МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»
Кафедра IT - инжиниринг

«ДОПУЩЕН К ЗАЩИТЕ»
Зав. кафедрой PhD, доцент Досжанова А.А
_____ «____» _____2020 г.

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка серверной части системы для автоматизации учебного процесса
Специальность 5В070400 – «Вычислительная техника и программное обеспечение»
Выполнил: Алтынов Д.М. Группа: ВТу-17-4
Научный руководитель: к.т.н., доцент Табултаев С.С

Консультанты:
по экономической части: Габелашвили К.Р.
_____ «____» _____2020 г.
по безопасности жизнедеятельности: Приходько Н.Г.
_____ «____» _____2020 г.
по программному обеспечению: Майкотов М.Н
_____ «____» _____2020 г.
Нормоконтролер: Абсатарова Б.Р.
_____ «____» _____2020 г.
Рецензент:_
_____
(учѐная степень, звание, Ф.И.О.)
_____ «____» _____2020 г.

Алматы 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Некоммерческое акционерное общество
«АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ
им. ГУМАРБЕКА ДАУКЕЕВА»

Институт систем управления и информационных технологий

Специальность 5В070400 – «Вычислительная техника и программное обеспечение»

Кафедра IT- инжиниринг

ЗАДАНИЕ
на выполнение дипломного проекта

Студенту Алтынов Дәулет Тасқынұлы

Тема проекта: Разработка серверной части системы для автоматизации учебного процесса

Утверждена приказом по университету № ___ от «___» _____2020 г.

Срок сдачи законченного проекта «___» _____ 2020 г.

Исходные данные к проекту (требуемые параметры результатов исследования (проектирования) и исходные данные объекта): Angular- среда разработки БД, Intellij idea – среда программирования, исходные правила обращения с информацией, список оборудования.

Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломного проекта:

а) Обработка и хранение данных;

б) Распределение ролей на сервере;

в) Отправка уведомлений на 3 языках, материалов и оценок.

Перечень графического материала (с точным указанием обязательных чертежей): представлены 12 таблиц, 50 иллюстраций, презентация (20 слайда).

Основная рекомендуемая литература:

1 Партыка, Т.Л. Информационная безопасность: Учебное пособие / Т.Л. Партыка, И.И. Попов. - М.: Форум, 2018. - 88 с.;

2 Баранова, Е.К. Информационная безопасность и защита информации: Учебное пособие / Е.К. Баранова, А.В. Бабаш. - М.: Риор, 2018. - 400 с.;

3 Основы безопасности жизнедеятельности. Алексеенко В.А., Матасова И.Ю., 2001. – 187 с "Безопасность в чрезвычайных ситуациях: Учебник" под ред. Н.К. Шишкина. – М., ГУУ, 2017. - 90 с.

Консультация по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант	Сроки	Подпись
Технико-экономическое обоснование проекта	к.э.н., ассоц. профессор Габелашвили К.Р.	06.04.2020 – 25.04.2020	
Безопасность жизнедеятельности	доц. Приходько Н. Г.	06.04 – 25.04.2020	
Программное обеспечение	ст.преп. Майкотов М.Н.	04.05 – 08.05.2020	
Нормоконтролер	ст.преп. Абсатарова Б.Р.	04.05 – 08.05.2020	

ГРАФИК
подготовки дипломного проекта

Наименования разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечания
Анализ предметной области	13.01.20	
Выбор программного обеспечения	23.02.20	
Сравнительный анализ аналогов	04.03.20	
Написание технического задания	26.03.20	
Создание UML диаграмм	14.04.20	
Реализация программного продукта	15.05.20	

Дата выдачи задания «13 января 2020г.

Заведующий кафедрой _____А.А.Досжанова

Научный руководитель проекта _____С.С.Табултаев

Задание принял к исполнению студент _____Д.Т.Алтынов

АНДАТПА

Бұл диплом жобасында мекеме үшін ақпараттық жүйені құру, электронды күнделік, хабарлама, материалды беру мәселелерімен айналысады. АЖ Java-да жасалынған, Postgres мәліметтер базасы үшін қолданылады, ал жобаны азайту және орналастыру үшін Docer қолданылды.

Бұл жобаның игерілуіне, экономикалық тиімділігі мен рентабельділігінің экономикалық есебі жасалды.

Тіршілік қауіпсіздігі мәселелері қаралды және шешілді. Кондиционерлеу және желдету жүйелері.

АННОТАЦИЯ

Данного дипломного прроекта рассматривает вопрос создания информационной системы для учреждений, электронный дневник, оповещение, передача материала. ИС разработана на языке Java, для базы данных используется Postgres, для уменьшения и развертывания проекта использовался Docer.

Произведен экономический расчет затрат на разработку, экономическая эффективность и рентабельность данного проекта.

Рассмотрены и решены вопросы безопасности жизнедеятельности. Рассчитаны системы кондиционирования и вентилирования помещения.

ANNOTATION

This thesis deals with the issue of creating an information system for institutions, an electronic diary, notification, transfer of material. The IS is developed in Java, Postgres is used for the database, and Docer was used to reduce and deploy the project.

The economic calculation of development costs, economic efficiency and profitability of this project was made.

Considered and resolved issues of life safety. Designed air conditioning and ventilation systems.

Содержание

Введение

Развитие IT-технологий не стоит на месте в связи с этим программисты имеют возможность использовать самые современные технологии для создания программных обеспечений. Примером таких технологий используемых в данной дипломной работе являются Angular, Docker, Spring и многие другие. Есть множество сайтов где люди делятся своими проектами, что позволяет испытать и понять, как работают те или иные инструменты. В отличии от стандартных инструментов, новые позволяют более гибко подходить к делу и имеют больший потенциал в создании проектов. Многие компании, сервисы и даже учебные заведения применяют такие разработки для разных целей, например, Docker используется в банковской и биржевой сфере, он позволяет уменьшает размер операционной системы убирая из нее сторонние библиотеки, функций и приложений, для работы приложений с доступом к терминалу. Это позволяет ускорить процесс разработки и пропадает необходимость в установке других инструментов.

В целом информационная система - это обработка информации и определенные ресурсы необходимые для ее работы (человеческие, технические, финансовые и т. д.). Информационная система предназначена для своевременного обеспечения информацией студентов и преподавателей, то есть для удовлетворения конкретных информационных потребностей в рамках определенной предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы и базы данных.

Мой проект служит целью подогнать под новые стандарты учебные заведения. В программе использовалось большое количество новых инструментов, что позволило более гибко и менее ресурсозатратно создать сайт. В отличии от нашего старого сайта он более легкий, гибкий и не требует больших знаний, так же он позволит более эффективно использовать ресурсы преподавателей и студентов. Так же обновление сайта позволит показать уровень нашей подготовки, которую предоставили нам преподаватели.

Для достижения этой цели необходимо выполнить следующие задачи:

- для анализа аналоговых продуктов
- разработать архитектуру веб-сервиса
- разработать веб-сервис
- развернуть веб-сервис в ресторане
- сделать тестирование и улучшить.

Для использования системы для пользователей не будет сложностей, так как проект обладает простыми и понятными функциями, кроме необходимости подключения к Интернету.

1 Аналитическая часть

1.1 Описание предметной области

Первый в мире Web сайт появился в 1991 году и состоял он из ссылок и текста, что для современного человека покажется не достаточно. Позже в 1993 году появляется первый графический браузер Mosaic. В нем появилась поддержка картинок, что уже приблизило сайт к привычному нам виду.

С 1994 по 1996 год стали появляется различные шрифты и анимированные GIF-изображения. В 1998 году появляется поддержка FLASH-технология, вследствие чего появляется возможность добавлять “мигающие” заставки. И уже начиная с 2000 годов идет активное слияние разных технологий продвигающих веб индустрию.

Web индустрия в нашей повседневной жизни несет огромное значение, большая часть информации находится именно в web ресурсах, тем самым уменьшая временные затраты на поиск необходимой материал. Большой упор в таких случаях идет не только на содержание сайта, но и его визуальную и функциональную часть. Чем проще и понятнее будет интерфейс, тем больший будет охват аудитории.

Web сайт это очень многогранный и сложный инструмент для взаимодействия разных компаний, учреждений или просто групп людей. Для освоения всего функционала потребуется не один год, но результат может превзойти все ожидания. Это будет связанно развитием множества навыков таких как дизайн, программирование, маркетинг, психология, моделирование. Ведь нужно заинтересовать пользователей и заставить их снова вернутся. Если это внутренний сайт, как например сайт университета, то необходимо создать такой функционал, который позволит всем без исключения использовать все его механизмы на полную.

1.2 Цель разработки

Цель разработки – разработать индивидуальный сайт для университета.

- обеспечение учебных заведений современной информационной системой;
- увеличение эффективности работы и администрировании системы;
- уменьшение материальных затрат на поддержание серверов и базы данных.

Проект должен соответствовать современным мировым стандартам, быть адаптивным, функциональным и безопасным. Так же в нем необходимо обеспечить своевременное оповещение студентов об их успеваемости и предстоящих рубежных контролях, экзаменов и праздников. Коммуникация между преподавателями и студентами. Возможностью настроить визуальную часть под себя.

1.3 Обоснование выбора ПО

Основой для создания проекта была выбрана среда разработки IntelliJ Idea. В нем представлен большой спектр поддерживаемых языков программирования таких как Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML и многих других. Так же помимо этого в нем представлены множества инструментов сборкой, управление и тестированием проектом: Maven, Gradle, Ant, JUnit, TestNG, Spock, и другими.

Как язык программирования был выбран Java и дополнительными к нему Angular, Docer, Spring, Nginx, Bootstrap, Postgres, Liquibase, Gradle, Rest.

Java выбрана за основу не спроста это очень гибкий надежный язык программирования позволяющий создавать большие проекты не создавая нагрузки, имея высокую безопасность и больше похож на общение двух людей, а не машины и человека. Так же Java является одним из самых распространённых языков программирования. Существует огромное количество созданной литературы и проектов, что позволяет сильно углубится в изучение этого языка.[1]

Разработка промышленных приложений на платформе J2EE всегда считалась достаточно непростым занятием. Утверждение, конечно, спорное, потому что сразу начнутся вопросы - кем считалась, когда и что тогда простое занятие... Но примем его на веру, потому что иначе трудно объяснить, зачем нужно было придумывать Spring Framework. Так вот, он придуман специально для того, чтобы упростить разработку enterprise-приложений.

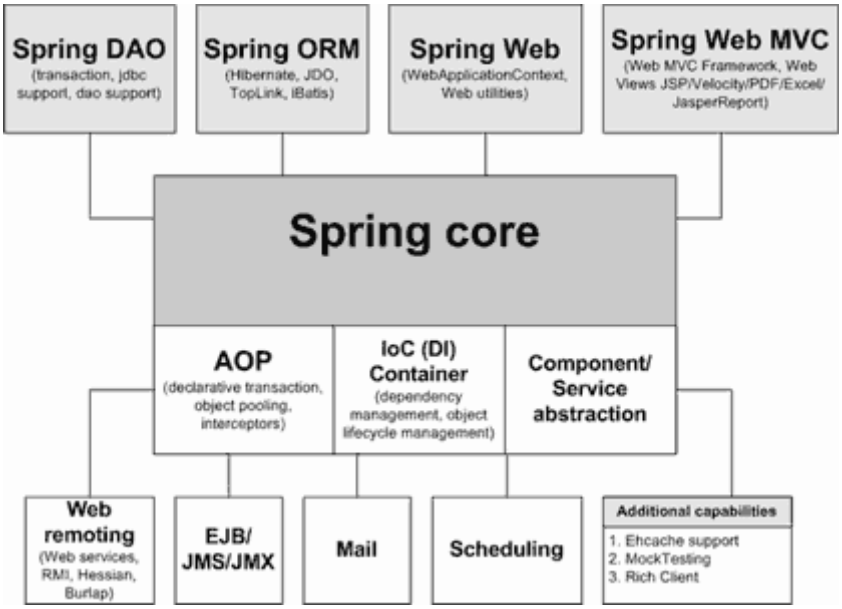
Spring - это, можно сказать, архитектурный фреймворк, потому что он придуман не столько для выполнения какой-то прикладной задачи (как, скажем, Struts, который нужен, чтобы писать web-приложения), а для обеспечения лучшей масштабируемости, возможности более простого тестирования и более простой интеграции с другими фреймворками (например, уже упомянутым Struts или Hibernate). Благодаря этому писать большие приложения становится проще - разработчики просто избегают ряда проблем, связанных с созданием enterprise-приложений, вместо того, чтобы их решать.[2]

Spring - достаточно крупный фреймворк. Потому что создатели этого фреймворка ухитрились охватить практически все аспекты программирования промышленных Java-приложений. Соответственно, и составных частей у Spring Framework немало. Например:

- IoC (Inversion of Control) контейнер;
- AOP-фреймворк (включая интеграцию с AspectJ);
- Data Access фреймворк;
- Transaction management;
- MVC-фреймворк;
- Remote Access фреймворк;
- Batch processing;

- фреймворк аутентификации и авторизации;
- Remote Management;
- Messaging-фреймворк;
- Testing-фреймворк.

На рисунке 1.1 можно увидеть схему фреймворка. Из этой схемы видно, что первые два компонента в нашем списке являются самыми важными - это, в некотором роде, "сердце" фреймворка. Поэтому, прежде чем говорить дальше о каждом из перечисленных следом за этими двумя компонентов, давайте подробнее разберёмся с тем, что лежит в его основе - то есть, с шаблоном проектирования Inversion of Control и аспектно-ориентироаванным программированием.[2]



Риуснок 1.1 - Схема фрейморка

При использовании объектно-ориентированного или процедурного программирования в программе всегда остаётся некоторая доля функциональности, которую невозможно выделить в отдельный модуль, который разрабатывал бы какой-то отдельный разработчик (пусть даже отдельная команда разработчиков). Это называется сквозной функциональностью программы - она как бы проходит через всю программу насквозь. Хрестоматийный пример сквозной функциональности - обработка исключительных ситуаций. Ещё один пример, уже ближе к функциональности Spring, - это ведение журналов событий (логов). Вот аспектно-ориентированное программирование и призвано выделять сквозную функциональность программы в отдельные, скажем так, сущности и обеспечивать программе в ещё большей степени модульную структуру. Вполне естественно, что аспектно-ориентированное программирование нашло понимание у создателей Spring, которые стремились максимально изолировать друг от друга отдельные части программы. [2]

Аспектом в АОП называется тот модуль (или класс), который и реализует сквозную функциональность. Он применяется остальным кодом приложения в так называемых точках соединения, набор которых называется срезом.

Но каким образом Spring Framework позволяет разработчикам получить все эти преимущества? Для этого нужно посмотреть на составные части фреймворка и поговорить о каждой из них в отдельности.

PostgreSQL - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.[3]

Надежность PostgreSQL является проверенным и доказанным фактом и обеспечивается следующими возможностями:

- полное соответствие принципам ACID - атомарность, непротиворечивость, изолированность, сохранность данных.
- многоверсионность (Multiversion Concurrency Control,MVCC) используется для поддержания согласованности данных в конкурентных условиях, в то время как в традиционных базах данных используются блокировки. MVCC означает, что каждая транзакция видит копию данных (версию базы данных) на время начала транзакции, несмотря на то, что состояние базы могло уже измениться. Это защищает транзакцию от несогласованных изменений данных, которые могли быть вызваны (другой) конкурентной транзакцией, и обеспечивает изоляцию транзакций.
- наличие Write Ahead Logging (WAL) - общепринятый механизм протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев.
- Point in Time Recovery (PITR) - возможность восстановления базы данных (используя WAL) на любой момент в прошлом, что позволяет осуществлять непрерывное резервное копирование кластера PostgreSQL.
- репликация также повышает надежность PostgreSQL. Существует несколько систем репликации, например, Slony (тестируется версия 1.1), который является свободным и самым используемым решением, поддерживает master-slaves репликацию.
- целостность данных является сердцем postgresql. помимо mvcc, postgresql поддерживает целостность данных на уровне схемы - это внешние ключи (foreign keys), ограничения (constraints).
- открытость кодов postgresql означает их абсолютную доступность для любого, а либеральная bsd лицензия не накладывает никаких ограничений на использование кода. [источник 4]

Производительность PostgreSQL основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе.

Поддержка индексов:

- стандартные индексы - b-tree, hash, r-tree, gist (обобщенное поисковое дерево);

- частичные индексы (partial indices) - можно создавать индекс по ограниченному подмножеству значений, например, create index idx_partial on foo (x) where x > 0;

функциональные индексы (expressional indices) позволяют создавать индексы используя значения функции от параметра, например, create index idx_functional on foo (length(x));

- планировщик запросов основывается на стоимости различных планов, учитывая множество факторов. он предоставляет возможность пользователю отлаживать запросы и настраивать систему;

- система блокировок поддерживает блокировки на нижнем уровне, что позволяет сохранять высокий уровень конкурентности при защите целостности данных. блокировка поддерживается на уровне таблиц и записей. на нижнем уровне, блокировка для общих ресурсов оптимизирована под конкретную ос и архитектуру;

- управление буферами и кэширование используют сложные алгоритмы для поддержания эффективности использования выделенных ресурсов памяти;

- tablespaces (табличные пространства) для управления хранения данных на уровне объектов, таких как базы данных, схемы, таблицы и индексы. это позволяет гибко использовать дисковое пространство и повышает надежность, производительность, а также способствует масштабируемости системы;

- масштабируемость основывается на описанных выше возможностях. низкая требовательность postgresql к ресурсам и гибкая система блокировок обеспечивают его шкалирование, в то время как индексы и управление буферами обеспечивают хорошую управляемость системы даже при высоких загрузках;

- расширяемость postgresql означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов,языков, индексов и операторов. объектно-ориентированность postgresql позволяет перенести логику приложения на уровень базы данных, что сильно упрощает разработку клиентов, так как вся бизнес логика находится в базе данных. функции в postgresql однозначно определяются названием, количеством и типами аргументов. [3]

Поддержка SQL.

Кроме основных возможностей, присущих любой SQL базе данных, PostgreSQL поддерживает:

- очень высокий уровень соответствия ANSI SQL 92, ANSI SQL 99 и ANSI SQL 2003.

Схемы, которые обеспечивают пространство имен на уровне SQL. Схемы содержат таблицы, в них можно определять типы данных, функции и операторы. Используя полное имя объекта можно одновременно работать с несколькими схемами. Схемы позволяют организовать базы данных

совокупность нескольких логических частей, каждая их которых имеет свою политику доступа, типы данных.

- Subqueries - подзапросы (subselects), полная поддержка SQL92. Подзапросы делают язык SQL более гибким и зачастую более эффективным.

Outer Joins - внешние связки (LEFT,RIGHT, FULL)

- Rules - правила, согласно которым модифицируется исходный запрос. Главное отличие от триггеров состоит в том, что rule работает на уровне запроса и перед исполнением запроса, а триггер - это реакция системы на изменение данных, т.е. триггер запускается в процессе исполнения запроса для каждой измененной записи (PER ROW). Правила используются для указания системе, какие действия надо произвести при попытке обновления view.

- Cursors - курсоры, позволяют уменьшить трафик между клиентом и сервером, а также память на клиенте, если требуется получить не весь результат запроса, а только его часть.

- Table Inheritance - наследование таблиц, позволяющее создавать объекты, которые наследуют структуру родительского объекта и добавлять свои специфические атрибуты.

- Stored Procedures - серверные (хранимые) процедуры позволяют реализовывать бизнес логику приложения на стороне сервера. Кроме того, они позволяют сильно уменьшить трафик между клиентом и сервером.

Триггеры позволяют управлять реакцией системы на изменение данных (INSERT,UPDATE,DELETE), как перед самой операцией (BEFORE), так и после (AFTER). Во время выполнения триггера доступны специальные переменные NEW (запись, которая будет вставлена или обновлена) и OLD (запись перед обновлением).

- Cluster table - упорядочивание записей таблицы на диске согласно индексу, что иногда за счет уменьшения доступа к диску ускоряет выполнение запроса. [4]

Типы данных

PostgreSQL поддерживает большой набор встроенных типов данных:

- булев тип;
- численные типы;
- целые;
- с фиксированной точкой;
- с плавающей точкой;
- денежный тип (отличается специальным форматом вывода, а в остальном аналогичен числам с фиксированной точкой с двумя знаками после запятой);
- символьные типы произвольной длины;
- двоичные типы;
- типы «дата/время» (полностью поддерживающие различные форматы, точность, форматы вывода, включая последние изменения в часовых поясах);
- перечисление;

- типы текстового поиска;
- составные типы;
- hstore (расширение, добавляющее ключ-значение в postgresql);
- массивы(различной длины и любого типа данных, включая текстовый и составной типы) размером до 1 гбайта;
- геометрические примитивы;
- сетевые типы;
- ip и ipv6-адреса;
- cidr-формат;
- mac-адрес;
- xml-данные с поддержкой запросов xpath;
- uuid-идентификатор;
- json (начиная с версии 9.2) и более быстрый jsonb (начиная с версии 9.4).

Кроме того, пользователи могут создавать свои собственные типы данных, которые обычно можно полностью индексировать через инфраструктуру индексирования PostgreSQL - GiST, GIN, SP-GiST. К ним относятся типы данных географической информационной системы (ГИС) из проекта PostGIS для PostgreSQL.

Существует также тип данных, называемый «домен», который является таким же, как и любой другой тип данных, но с необязательными ограничениями, определенными создателем этого домена. Это означает, что любые данные, введенные в столбец с использованием домена, должны соответствовать тем ограничениям, которые были определены как часть домена.

Начиная с PostgreSQL 9.2, может использоваться тип данных, представляющий диапазон данных, которые называются типами диапазонов. Это могут быть дискретные диапазоны (например, все целые значения от 1 до 10) или непрерывные диапазоны (например, любой момент времени между 10:00 и 11:00). Доступные типы доступных диапазонов включают диапазоны целых чисел, большие целые числа, десятичные числа, отметки времени (с часовым поясом и без него) и даты.

