая версия Embedded DirectX версии 11 добавляет следующие улучшения: Новая подсветка компьютера поддерживает многопотоковый рендеринг мозаики, новый алгоритм сжатия текстур.

Сетевые технологии могут использовать выбор файлов путем временного хранения кэшированного трафика. Если пользователю локальной сети необходимо передавать файлы по сети, он не может загружать каналы из локального кэша и использовать каналы с ограниченной пропускной способностью. Эта технология предназначена для использования в больших сетях и больших операционных системах.

Вы можете создать все конфигурации AIK и образы конфигурации Windows. Поэтому в Windows 7 используются спецификации USB 3.0, Bluetooth 4.0 и DirectX 11.1. Поддерживает .NET Framework 4.5.

Windows 7 также получила поддержку UEFI OEM и образов продуктов UEFI для Windows 7 SP1. При необходимости установщик UEFI для Windows 7 может выполнить запись на USB-устройство. Вы можете добавить полную поддержку UEFI, используя Windows AIK.

В отличие от Windows Vista, Windows 7 предлагает более гибкую конфигурацию управления пользователями (UAC) с двумя аргументами. Если вы попытаетесь сменить компьютер, отобразится только местоположение по умолчанию. Уведомление о смене компьютера (рабочий стол выглядит темным)

BitLocker был добавлен для шифрования съемных носителей с использованием технологии шифрования BitLocker, но TPM не поддерживается, поскольку съемный носитель зашифрован.

Эти события также влияют на брандмауэр Windows. Пользователь вернул предупреждение, чтобы разрешить доступ к сети.

Групповая политика и блокировка приложений могут блокировать выполнение определенных программ.

Лицензионная версия может блокировать удаленные подключения к компьютерам, которые требуют аутентификации пользователя. Кроме того, адрес подключенного хоста заблокирован для связи и управления авторизацией между консолью и компьютером.

С DirectAccess вы можете установить безопасное соединение с сервером в фоновом режиме, а не с VPN, если вам нужно взаимодействовать с пользователем. DirectAccess может применять групповую политику, прежде чем пользователи войдут в систему.

## 3.2 Прикладное программное обеспечение

### 3.2.1 Браузер Google Chrome

Google Chrome - это браузер, разработанный Google на основе бесплатного браузера и движка Blink. Первая версия была выпущена 11 декабря 2008 года. По данным StatCounter, Chrome - самый популярный браузер в мире, которым пользуются около 300 миллионов пользователей Интернета.

Secure Chrome регулярно получает обновления (сайты, содержащие вредоносный код) и предупреждает, когда пользователь пытается посетить вредоносную страницу.

Плагины (наиболее часто используемые Adobe Flash Player) не являются частью браузера и не могут быть включены в песочницу. Таким образом, безопасное выполнение кода страницы отключается плагинами, операционная система которых не имеет дополнительной защиты.

30 марта 2010 года Adobe Flash включается в браузер и не требует от пользователей его загрузки и установки. Flash Player обновляется автоматически.

Adobe Flash Player интегрирован с версии 5.0.

Начиная с версии 4.2, Chrome прекратил обслуживание плагина NPAPI. Этот плагин поддерживает новый формат PPAPI (Pepper API), который специально разработан для запуска плагина в отдельном процессе (песочнице). Таким образом,

эта версия уязвимости в самом плагине не может проникнуть вредоносный код в систему.

Браузер поддерживает секретный режим. Страницы, видимые в окне «Инкогнито», не отображаются в истории браузера или в истории поиска и, как файлы cookie, не оставляют других следов на вашем компьютере. Пользователи автоматически удаляются после закрытия этого окна. Однако все загруженные файлы или созданные закладки будут сохранены.

Как предупреждает браузер весной 2015 года, переход в «секретный режим» не влияет на работу других пользователей, серверов или программ и не препятствует следующему:

- интернет-сайты, которые собирают информацию или сообщают ее другим;
- интернет-провайдеры и сотрудники, которые отслеживают доступ к веб-сайтам;
- вредоносное программное обеспечение, которое отслеживает нажатия клавиш.