Пользовательские типы диапазонов могут быть созданы для обеспечения доступности новых типов диапазонов, таких как диапазоны IP-адресов, с использованием типа inet в качестве базы или диапазонов с плавающей точкой, используя тип данных float в качестве базы. Типы диапазонов поддерживают включенные и исключительные границы диапазона, используя символы и () соответственно. (например, представляет все целые числа, начиная с 4 включительно, но не включая 9.) Типы диапазонов также совместимы с существующими операторами, используемыми для проверки наложения, сдерживания, права и т. *д. [5]*

Наследование

Таблицы могут наследовать свои характеристики от "родительской" таблицы. Данные в дочерних таблицах будут существовать в родительских

таблицах, если данные не выбираются из родительской таблицы, с помощью ключевого слова ONLY, т.е. SELECT * FROM ONLY parent_table;. Добавление столбца в родительскую таблицу будет причиной того, что столбец появится в дочерней таблице.

Наследование может быть использовано для реализации разбиения таблиц, с использованием либо триггеров либо правил, чтобы направить вставки в родительской таблице в соответствующие дочерние таблицы.

По состоянию на 2010, эта функция поддерживается не полностью, но, в частности, таблица ограничений в настоящее время не наследуемая. Все проверочные ограничения и ненулевые ограничения на родительской таблице автоматически наследуются его детьми. Другие типы ограничений (уникальные, первичный ключ, и иностранные ключевые ограничения) не наследуются.

Наследование обеспечивает способ отображения особенности иерархий обобщения, изображенных на сущность-связь диаграммах (ERD) непосредственно в базе данных PostgreSQL. [5]

- функции запроса;
- операции;
- полнотекстовый поиск;
- просмотры;
- материализованные взгляды;
- обновляемые виды;
- рекурсивные взгляды;
- внутренний, внешний (полный, левый и правый) и крест- соединения;
- суб выбирает;
- коррелированные подзапросы;
- регулярные выражения;
- общие выражения таблиц и записываемые общие табличные выражения
- зашифрованные соединения через tls (текущие версии не используют уязвимый ssl, даже с этой конфигурацией);
- домены;
- точки сохранения;
- двухфазное принятие.

TOAST (метод хранения с ограниченными возможностями) используется для прозрачного хранения больших атрибутов таблицы (таких как большие вложения MIME или XML-сообщения) в отдельной области с автоматическим сжатием.

Встроенный SQL реализован с использованием препроцессора. Код SQL сначала записывается в C-код. Затем код запускается через препроцессор ECPG, который заменяет SQL на вызовы в библиотеку кода. Затем код может быть скомпилирован с использованием компилятора C.[5]

Большая часть приложений, с которыми мне приходилось сталкиваться в ходе моей профессиональной деятельности, являлись корпоративными

приложениями, оперирующими большими массивами данных. Команды разработчиков, работающих над такими проектами, часто рассматривали базу данных как отдельную сущность, независимую от приложения. Такая ситуация может возникать из-за организационной структуры проекта, когда разработкой базы данных и разработкой приложения занимаются отдельные команды разработчиков. Иногда это просто привычка разработчиков. Как бы то ни было, я заметил, что подобное разделение проектов приводит к следующим проблемам:

- ручное внесение изменений в БД;
- разные версии БД у разных участников команды разработчиков;
- непоследовательные подходы к внесению изменений (в базу данных или данные);
- неэффективные механизмы ручного управления изменениями при переходах между версиями баз данных.

Все вышеперечисленные неэффективные действия мешают разработчикам синхронизировать БД. Кроме того, из-за них у конечных пользователей приложения могут возникнуть проблемы, связанные несовместимостью или повреждением данных.

Ручной подход, который часто используется в проектах по разработке программного обеспечения. Ручные изменения часто несовместимы и ошибочны, кроме того, из-за них может быть трудно отменить уже внесенные изменения или проанализировать историю изменений в базе данных за какой-либо период. Например, администратор базы данных может внести какие-то изменения в таблицу поиска, а разработчик зафиксировать эти изменения в следующей версии.

Избежать неудобств ручного подхода позволяет стратегия изменения базы данных, при которой вмешательство человека минимизируется. При помощи комбинации методик и инструментов можно организовать непротиворечивое и повторяемое управление изменениями в структуре базы данных и в самих данных. В этой статье будут затронуты следующие темы:

- использование инструмента LiquiBase для миграции между различными версиями одной и той же базы данных;
- автоматическое выполнение миграции базы данных;
- способы последовательного внесения изменений в базе данных;
- выполнение рефакторинга базы данных при помощи LiquiBase.

Контейнеризация является отличной альтернативой аппаратной виртуализации. Все процессы в ней протекают на уровне операционной системы, что позволяет существенно экономить ресурсы и увеличивать эффективность работы с приложениями.

Одним из наиболее популярных инструментов для программной виртуализации является Docker — автоматизированное средство управления виртуальными контейнерами. Он решает множество задач, связанных с созданием контейнеров, размещением в них приложений, управлением процессами, а также тестированием ПО и его отдельных компонентов.

Docker (Докер) — программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации. Он нужен для более эффективного использование системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы.

Выражаясь простыми словами, Docker это платформа, облегчающая процесс сборки, запуска, управления и дистрибуции приложений. Все это он делает путем виртуализации операционной системы компьютера, на котором он установлен и запущен.

Написан Docker на языке программирования Go, первая его версия была выпущена в 2013 году.

Преимущества использования Docker:

- минимальное потребление ресурсов - контейнеры не виртуализируют всю операционную систему (ос), а используют ядро хоста и изолируют программу на уровне процесса. последний потребляет намного меньше ресурсов локального компьютера, чем виртуальная машина;

- скоростное развертывание - вспомогательные компоненты можно не устанавливать, а использовать уже готовые docker-образы (шаблоны). например, не имеет смысла постоянно устанавливать и настраивать linux ubuntu. достаточно 1 раз ее инсталлировать, создать образ и постоянно использовать, лишь обновляя версию при необходимости;

- удобное скрытие процессов - для каждого контейнера можно использовать разные методы обработки данных, скрывая фоновые процессы.

- работа с небезопасным кодом - технология изоляции контейнеров позволяет запускать любой код без вреда для ОС;

- простое масштабирование - любой проект можно расширить, внедрив новые контейнеры;

- удобный запуск - приложение, находящееся внутри контейнера, можно запустить на любом docker-хосте;

- оптимизация файловой системы - образ состоит из слоев, которые позволяют очень эффективно использовать файловую систему.

Компоненты Docker:

- docker-демон (Docker-daemon) — сервер контейнеров, входящий в состав программных средств Docker. Демон управляет Docker-объектами (сети, хранилища, образы и контейнеры). Демон также может связываться с другими демонами для управления сервисами Docker;

- docker-клиент (Docker-client / CLI) — интерфейс взаимодействия пользователя с Docker-демоном. Клиент и Демон – важнейшие компоненты «движка» Докера (Docker Engine). Клиент Docker может взаимодействовать с несколькими демонами;

- docker-образ (Docker-image) — файл, включающий зависимости, сведения, конфигурацию для дальнейшего развертывания и инициализации контейнера;

- docker-файл (Docker-file) — описание правил по сборке образа, в котором первая строка указывает на базовый образ. Последующие команды выполняют копирование файлов и установку программ для создания определенной среды для разработки;

- docker-контейнер (Docker-container) — это легкий, автономный исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска приложения: код, среду выполнения, системные инструменты, системные библиотеки и настройки;

- том (volume) — эмуляция файловой системы для осуществления операций чтения и записи. она создается автоматически с контейнером, поскольку некоторые приложения осуществляют сохранение данных;

- реестр (docker-registry) — зарезервированный сервер, используемый для хранения docker-образов. примеры реестров;

- центр docker — реестр, используемый для загрузки docker-image. он обеспечивает их размещение и интеграцию с github и bitbucket;

- контейнеры azure — предназначен для работы с образами и их компонентами в директории azure (azure active directory);

- доверенный реестр docker или dtr — служба docker-реестра для инсталляции на локальном компьютере или сети компании;

- docker-хаб (Docker-hub) или хранилище данных — репозиторий, предназначенный для хранения образов с различным программным обеспечением. Наличие готовых элементов влияет на скорость разработки;

- docker-хост (Docker-host) — машинная среда для запуска контейнеров с программным обеспечением;

- docker-сети (Docker-networks) — применяются для организации сетевого интерфейса между приложениями, развернутыми в контейнерах;

Элементы Docker Engine

- сервер выполняет инициализацию демона (фоновой программы), который применяется для управления и модификации контейнеров, образов и томов;

- rest api — механизм, отвечающий за организацию взаимодействия докер-клиента и докер-демона;

- клиент — позволяет пользователю взаимодействовать с сервером при помощи команд, набираемых в интерфейсе (cli);

Как работает Docker

Работа Docker основана на принципах клиент-серверной архитектуры, которая основана на взаимодействии клиента с веб-сервером (хостом). Первый отправляет запросы на получение данных, а второй их предоставляет.

1) Пользователь отдает команду с помощью клиентского интерфейса Docker-демону, развернутому на Docker-хосте. Например, скачать готовый образ из реестра (хранилища Docker-образов) с помощью команды docker pull. Взаимодействие между клиентом и демоном обеспечивает REST API. Демон может использовать публичный (Docker Hub) или частный реестры.

2) Исходя из команды, заданной клиентом, демон выполняет различные операции с образами на основе инструкций, прописанных в файле Dockerfile. Например, производит их автоматическую сборку с помощью команды docker build.

3) Работа образа в контейнере. Например, запуск docker-image, посредством команды docker run или удаление контейнера через команду docker kill.

Docker-image — шаблон только для чтения (read-only) с набором некоторых инструкций, предназначенных для создания контейнера. Он состоит из слоев, которые Docker комбинирует в один образ при помощи вспомогательной файловой системы UnionFS. Так решается проблема нерационального использования дисковой памяти. Параметры образа определяются в Docker-file.

Для многократного применения Docker-image следует пользоваться реестром образов или Докер-реестром (Docker-registry), позволяющим закачивать готовые образы с внешнего репозитория сервиса и хранить их в реестре Докер-хоста. Рекомендуемый вариант — официальный реестр компании Docker Trusted Registry (DTR).

Если требуется файл, то скачиваться будут только нужные слои. Например, разработчик решил доработать программное обеспечение и модифицировать образ, изменив несколько файлов. После загрузки на сервер будут отправлены слои, содержащие только модифицированные данные.

Каждый контейнер строится на основе Docker-образов. Контейнеры запускаются напрямую из ядра операционной системы Linux. Благодаря этому, они потребляют гораздо меньше ресурсов, чем при аппаратной виртуализации.

Изоляция рабочей среды осуществляется при помощи технологии namespace. Для каждого изолированного пространства (контейнера) создается уникальное пространство имен, которое и обеспечивает к нему доступ. Любой процесс, выполняемый внутри контейнера, ограничивается namespace.

В ОС Linux посредством Docker Engine используется немного другая технология — контрольные группы (cgroups). При этом приложение ограничивается некоторым набором ресурсов. Cgroups осуществляют обмен доступных аппаратных ресурсов с контейнерами, на которые дополнительно устанавливаются необходимые ограничения (использование памяти, прав доступа к другому ресурсу и т. д.).

Движок Docker объединяет пространство имен (namespace), контрольные группы (cgroups) и файловую систему (UnionFS) в формат контейнера. В будущем планируется поддержка других форматов посредством интеграции технологий BSD Jails или Solaris Zones.

Запуск контейнера

Происходит запуск образа (Docker-image). Docker Engine проверяет существование образа. Если образ уже существует локально, Docker

использует его для нового контейнера. При его отсутствии выполняется скачивание с Docker Hub.

- создание контейнера из образа;
- разметка файловой системы и добавление слоя для записи;
- создание сетевого интерфейса;
- поиск и присвоение IP-адреса;
- запуск указанного процесса;
- захват ввода/вывода приложения.

Для управления несколькими контейнерами, из которых состоит проект, используют пакетный менеджер - Docker Compose.

Он применяется не во всех случаях. Если проект является простым приложением, не требующим использования сторонних сервисов, то для его развертывания можно ограничиться только Docker. Docker Compose рекомендуется использовать при проектировании сложных программных продуктов, включающих в себя множество процессов и сервисов.

При преобразовании хостов в кластер нужно воспользоваться утилитой кластеризации Docker Swarm. Хост, находящийся в его составе, называется «узлом» (node), который бывает управляющим или рабочим. Один кластер содержит только один управляющий «узел».

- управление нагрузочными характеристиками - осуществляется оптимизация рассылки запросов между хостами, обеспечивая на них равномерную нагрузку;
- динамическое управление - допускается добавление элементов в swarm-кластер без дальнейшего его перезапуска;
- возможность масштабирования - позволяет добавлять или удалять docker - образ для автоматического создания контейнера;
- восстановление «узла» после сбоя - работоспособность каждого хоста постоянно контролируется управляющим «узлом». При сбое кластера происходит его восстановление и перезапуск;
- rolling-update — выполняет обновление контейнеров. Процедура может выполняться в определенной последовательности и с временной задержкой для запуска другого контейнера. Параметр указывается в настройках. Если произойдет сбой обновления, то Docker Swarm выдаст ошибку и процесс повторится заново;

Окружение для разработки Docker применяется во множестве сфер - от обработки больших массивов данных, до работы с микросервисами, основанных на распределенной архитектуре.

Чтобы понять, как можно применять Докер на практике, разберем основные примеры использования для чайников.

- быстрая доставка приложений (команды docker pull и docker push) позволяет организовать коллективную работу над проектом. Разработчики могут работать удаленно на локальных компьютерах и выполнять пересылку фрагментов кода в контейнер для тестов;

- развертывание и масштабирование - контейнеры работоспособны на локальных компьютерах, серверах, в облачных онлайн-сервисах. Их можно загружать на хостинг для дальнейшего тестирования, создавать (docker run), останавливать (docker stop), запускать (docker start), приостанавливать и возобновлять (docker pause и docker unpause соответственно);

- множественные нагрузки - осуществление запуска большого количества контейнеров на одном и том же оборудовании, поскольку Docker занимает небольшой объем дисковой памяти;

- диспетчер процессов - возможность мониторинга процессов в Docker посредством команд docker ps и docker top, имеющими схожий синтаксис с Linux;

- удобный поиск - в реестрах Docker он осуществляется очень просто. Для этого следует использовать команду docker search.

1.4 Сравнительный анализ аналогов

В Казахстане есть несколько сайтов предлагающих услуги учреждениям. Одним из ярких представителей является Platonus. Его использую в университетах, таких как АУЭС, КазНУ, ЕНУ и многие другие.

В целом критерием для такого сайта с серверной части является:

- современные методы обработки и хранением данных;
- поддержка большого количества одновременных пользователей;
- обеспечение гибкой функциональности для сайта.

И Platonus соответствует им, но за каждую функцию или дополнительное место нужно оплачивать дополнительно. Так же стабильность и защита сайта не всегда работают как надо.

Дипломный проект Herodotus в свою же очередь будет располагаться внутри учреждений что позволит обезопасить и стабилизировать работу сайта, снизив время отклика. Благодаря базе, созданной через Postgres, будет легче создавать запросы для новых функций. А развитие языка Java и объединением его с разными компонентами дает не ограниченные функции, которые не сильно загружают сайт.

2 Проектная часть

2.1 Создание проекта

Установка программы intellij idea. После скачивания дистрибутива с официального сайта jetbrains.com, выйдет окно выбора папки куда установится программа.

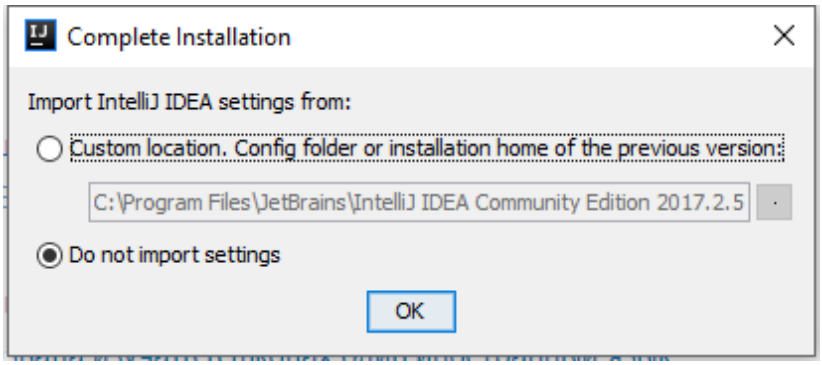


Рисунок 2.1 - Место установки программы

Далее идет выбор темы. По умолчанию их 2.

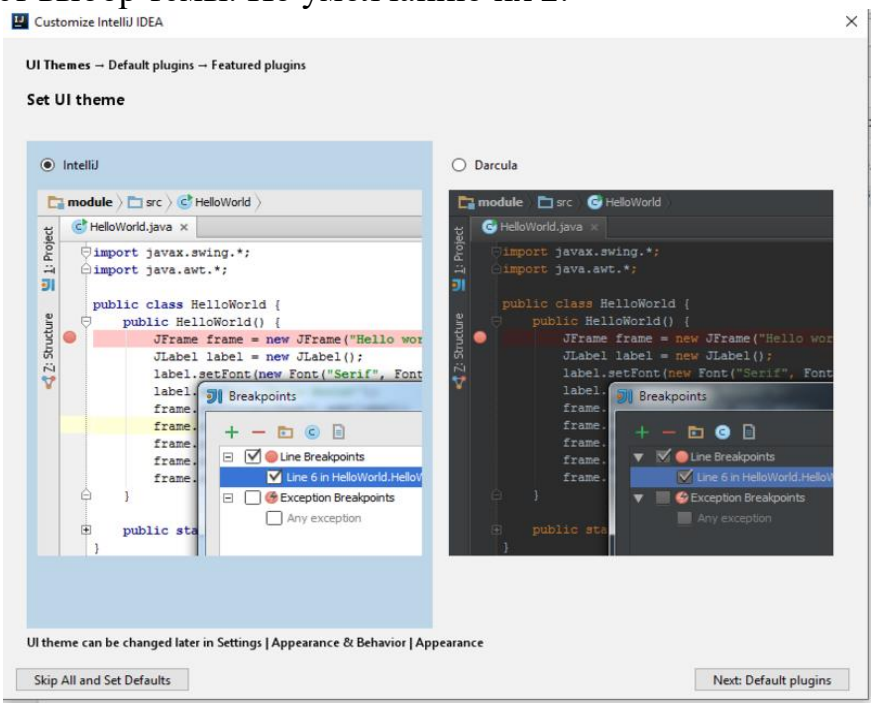


Рисунок 2.2 - Тема программы

После этого будет предложено меню вспомогательных инструментов. Их можно отключить и позже подключить те что необходимы.

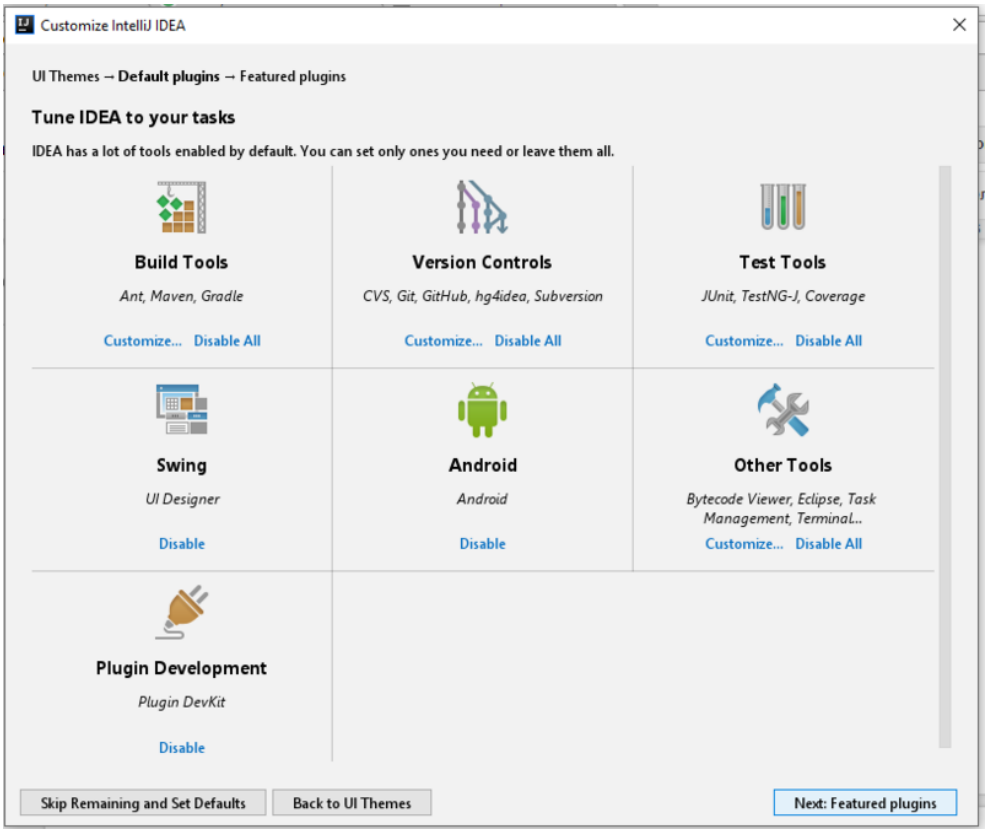


Рисунок 2.3 - Дополнительные инструменты

И в конце установи программа предлагает создать, импортировать или открыть папку с проектом.