Speed Chrome использует высокопроизводительный обработчик JavaScript V8. Кроме того, Chrome может использовать функцию предварительной выборки DNS для ускорения загрузки страницы.

Надежность. Chrome использует архитектуру многих процессов, и часто каждая карта или плагин содержит отдельный процесс. Эта процедура называется изоляцией, которая исключает вмешательство в закладки.

Chrome сможет выполнять задачи в соответствии с потребностями диспетчера задач, который может просматривать сайты и плагины, использующие пользовательскую память, процессоры и онлайн-каналы.

Адресная строка омнибокса вверху каждой карточки. Это комбинация адресной строки и строки поиска. Если адрес омнибокса не соответствует написанию URL-адреса, он перенаправляет поисковый запрос в поисковую систему, например, если в начале пропущенного адреса в адресе адреса нет места. Автозаполнение также помогает искать закладки и историю ранее посещенных страниц. Браузер может вызывать разные поисковые системы прямо из адресной строки.

## 3.3 Инструментальное программное обеспечение

### 3.3.1 Среда разработки Intellij Idea

IntelliJ IDEA - это интегрированная среда разработки программного обеспечения для многих языков программирования, разработанная JetBrains, в частности Java и JavaScript, Python. Успешное внедрение Java, Groovy, Scala, Clojure, Erlang, Инструменты анализа качества кода, простая навигация, усовершенствованный рефакторинг и форматирование.

Профессиональный набор инструментов для разработки приложений для Android.

Поддерживает интеграцию с Java FX 2.0 SceneBuilder. Swing Interface Designer

Интеграция с автоматизированными инструментами сборки и управления проектами, такими как Maven, Gradle и Ant.

Инструменты тестирования, поддерживающие JUnit, TestNG, Spock, ScalaTest и spec2.

Интеграция с Git, Subversion, Mercurial и CSV.

### 3.3.2 Java Development Kit

Java Development Kit (JDK) - это один из трех основных технологических пакетов, используемых для Java-приложений виртуальной машины Java (JVM) и Java Runtime Environment (JRE). Важно различать их с точки зрения актуальности этих трех технологий.

JVM является компонентом платформы Java, которая реализует программу.

JRE является частью Java на диске для создания JVM.

С JDK разработчики могут создавать приложения Java, работающие на JVM и JRE.

JDK - это инструмент разработки программного обеспечения на основе Java.

У каждого JDK есть Java-компилятор. Компилятор - это программа, которая обрабатывает сырые файлы .java в виде простого текста и делает их исполняемыми. class.

## 3.3 Функциональное назначение

Описание функционального назначения структуры приложения.

Базовой модуль Maven:

- POM.xml – хранит, набор зависимостей (библиотек, драйверов, плагинов), необходимых для работы проекта.

Классы:

- MvcConfig - определяет методы для настройки конфигурации на основе Java для Spring MVC, включенной через @EnableWebMvc;
- WebSecurityConfig – реализует методы веб-безопасности Spring Security;
- Customer – сущность клиента;
- Employee - сущность сотрудника;
- Role – сущность роли сотрудника;
- Credit – сущность кредита;
- BpmSystemApplication – класс для запуска Spring Boot приложения;
- Position – сущность должности сотрудника.

Классы контроллеров:

- AccountantController – реализует функционал роли бухгалтера;
- CustomerController - реализует функционал роли сорудника;
- EmployeeController - реализует функционал роли менеджера;
- RegistrationController – осуществляет авторизацию сотрудника;
- SecurityController – осуществляет функционал роли службы безопасности;

Интерфейсы:

- CustomerRepository – реализует интерфейс для операций с базой данных, связанных с клиентами;
- EmployeeRepository – реализует интерфейс для операций с базой данных, связанных с сотрудниками.

Сервисы:

- EmployeeService – реализует бизнес-логику сотрудника;
- CustomerService - реализует бизнес-логику клиента.

Веб-страницы:

- add_employee.mustache – страница добавления нового сотрудника;
- addCustomer.mustache - страница добавления нового клиента;
- emp_page.mustache – главная страница сотрудника;
- login.mustache – страница входа в систему;
- registration.mustache – страница регистрации сотрудника;
- revision.mustache – страница для доработки;
- step_2.mustache – страница второго шага проверки;
- step_3.mustache – страница третьего шага проверки;
- step_4.mustache – страница четвертого шага проверки;

Параметры для подключения к базе данных - application.properties.

### 3.4 Вызов и загрузка

Приложение запускается через один из браузеров (Internet Explorer, Opera, Firefox, Google Chrome) по адресу http://localhost/8080.

### 3.5 Используемые технические средства

При разработке ПО использовался персональный компьютер со следующими характеристиками:

- процессор Intel® Core™ i3-5005U CPU @ 2.00 GHz;
- оперативная память 4,00 ГБ DDR3L 1600 МГц;
- жесткий диск HDD 500 ГБ;
- оптический привод встроенный DVD-RW; поддержка Double Layer;
- видеокарта NVIDEA GeForce 920m.

# ЗАКЛЮЧЕНИЕ

Увеличение использования интернета среди компаний и привело к широкому распространению веб-приложений. Веб-приложения дают возможность оптимизировать свои операции, повысить эффективность и сократить расходы. Также у них есть дополнительное преимущество: они работают на разных платформах, имеют более широкий охват и легко доступны из любой точки мира. Успешность компании (банков), в современном мире, зависит от грамотно реализованной информационной системы.

В ходе выполнения дипломного проекта в соответствии с заданием была разработана информационная система, под названием «Фронтальная система банковского обслуживания». Была разработана структура базы данных в СУБД MySQL. Были включены функции добавления, редактирования записей, функции поиска, авторизации и аутентификации, с учетом всех выдвинутых требований.

Работа была выполнена с использованием новых и актуальных технологий разработки веб-приложений, как Spring boot, Hibernate.

Разработанная программа устойчиво выполняет все свои функции.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ


1 Уоллс К. У62 Spring в действии. – М.: ДМК Пресс, 2016. – 752 с.: ил.

2 Шилдт, Герберт Ш57 Java 8: руководство для начинающих, 6-е изд.: Пер с англ. – М.: ООО “И.Д.Вильямс”,2016 -720с.: ил. – Парал. тит.англ.

3 Лонг Джош, Бастани Кеннет Л76 Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry. – СПб.: Питер, 2019. – 624 с.: ил.

4 Пьюривал С. П96 Основы разработки веб-приложений. – СПб.: Питер, 2015. – 272 с.: ил.

5 Кузнецов М. В., Симдянов И. В. К89 Самоучитель MySQL 5. — СПб.: БХВ-Петербург, 2006. — 560 с.: ил. ISBN 978-5-94157-754-5.

6 Адам Фримен Ш74 Pro ASP.NET MVC 5. — СПб.: Питер, 2017. – 537 с.: ил.

7 Гонсалвес Э. Изучаем java EE7- СПб.: Питер, 2014 – 640.: ил.

8 Хеник Б. П38 HTML и CSS: путь к совершенству. - СПб.: Питер, 2011. – 336 с.: ил.

9 Юлиана Козмина, Роб Харроп, Крис Шефер, Кларенс Хо Spring 5 для профессионалов, издательство - ДИАЛЕКТИКА, 2019. – 1150 с.

10 Бауэр, Кинг, Грегори Java Persistence API и Hibernate. – издательство ДМК-пресс, 2017 – 637 с.

11 Роберт Мартин Чистый код. Создание, анализ и рефакторинг. – СПб.: Питер, 2015– 464 с.

12 Дейт, К. Дж. Д27 Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328 с.: ил. — Парал. тит. англ.