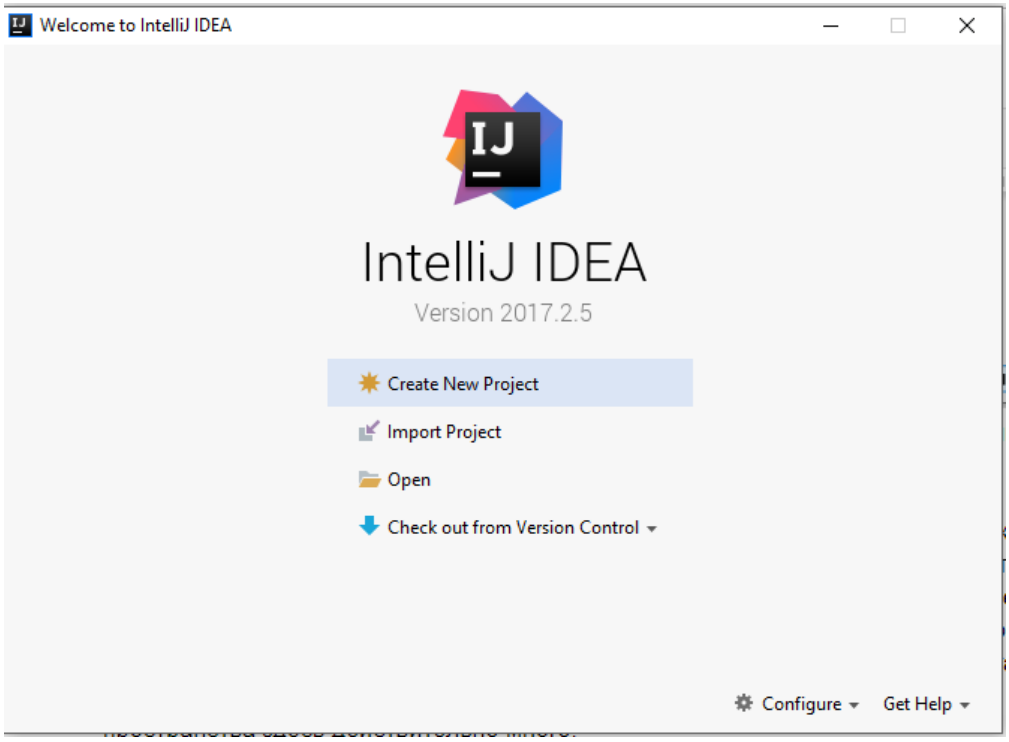


Рисунок 2.4 - Управление проектом

После установки intellij idea, нужно выбрать язык программирования, в проекте выбрана java.

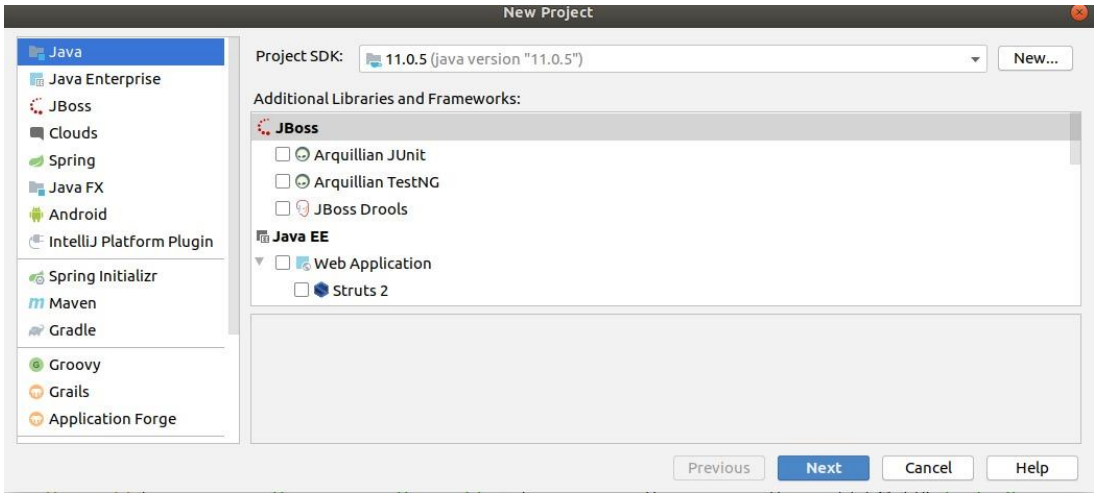


Рисунок 2.5 - Выбор языка программирования

Далее идет выбор шаблона.

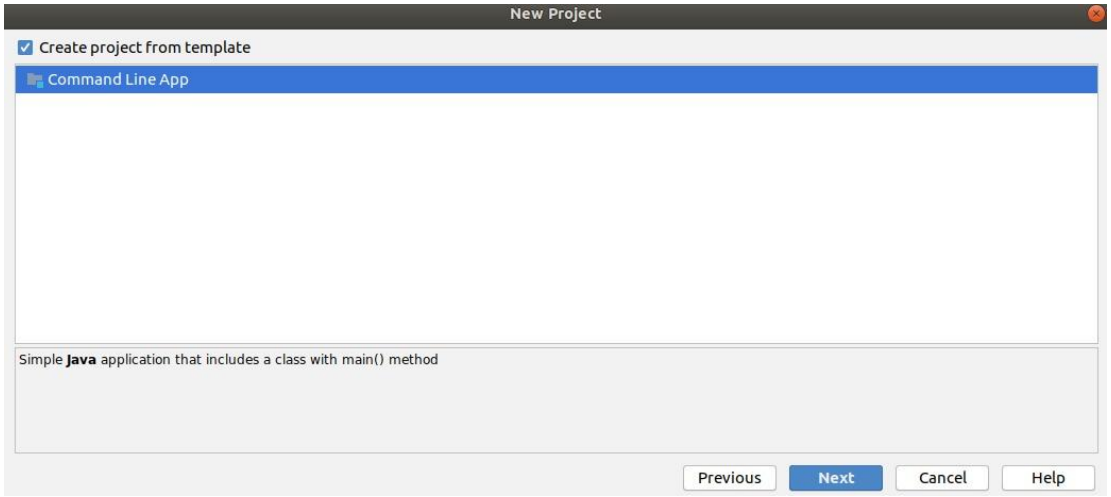


Рисунок 2.6 - Шаблон

Следующим шагом идет название проекта, его место сохранение и название внутренней папки хранения.

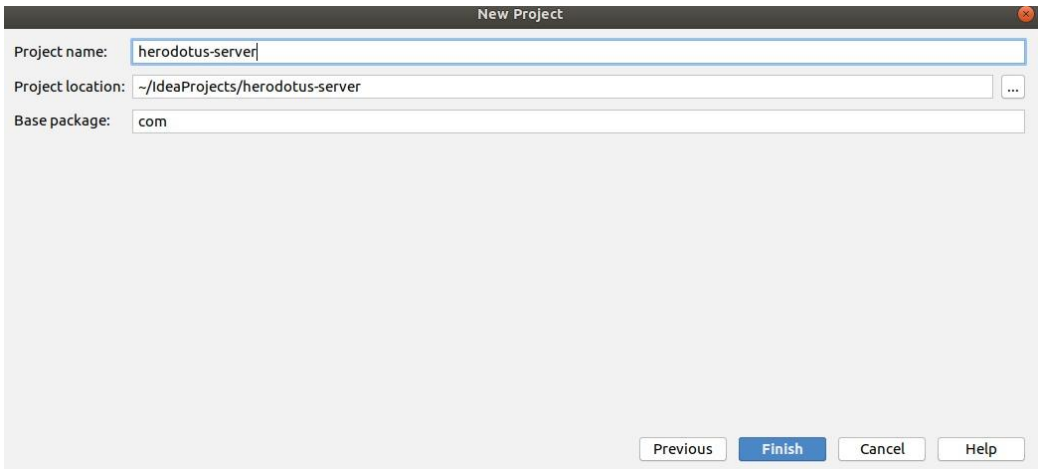


Рисунок 2.7 Название и место хранение проекта

После всех шагов откроется проект, для которого необходимо настроить подключение к серверу с базами данных.

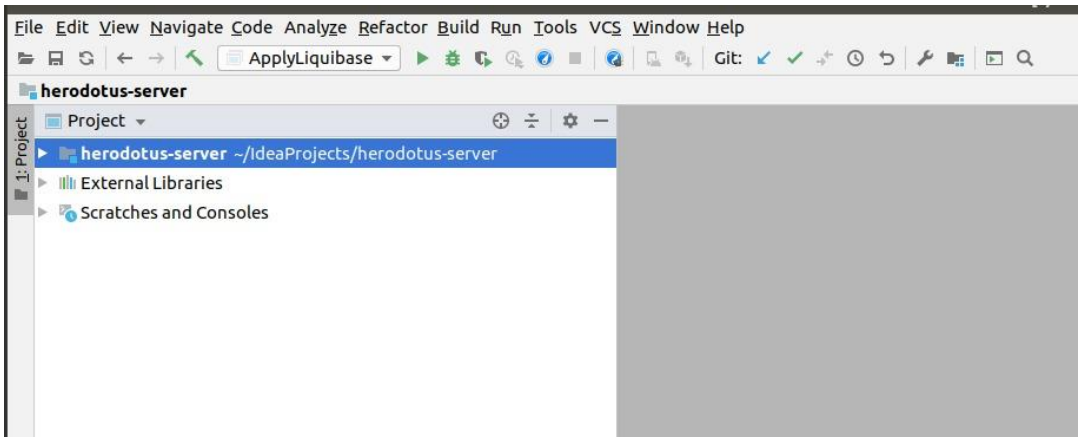


Рисунок 2.8 - Проект

Настройка сервера происходит в 2 этапа: ввод данных: название хоста, пользователя, выбор порта, пароль, название базы данных и url.

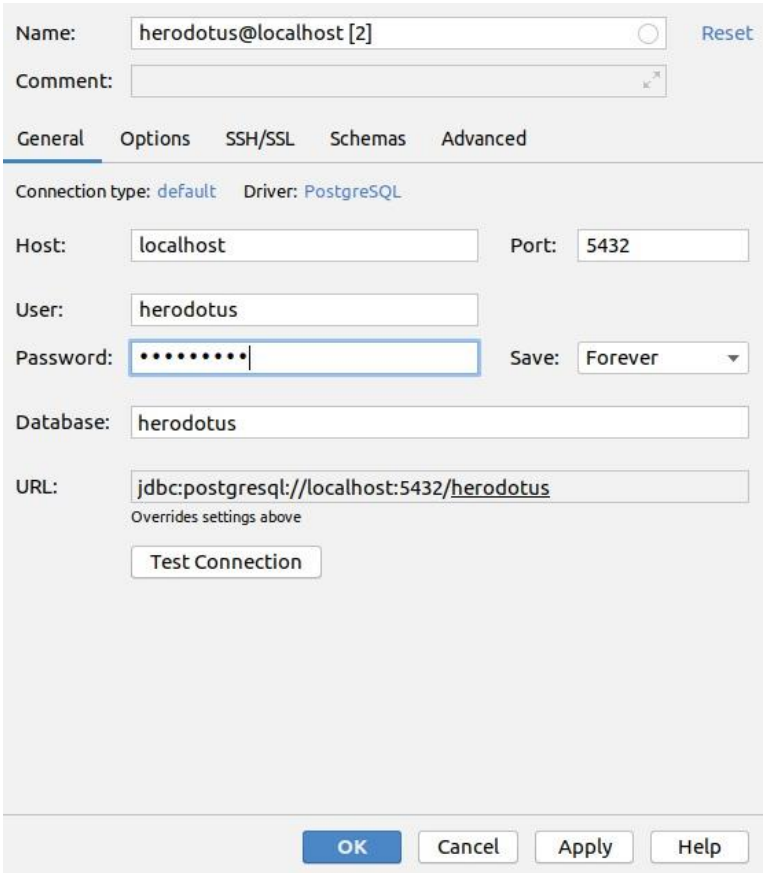


Рисунок 2.9 - Настройка подключения к серверу

Далее введённые данные проверяются.

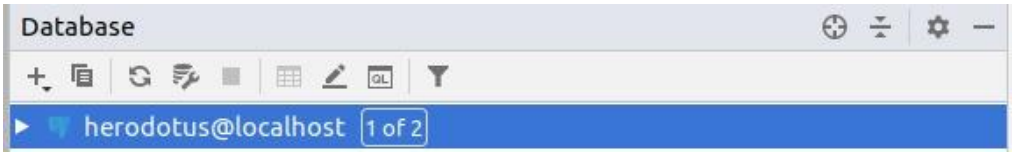


Рисунок 2.10 - проверка данных

Создание класса main

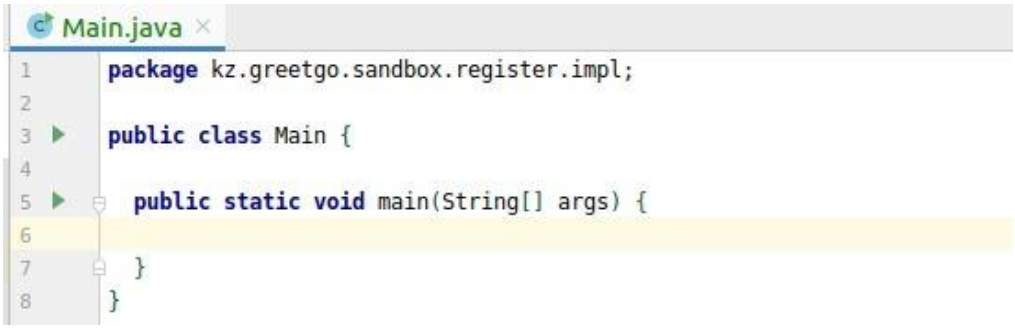


Рисунок 2.11 - Класс main

После этого необходимо подключить отладчик, он позволяет управлять проектом, находить ошибки и дает возможность добавления сторонних библиотек на сервер.

DebugServer.java

```
1  package kz.greetgo.sandbox.debug_server.launcher;
2
3  import ...
17
18 @EnableCaching
19 @SpringBootApplication
20 @Import(BeanConfigForDebugServer.class)
21 public class DebugServer {
22
23   @Autowired
24   private ApplicationFinisher applicationFinisher;
25
26   @Autowired
27   private ShowConfigValuesOnStartup showConfigValuesOnStartup;
28
29   @Autowired
30   private SchedulerManager schedulerManager;
31
32   @Autowired
33   private SlowControllerAspect slowControllerAspect;
34
35   @PostConstruct
36   public void init() throws Exception {
37     showConfigValuesOnStartup.show();
38     schedulerManager.start();
39     slowControllerAspect.init();
40   }
41
42   public static void main(String[] args) {
43     LaunchLogoGreetgo.print(System.out);
44     SpringApplication.run(DebugServer.class, args);
45   }
46
47   @PreDestroy
48   public void finishApplication() {
49     System.out.println("4b0kVJTtHn :: finishApplication");
50     applicationFinisher.finishApplication();
51   }
52
53 }
54
```

Рисунок 2.12 - Отладчик Debug

После первого запуска программы будет предложено выбрать набор средств установки sdk. Последняя версия 11.0.5.

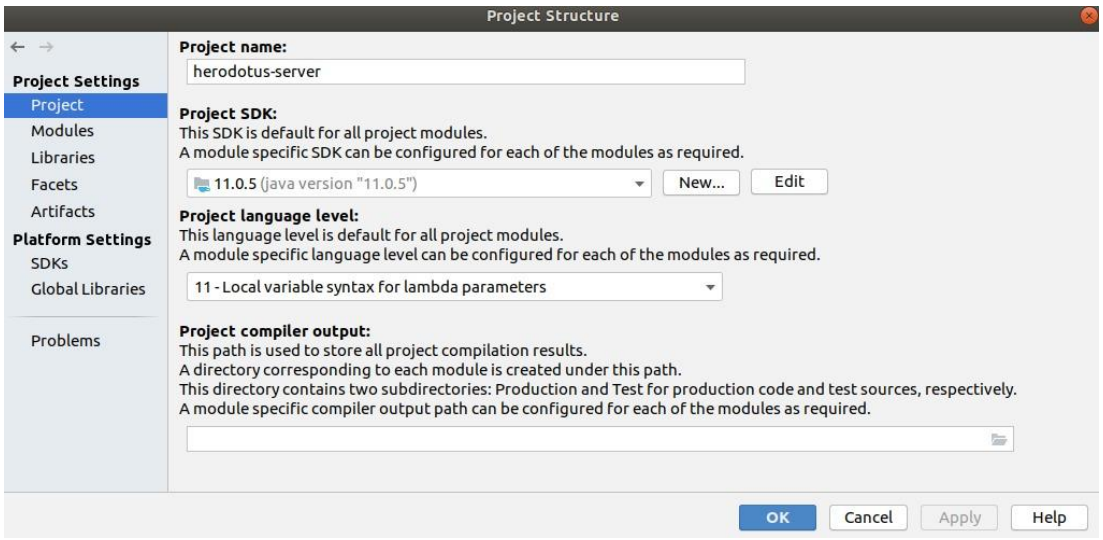


Рисунок 2.12 - Выбор sdk

2.2 UML диаграмма

В этом подразделе будут представлены диаграммы деятельности UML.

На рисунке 2.1 представлен процесс входа пользователя под определенной ролью. Пользователь заходит на сайт, идет авторизация и проверка роли пользователя на сайте.

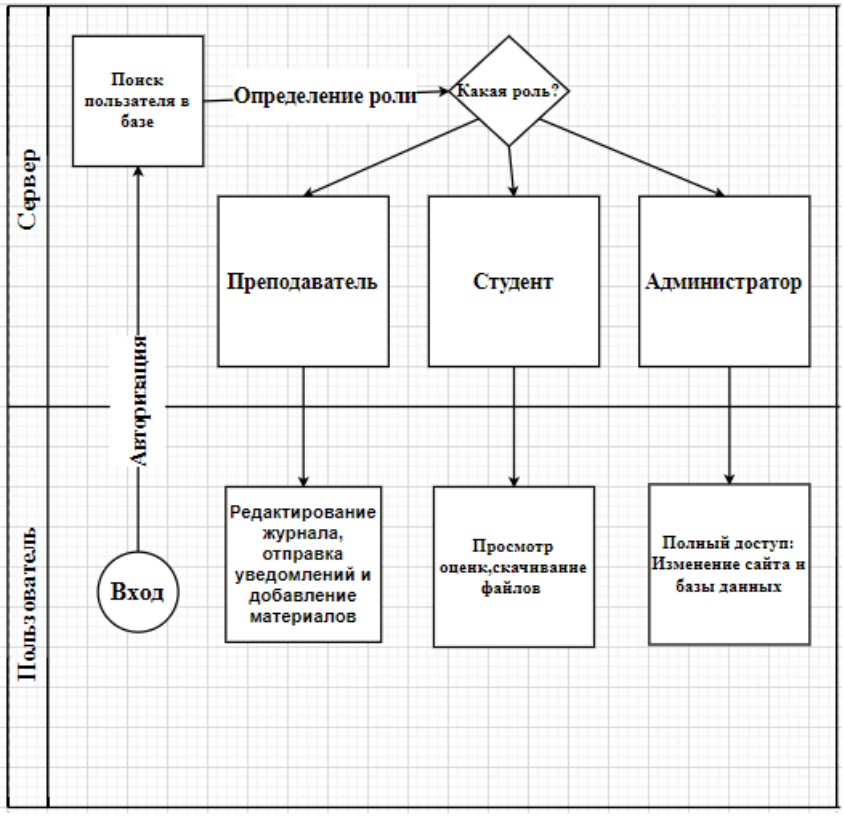


Рисунок 2.13 - Схема ролей на сайте

На рисунке 2.2 представлена диаграмма sql запросов базы данных

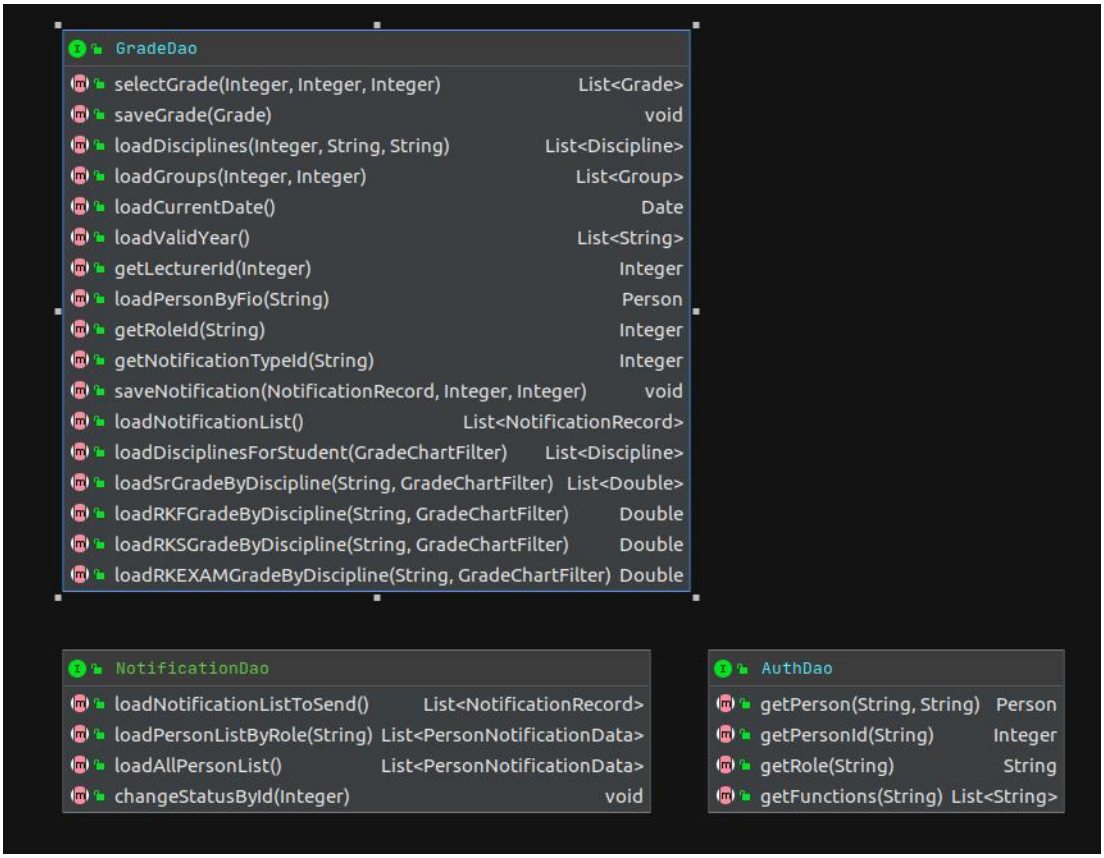


Рисунок 2.14 - Запросы базы данных

На рисунке 2.3 представлена диаграмма ролей

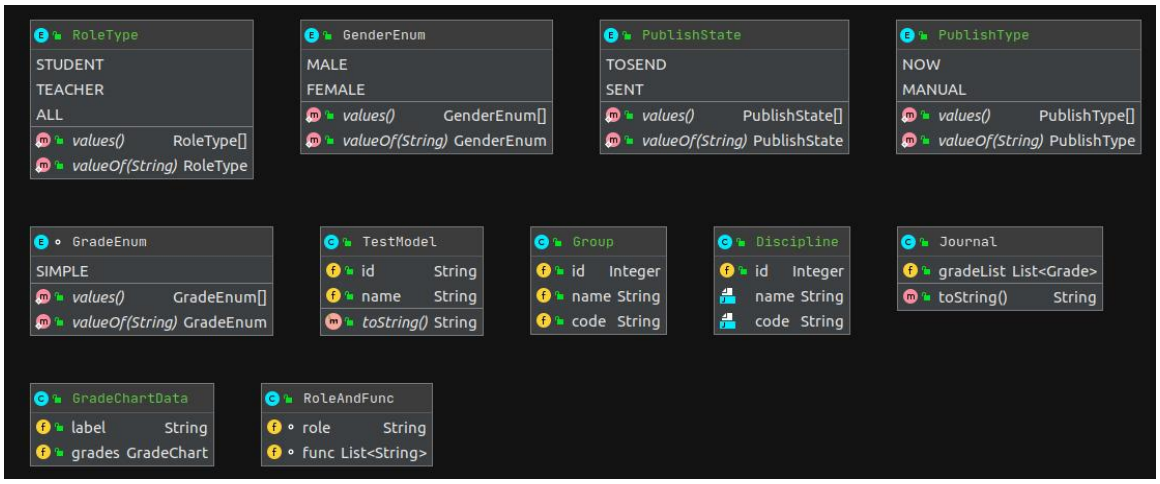


Рисунок 2.15 - Роли

На рисунках 2.4-2.7 представлены диаграммы связей между таблицами.

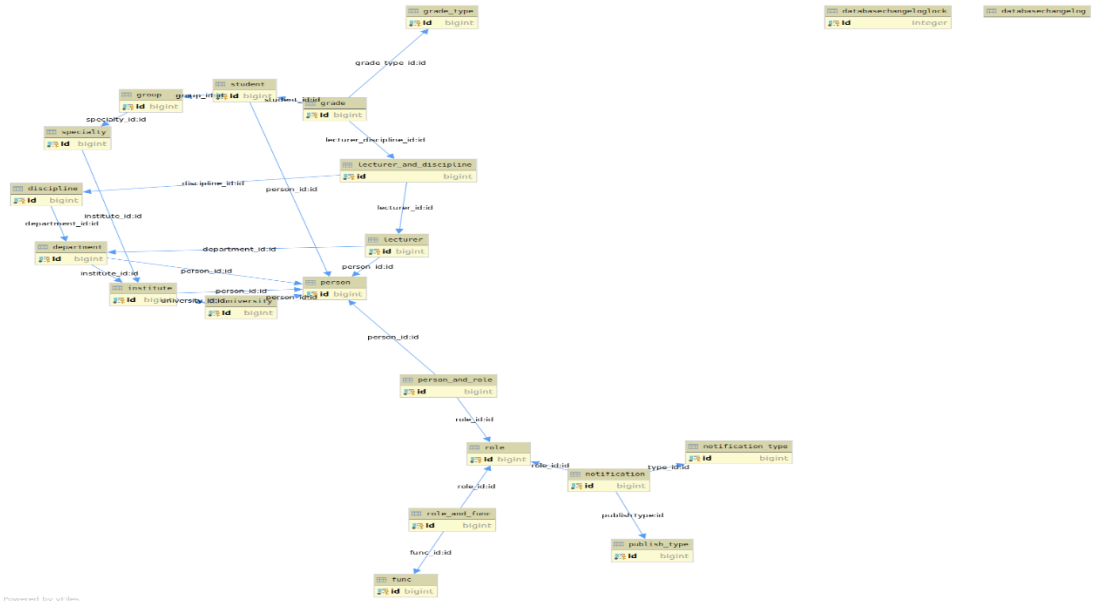


Рисунок 2.16 - Диаграмма связей между таблицами

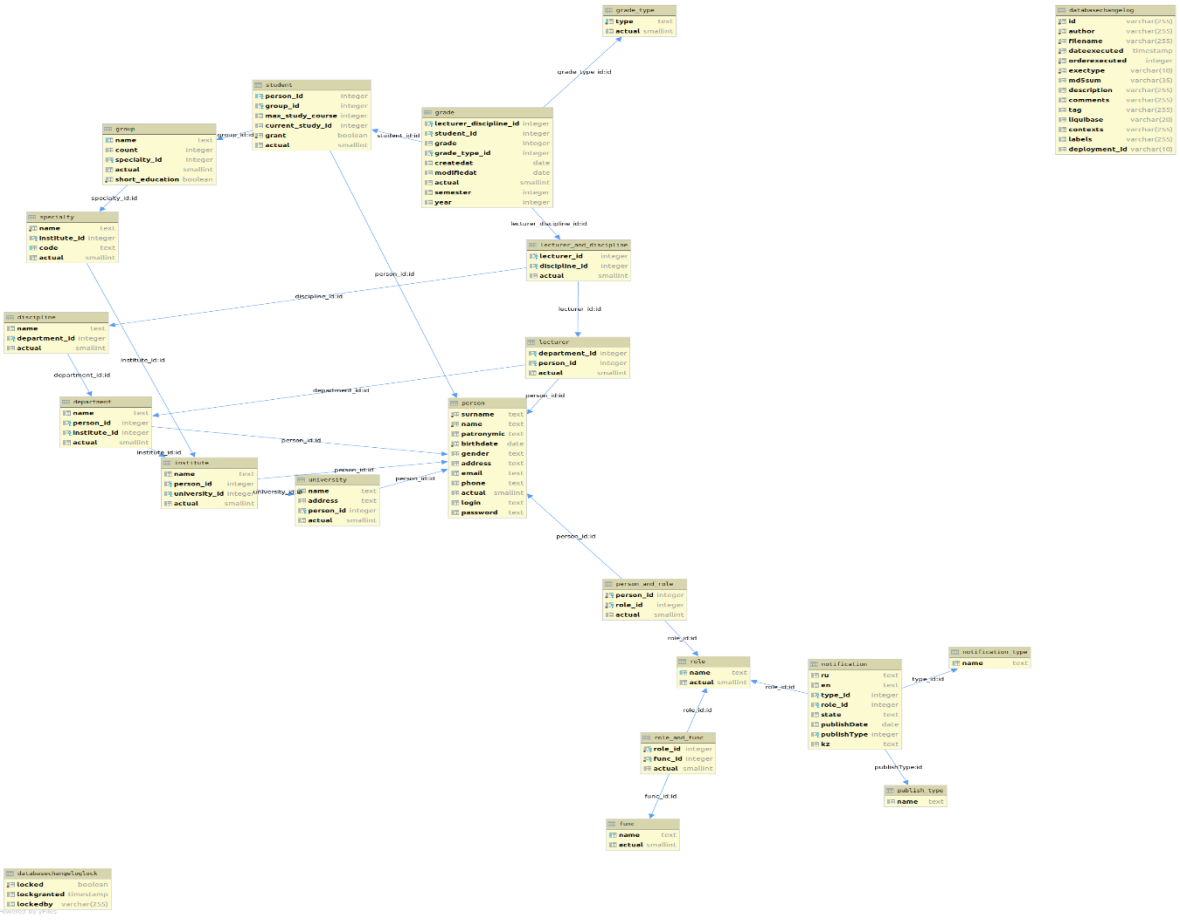


Рисунок 2.17 - Диаграмма связей таблицами и столбцами

2.3 Программная часть

Отправной точкой процесса проектирования информационной системы является построение исходной модели рассматриваемой системы и

используемых в настоящее время информационных систем. Эти модели являются источником извлечения метрических характеристик из начальных требований и ограничений в будущем ИС.

По результатам моделирования определяется, какие задачи и требования могут быть выполнены на основе реализации того или иного варианта архитектуры вычислительных комплексов и телекоммуникаций. Более того, определить, какие изменения потребуются в структуре информационной системы (если она существует) и распределении прикладных задач, подлежащих выполнению, а затем сформировать соответствующие требования для новой информационной системы.

Вспомогательная модель каркаса - это главное, что нужно сделать на начальном этапе структурирования каркаса данных. В то же время, взаимосвязи между компонентами и их основными моментами (параметры торговой марки, техника, препятствия и критерии) изучаются и формируются. Из-за воздействия внешних воздействий особое внимание уделяется связи между различными предметами и статьями, которые являются вертикальными (высшие ассоциации, социальные структуры и т. Д.), Которые имеют прогрессивную структуру подчинения и даже. Доступны реляционные и межгрупповые отношения. Базовая модель объединения зависит от стандартов репрезентативной картины. Каждый компонент каждого класса относится к определенному персонажу. Наиболее важным моментом здесь является решение о важности компонентов структуры. Такое позиционирование позволяет расшифровать субъективное понимание центральности в количественную структуру, чтобы соединиться с помощью жизненно важных квалифицирующих компонентов исследуемой или созданной структуры.

Каждая сетевая модель имеет значение для распознавания изображения базы данных и самой базы данных. Смысл базы данных мы называем шаблоном базы данных, который является явным в плане базы данных, и там вы не можете преобразовывать его время от времени. В большинстве информационных моделей есть отображения, отображающие в виде контуров.

Системная модель, известная как сортировка множеств, указывает информацию в виде типов записей и, кроме того, демонстрирует конкретный оценочный вид отношения 1: N. Один пример записи для многих конструкций записей, использующих в этих моделях некоторую систему соединения указателей, идентифицирует отношение A 1: N или «один ко многим».

Хорошая основанная и структурированная структура данных опирается на осведомленное учреждение, которое лежит в основе гибких изменений. Он используется в качестве основы структуры данных, систем вещательной связи и информации, используемой руководителями. В условиях глобализации бизнеса фонд ассоциации регулярно пересекает многочисленные национальные границы. Фонд и правление структуры начинают изменения, слияния и поглощения. Основу структуры данных следует создать так, чтобы сделать важный выбор для будущего корпоративного улучшения.

Разработка структуры данных характеризует эту структуру в отношении частей и связей между этими сегментами с точки зрения явных частей этой структуры и зависит от явных организационных стандартов. На рисунке 2.3 показан MVC фреймворка.

Это зависит от частей, которые включают сегменты и их взаимосвязи, так же, как свойства этих сегментов.

Каркас двух маршрутов переписывается со всем вокруг него. Компонент обычно имеет меньшее значение при использовании для создания программ. В таком сегменте определенные рекомендации выполняются и, кроме того, имеют определенный интерфейс. Не пропустите, что это указывает на различные структуры и компоненты техники. Значение частей, которые показывают, как они действуют на сегменты, когда они связаны друг с другом в ограниченных автоматах.

В UML граф случая использования - это динамическая диаграмма или диаграмма поведения. Использование коллаборатороов, учителей, ассоциаций, а также менеджера и других вариантов использования демонстрирует полезность диаграмм вариантов использования для просмотра веб-сайта. Возможности, что структура должна быть сделана и много действий, администрация, варианты использования. Почему UCD был сделан? Диаграмма вариантов использования полезна для расшифровки полезных обязательств структуры, которая преобразуется в альтернативы структуры и потребности проектирования. Они также помогают обнаружить любые внутренние или внешние компоненты, которые могут повлиять на структуру и должны быть приняты к сведению. Они дают идеальное аномальное состояние расследования извне. Диаграммы прецедентов показывают, как сайт платформы запросов взаимодействует с правой рукой, преподавателем, ассоциациями и менеджером, не мучаясь при этом из-за тонкостей выполнения этой полезности. На рисунке 2.8 вы можете заявить, что вы демонстрируете полезность.

Мы записали информацию, необходимую для базы данных, ниже, и после этого сделали свой прикладной проект, когда представляем идеи отображения модели ER. База данных поисковой системы Research Assistant отслеживает ассоциации системы поиска, их новости, открытые возможности, людей (преподавателей и партнеров), веб-сайты и мероприятия. Предположим, что после этапа сбора предварительных условий и исследования мы дали сопроводительное описание мини-мира - части структуры, о которой будет рассказано в базе данных.

1) На нашем сайте есть краткое изложение ассоциаций. Каждая ассоциация имеет уникальный идентификатор, название ассоциации, изображение, логотип, сайт, изображения. Мы отслеживаем количество представителей и преподавателей, когда ассоциация начала заниматься своим профилем на нашем сайте. Ассоциация может иметь несколько межличностных организаций.

2) Мы храним идентификатор каждого человека, его имя, логин, должность, рабочую силу, сайт, фотографию, связи, электронную почту, номер телефона, работу и последний наблюдаемый. Человек отводится в одну ассоциацию, но может иметь дело с несколькими видами деятельности, которые на самом деле не ограничены подобной ассоциацией. Человек также может опубликовать просмотр статей.

Система Spring, или, по сути, Spring, является одной из самых понятных структур для структурных веб-приложений на Java. Структура - это что-то вроде библиотеки, но есть нечто определенное. Используя библиотеку, вы в основном создаете объекты классов, которые в ней находятся, называете стратегии, которые вам нужны, и таким образом получаете результат, который вам нужен. То есть, существует прогрессивно базовая методология: вы явно демонстрируете в своей структуре, в какую конкретную минуту вы должны сделать какой объект, в какую минуту вызываете конкретную технику и т.д. Со структурами вещи незначительно необычны. Вы едва делаете часть классов владения, регистрируете там какое-то обоснование, и делаете объекты классов и рассматриваете стратегии для вас на данный момент структуру. Регулярно ваши классы выполняют несколько интерфейсов из структуры или получают несколько классов из нее, следовательно, получают часть полезности, официально составленной для вас. Как бы то ни было, не совсем так.

Система контроллера просмотра веб-страниц Spring (MVC) построена вокруг DispatcherServlet, который отправляет запросы обработчикам, с настраиваемыми сопоставлениями обработчиков, см. Цели, поддержку регионов для передачи записей. Обработчик по умолчанию зависит от комментариев @Controller и @RequestMapping, предлагая широкий диапазон адаптируемой работы со стратегиями. В представлении Spring элемент управления @Control дополнительно позволяет создавать успокаивающие сайты и приложения с помощью комментария @PathVariable и различных основных моментов.

В Spring Web MVC вы можете использовать любой элемент в качестве объекта направления или структуры; вам не нужно выполнять системный явный интерфейс или базовый класс. Ограничение информации Spring исключительно адаптируемо: например, он рассматривает перепутывания типов как ошибки утверждения, которые могут быть оценены приложением, а не как грубые ошибки в работе инфраструктуры. Таким образом, вам не нужно копировать свойства ваших бизнес-статей в виде базовых нетипизированных строк в вопросах структуры, по существу, чтобы иметь дело с недопустимыми записями или соответствующим образом изменять строки.

В качестве сервера приложений мы использовали Apache Tomcat.

Разработка на стороне сервера

Сервер это программный код, который «разворачивается» в фреймворке и настраивается на порты. Дайте нам возможность рассмотреть 2 варианта

того, что сервер может сделать любым способом: он может дать информацию (GET) - просто вернуть число, страницу или что-то еще. В любом случае, есть дополнительно POST - он также возвращает информацию, но, кроме того, получает ее от клиента до нее.

Объект управления на стороне сервера определяет и формирует тонкости пользовательского интерфейса на стороне клиента для отображения в нашей среде. Кучи элементов управления на стороне сервера могут быть объединены в организацию протестов управления на стороне сервера, которые существуют для планирования отображения кода языка, например HTML, для отображения в нашей среде на клиенте. Средства того, как функционирует компонент пользовательского интерфейса на стороне клиента, во всяком случае, один из случаев, связанных с активностью, информацией обратной передачи, отвечающей за активность, задачей ограничения информации и состоянием активности форума. Состояние задачи руководителей ссылается на состояние объекта управления на стороне сервера.

Во времена компьютерных инноваций существует множество организаций, занимающихся совершенствованием программирования. Таким образом, каждая из этих организаций имеет различные группы улучшений, которые создают и тестируют создаваемый продукт. Чтобы улучшить способ создания и поддержки программ, гарантирующих, важно механизировать эти процедуры, что было основной причиной создания этого веб-приложения. Гарантия характера разряженного программирования - одно из основных заданий, стоящих перед продуктом бизнеса. На сегодняшний день существует значительное количество аппаратов подтверждения экстраординарного качества, которые изолированы по причине к сопровождающим:

- процесс разработки рамок для руководителей: задачи, следующие за рамками, и структуры для руководителей;
- следование дефектам и изменение структуры спроса;
- тестирование инструментов.

Созданное веб-приложение предназначено для отслеживания созданного программирования.

Веб-приложение - это приложение клиент-сервер, в котором клиент - это программа, а сервер - веб-сервер. Обоснование веб-приложения передается между сервером и клиентом, информация в основном размещается на сервере, данные передаются по системе.

Веб-приложение актуализирует инновации клиент-сервер, поскольку состоит из клиентской и серверной частей.

Клиентская часть актуализирует пользовательский интерфейс, создает запросы к серверу и реагирует на него процедурами. Для продвижения клиентской части приложения используются такие технологии, как: JS, Ajax, AngularJS, CSS и HTML.

Серверная часть получает запрос от клиента, выполняет важные действия, в этот момент создает страницу веб-сайта и отправляет ее клиенту

по системе, используя соглашение HTTP. Для улучшения серверной части приложения были внедрены инновации, например, Spring MVC, Java, Sleep, Spring Security, Log4J, Tomcat и Derby.

Чтобы начать работу с этим приложением, достаточно заручиться поддержкой клиента, определить его права (например, дать права менеджера). Затем вы должны сделать задачу, округляя все основные данные для этого. Чтобы сделать обязательство, вы должны выбрать предприятие, которому будет поручено настоящее поручение, и заполнить различные поля, предоставленные клиенту (предмет, тип, потребность, цель задания). По поручению можно связаться с объединенным материалом - записями, фотографиями или замечаниями.

Клиент может включать, стирать и изменять выполненные обязательства и задания, изменять статус, например, с открытого на закрытое, включать новых участников венчурного бизнеса и контролировать размер доступа включенных клиентов. Кроме того, каждый клиент подходит к индивидуальному профилю, который показывает индивидуальные данные, движение предприятия, рабочие места на предприятиях и распределенные назначения для клиента. Любые изменения (например, изменения состояния в случае обязательств PolesGU 402) в приложении подписываются в журнале, который показывается всем клиентам, с целью, чтобы клиенты могли понять, что происходит.

Для работы с этим приложением нет веских причин вводить какое-либо программирование продукта, так как оно вполне может быть выполнено в любой программе. Соответственно, это предприятие добавляет глобализации, позволяя ассоциациям создавать границы прошлого и часовые пояса. С дальнейшим расширением этой задачи возможна интернационализация предприятия, чтобы упростить адаптацию клиентов различных стран к интерфейсу приложения.

Полезность выполненного приложения может быть расширена и улучшена в контексте до более высокого уровня, кроме того, возможно реализовать сотрудничество этого приложения с администрациями сторонних организаций, чтобы дополнительно расширить сущность данных, привлекательность и удобство приложения, в зависимости от обстоятельств. на потребности дизайнера.

Интернет-браузеры взаимодействуют с веб-серверами, используя соглашение о передаче гипертекста (HTTP). Когда вы нажимаете на соединение на странице, округляете структуру или начинаете преследование, запрос HTTP отправляется из вашей программы на целевой сервер. Запрос включает в себя URL-адрес, который распознает актив, на который влияют, метод, который определяет идеальную деятельность (например, получение, удаление или распространение актива) и может включать дополнительные данные, закодированные в параметрах URL (наборы оценки поля, отправленные в виде строки запроса), как POST-запрос (информация отправляется в HTTP-позиции POST), или как обрабатывается.

Веб-серверы ожидают сообщения с требованиями клиентов, обрабатывают их при посадке и реагируют на интернет-браузер с помощью сообщения HTTP-реакции. Реакция содержит строку состояния, показывающую, было ли ходатайство плодотворным или нет (например, «HTTP / 1.1 200 в порядке», если он эффективен). Тело плодотворной реакции на ходатайство может содержать упомянутую информацию (например, другую страницу HTML, или изображение и т. д.), которое можно показать с помощью интернет-браузера.