# ПРИЛОЖЕНИЕ

## Листинг MvcConfig.java

```
package com.myspring.bpmsystem.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {
   public void addViewControllers(ViewControllerRegistry registry) {
      registry.addViewController("/login").setViewName("login");
   }
}
```

## Листинг WebSecurityConfig.java

```
package com.myspring.bpmsystem.config;

import com.myspring.bpmsystem.services.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManager
Builder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSec
urity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAda
pter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Autowired
   private EmployeeService employeeService;

   @Override
   protected void configure(HttpSecurity http) throws Exception {
```

```
        http
                .authorizeRequests()
                .antMatchers("/", "/registration").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
                .logout()
                .permitAll();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(employeeService)
                .passwordEncoder(NoOpPasswordEncoder.getInstance());
    }
}
```

Листинг AccountantController.java

```
package com.myspring.bpmsystem.controllers;

import com.myspring.bpmsystem.models.Customer;
import com.myspring.bpmsystem.services.CustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Controller
@PreAuthorize("hasAuthority('ACCOUNTANT') or hasAuthority('ADMIN')")
@RequestMapping("/accountant")
public class AccountantController {

    @Autowired
    private CustomerService customerService;

    @GetMapping("/step_3")
    public String step3(Model model){
        Customer customer = customerService.takeStep3();
```

```
        model.addAttribute("customer", customer);
        return "step_3";
    }

    @PostMapping("/revision")
    public String revision(Customer customer){
        customerService.sendRevision3(customer);
        return "step_3";
    }

    @PostMapping("/refuse")
    public  String refuse(Customer customer){
        customerService.sendRefused(customer);
        return "step_3";
    }

    @PostMapping("/step_4")
    public String step4(Customer customer,
                        Model model){
        String message = null;
        if (customer!=null){
            customerService.sendStep4(customer);
            message = "customer send!";
        }
        model.addAttribute("message", message);
        return "step_3";

    }

}
```

Листинг CustomerController.java

```
package com.myspring.bpmsystem.controllers;

import com.myspring.bpmsystem.models.Customer;
import com.myspring.bpmsystem.services.CustomerService;
import com.myspring.bpmsystem.services.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;

@Controller
@RequestMapping("/customer")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    @Autowired
    private EmployeeService employeeService;

    @GetMapping("/")
    public String mainPage(){
        return "mainCustomer";
    }

    @GetMapping("/addCustomer")
    public String addCustomerPage(Model model){
//        Customer customer = customerService.takeRevision();
//        if (customer!=null){
//            model.addAttribute("customer", customer);
//        }

        return "addCustomer";
    }

//    @GetMapping("/step_2")
//    public String step2Page(Map<String, Customer> model){
//        Customer customer = customerService.takeStep2();
//        model.put("customer", customer);
//        return "step_2";
//    }

    @PostMapping("/step_2")
    public String step2( Customer customer,
                         Model model){

        String message = null;
        if(customer!=null){
//            customer.setId(0);
            customerService.sendStep2(customer);
            message = "customer send!";
        }
```

```
    model.addAttribute("message", message);
    return "addCustomer";
}

@GetMapping("/refused_customers")
public String refusedCustomers(Model model){
    List<Customer> refusedCustomers = customerService.refusedList();
    model.addAttribute("customers", refusedCustomers);
    return "mainCustomer";

}

@GetMapping("/approved_customers")
public String approvedCustomers(Model model){
    List<Customer> approvedCustomers = customerService.approvedList();
    model.addAttribute("customers", approvedCustomers);
    return "mainCustomer";
}

@GetMapping("/revision")
public String revision(Model model){
    Customer customer = customerService.takeRevision();
    model.addAttribute("customer", customer);
    return "revision";
}


@GetMapping("/read")
public List<Customer> read() {
    return customerService.read();
}

@PutMapping(value = "/create")
public Customer create(@RequestBody Customer customer) {
    return customerService.create(customer);
}

@PostMapping(value = "/update")
public Customer update(@RequestBody Customer customer) {
    return customerService.update(customer);
}

@DeleteMapping(value = "/delete")
public void delete(@RequestParam(name = "id") Integer id) {
    customerService.delete(id);
```

```
    }

}
```

Листинг EmployeeController.java

```
package com.myspring.bpmsystem.controllers;

import com.myspring.bpmsystem.models.Customer;
import com.myspring.bpmsystem.models.Employee;
import com.myspring.bpmsystem.services.CustomerService;
import com.myspring.bpmsystem.services.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@PreAuthorize("hasAuthority('ADMIN') " +
        "or hasAuthority('MANAGER')")
@RequestMapping("/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @Autowired
    private CustomerService customerService;

    @GetMapping("/step_2")
    public String step2Page(Model model){
        Customer customer = customerService.takeStep2();
        model.addAttribute("customer", customer);
        return "step_2";
    }

    @PostMapping("/revision")
    public String revision( Customer customer){
        customerService.sendRevision(customer);
        return "redirect:/employee/step_2";
    }
```

```
@PostMapping("/refuse")
public String refuse(Customer customer){
   customerService.sendRefused(customer);
   return "redirect:/employee/step_2";
}

@PostMapping("/step_3")
public String step3(Customer customer,
                    Model model){
   customerService.sendStep3(customer);
   String message = null;
   if(customer!=null){
      customerService.sendStep2(customer);
      message = "customer send!";
   }
   model.addAttribute("message", message);
   return "step_2";
}

@GetMapping("/main")
public String employeePage(){
   return "emp_page";
}

@GetMapping("/read")
public List<Employee> read(){
   return employeeService.read();
}

@PutMapping("/create")
public Employee create(@RequestBody Employee employee){
   return employeeService.create(employee);
}

@PostMapping("/update")
public Employee update(@RequestBody Employee employee){
   return employeeService.update(employee);
}

@DeleteMapping("/delete")
public void delete(@RequestParam(name = "id") Integer id){
   employeeService.delete(id);
}
```

```
}
```

Листинг RegistrationController.java

```
package com.myspring.bpmsystem.controllers;

import com.myspring.bpmsystem.models.Employee;
import com.myspring.bpmsystem.models.Role;
import com.myspring.bpmsystem.repositories.EmployeeRepository;
import com.myspring.bpmsystem.services.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.Arrays;
import java.util.Collections;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

import static java.util.Collections.singleton;

@Controller
public class RegistrationController {
   @Autowired
   private EmployeeRepository employeeRepository;

   @Autowired
   private EmployeeService employeeService;

   @GetMapping("/registration")
   public String registration(Model model) {

      model.addAttribute("roles", Role.values());
      return "registration";
   }

   @PostMapping("/registration")
   public String addUser(Employee employee, Map<String, Object> model,
                         @RequestParam(name = "role") Role role) {
```

```
        Employee employeeFromDb =
employeeRepository.findByUsername(employee.getUsername());

        if (employeeFromDb != null) {
            model.put("message", "User exists!");
            return "registration";
        }


        employee.setActive(true);
//        employee.setRoles(Collections.singleton(Role.USER));
        employee.setRoles(Collections.singleton(role));

        employeeService.create(employee);

        return "redirect:/login";
    }
}
```

Листинг RegistrationController.java

```
package com.myspring.bpmsystem.controllers;

import com.myspring.bpmsystem.models.Customer;
import com.myspring.bpmsystem.services.CustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@PreAuthorize("hasAuthority('SECURITY') or hasAuthority('ADMIN')")
@RequestMapping("/security")
public class SecurityController {

    @Autowired
    private CustomerService customerService;

    @GetMapping("/step_4")
    public String step4(Model model){
        Customer customer = customerService.takeStep4();
        model.addAttribute("customer", customer);
        return "step_4";
```

```
    }

    @PostMapping("/revision")
    public String revision(Customer customer){
        customerService.sendRevision4(customer);
        return "step_4";
    }

    @PostMapping("/refuse")
    public  String refuse(Customer customer){
        customerService.sendRefused(customer);
        return "step_4";
    }

    @PostMapping("/confirm")
    public String confirm(Customer customer,
                          Model model){
        String message = null;
        if (customer!=null){
            customerService.sendApproved(customer);
            message = "customer send!";
        }
        model.addAttribute("message", message);
        return "step_4";

    }
}
```