Программная часть:

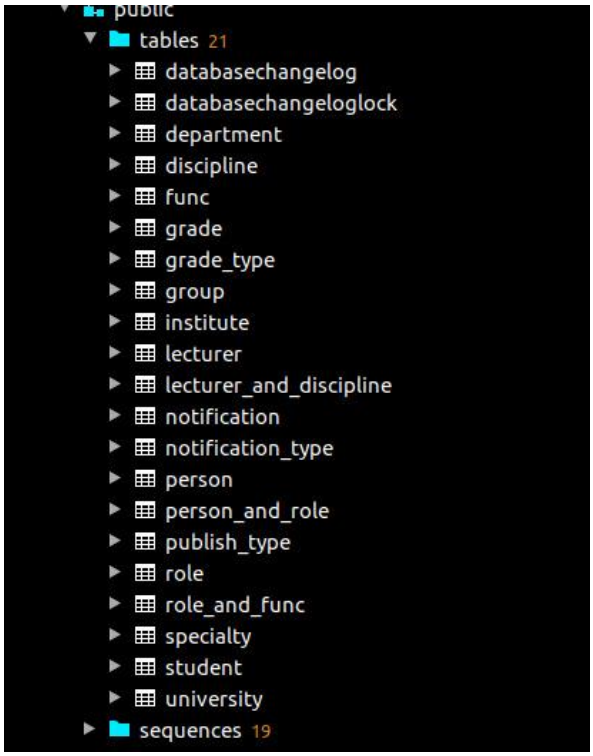


Рисунок 2.18 - Базы данных

Работа с уведомлениями:

Рисунок 2.19 - Отправка уведомлений

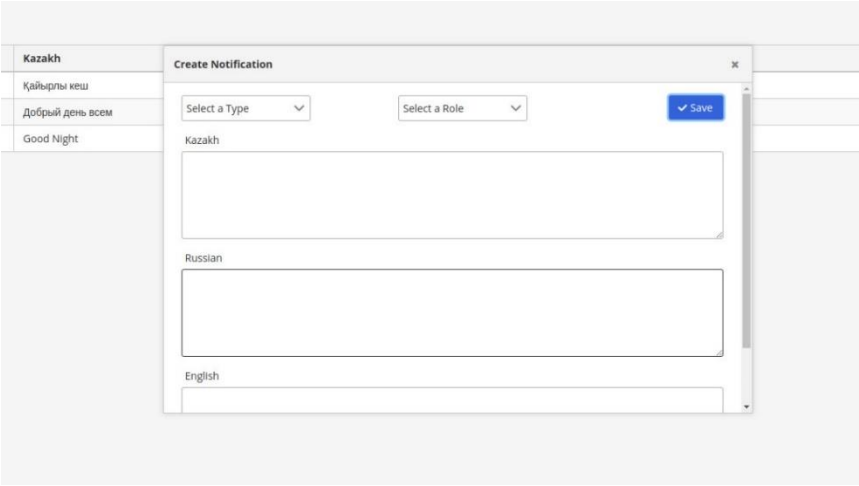


Рисунок 2.20 - Оформление на 3 языках

Планировщик, который отправляет каждую минуту неотправленные уведомления и меняет статус на отправлено

```
@Component
public class NotificationScheduler implements HasScheduled {
  @Autowired
  NotificationRegister notificationRegister;
  @Scheduled("repeat every 1 minute")
  public void sendNotification() {
    notificationRegister.sendNotification();
  }
}
```

Рисунок 2.21 - Планировщик

Таблица на рисунке 2.22 относится к библиотеке liquidbase предназначена для регистрации изменений в базе данных.

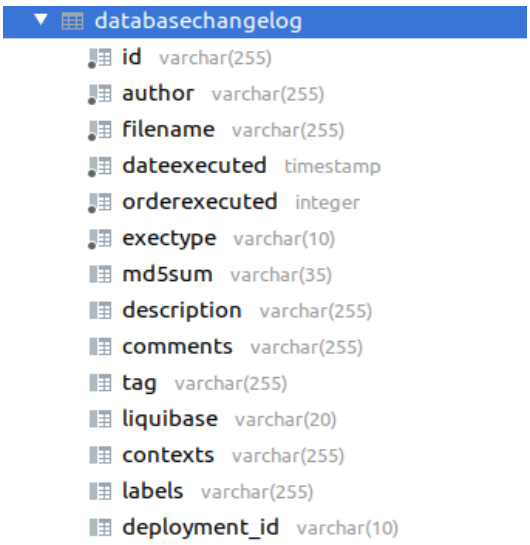


Рисунок 2.22 - Регистрация изменений в базе

Таблица на рисунке 2.23 относится к библиотеке liquidbase предназначена для регулирования нескольких изменений в одно время.

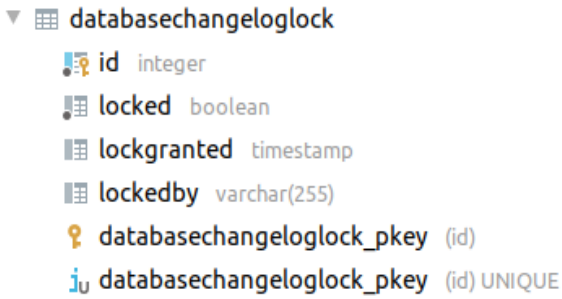


Рисунок 2.23 - Регулирование нескольких изменений

Таблица на рисунке 2.24 отражает сущность кафедры учебного заведения

Рисунок 2.24 - Сущность кафедры

Таблица на рисунке 2.25 отражает сущность дисциплины по которой обучаются обучающиеся.

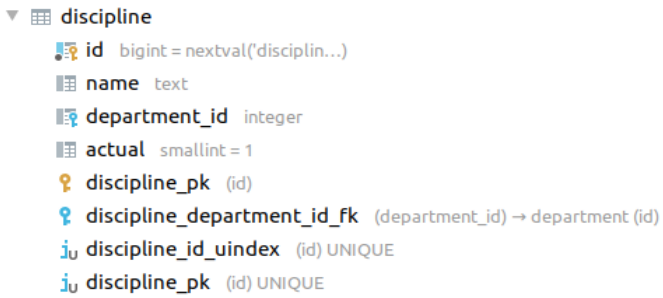


Рисунок 2.25 - Сущность дисциплины

Таблица на рисунке 2.26 отражает сущность функции которыми оперирует каждая отдельно взятая ролью.

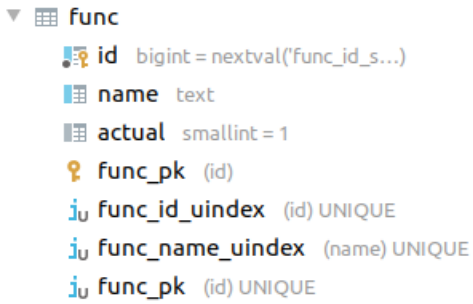


Рисунок 2.26 - Сущность функций отдельных ролей

Таблица на рисунке 2.27 представляет из себя все оценки по дисциплинам обучающихся за каждый конкретный период времени.

Рисунок 2.27 - Оценки за определенный период

Таблица на рисунке 2.28 определяет тип балла, который получил обучающийся.

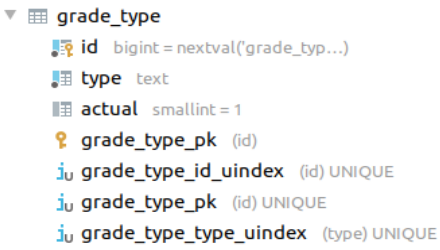


Рисунок 2.28 - Тип балла

Таблица на рисунке 2.29 представляет из себя группу в которой обучаются обучающиеся.

Рисунок 2.29 - Группа студентов

Таблица на рисунке 2.30 представляется из себя институт как подраздел университета.

Рисунок 2.30 - Институт

Таблица на рисунке 2.31 представляется из себя сущность преподавателей, которые имеют характерные черты в отличие от пользователей.

Рисунок 2.31 - Сущность преподавателей

Таблица на рисунке 2.32 представляется из себя сущность связь преподаватели и дисциплина дабы избавиться от связей многие ко многим.

Рисунок 2.32 Связь преподавателя и дисциплины

Таблица на рисунке 2.33 представляется из себя сущность уведомлений, которые отправляются из системы.

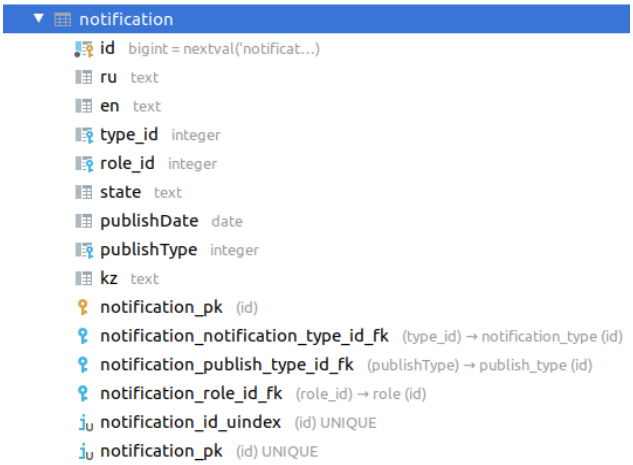


Рисунок 2.33 - Сущность уведомлений

Таблица на рисунке 2.34 представляется из себя сущность типов уведомлений для того чтобы понимать какой вид уведомлений нужно отправлять.

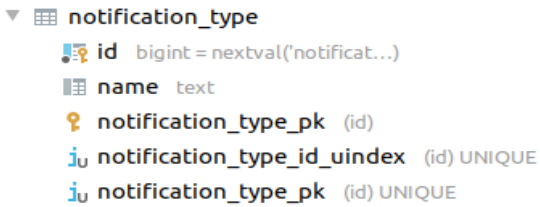


Рисунок 2.34 - Сущность типов уведомлений

На рисунке 2.35 главная таблица где хранятся пользователи.

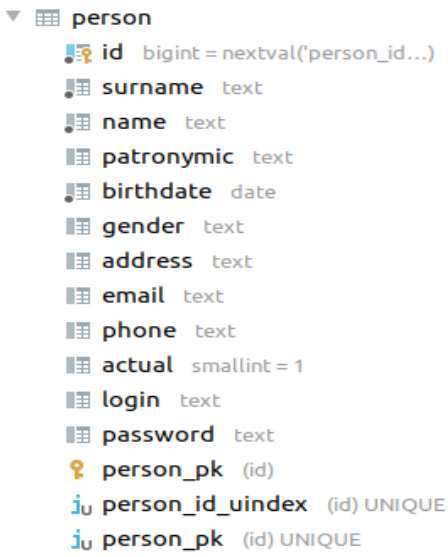


Рисунок 2.35 - Таблица пользователей

Таблица на рисунке 2.36 представляется из себя сущность связь пользователи и роли дабы избавиться от связей многие ко многим.

Рисунок 2.36 - Связь пользователей и ролей

Таблица на рисунке 2.37 представляется из себя сущность типов времени отправки для того чтобы понимать, когда уведомления нужно отправлять.

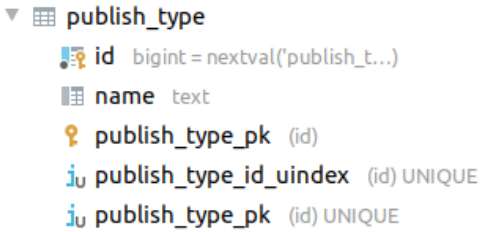


Рисунок 2.37 - Сущность типов времени отправки

Таблица на рисунке 2.38 представляется из себя сущность ролей для того чтобы понимать какой пользователь вошел в систему и какие роли он выполняет.

Рисунок 3.38 - Сущность ролей

Таблица на рисунке 2.39 представляется из себя сущность связь роли и функции дабы избавиться от связей многие ко многим.

Рисунок 3.39 - Сущность связи роли и функции

Таблица на рисунке 2.40 представляется из себя сущность специальность чтобы понимать по какой специальности учится обучающийся.

Рисунок 2.40 - Сущность специальностей

Таблица на рисунке 2.41 представляется из себя сущность студент которые имеют характерные черты в отличие от пользователей.

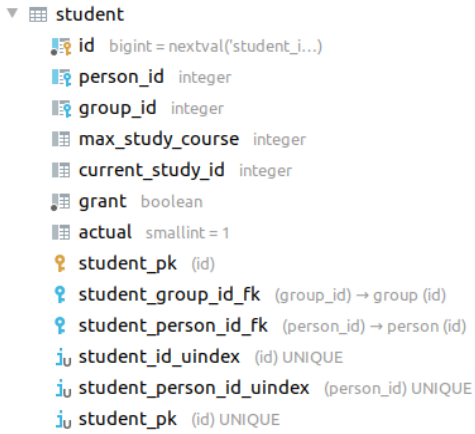


Рисунок 2.41 - Сущность студент

Таблица на рисунке 2.42 представляет из себя сущность университет, который содержит в себе все остальные единицы.

Рисунок 2.42 - Сущность университет

3 Экспериментальная часть

3.1 Запросы базы данных

Для начала необходимо проверить правильность обработки информации сервером для этого создал sql запросы:

```
@Select(" select gr.id    as id, " +
  "       (ps.surname || ' ' || SUBSTRING(ps.surname FROM 1 FOR 1) || '.' || SUBSTRING(ps.patronymic FROM 1 FOR 1) || " +
  "        '.')    as fio, " +
  "       d.name   as discipline, " +
  "       gr.grade as grade, " +
  "       g.name   as groupName, " +
  "       gt.type  as type " +
  " " +
  "from grade gr " +
  "       join lecturer_and_discipline lad " +
  "            on gr.lecturer_discipline_id = lad.id " +
  "       join lecturer l on lad.lecturer_id = l.id " +
  "       join person p on l.person_id = p.id " +
  "       join grade_type gt on gr.grade_type_id = gt.id " +
  "       join discipline d on lad.discipline_id = d.id " +
  "       join student s on gr.student_id = s.id " +
  "       join person ps on s.person_id = ps.id " +
  "       join public.group g on s.group_id = g.id " +
  "  where lecturer_id = #{lecturerId} " +
  "  and discipline_id = #{disciplineId} " +
  "  and g.id = #{groupId} " +
  "order by ps.surname, gr.id")
List<Grade> selectGrade(@Param("groupId") Integer groupId,
@Param("disciplineId") Integer disciplineId,
@Param("lecturerId") Integer lecturerId);

@Update("update grade set grade = #{grade.grade} where id = #{grade.id} ")
void saveGrade(@Param("grade") Grade grade);
```

Рисунок 3.1 - Запрос по дисциплинам

```
@Select("select d.id   as id,  " +
  " d.name as name,  " +
  " d.name as code  " +
  "from person  " +
  " join lecturer l on person.id = l.person_id  " +
  " join lecturer_and_discipline on l.id = lecturer_and_discipline.lecturer_id  " +
  " join discipline d on lecturer_and_discipline.discipline_id = d.id  " +
  " left join grade g on lecturer_and_discipline.id = g.lecturer_discipline_id  " +
  "where person.id = #{personGuid}  " +
  "  and year = ${year}  " +
  "  and semester = ${semester}  " +
  "group by d.id, d.name  ")
List<Discipline> loadDisciplines(@Param("personGuid") Integer personGuid,
@Param("year") String year,
@Param("semester") String semester);
```

Рисунок 3.2 - Запрос по дисциплине

```
@Select("select distinct g2.id as id, g2.name as name, g2.name as code " +
  "  from person p join lecturer  on p.id = lecturer.person_id " +
  "  join lecturer_and_discipline lad on lecturer.id = lad.lecturer_id " +
  "  join grade g on lad.id = g.lecturer_discipline_id " +
  "  join student s on g.student_id = s.id " +
  "  join public.group g2 on s.group_id = g2.id " +
  "  where p.id = #{personGuid} " +
  "  and lad.discipline_id = #{disciplineId} ")
List<Group> loadGroups(@Param("personGuid") Integer personGuid, @Param("disciplineId") Integer disciplineId);

@Select("select now();")
Date loadCurrentDate();

@Select("select year from grade group by year;")
List<String> loadValidYear();

@Select(" select id from lecturer where person_id = #{personGuid}")
Integer getLecturerId(@Param("personGuid") Integer personGuid);
```

Рисунок 3.3 - Запрос по дате

```
@Select("select * from person ps where (ps.surname ||" +
  " ' ' || SUBSTRING(ps.surname FROM 1 FOR 1) ||" +
  " '.' || SUBSTRING(ps.patronymic FROM 1 FOR 1) || '.') = #{fio} ")
Person loadPersonByFio(@Param("fio") String fio);

@Select("select id from role where name = #{role}")
Integer getRoleId(@Param("role") String role);

@Select("select id from notification_type where name = #{type}")
Integer getNotificationTypeId(@Param("type") String type);

@Insert(" insert into notification" +
  " (id, ru, en, kz, type_id, role_id, state, \"publishDate\", \"publishType\") " +
  "values (DEFAULT, #{notification.ru}, #{notification.en},#{notification.kz}, " +
  " #{typeId}," +
  "  #{roleId}, " +
  " 'TOSEND', now(), 1) ")
void saveNotification(@Param("notification") NotificationRecord notification,
@Param("typeId") Integer typeId,
@Param("roleId") Integer roleId);
@Select("select n.id, " +
  "       nt.name as type, " +
  "       kz, " +
  "       ru, " +
  "       en, " +
  "       r.name as role, " +
  "       pt.name as publishType, " +
  "       n.\"publishDate\" as publishDate, " +
  "       state " +
  "from notification n " +
  "       join publish_type pt on n.\"publishType\" = pt.id " +
  "       join notification_type nt on n.type_id = nt.id " +
  "       join role r on n.role_id = r.id")
List<NotificationRecord> loadNotificationList();
```

Рисунок 3.4 - Запрос оценок по всем дисциплинам студента

Рисунок 3.5 - Запрос по всем дисциплинам одной группы

Так же проверялась нагрузка при одновременной работе пользователей как со стороны студентов, так и со стороны преподавателей. Пока есть небольшая задержка при отправке смс при работе через twilio, но при отправке на почту проблем не наблюдалось.

4 Технико-экономическое обоснование проекта

В экономической части данного дипломного проекта будет приведено технико-экономическое обоснование разработки информационной системы для учебных заведений. Данная информационная система предназначена для управления серверной частью сайта. Основные цели создания такой информационной системы – обеспечение учебных заведений современной информационной системой, увеличение эффективности работы и администрировании системы, уменьшение материальных затрат на поддержание серверов и базы данных.

4.1 Расчет трудоемкости разработки ИС

Для расчета затрат на разработку ПО является объем программного продукта и используемые программы. Общий объем (VO) программного продукта определяется исходя из количества и объема функций, реализуемых программой:

$$V_0 = \sum_{i=1}^{n} V_i \quad (4.1)$$

где V_i - объем отдельной функции ПО;

n - общее число функций.

В экономической части дипломной работе в качестве единиц измерения объема ПО используется строка исходного кода (LOC).

Расчет объема ПП рассчитывается в количества строк кода и предполагает определение вида программного обеспечения, функциональностью проекта. На стадии технико-экономического обоснования проекта могут быть получены только примерные оценки на основе похожих проектов, существующих на рынке, или путем применения действующих нормативов.

$$V_y = \sum_{i=1}^{n} V_{yi} \quad (4.2)$$

где V_{yi} - уточненный объем отдельной функции ПО (LOC).

По уточненному объему ПО и нормативам затрат труда в расчете на единицу объема определяются нормативная и общая трудоемкость разработки ПО.

Для подсчета LOC использовалось расширение для среды разработки Intellij Idea. В соответствии с ним, проект состоит из 3 модулей, общее число строк всех этих компонентов – 4352.

Нормативная трудоемкость разработки ПО ($Т_н$) определяется на основании принятого к расчету объема (V_y) и категории сложности, которая уточняется с учетом сложности и новизны проекта и степени использования стандартных модулей при разработке.

Разрабатываемая информационная система относится ко 2 категории сложности, тогда нормативная трудоемкость её разработки составит $Т_н$=23 чел./дн.

Коэффициент сложности рассчитывается по формуле:

$$К_С = 1 + \sum_{i=1}^{n} K_i \quad (4.3)$$

где Ki - коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;

n - количество учитываемых характеристик.

Тогда, коэффициент сложности составит:

$$К_С = 1 + 0{,}06 + 0{,}07 + 0{,}12 = 1{,}25$$

Поправочный коэффициент, учитывающий степень использования при разработке ПО стандартных модулей ($К_m$), определяется удельным весом этих модулей в общем объеме проектируемого продукта (см. таблицу 2 [1]) $К_m$=0,8.

Поправочный коэффициент, учитывающий новизну разрабатываемого ПО ($К_Н$). Разрабатываемая ИС относится к Б-категории новизны ПО и является развитием определенного параметрического ряда ПО, не используется на основе новых типов ПК и в средах новых ОС. Значение коэффициента новизны соответствующее этим факторам равно $К_Н$=0,9.

Разработка информационной системы для управления проектированием реализуется одним исполнителем.

Для таких проектов рассчитывается общая трудоемкость с учетом стадий разработки:

$$Т_О = \sum_{i=1}^{n} T_i \quad (4.4)$$

где T_i – трудоемкость разработки ПО на i-й стадии (чел./дн.);

n – количество стадий разработки.

Трудоёмкость стадий определяется на основе нормативной трудоемкости с учетом сложности, новизны и степени использования в разработке стандартных модулей ПО и удельного веса трудоемкости каждой стадии:

$$Т_{уi} = Т_Н \times d_{cmi} \times К_С \times К_m \times К_Н$$

где $Т_{уi}$ – уточненная трудоемкость разработки ПО на i-й стадии;

$Т_Н$ – нормативная трудоемкость;

d_{cmi} – удельный вес трудоемкости i-й стадии разработки ПО в общей трудоемкости разработки ПО (определяется по таблице 4 [1] в зависимости от степени новизны);

$К_С$ – коэффициент, учитывающий сложность ПО, вводится на всех стадиях;

$К_m$ – коэффициент, учитывающий степень использования стандартных модулей ПО, вводится только на стадии рабочего проекта;

$К_Н$ – коэффициент, учитывающий степень новизны ПО, вводится на всех стадиях.