Листинг Customer.java

```
package com.myspring.bpmsystem.models;

import javax.persistence.*;

@Entity
@Table(name = "customers")
public class Customer implements Comparable<Customer> {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private String name;
```

```
private String surname;
private String patronymic;
private String birthDate;
private String iin;
private String job;
private String position;
private String salary;
private String sum;

public Customer() {
}

public Integer getId() {
   return id;
}

public void setId(Integer id) {
   this.id = id;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public String getSurname() {
   return surname;
}

public void setSurname(String surname) {
   this.surname = surname;
}

@Override
public int compareTo(Customer o) {
   return 0;
}

public String getPatronymic() {
   return patronymic;
}
```

```
public void setPatronymic(String patronymic) {
    this.patronymic = patronymic;
}

public String getBirthDate() {
    return birthDate;
}

public void setBirthDate(String birthDate) {
    this.birthDate = birthDate;
}

public String getIin() {
    return iin;
}

public void setIin(String iin) {
    this.iin = iin;
}

public String getJob() {
    return job;
}

public void setJob(String job) {
    this.job = job;
}

public String getPosition() {
    return position;
}

public void setPosition(String position) {
    this.position = position;
}

public String getSalary() {
    return salary;
}

public void setSalary(String salary) {
    this.salary = salary;
}

public String getSum() {
```

```
        return sum;
    }

    public void setSum(String sum) {
        this.sum = sum;
    }
}
```

Листинг Employee.java

```
package com.myspring.bpmsystem.models;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import com.myspring.bpmsystem.models.Role;
import javax.persistence.*;
import java.util.Collection;
import java.util.Set;

@Entity
@Table(name = "employees")
public class Employee implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String username;
    private String password;
    private String employee_name;
    private String surname;
    private String patronymic;
    private String birthDate;
    private String position;
    private String iin;
    private boolean active;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "employee_role", joinColumns = @JoinColumn(name =
"employee_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;

    public Employee() {
    }

    public Long getId() {
        return id;
```

```
}

public void setId(Long id) {
   this.id = id;
}

public String getEmployee_name() {
   return employee_name;
}

public void setEmployee_name(String employee_name) {
   this.employee_name = employee_name;
}

public String getSurname() {
   return surname;
}

public void setSurname(String surname) {
   this.surname = surname;
}

public String getPatronymic() {
   return patronymic;
}

public void setPatronymic(String patronymic) {
   this.patronymic = patronymic;
}

public String getBirthDate() {
   return birthDate;
}

public void setBirthDate(String birthDate) {
   this.birthDate = birthDate;
}

public String getPosition() {
   return position;
}

public void setPosition(String position) {
   this.position = position;
}
```

```
public String getIin() {
    return iin;
}

public void setIin(String iin) {
    this.iin = iin;
}

public String getUsername() {
    return username;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return isActive();
}

public void setUsername(String username) {
    this.username = username;
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return (Collection<? extends GrantedAuthority>) getRoles();
}

public String getPassword() {
    return password;
}
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }
}
```

Листинг Role.java

```
package com.myspring.bpmsystem.models;

import org.springframework.security.core.GrantedAuthority;

public enum Role implements GrantedAuthority {
    USER, ADMIN, MANAGER, ACCOUNTANT, SECURITY;

    @Override
    public String getAuthority() {
        return name();
    }
}
```

Листинг CustomerRepository.java

```
package com.myspring.bpmsystem.repositories;

import com.myspring.bpmsystem.models.Customer;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
public interface CustomerRepository extends CrudRepository<Customer, Integer> {

}
```