Трудоемкость стадий ПО рассчитывается по следующим формулам:

Трудоемкость стадии ТЗ:

$$Т_{уз}=T_H \times K_С \times d_з \times K_Н = 23 \cdot 1{,}25 \cdot 0{,}1 \cdot 0{,}9 = 2{,}58 \text{ чел/дн}$$

Трудоемкость стадии ЭП:

$$Т_{уэ}=T_H \times K_С \times d_э \times K_Н = 23 \cdot 1{,}25 \cdot 0{,}08 \cdot 0{,}9 = 2{,}07 \text{ чел/дн}$$

Трудоемкость стадии ТП:

$$Т_{ут}=T_H \times K_С \times d_р \times K_Н = 23 \cdot 1{,}25 \cdot 0{,}09 \cdot 0{,}9 = 2{,}32 \text{ чел/дн}$$

Трудоемкость стадии РП:

$$Т_{ур}=T_H \times K_С \times d_з \times K_Н \times K_m = 23 \cdot 1{,}25 \cdot 0{,}58 \cdot 0{,}9 \cdot 0{,}8 = 12{,}006 \text{ чел/дн}$$

Трудоемкость стадии ВН:

$$Т_{ув}=T_H \times K_С \times d_в \times K_Н = 23 \cdot 1{,}25 \cdot 0{,}15 \cdot 0{,}9 = 3{,}88 \text{ чел/дн}$$

Общая трудоемкость определяется как сумма трудоемкостей по стадиям:

$$Т_у=Т_{уз}+Т_{уэ}+Т_{ут}+Т_{ур}+Т_{ув} = 2{,}58 + 2{,}07 + 2{,}32 + 12{,}006 + 3{,}88 = 23{,}54 \text{ чел/дн}$$

Таблица 4.1 Трудоемкость разработки программного продукта

Стадии разработки	Содержание работ	Трудоемкость чел/дн
1. Техническое задание	Постановка задачи.	2,58
	Сбор исходных материалов.	
	Выбор и обоснование критериев эффективности и качества разрабатываемой программы.	
	Определение требований к программе.	
	Разработка технико-экономического обоснования разработки программы.	
	Выбор языков программирования.	
	Согласование и утверждение технического задания	
	Предварительная разработка структуры входных и выходных данных.	
2. Эскизный проект	Разработка макета ИС	2,07
	Разработка общего описания алгоритма решения задачи.	
	Разработка технико-экономического обоснования	

Продолжение таблицы 4.1

3. Технический проект	Разработка прототипа ИС	2,32
	Определение формы представления входных и выходных данных.	
	Разработка структуры программы.	
4. Рабочий проект	Программирование и отладка программы.	12,006
	Разработка, согласование и утверждение программы и методики испытаний.	
5. Внедрение	Оформление и утверждение акта о передаче программы на сопровождение и (или) изготовление.	3,88
	Передача программы на сопровождение	
Итого		23,54

Трудоемкость в чел/ч при 9-часовом рабочем дне Ту = 23,54 · 9 = 211,86 чел/ч

4.2 Расчет затрат на разработку ИС

Расчет полных затрат на разработку проектного решения в виде информационных технологий (Cni) осуществляется по формуле:

$$Cni = \text{Зфот} + \text{Зсз}i + \text{М}i + \text{Р}ci + \text{Рм}i + \text{Рнк}i + \text{Пз}i + \text{Рн}i \qquad (4.5)$$

где Зфот – общий фонд оплаты труда разработчиков, тенге;

Зсзі – отчисления по социальному налогу, тенге;

Мі – затраты на материалы, тенге;

Рсі – затраты на специальные программные средства, необходимые для разработки проектного решения, тенге;

Рмі – затраты, связанные с эксплуатацией техники, тенге;

Рнкі – затраты на научные командировки, тенге;

Пзі – прочие затраты, тенге;

Рні – накладные расходы, тенге.

Размер фонда оплаты труда разработчиков (Зфот) рассчитывается по формуле:

$$З_{ФОТ} = З_{oi} + З_{дi}\ , \qquad (4.6)$$

где $З_{oi}$ – основная заработная плата, тенге;

$З_{дi}$ – дополнительная заработная плата, тенге.

Основная заработная плата исполнителей по конкретному ПО рассчитывается по формуле:

$$З_{oi} = \sum_{i=1}^{n} T_{чi} \times T_{ч} \times Ф_{П} \times K, \quad (4.7)$$

где п - количество исполнителей, занятых разработкой конкретного ПО;

$Tчi$ - часовая тарифная ставка i–го исполнителя (тенге/тыс. тенге);

Фп - плановый фонд рабочего времени i-го исполнителя (дней);

$Tч$ - количество часов работы в день (час);

К - коэффициент премирования.

Базой для расчета основной заработной платы являются общая трудоемкость, плановая численность работников и плановые сроки разработки ПО.

По данным о специфике и сложности выполняемых функций составляется штатное расписание группы специалистов-исполнителей, участвующих в разработке ПО, с определением образования, специальности, квалификации и должности. Заработная плата full-stack разработчика согласно headhunter.kz начинается с 150 000 тенге в месяц при 9-часовом рабочем дне.

Таблица 4.2 Сведения по работникам, задействованным в проекте

Специалист - Исполнитель	Количество, человек	**Заработная плата в месяц, тенге**
Программист	1	150 000
Итого		150 000

Часовая тарифная ставка рассчитывается путем деления месячной тарифной ставки на установленную при 40-часовой недельной норме рабочего времени расчетную среднемесячную норму рабочего времени в часах ($Ф_Р$):

$$Т_Ч = \frac{Т_М}{Ф_Р} = \frac{150000}{9 \cdot 21} = 793{,}65 \text{ тг/ч}$$

где ТЧ - часовая тарифная ставка (тыс. тенге);

TM - месячная тарифная ставка (тыс. тенге).

Таблица 4.3 Затраты на оплату труда

Категория работника	Трудоемкость разработки ПП, чел.×ч	Часовая ставка, тг/ч	Сумма, тг
Разработчик	211,86	793,65	168 778
ИТОГО затраты на оплату труда			168 778

$З_{oi} = 168\,778$ тенге.

Дополнительная заработная плата составляет в среднем 10% от основной заработной платы и рассчитывается по формуле:

$Зд_i = Зо_i \times Н_д / 100 = 168\,778 \cdot 0{,}1 \approx 16\,877$ тенге

где $Нд$ - коэффициент дополнительной заработной платы разработчиков.

$$З_{ФОТ} = З_{oi} + З_{дi} = 168\,778 + 16887 = 185\,665 \text{ тенге}$$

Социальный налог для юридических лиц рассчитывается:

$$З_{СН} = (З_{ФОТ} - ОПВ - ВОСМС) \cdot 0{,}095 - СО$$

где ОПВ – обязательный пенсионный взнос (10% от заработной платы);

ВОСМС – взнос на ОСМС (1% от заработной платы);

СО – социальные отчисления (3,5% от заработной платы с вычетом пенсионного взноса).

$$ОПВ = 185\,665 \cdot 0{,}1 = 18\,566$$

$$ВОСМС = 185\,665 \cdot 0{,}02 = 3\,713$$

$$СО = (185\,665 - 18566) \cdot 0{,}035 = 5\,848$$

$$СН = (185\,665 - 18566 - 3713) \cdot 0{,}095 - 5848 = 9\,673 \text{ тенге}$$

$$Зсзi = СН + СО + ВОСМС = 9673 + 5848 + 3713 = 19234 \text{ тенге}$$

Величина затрат на материалы на основании исходных данных определяется по формуле:

$$Mi = \frac{З_{осн} \cdot Н_{мз}}{100} \quad (4.8)$$

где $Н_{мз}$ - норма расхода материалов от основной заработной платы (3-5%).

Величина затрат на материалы при норме расхода – 3%:

$$M_i = \frac{168\,778 \cdot 3}{100} = 5\,063 \text{ тенге}$$

Расходы по статье «Спецоборудование» включают затраты средств на приобретение вспомогательных специального назначения технических и программных средств, необходимых для разработки конкретного ПО, включая расходы на их проектирование, изготовление, отладку, установку и эксплуатацию:

$$P_{ci} = \sum_{i=1}^{n} Ц_{ci} \quad (4.9)$$

где $Ц_{ci}$ - стоимость конкретного специального оборудования (тыс. тенге);

n - количество применяемого специального оборудования.

Таблица 4.4 Затраты на спецоборудование

Наименование материального ресурса	Единица измерения	Количество израсходованного материала	Цена за единицу, тг	Сумма, тг
Ноутбук Lenovo ideapad 510-15ikb	шт	1	267 000	267 000

Продолжение таблицы 4.4

Мышь компьютерная Bloody V8m	шт	1	8 000	8 000
Принтер лазерный hp laserjet pro mfp m130a	шт	1	53 000	53 000
ОС Windows 10 Pro (ключ активации)	шт	1	8000	8000
intellij idea (лицензия на пользователя в месяц)	шт	1	10000	10000
Итого				346 000

Научные командировки не предусмотрены.

Расходы по статье «Прочие затраты» (Пзi) на конкретное ПО включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу, разрабатываемому в целом по организации, в процентах к основной заработной плате:

$$П_{зi} = З_{oi} \cdot \frac{Н_{пз}}{100} \quad (4.10)$$

где $Н_{пз}$ - норматив прочих затрат в целом по организации в (%), в дипломной работе нужно брать 20%.

$$П_з = 168\,778 \cdot 0{,}2 = 33\,755 \text{ тенге}$$

Затраты по статье «Накладные расходы» ($Р$нi), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды ($Р$нi), относятся на конкретное ПО по нормативу ($Н$нр) в процентном отношении к основной заработной плате исполнителей. Норматив устанавливается в целом по организации:

$$Р_{нi} = З_{oi} \cdot \frac{Н_{нр}}{100} \quad (4.11)$$

где $Р$нi - накладные расходы на конкретную ПО (тыс. тенге);

$Н$нр - норматив накладных расходов в целом по организации в (%), в дипломной работе нужно брать 70%.

$$Р_{нi} = 168\,778 \cdot 0{,}7 = 118\,144 \text{ тенге}$$

Таблица 4.5 Затраты на разработку

Затраты на разработку	Условное обозначение	Значение, тенге	В процентах от общей суммы
Фонд оплаты труда	Зфот	185 665	30%
Социальные отчисления и налоги	Зсзi	19234	3,1%
Материалы	Мi	5 063	0,9%
Спецоборудование	Рсi	346 000	41,6%
Прочие затраты	Пзi	33 755	5,4%
Накладные расходы	Рнi	118 144	19%
Итого	C_{ni}	680 861	100%

Рентабельность и прибыль по создаваемому ПО (Псi) определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей дополнительно налог на добавленную стоимость. В случае разработки ПО для использования внутри организации оценка программного продукта производится по действующим правилам и показателям внутреннего хозрасчета (по ценам, устанавливаемым для расчета за услуги между подразделениями). Прибыль рассчитывается по формуле:

$$П_{Oi} = C_{ni} \cdot \frac{У_{рni}}{100} \quad (4.12)$$

где ПОi - прибыль от реализации ПО заказчику (тыс. тенге);

Урni - уровень рентабельности ПО (%), в дипломной работе брать 40-60%;

Cni - себестоимость ПО (тыс. тенге).

$$П_{Oi} = 680\ 861 \cdot\ 0.4 = 272\ 344 \text{ тенге}$$

Прогнозируемая цена ПО без налогов (Цпi):

$$Ц_{пi} = C_{ni} + П_{Oi} = 680\ 861 +\ 272\ 344 = 953\ 205 \text{ тенге}$$

Прогнозируемая отпускная цена *($Ц_{o1}$):*

$$Ц_{01} = Ц_{ni} + НДС$$

Ставка налога на добавленную стоимость НДС в РК на 2020 год составляет 12% от отпускной цены ПО.

$$Ц_{01} = 953\ 205 + \frac{953\ 205\ \cdot 12}{100} = 1\ 067\ 589 \text{ тенге}$$

Организация-разработчик участвует в освоении ПО и несет соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Затраты на освоение определяются по нормативу

($Н_О$=*10%)* от себестоимости ПО в расчете на 3 месяца и рассчитываются по формуле:

$$P_{Oi} = C_{ni} \cdot \frac{Н_О}{100} = 680\,861 \cdot 0{,}1 = 68\,086 \text{ тенге}$$

Затраты на сопровождение ПО ($Р_{Ci}$). Организация-разработчик осуществляет сопровождение ПО и несет соответствующие расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Затраты на сопровождение определяются по установленному нормативу (НС=*20%)* от себестоимости ПО (в расчете на год) и рассчитываются по формуле:

$$P_{Ci} = C_{ni} \cdot \frac{Н_С}{100} = 680\,861 \cdot 0{,}2 = 136\,172 \text{ тенге}$$

Капиталовложения программного обеспечения с учетом затрат на освоение и сопровождение будет:

$$K = 1\,067\,589 + 68\,086 + 136\,172 = 1\,408\,019 \text{ тенге.}$$

4.3 Расчет эксплуатационных затрат

Годовые эксплуатационные текущие затраты в условиях функционирования информационных технологий (C) рассчитываются по формуле:

$$С = ЗП + ОТ + А + М + НР \quad (4.13)$$

где ЗП - годовые затраты на оплату труда специалистов при выполнении ими своих функций в рамках автоматизируемого процесса после внедрения ИТ, тенге;

ОТ - отчисления по социальному налогу, тенге;

А - затраты на амортизацию, тенге;

М - годовые материальные затраты на сопровождение программного продукта, тенге;

НР - накладные расходы, тенге.

Годовые затраты по заработной плате специалистов после внедрения ИТ определяются по формуле:

$$ЗП = \frac{О_с \cdot Ч_с \cdot 12}{Ф_{р.в.}} \cdot t_{общ} \cdot 12 \cdot (1 + К_Д) \quad (4.14)$$

где $О_с$ - оклад специалиста, тенге/мес.;

$Ч_с$- численность специалистов, участвующих в процессе, чел.;

$Ф_{р.в.}$ - годовой фонд рабочего времени, час;

$t_{общ}$- трудоемкость решения задач в условиях функционирования ИТ в месяц, час;

$К_д$ - коэффициент дополнительной заработной платы.

Трудоемкость решения задач в условиях функционирования ИТ в месяц вычисляется следующим образом:

$$t_{\text{общ}} = \sum_{\beta=1}^{n} t_{\beta} \cdot K_{\beta} \quad (4.15)$$

где t_{β} - затраты времени на решение β-й задачи, час:

K_{β} - количество решаемых β -х задач в месяц, ед.

В ходе эксплуатации системы предполагается появление серверных ошибок, на обработку одной ошибки в среднем уходит 1.5 часа, на протяжении месяца эксплуатации может возникать до 70 ошибок, помимо этого ожидается внесение изменений в интерфейс и доработка компонентов по требованию руководства. Всего одулей в системе 3, на доработку одного уходит в среднем 20 часов.

$$t_{\text{общ}} = \ 1.5 \ \cdot 70 + 20 \ \cdot 3 = 105 + 60 = 165 \text{ часов}$$

$$\text{ЗП} = \frac{150\,000 \cdot 1 \cdot 12}{1968} \cdot 165 \cdot 12 \ \cdot 1 = 1\,810\,975 \text{ тенге}$$

Материальные затраты определяются по формуле:

$$M = \sum_{\mu=1}^{n} \text{Ц}_{\mu} \cdot N_{\mu} \quad (4.16)$$

где Цμ - цена μ-го вида единицы материальных затрат в условиях функционирования ИТ, тенге.

$N\mu$ - используемое количество μ-го вида материальных затрат в месяц, ед.

В условиях функционирования разрабатываемой ИС, не предполагается материальных затрат.

Амортизационные отчисления производятся по установленным нормам амортизации, выражаются в процентах к балансовой стоимости оборудования и рассчитываются по формуле:

$$\text{А} = \text{С}_{\text{обор}} \cdot \text{Н}_{\text{а}} \quad (4.17)$$

где $\text{Н}_{\text{а}}$ — норма амортизации (25 %);

$C_{\text{обор}}$ — первоначальная стоимость оборудования;

$$\text{А} = (267000 \ \cdot 0{,}25) + (8\,000 \ \cdot 0{,}25) + (53\,000 \ \cdot 0{,}25) = 82\,000$$

Отчисления по социальному налогу и накладные расходы рассчитываются так же, как и при разработке информационных технологий.

$$\text{СН} = (\text{ЗП} - \text{ОПВ} - \text{ВОСМС}) \cdot 0{,}095 - \text{СО}$$

$$\text{ОПВ} = 1\,810\,975 \ \cdot 0{,}1 = 181\,097 \text{ тенге}$$

$$\text{ВОСМС} = 1\,810\,975 \ \cdot 0{,}02 = 36\,219 \text{ тенге}$$

$$\text{СО} = (1\,810\,975 - 181\,097\ ) \cdot 0{,}035 = 57\,045 \text{ тенге}$$

$$\text{СН} = (1\,810\,975 - 181\,097 - 36\,219) \cdot 0{,}095 - 57\,045 = 94\,352 \text{ тенге}$$

$$\text{ОТ} = \text{СН} + \text{СО} + \text{ВОСМС} = 94\,352 + 57\,045 + 36\,219 = 187\,616 \text{ тенге}$$

$$НР = ЗП \cdot \frac{Н_{нр}}{100} = 1\,810\,975 \cdot 0{,}7 = 1\,267\,682 \text{ тенге}$$

Таблица 4.7 – Эксплуатационные затраты

Затраты на эксплуатацию	Условное обозначение	Значение, тенге	В процентах от общей суммы
Заработная плата	ЗП	1 810 975	54,6%
Социальные отчисления и налоги	ОТ	187 616	5,6%
Материальные затраты	М	0	0%
Амортизационные отчисления	А	57 045	1,7%
Накладные расходы	НР	1 267 682	38,1%
Итого		3 323 318	

$$С = 3\,323\,318 \text{ тенге}$$

4.4 Расчет результатов от создания и использования ИС

Информационная система для учебных заведений предназначена в первую очередь для удобной работы программистов. Она облегчает процесс работы с большим массивом данных, а также способна повысить скорость и точность работы программистов, за счет широкого фуункционала.

Для оценки экономии от использования разрабатываемой ИС необходимо сравнить эксплуатационные расходы с ее применением и без.

Статьи затрат при применении ИС включают в себя:

- заработная плата специалиста, осуществляющего поддержку и сопровождение системы;
- износ оборудования;
- накладные расходы.

Данная система не предполагает расхода каких-либо материалов.

В организации имеется специалист по проектам (месячная заработная плата – 140 000 тенге), основной задачей которых является осуществление контроля работы проектной группы.

ЗП = 2 500 000 тенге

ОПВ=2 500 000 ·0,1=250 000

СО = (2 500 000 – 250 000) · 0,035=78 750

ВОСМС=2 500 000 · 0,02 = 50 000

СН= (ЗП-ОПВ-ВОСМС) ·0,095 – СО = 130 250

ОТ=СО+СН+ВОСМС= 78 750 + 50 000 + 130 250= 259 000 тенге

В их распоряжении находятся два персональных компьютера (стоимость одного ~ 150 000 тенге), принтер (80 000 тенге) и прочая компьютерная периферия (~20 000 тенге); итого затрат на оборудование – 250 000 тенге.

Износ оборудования рассчитывается исходя из 25 % амортизационных отчислений за год.

$$A = (300\,000 \cdot 0{,}25) + (80\,000 \cdot 0{,}25) + (20\,000 \cdot 0{,}25) = 100\,000 \text{ тенге}$$

На заправку картриджей, бумагу прочую канцелярию в месяц уходит порядка 25 000 тенге.

Общие накладные расходы составят:

$$НР = ЗП \cdot \frac{Н_{нр}}{100} = 2\,500\,000 \cdot 0{,}7 = 1\,750\,000 \text{ тенге}$$

Статьи затрат без применения ИС включают в себя:

- заработная плата специалистов – кураторов проектов, которые осуществляют управление проектами;
- износ используемого ими оборудования;
- расход материалов (к примеру канцелярия – бумага, картридж и т.д.)
- накладные расходы.

Таблица 4.8 Годовые эксплуатационные затраты

Статьи	Без применения ИС (1 сотрудник – куратор проекта)	С применением ИС (1 сотрудник – программист, сопровождающий систему)
Годовая заработная плата	2 500 000	1 267 682
Социальные отчисления и налоги	130 250	18 7616
Расходуемые материалы	250 000	0
Амортизационные отчисления	100 000	57 045
Накладные расходы	1 750 000	1267682
Всего	4 730 250	3 323 318

Ожидаемая условно-годовая экономия определяется по формуле:

$$Э_{уг} = С_1 - С_2 + \sum Э_i \qquad (4.18)$$

где $Э_{уг}$- величина экономии, тенге;

$С_1$и $С_2$ – показатели текущих затрат по базовому и внедряемому вариантам, тенге:

$\Sigma Э_i$ – ожидаемый дополнительный эффект от различных факторов, тенге.

$Э_{уг} = С_1 - С_2$= 1 406 932тенге

4.5 Расчет основных показателей экономической эффективности

Так как разработанная информационная система несет более социальный эффект, чем экономический, целесообразно оценивать его эффективность за счет экономии в сравнении с предыдущим периодом работы без использования ИС.

Величина ожидаемого годового экономического эффекта от внедрения ИС рассчитывается по формуле:

$$Э_г = Э_{уг} - К \cdot Е_Н \quad (4.19)$$

где Эг - ожидаемый годовой экономически й эффект, тенге:

Эуг — ожидаемая условно-годовая экономия, тенге;

К — капитальные вложения, тенге;

$Е_Н$ - нормативный коэффициент экономической эффективности капитальных вложений.

Нормативный коэффициент экономической эффективности капитальных вложений определяется по формуле:

$$Е_Н = \frac{1}{Т_Н} \quad (4.20)$$

где Тн — нормативный срок окупаемости капитальных вложений, лет.

Нормативный срок окупаемости капитальных вложений. принимается исходя из срока морального старения -технических средств и проектных решений ИС (Тн=1,2,3...n), для программных продуктов срок окупаемости принимаем равным 4 года.

$$Е_Н = \frac{1}{4} = 0{,}25 \quad (4.21)$$

$$Э_г = 1\,406\,932 - 1\,408\,019 \cdot 0{,}25 = 1\,054\,927 \text{ тенге}$$

Расчетный коэффициент экономической эффективности капитальных вложений составляет:

$$Е_р = \frac{Э_{уг}}{К} \quad (4.22)$$

где Ер - расчетный коэффициент экономической эффективности капитальных вложений;

Эуг — ожидаемая условно-годовая экономия, тенге;
К — капитальные вложения на создание системы, тенге.

$$E_р = \frac{1\,406\,932}{1\,408\,019} = 0{,}99$$

Расчетный срок окупаемости капитальных вложений составляет:

$$T_Р = \frac{1}{E_Р} \qquad (4.23)$$

где Ер - коэффициент экономической эффективности капитальных вложений.

$$T_Р = \frac{1}{0{,}99} = 1{,}01 \text{ года} \approx 12{,}12 \text{ месяцев}$$

Таблица 4.9 Показатели сравнительной экономической эффективности от внедрения программного продукта

Наименование показателей	Значение
Условная годовая экономия затрат, тенге	1 406 932
Коэффициент экономической эффективности капитальных вложений (Ер)	0,99
Срок окупаемости капитальных вложении (Тр)	1,01

Таким образом, разработанная информационная система позволила упросить и сэкономить проект. Ожидаемый годовой экономический эффект составил 1 406 932 тенге. Приложение окупится в первые 12 месяцев использования.

Вывод

Суммарные затраты на разработку программного продукта составляют 2 578 548 тенге, цена реализации – 3 455 789,2 тенге с учетом НДС. Основную долю в себестоимости программного продукта составляют машинное время – 1 145 147 тенге или 87,6%, и фонд оплаты труда – 185 125 тенге или 8,4%.

Приложение является собственностью разработчика, может свободно меняться, дополняться и улучшаться при сравнительно небольших дополнительных вложениях.

5 Безопасность жизнедеятельности

5.1 Анализ потенциально опасных и вредных факторов в офисе, воздействующих на персонал

5.1.1 Рабочее помещение

Рассматривается рабочее помещения, расположенное в здании, которое находится в непосредственной близости от к проезжей части поэтому есть внешние источники шума, влияющих на процесс работы. Ниже описываются характеристики выбранного помещения:

a) рабочее помещение находится на третьем этаже четырехэтажного здания;

b) размеры рабочего помещения (комнаты): длина 30 м, ширина 20 м, высота 3 м;

c) остекление помещения – со стороны улицы окно от потолка до пола, двойной стеклопакет;

d) искусственное освещение – светильники: 50 светильников, в каждом по 1 люминесцентной лампе;

e) внутренняя отделка стен – светлая;

f) помещение по зрительным условиям работы относится к IV разряду

g) в здание предусмотрена вытяжная система.

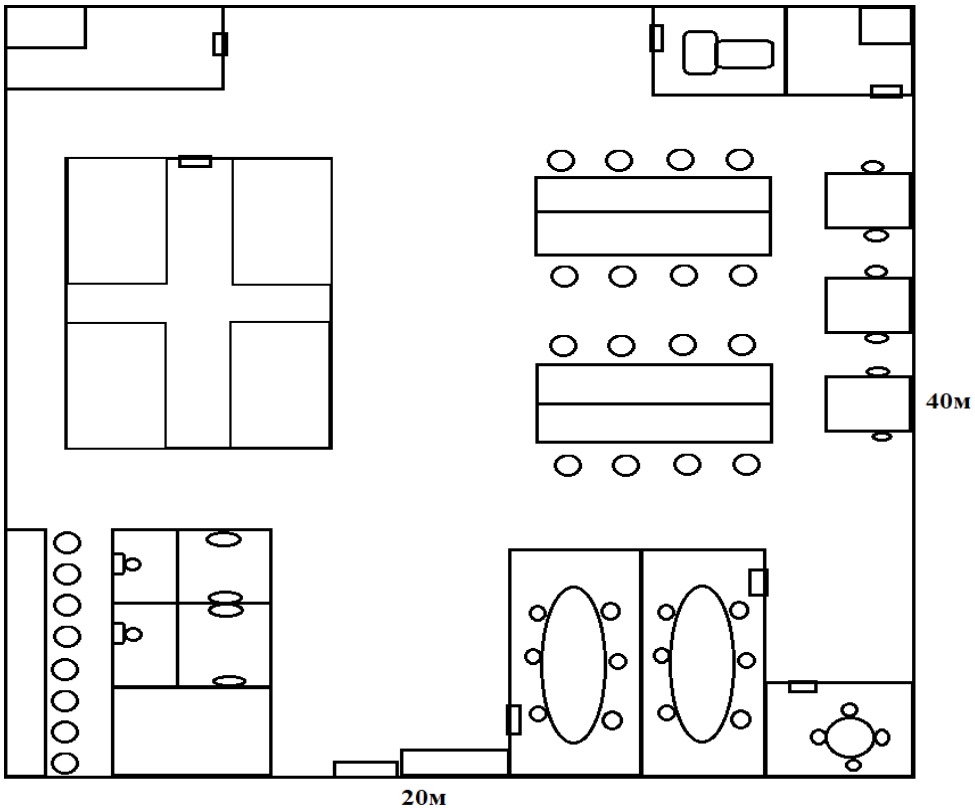


Рисунок 5.1 - Рабочее помещение

5.1.2 Характеристика источника опасных и вредных факторов

В наше время люди все чаще переходят от стационарных компьютеров, к более практичным ноутбукам. Но опасность от этого не снижается, хоть и некоторые факторы воздействуют меньше. Основные факторы, плохо сказывающиеся на здоровье человека при работе с ноутбуком:

- физические: Не правильная освещенность, не правильное положение за рабочим местом, запыленность и др;

- химические: содержание в воздухе углекислого газа, аммиака, оксида углерода и др.;

- психофизиологические: напряжение зрения, памяти, внимания, эмоциональные перегрузки.

Особенности режима и характера работы, значительные умственные напряжения и другие нагрузки при неправильном расположении элементов рабочего места чаще всего способствуют в необходимости находится в одной позе долгое время. Длительный дискомфорт при работе вызывает развитие общего утомления и снижения работоспособности.

Хоть с каждым годом и улучшается технология создания экранов при этом и все же при длительной работе за экраном монитора возникает напряжение на глаза. При неправильном выборе яркости и освещенности экрана, контрастности знаков, цветов знаков и фона, при наличии бликов на экране, дрожании и мелькании изображения работа на мониторе приводит к зрительному утомлению, головным болям, раздражительности, нарушению сна, усталости и болезненному ощущению в глазах, пояснице, в области шеи, рук и т.д.

Все вышеописанные воздействия на организм человека усугубляются, если не соблюдается режим труда и отдыха, не проводится производственная гимнастика и витаминизация организма.

Наличие электростатического поля приводит к уменьшению

содержания отрицательных ионов в воздухе помещения и загрязнению экрана в результате притягивания к нему отрицательных ионов и мелких частиц пыли. Длительная работа компьютера приводит к снижению концентрации кислорода, повышению концентрации озона. Озон является сильным окислителем, и концентрация его выше предельно допустимых величин может привести к неблагоприятным обменным реакциям организма, изменяя активность ряда ферментов, способствует нарушению зрения.

5.1.3 Основная опасность на рабочем месте

Основная опасность на рабочем месте являются:

высокое количество пыли и вредных газов;

высокая и низка температура;

- повышенная или пониженная влажность и подвижность воздуха;

- высокий уровень шума и вибрации;
- повышенный уровень различных электромагнитных излучений;
- отсутствие или недостаток естественного света;
- недостаточная освещенность рабочей зоны.

Границы производственной санитарии:

- оздоровление воздушной среды и нормализация микроклимата;
- защита и уменьшение воздействий шума, вибрации и электромагнитных излучений;
- соблюдение нормативов освещения;
- поддержание санитарных норм в помещен

Для поддержания комфортного микроклимата необходимо кондиционирование и отопление в помещении. В современных зданиях что бы обойтись без кондиционеров, по всему зданию проводят воздуховоды. Они отлично контролируют воздушный поток, температуру и влажность. Отопление же в свою очередь немного высушивает воздух и поддерживает температуру в офисе, так как в холодное время года через окна и стены идет большая тепло потеря.

Так же размеры помещения должны соответствовать необходимым нормам, исходя из числа сотрудников и оборудования. На одно рабочее место должно приходится не менее 15м объема производственного помещения и не менее 4,5 м2 площади.

Таблица 5.1 Нормы температуры, относительной влажности и скорости движения воздуха офисе.

Температура окружающего воздуха	Оптимальные параметры воздушной среды на постоянных рабочих местах		
	Температура, °С	Относительная влажность, %	Скорость движения воздуха, м/с не более
Ниже+10°С	20-22	40-60	0,2
Выше+10°С	22-25	40-60	0,5

Таблица 5.2 Нормы уровня шумов

Величина	Уровни звукового давления, Дб, в октавных полосах со среднегеометрическими частотами, Гц							
	63	125	250	500	1000	2000	4000	8000
Помещение пользователей ПК	73	63	57	52	48	45	43	41

На рабочем месте мы часто используем экраны ноутбуков (ПК)и пожтому необходимо уменьшить действий электромагнитных и рентгеновски х излучений, которые являются очень опасными для человека. Для этого

необходимо сотрудникам делать перерывы для снятия напряжения с глаз и размять мышцы. Существуют нормы предельного электромагнитного и рентгеновского излучения, при которых действие излучений не оказывает существенного влияния на организм человека. Главным источником излучения являются мониторы ноутбуков. Современные стандарт мониторов ТСО'95 и ТСО'99 является наиболее подходящим.

Таблица 5.3 Предел допустимого напряжения электрического поля радиочастот на рабочих местах

Допустимый диапазон частот в помещении, МГц	Предел допустимой напряженности в течении рабочего дня
Электрическая составляющая В/м	
0,06	50
3-30	20
30-50	10
50-300	5
Магнитная составляющая А/м	
0,06-1,5	5
30-50	0,3

Таблица 5.3 Предельно допустимые дозы облучения

Категория облучаемых лиц	Высшее облучение	
	Бэр в неделю	Бэр в год
Программисты работающие непосредственно на ноутбуках	0,1	5

Рабочее место и взаимное расположение всех его элементов должно соответствовать всем современным требованиям. Главным фактором является характер работы специалиста. На примере рабочего места программиста соблюдаются следующие основные условия: наилучшее размещение оборудования, входящего в состав рабочего места и достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения. Эргономическими аспектами проектирования видеотерминальных рабочих мест, в частности, являются: высота рабочей поверхности, размеры пространства для ног, требования к расположению документов на рабочем месте (наличие и размеры подставки для документов, возможность различного размещения документов, расстояние от глаз пользователя до экрана, документа, клавиатуры и т.д.), характеристики рабочего кресла, требования к поверхности рабочего стола, регулируемость элементов рабочего места.

5.2 Расчеты

5.2.1 Расчет системы кондиционирования

Определим необходимое количество кондиционеров для создания комфортных условий труда в помещении. В помещении за счёт тепловыделений оборудования (мониторы, ноутбуки, сотрудники, тепло от окон и т.д.) могут иметь место значительные избытки тепла, удаление которых, прежде всего, должна обеспечить система вентиляции.

Избыточное тепло:

$$Q_{ИЗБ} = Q_{ОБ} + Q_{ОСВ} + Q_{Л} + Q_{Р} - Q_{ОТД} \quad (5.1)$$

Где $Q_{ОБ}$, $Q_{ОСВ}$, $Q_{Л}$ - тепло, выделяемое производственным оборудованием, системой искусственного освещения помещения и сотрудниками соответственно, ккал/ч;

$Q_{Р}$ – тепло, вносимое в помещение солнцем (солнечная радиация), ккал/ч;

$Q_{ОТД}$ - теплоотдача естественным путём, ккал/ч.

Тепло, выделяемое производственным оборудованием

$$Q_{ОБ} = 530 * P_{ОБ} * \eta \quad (5.2)$$

где 530 тепловой эквивалент 1кВт/ч; 1. (СН РК 4.02-01-2011, с.36)

$P_{ОБ}$ – мощность, потребляемая оборудованием, кВт/ч;

η - коэффициент перехода тепла в помещение.

Для 1 ноутбука имеем

$$Q_{ОБ} = 530 * 0{,}25 * 0{,}95 = 125{,}87 \text{ ккал/ч};$$

Значение η=0,95 – норма потерь потребляемой мощности на тепловыделения компьютерного оборудования. (СН РК 4.02-01-2011, с.38)

Тепло, выделяемое осветительными приборами

$$Q_{ОСВ} = 370 * \eta * N \quad (5.3)$$

где N расходуемая мощность светильников, кВт;

Значение η=0,55 – норма потерь потребляемой мощности на тепловыделения люминесцентных ламп. (СН РК 4.02-01-2011, с.41)

$$Q_{ОСВ} = 370 * 0{,}55 * 0{,}52 = 105{,}82 \text{ ккал/ч};$$

Тепло, выделяемое людьми

$$Q_{Л} = K_{Л} * (q - q_{ИСП}) \quad (5.4)$$

где Кл-количество работающих;

(q-qисп) – явное тепло, ккал/ч;

q-тепловыделения одного человека при данной категории работ I-III, ккал/ч.

Работа, производимая в помещении, относится к I категории работ:

q=100 Вт, или 0,1 кВт для офисных помещений; (СНиП 2.04.05-86, с.24)

$$Q_Л = 1*860*0,1=86 \text{ ккал / ч.}$$

Тепло, вносимое солнечной радиацией

где m – количество окон в помещении;

F – площадь одного окна, м2;

qост – солнечная радиация через остеклённую поверхность, т.е. количество тепла, вносимое за один час через остеклённую поверхность площадью 1 м2.

Одно окно с двойным стеклопакетом, $q_{ОСТ}=105$ (окна выходят на юг). Количество окон около 30. Площадь окна $3*2=6$ м^2

$$Q_Р=1*6*105=630 \text{ ккал / ч.}$$

Для тёплого периода года при расчётах можно принять $Q_{ОТД}=0$.

$$Q_{ИЗБ=}125,87+105,82+86+630=947,69 \text{ ккал / ч.}$$

При наличии тепло избытков количество воздуха, которое необходимо удалить из помещения

$$L_b = Q_{изб} / C_b * \Delta t * \gamma_b \qquad (5.5)$$

где $Q_{изб}$ - избыточное тепло, ккал/ч;

C_b - теплоёмкость воздуха (0,24 ккал/кг°С); (СНиП 2.04.05-86, с.35)

$\Delta t=t_{вых} – t_{вх}$;

$t_{вых}$ – температура воздуха выходящего из помещения, °С;

$t_{вх}$ – температура воздуха поступающего в помещение, °С;

γ_b =1,206 кг/ м3 – удельная масса приточного воздуха. (СН РК 4.02-01-2011, с.64)

Величина Δt при расчётах выбирается в зависимости от тепло напряжённости воздуха

$$Q_н=Q_{ИЗБ} / V_П \qquad (5.6)$$

$$Q_н = 947,69 / 54 = 17,5 \text{ ккал / м}$$

Если тепло напряжённость воздуха $Q_н$<20 ккал / $м^3$, то принимаем $\Delta t=6^0C$, а при $Q_н$>20 ккал / $м^3$, то принимаем $\Delta t=8^0C$(СН РК 4.02-01-2011, с.66)

$$L_b=947,69/(0,24*6*1,206)= 545,9 м^3/ч$$

Так как система воздуховодов по всему кабинету, то будет достаточным охлаждением воздуха.

5.2.2 Расчет вентилирования помещения

Существует 2 типа вентилирования помещения: механическая и естественный. В свою очередь естественная вентиляция делится на неорганизованной естественной системой вентиляции и организованной естественной вентиляцией. Объединяет их похожий механизм за счёт разницы снаружи и внутри здания, отличает же то что неорганизованная достигает этого за счет открытых форточек и дверей, а организованная через специально устроенные приточные и вытяжные проемы, степень открытия которых регулируется.

При механической вентиляции воздухообмен происходит за счет разности давления, создаваемой вентилятором или эжектором. Этот способ вентиляции более эффективен, так как воздух предварительно может быть очищен от пыли и доведен до требуемой температуры и влажности.

В нашем случае необходима механическая. При создании системы вентиляции необходимо определить необходимый воздухообмен, исходя из объема помещения, типа помещения и численности человек. Расчет воздухообмена по кратности:

$$L = n * S * H, \quad (5.7)$$

где L - требуемая производительность приточной вентиляции, м3/ч;

n - нормируемая кратность воздухообмена: для офисов n = 2,5; (СНиП 2.04.05-86, с.40)

S - площадь помещения, (800м2); H — высота помещения, (3 м);

$$L=2,5*800*3=6000 м3/ч.$$

Расчет воздухообмена по количеству людей:

$$L = N * Lнорм, \quad (5.8)$$

где L — требуемая производительность приточной вентиляции, м3/ч;

N — количество людей;

Lнорм — норма расхода воздуха на одного человека (при работе в офисе Lнорм = 60 м3/ч). (СНиП 2.04.05-86, с.53)

$$L=30*60=1800 \text{ м3/ч.}$$

После определения производительности вентиляции можно переходить к проектированию воздухораспределительной сети, которая состоит из воздуховодов, фасонных изделий (переходников, разветвителей, поворотов), дроссель-клапанов и распределителей воздуха (решеток или диффузоров). Расчет воздухораспределительной сети начинают с составления схемы воздуховодов. Схему составляют таким образом, чтобы при минимальной общей длине трассы система вентиляции могла подавать расчетное количество воздуха во все обслуживаемые помещения. Далее по этой схеме рассчитывают размеры воздуховодов и подбирают воздухораспределители.

Для расчета размеров (площади сечения) воздуховодов нам нужно знать объем воздуха, проходящий через воздуховод в единицу времени, а также максимально допустимую скорость воздуха в канале. При увеличении скорости воздуха размеры воздуховодов уменьшаются, но уровень шума и сопротивление сети возрастают. На практике для квартир и коттеджей скорость воздуха в воздуховодах ограничивают на уровне 3–4 м/с, поскольку при более высоких скоростях воздуха шум от его движения в воздуховодах и распределителях может стать слишком заметным. Следует также учитывать, что использовать «тихие» низкоскоростные воздуховоды большого сечения не всегда возможно, поскольку их сложно разместить в запотолочном пространстве. Снизить высоту запотолочного пространства позволяет применение прямоугольных воздуховодов, которые при одинаковой площади сечения имеют меньшую высоту, чем круглые (например, круглый воздуховод диаметром 160 мм имеет такую же площадь сечения, как и прямоугольный размером 200×100 мм). Так что поэтому применим прямоугольные.

$$Sc = L * 2{,}778 / V, \qquad (5.9)$$

где Sc — расчетная площадь сечения воздуховода, м²;

L — расход воздуха через воздуховод, м³/ч;

V — скорость воздуха в воздуховоде, м/ч;

2,778 — коэффициент для согласования различных размерностей (часы и секунды, метры и сантиметры). (СН РК 4.02-01-2011, с.73)

$$Sc=350*2{,}778/2*3600=0{,}135 \text{ м2.}$$

Фактическая площадь сечения воздуховода определяется по формуле:

$$S = \pi * D^2 / 400 \text{ — для круглых воздуховодов,} \qquad (5.10)$$

$$S = A * B / 100 \text{ — для прямоугольных воздуховодов,} \qquad (5.11)$$

где S — фактическая площадь сечения воздуховода;
D — диаметр круглого воздуховода, м;
A и B — ширина и высота прямоугольного воздуховода, м.

S=0,125*0,4/100=0,0005 м2

Зная расход воздуха можно подобрать по каталогу воздухораспределители с учетом соотношения их размеров и уровня шума (площадь сечения воздухораспределителя, как правило, в 1,5–2 раза больше площади сечения воздуховода). Для примера рассмотрим параметры популярных воздухораспределительных решеток Арктос серий АМН, АДН, АМР, АДР.

Теперь мы можем рассчитать требуемую мощность калорифера. Для этого нам понадобятся значения температуры воздуха на выходе системы и минимальной температуры наружного воздуха в холодный период года. Температура воздуха, поступающего в жилое помещение, должна быть не ниже +18°C. Минимальная температура наружного воздуха зависит от климатической зоны и для Алматы принимается равной -20°C. Таким образом, при включении калорифера на полную мощность, он должен нагревать поток воздуха на 40°C. Поскольку сильные морозы в Алмате непродолжительны, можно использовать калорифер меньшей мощности, при условии, что система вентиляции имеет регулировку производительности: это позволит в холодный период поддерживать комфортную температуру воздуха за счет снижения скорости вентилятора .

Мощность калорифера рассчитывается по формуле:

$$P = \Delta T * L * C_v / 1000, \quad (5.12)$$

где P — мощность калорифера, кВт;
ΔT — разность температур воздуха на выходе и входе калорифера, C;
Для Алматы ΔT=40°C, для других регионов — определяется по СНиП;
L — производительность вентиляции, м³/ч;
Cv — объемная теплоемкость воздуха, равная 0,336 Вт·ч/м³/°C. Этот параметр зависит от давления, влажности и температуры воздуха (СН РК 4.02-01-2011, с.82)

P=40*350*0,336/1000=4 704 Вт.(4,704 кВТ)

После расчета мощности калорифера нужно выбрать напряжение питания (для электрического калорифера. При мощности калорифера свыше 4–5 кВт желательно использовать 3-х фазное подключение. Максимальный ток, потребляемый калорифером, можно рассчитать по формуле:

$$I = P / U, \quad (5.11)$$

где I — максимальный потребляемый ток, А;
P — мощность калорифера, Вт; U — напряжение питание:
220В — для однофазного питания;
660В (3 × 220В) — для трехфазного питания.

$$I=4\ 704/660=7{,}12\ А$$

Типичные значения от 5 до 50 кВт для офисов. При высокой расчетной мощности лучше устанавливать водяной калорифер, который использует в качестве источника тепла воду из системы центрального или автономного отопления.

Вывод

В части безопасность жизнедеятельности рассмотрена современная организация рабочего места. Так как в современных зданиях изменены размеры окон, переработана вентиляция, более легкая и минималистичная планировка, меняются расчеты освещенности, вентиляции и шума. Увеличивается естественная освещенность, упрощается контроль микроклиматом и уменьшается шум. В работе приведены расчеты кондиционирования и вентиляции помещения, по ним определяется эффективность современных методов поддержания микроклимата.