## Листинг EmployeeRepository.java

```
package com.myspring.bpmsystem.repositories;

import com.myspring.bpmsystem.models.Employee;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Integer> {
   Employee findByUsername(String username);
}
```

## Листинг CustomerService.java

```
package com.myspring.bpmsystem.services;

import com.myspring.bpmsystem.models.Customer;
import com.myspring.bpmsystem.models.Employee;
import com.myspring.bpmsystem.repositories.CustomerRepository;
import org.hibernate.ObjectDeletedException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
import java.util.stream.Collectors;

@Service
public class CustomerService {

   @Autowired
```

```
    private CustomerRepository customerRepository;

//    private List<String> ListOfKeys = null;
//    private List<Object> ListOfValues = null;

    PriorityQueue<Customer> sendCustomers2 = new PriorityQueue<Customer>();

    PriorityQueue<Customer> sendCustomers3 = new PriorityQueue<Customer>();

    PriorityQueue<Customer> sendCustomers4 = new PriorityQueue<>();

    PriorityQueue<Customer> revision = new PriorityQueue<>();

    PriorityQueue<Customer> revisionStep3 = new PriorityQueue<>();

    PriorityQueue<Customer> revisionStep4 = new PriorityQueue<>();

    List<Customer> refused = new ArrayList<>();

    List<Customer> approved = new ArrayList<>();

    public List<Customer> read(){
        return (List<Customer>) customerRepository.findAll();
    }

    public Customer create(Customer customer){
        return customerRepository.save(customer);
    }

    public Customer update(Customer customer){
        return customerRepository.save(customer);
    }

    public void delete(Integer id){
        customerRepository.deleteById(id);
    }

    public void sendStep2(Customer customer){
//        this.sendCustomers.add(customer);
        this.sendCustomers2.add(customer);
    }

    public Customer takeStep2(){
//        Customer customer = sendCustomers.get(0);
//        sendCustomers.remove(0);
```

```
    Customer customer = sendCustomers2.peek();
    return customer;
}

public void sendStep3(Customer customer){
    this.sendCustomers3.add(customer);
}

public Customer takeStep3(){
    Customer customer = sendCustomers3.peek();
    return customer;
}

public void sendStep4(Customer customer){
    this.sendCustomers4.add(customer);
}

public Customer takeStep4(){
    Customer customer = sendCustomers4.peek();
    return customer;
}

public void sendRevision(Customer customer){
    this.revision.add(customer);
}

public Customer takeRevision(){
    return this.revision.peek();
}

public void sendRevision3(Customer customer){
    this.revisionStep3.add(customer);
}

public Customer takeRevision3(){
    return this.revisionStep3.peek();
}

public void sendRevision4(Customer customer){
    this.revisionStep4.add(customer);
}

public Customer takeRevision4(){
    return this.revisionStep4.peek();
}
```

```
    public void sendRefused(Customer customer){
        this.refused.add(customer);
    }

    public void sendApproved(Customer customer){
        this.approved.add(customer);
    }

    public List<Customer> refusedList(){
        return this.refused;
    }

    public List<Customer> approvedList(){
        return this.approved;
    }
```

Листинг EmployeeService.java

```
package com.myspring.bpmsystem.services;

import com.myspring.bpmsystem.models.Employee;
import com.myspring.bpmsystem.repositories.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class EmployeeService implements UserDetailsService {

    @Autowired
    private EmployeeRepository employeeRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return employeeRepository.findByUsername(username);
    }
```

```
    public List<Employee > read(){
        return (List<Employee>) employeeRepository.findAll();
    }

    public Employee create(Employee employee){
        return employeeRepository.save(employee);
    }

    public Employee update(Employee employee){
        return employeeRepository.save(employee);
    }

    public void delete(Integer id){
        employeeRepository.deleteById(id);
    }
}
```

Листинг BpmSystemApplication.java

```
package com.myspring.bpmsystem;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BpmSystemApplication {

        public static void main(String[] args) {
                SpringApplication.run(BpmSystemApplication.class, args);
        }

}
```