Заключение

В ходе выполнения данного дипломного проекта была разработана серверная часть сайта для учреждений. Название сайта herodotus.

Поставленные нами цели были достигнуты:

Анализ был сделан путем раскрытия предыдущих представлений об опыте студентов с онлайн сайтами учебных заведений.

Архитектурный дизайн веб-сервиса был выполнен с учетом всех требований и прокладывает путь для будущих планов расширения и масштабируемости.

Основные функции реализованные в проекте:

- использование веб-технологий, позволяющих использовать систему как в локальной сети, так и в интернете;
- первый взгляд на онлайн дневник и своевременные уведомления;
- распределение пользователей по ролям;
- зайти в журнал можно с любого устройства благодаря адаптивности системы;
- просматривать и скачивать материал, размещенный преподавателями;
- Возможность администратора изменять роли, добавлять новые функции, изменять дизайн;

Было проведено тестирование для выявления ошибок и немедленного реагирования для их устранения.

Целью проекта было разработать сайт, где студенты могли беспрепятственно использовать журнал, материалы сайта и своевременно получать уведомления о предстоящих событиях. Эта цель, которая была определена в начале проекта, была достигнута. Однако есть некоторые пробелы, которые требуют дальнейшего развития.

При большем времени для работы над этим проектом, можно добавить множество полезных и новых функций на сайт, а так же новых способах их решения. Весь процесс разработки программного обеспечения был продуктивным и интересным, создавая совместный проект было необходимо не только создавать и развертывать веб-службы, но и беседовать с людьми учащимися и работающими в сфере образования. Разрабатывая каждый интерфейс, приходилось примерять роли как студентов так и преподавателей что, возможно, было лучшим элементом управления пользовательского интерфейса для такой-то функциональности.

В целом, проект созданный с помощью новых технологий обладает рядом преимуществ, связанных гибкостью и доступностью инструментов.

Список литературы

1 Фуфаев Э. В., Фуфаев Д. Э. Базы данных. – М.:ACADEM A, 2016 – 300 с.

2 PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.: А

3 Дремина, Марина Анатольевна Проектный подход к разработке и внедрению систем менеджмента качества / Дремина Марина Анатольевна. - М.: Лань, 2017. - 142 с.

4 Методические указания к выполнению экономической части дипломных работ для студентов специальности 5В070400 Вычислительной техника и программного обеспечения Алматы, АУЭС 2016 – 40с.

5 Аманбаев У.А. Экономика предприятия – Алматы «Бастау» 2015 г.

6 Буров В. П. Бизнес план фирмы, - М., «Инфра-М» 2015г.

7 Куатова Д. Я. Экономика предприятия – Алматы «Экономика», 2011

8 Стешин А. Изучаем сервер под windows. – СПб: Питер, 2017. – 209

9 СН РК 4.02-01-2016 «Отопление, вентиляция и кондиционирование воздуха»

10 СП РК 4.02-106-2017 Автономные источники теплоснабжения.

11 Безопасность жизнедеятельности. Павлов В.Н., Буканин В.А. и др. (2015, 336с.)

12 Безопасность жизнедеятельности. Калюжный Е.А., Михайлова С.В. и др. (АГПИ, 2017, 316с.)

13 Безопасность жизнедеятельности. (для ссузов) Микрюков В.Ю. (2015, 464с.)

14 Гонсалвес, Энтони Изучаем Java EE 7 / Энтони Гонсалвес. - М.: Питер, 2016. - 640 с.

15 Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 689 с.

16 Глушаков, С. В. Программирование Web-страниц. JavaScript. VBScript / С.В. Глушаков, И.А. Жакин, Т.С. Хачиров. - М.: Фолио, 2015. - 390 с.

17 Козловский, Павел Разработка веб-приложений с использованием AngularJS / Павел Козловский , Питер Бэкон Дарвин, Павел Козловский. - Москва: Мир, 2015. - 394 с.

18 Саймон, Ригс Администрирование PostgreSQL 9. Книга рецептов / Ригс Саймон. - М.: ДМК Пресс, 2018. - 806 с.

19 Стоунз PostgreSQL. Основы / Стоунз, Мэттью Ричард; , Нейл. - М.: СПб: Символ-Плюс, 2002. - 640 с.

20 http://ru.wikipedia.org/wiki/

21 http://www.ibresource.ru

Приложение А
(листинг программы)

```
//журнал
package kz.greetgo.sandbox.register.impl;
import kz.greetgo.sandbox.register.dao.GradeDao;
import kz.greetgo.sandbox.register.logging.LOG;
import kz.greetgo.sandbox.register.model.dao.Discipline;
import kz.greetgo.sandbox.register.model.dao.Grade;
import kz.greetgo.sandbox.register.model.dao.GradeChart;
import kz.greetgo.sandbox.register.model.dao.GradeChartData;
import kz.greetgo.sandbox.register.model.dao.GradeChartFilter;
import kz.greetgo.sandbox.register.model.dao.Group;
import kz.greetgo.sandbox.register.model.dao.Journal;
import kz.greetgo.sandbox.register.model.dao.NotificationRecord;
import kz.greetgo.sandbox.register.model.dao.Person;
import kz.greetgo.sandbox.register.model.dao.StudentJournalFilter;
import kz.greetgo.sandbox.register.register.GradeRegister;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
@Component
public class GradeRegisterImpl implements GradeRegister {
  private final LOG logger = LOG.byName("GRADE");
  @Autowired
  private GradeDao gradeDao;
  @Override
  public Journal gradeList(StudentJournalFilter filter) {
    long start = System.currentTimeMillis();
    logger.info(() -> "group id: " + filter.getGroupId());
    Integer lecturerId = gradeDao.getLecturerId(filter.getPersonGuid());
    Journal journal = new Journal();
    journal.gradeList = gradeDao.selectGrade(filter, lecturerId);
    logger.info(() -> "journal: " + journal + " time: " + (System.currentTimeMillis() - start));
    return journal;
  }
  @Override
  public void saveGrade(Grade grade) {
    Person person = gradeDao.loadPersonByFio(grade.fio);
    try {
//      SmsSender smsSender = new SmsSender();
//       smsSender.sendSms(person.getPhone(), " Оценка по предмету " + grade.discipline
+ " балл : " + grade.grade);
      EmailSender emailSender = new EmailSender();
      emailSender.sendEmail(person.getEmail(), " Оценка по предмету " + grade.discipline
+ " балл : " + grade.grade);
    } catch (Exception e) {
      logger.error(e.getMessage());
    }
```

```
 gradeDao.saveGrade(grade);
}
@Override
public List<Discipline> loadDisciplines(Integer personGuid, String year, String semester) {  return gradeDao.loadDisciplines(personGuid, year, semester); }
@Override
public List<Group> loadGroups(Integer personGuid, Integer disciplineId) {
  return gradeDao.loadGroups(personGuid, disciplineId);  }
@Override
public Date loadCurrentDate() { Date date = gradeDao.loadCurrentDate();
  if (date == null) {
    return new Date(); }
  return date;  }
@Override
public List<String> loadValidYears() {
  return gradeDao.loadValidYear();}
@Override
public void saveNotification(NotificationRecord notificationToSave) {
  Integer roleId = gradeDao.getRoleId(notificationToSave.role.name());
  Integer typeId = gradeDao.getNotificationTypeId(notificationToSave.type.name());
  gradeDao.saveNotification(notificationToSave, typeId, roleId); }
@Override
public List<NotificationRecord> loadNotificationList() {
  return gradeDao.loadNotificationList();}
@Override
public List<GradeChartData> loadGradeChartData(GradeChartFilter filter) {
  List<GradeChartData> gradeChartData = new ArrayList<>();
  List<Discipline> disciplines = null;
  try {
    disciplines = gradeDao.loadDisciplinesForStudent(filter);
  } catch (Exception e) {
    logger.error(e.getMessage());}
  for (Discipline x : disciplines) {
    GradeChartData data = new GradeChartData();
    List<Double> srList = gradeDao.loadSrGradeByDiscipline(x.name, filter);
    data.label = x.name;
    GradeChart grade = new GradeChart();
    Double all = 0.0;
    for (Double y : srList) {
      all += y;}
    grade.sr = all / srList.size();
    grade.rk1 = gradeDao.loadRKFGradeByDiscipline(x.name, filter);
    grade.rk2 = gradeDao.loadRKSGradeByDiscipline(x.name, filter);
    grade.rd = (((grade.rk1 + grade.rk2) / 2) * 0.2) + (grade.sr * 0.8);
    grade.ex = gradeDao.loadRKEXAMGradeByDiscipline(x.name, filter);
    grade.sum = (grade.rd * 0.6) + (grade.ex * 0.4);
    data.grades = grade;
    gradeChartData.add(data);}
  return gradeChartData; }}
```

```
//смс уведомления
package kz.greetgo.sandbox.register.impl;
import kz.greetgo.sandbox.register.dao.GradeDao;
import kz.greetgo.sandbox.register.dao.NotificationDao;
import kz.greetgo.sandbox.register.logging.LOG;
import kz.greetgo.sandbox.register.model.dao.NotificationRecord;
import kz.greetgo.sandbox.register.model.dao.NotificationType;
import kz.greetgo.sandbox.register.model.dao.PersonNotificationData;
import kz.greetgo.sandbox.register.model.dao.RoleType;
import kz.greetgo.sandbox.register.register.NotificationRegister;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import java.util.List;
import java.util.stream.Collectors;
@Component
public class NotificationRegisterImpl implements NotificationRegister {
  private final LOG logger = LOG.byName("NOTIFICATION");
  @Autowired
  NotificationDao notificationDao;
  @Autowired
  GradeDao gradeDao;
  SmsSender smsSender = new SmsSender();
  EmailSender emailSender = new EmailSender();
  @Override
  public void sendNotification() {
    List<NotificationRecord>                    notificationRecords                    =
notificationDao.loadNotificationListToSend();
    if (notificationRecords.size() > 0) {
     try {
      notificationRecords.forEach(x -> {
       List<PersonNotificationData> personDataList;
       if (x.role == RoleType.ALL) {
        personDataList = notificationDao.loadAllPersonList();
       } else {
        personDataList = notificationDao.loadPersonListByRole(x.role.name());
       }
       for (PersonNotificationData y : personDataList) {
        logger.info(() -> "SEND SMS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        if (x.type == NotificationType.ALL) {
         smsSender.sendSms(y.getPhone(), x.kz + "\n\n" + x.ru + "\n\n" + x.en);
         logger.info(() -> "SMS ОТПРАВЛЕНО : " + y.getPhone());
         emailSender.sendEmail(y.getEmail(), x.kz + "\n\n" + x.ru + "\n\n" + x.en);
        } else {
         if (x.type == NotificationType.EMAIL) {
          emailSender.sendEmail(y.getEmail(), x.kz + "\n\n" + x.ru + "\n\n" + x.en);
         } else {
          smsSender.sendSms(y.getPhone(), x.kz + "\n\n" + x.ru + "\n\n" + x.en);
          logger.info(() -> "SMS ОТПРАВЛЕНО : " + y.getPhone());
         }
```

```
        }
              }

               });
              }catch (Exception e){
               logger.error(e.getMessage());
               return;
              }
              logger.info(() -> "Notification sended");
              List<Integer>          collect          =          notificationRecords.stream().map(x          ->
x.id).collect(Collectors.toList());
              collect.forEach(x -> notificationDao.changeStatusById(x));
              logger.info(() -> "Status changed");
             }
            }
           }
           //почтовые уведомления
           package kz.greetgo.sandbox.register.impl;
           import java.util.Properties;
           import javax.mail.Message;
           import javax.mail.MessagingException;
           import javax.mail.PasswordAuthentication;
           import javax.mail.Session;
           import javax.mail.Transport;
           import javax.mail.internet.InternetAddress;
           import javax.mail.internet.MimeMessage;
           public class EmailSender {
            public static void main(String[] args) {
             // Recipient's email ID needs to be mentioned.
             String to = "daltynov@greet-go.com";
             // Sender's email ID needs to be mentioned
             String from = "zolotosss77796@gmail.com";
             // Assuming you are sending email from through gmails smtp
             String host = "smtp.gmail.com";
             // Get system properties
             Properties properties = System.getProperties();
             // Setup mail server
             properties.put("mail.smtp.host", host);
             properties.put("mail.smtp.port", "465");
             properties.put("mail.smtp.ssl.enable", "true");
             properties.put("mail.smtp.auth", "true");
             // Get the Session object.// and pass username and password
             Session session = Session.getInstance(properties, new javax.mail.Authenticator() {
              protected PasswordAuthentication getPasswordAuthentication() {
               return new PasswordAuthentication("test.univer111@gmail.com", "Zoloto13");
              }
             });
             // Used to debug SMTP issues
             session.setDebug(true);
```

```
  try {
    // Create a default MimeMessage object.
    MimeMessage message = new MimeMessage(session);
    // Set From: header field of the header.
message.setFrom(new InternetAddress(from));
    // Set To: header field of the header.
    message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
    // Set Subject: header field
    message.setSubject("This is the Subject Line!");
    // Now set the actual message
    message.setText("This is actual message");
    System.out.println("sending...");
    // Send message
    Transport.send(message);
    System.out.println("Sent message successfully....");
  } catch (MessagingException mex) {
    mex.printStackTrace();
  }
}
public void sendEmail(String email, String messageBody){
  // Recipient's email ID needs to be mentioned.
  String to = email;
  // Sender's email ID needs to be mentioned
  String from = "zolotosss77796@gmail.com";
  // Assuming you are sending email from through gmails smtp
  String host = "smtp.gmail.com";
  // Get system properties
  Properties properties = System.getProperties();
  // Setup mail server
  properties.put("mail.smtp.host", host);
  properties.put("mail.smtp.port", "465");
  properties.put("mail.smtp.ssl.enable", "true");
  properties.put("mail.smtp.auth", "true");
  // Get the Session object.// and pass username and password
  Session session = Session.getInstance(properties, new javax.mail.Authenticator() {
    protected PasswordAuthentication getPasswordAuthentication() {
      return new PasswordAuthentication("test.univer111@gmail.com", "Zoloto13");
    }
  });
  // Used to debug SMTP issues
  session.setDebug(true);
  try {
    // Create a default MimeMessage object.
    MimeMessage message = new MimeMessage(session);
    // Set From: header field of the header.
    message.setFrom(new InternetAddress(from));
    // Set To: header field of the header.
    message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
    // Set Subject: header field
```

```
 message.setSubject("Test Grade");
      // Now set the actual message
      message.setText(messageBody);
      System.out.println("sending...");
      // Send message
    Transport.send(message);
      System.out.println("Sent message successfully....");
    } catch (MessagingException mex) {
      mex.printStackTrace();
    }
  }
}
//регистратор данных
package kz.greetgo.sandbox.register.logging;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.function.Supplier;
public class LOG {
  private final Logger logger;
  private LOG(Logger logger) {
    this.logger = logger;
  }
  public static LOG forClass(Class<?> aClass) {
    return new LOG(LoggerFactory.getLogger(aClass));
  }
  public static LOG byName(String name) {
    return new LOG(LoggerFactory.getLogger(name));
  }
  public boolean isTraceEnabled() {
    return logger.isTraceEnabled();
  }
  public boolean isErrorEnabled() {
    return logger.isErrorEnabled();
  }
  public boolean isInfoEnabled() {
    return logger.isInfoEnabled();
  }
  public boolean isDebugEnabled() {
    return logger.isDebugEnabled();
  }
  public void trace(Supplier<String> message) {
    if (isTraceEnabled()) {
      logger.trace(message.get());
    }
  }
  public void debug(Supplier<String> message) {
    if (isDebugEnabled()) {
      logger.debug(message.get());
    }
```

```
  }
  public void info(Supplier<String> message) {
    if (isInfoEnabled()) {
      logger.info(message.get());
    }
  }
  public void error(String message, Throwable throwable) {
    if (isErrorEnabled()) {
      logger.error(message, throwable);
    }
  }
  public void error(String message) {
    if (isErrorEnabled()) {
      logger.error(message);
    }
  }
  public void errorOnly(Throwable throwable) {
    if (isErrorEnabled()) {
      if (throwable == null) {
        logger.error("throwable == null");
      } else {
        logger.error(throwable.getMessage(), throwable);
      }
    }
  }
  public static void resetThread() {
    LogIdentity.resetThread();
  }
}
//запросы для базы данных
package kz.greetgo.sandbox.register.dao;
import kz.greetgo.sandbox.register.model.dao.Discipline;
import kz.greetgo.sandbox.register.model.dao.Grade;
import kz.greetgo.sandbox.register.model.dao.GradeChartFilter;
import kz.greetgo.sandbox.register.model.dao.Group;
import kz.greetgo.sandbox.register.model.dao.NotificationRecord;
import kz.greetgo.sandbox.register.model.dao.Person;
import kz.greetgo.sandbox.register.model.dao.StudentJournalFilter;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;
import org.springframework.data.relational.core.sql.In;
import javax.annotation.security.PermitAll;
import java.util.Date;
import java.util.List;
import java.util.zip.DataFormatException;
public interface GradeDao {
```

```
        @Select(" select gr.id    as id, " +
          "             (ps.surname || ' ' || SUBSTRING(ps.surname FROM 1 FOR 1) || '.' ||
SUBSTRING(ps.patronymic FROM 1 FOR 1) || " +
          "       '.')    as fio, " +
          "       d.name   as discipline, " +
          "       gr.grade as grade, " +
          "       g.name   as groupName, " +
          "       gt.type  as type " +
          " " +
          "from grade gr " +
          "       join lecturer_and_discipline lad " +
          "           on gr.lecturer_discipline_id = lad.id " +
          "       join lecturer l on lad.lecturer_id = l.id " +
          "       join person p on l.person_id = p.id " +
          "       join grade_type gt on gr.grade_type_id = gt.id " +
          "       join discipline d on lad.discipline_id = d.id " +
          "       join student s on gr.student_id = s.id " +
          "       join person ps on s.person_id = ps.id " +
          "       join public.group g on s.group_id = g.id " +
          "  where lecturer_id = ${lecturer} " +
          "  and discipline_id = ${filter.discipline} " +
          "  and g.id = ${filter.groupId} " +
          "  and gr.year = ${filter.year} " +
          "  and gr.semester = ${filter.semester} " +
          "order by ps.surname, gr.id")
        List<Grade>          selectGrade(@Param("filter")          StudentJournalFilter          filter,
@Param("lecturer") Integer lecturerId);
        @Update("update grade set grade = #{grade.grade} where id = #{grade.id} ")
        void saveGrade(@Param("grade") Grade grade);
        @Select("select d.id   as id,  " +
          " d.name as name,  " +
          " d.name as code  " +
          "from person  " +
          " join lecturer l on person.id = l.person_id  " +
          " join lecturer_and_discipline on l.id = lecturer_and_discipline.lecturer_id  " +
          " join discipline d on lecturer_and_discipline.discipline_id = d.id  " +
          " left join grade g on lecturer_and_discipline.id = g.lecturer_discipline_id  " +
          "where person.id = #{personGuid}  " +
          "  and year = ${year}  " +
          "  and semester = ${semester}  " +
          "group by d.id, d.name  ")
        List<Discipline> loadDisciplines(@Param("personGuid") Integer personGuid,
                                         @Param("year") String year,
                                         @Param("semester") String semester);
        @Select("select distinct g2.id as id, g2.name as name, g2.name as code " +
          "  from person p join lecturer  on p.id = lecturer.person_id " +
          "  join lecturer_and_discipline lad on lecturer.id = lad.lecturer_id " +
          "  join grade g on lad.id = g.lecturer_discipline_id " +
          "  join student s on g.student_id = s.id " +
```

```
            "  join public.group g2 on s.group_id = g2.id " +
            "  where p.id = #{personGuid} " +
            "  and lad.discipline_id = #{disciplineId} ")
          List<Group>          loadGroups(@Param("personGuid")          Integer          personGuid,
@Param("disciplineId") Integer disciplineId);
          @Select("select now();")
          Date loadCurrentDate();
          @Select("select year from grade group by year;")
          List<String> loadValidYear();
         @Select(" select id from lecturer where person_id = #{personGuid}")
          Integer getLecturerId(@Param("personGuid") Integer personGuid);

          @Select("select * from person ps where (ps.surname ||" +
            " ' ' || SUBSTRING(ps.surname FROM 1 FOR 1) ||" +
            " '.' || SUBSTRING(ps.patronymic FROM 1 FOR 1) || '.') = #{fio} ")
          Person loadPersonByFio(@Param("fio") String fio);
          @Select("select id from role where name = #{role}")
          Integer getRoleId(@Param("role") String role);
          @Select("select id from notification_type where name = #{type}")
          Integer getNotificationTypeId(@Param("type") String type);
          @Insert(" insert into notification" +
            " (id, ru, en, kz, type_id, role_id, state, \"publishDate\", \"publishType\") " +
            "values (DEFAULT, #{notification.ru}, #{notification.en},#{notification.kz}, " +
            " #{typeId}," +
            "  #{roleId}, " +
            " 'TOSEND', now(), 1) ")
          void saveNotification(@Param("notification") NotificationRecord notification,
                                @Param("typeId") Integer typeId,
                                @Param("roleId") Integer roleId);
          @Select("select n.id, " +
            "       nt.name as type, " +
            "       kz, " +
            "       ru, " +
            "       en, " +
            "       r.name as role, " +
            "       pt.name as publishType, " +
            "       n.\"publishDate\" as publishDate, " +
            "       state " +
            "from notification n " +
            "       join publish_type pt on n.\"publishType\" = pt.id " +
            "       join notification_type nt on n.type_id = nt.id " +
            "       join role r on n.role_id = r.id")
          List<NotificationRecord> loadNotificationList();
          @Select(" select distinct d.id as id, " +
            "              d.name as name, " +
            "              d.name as code " +
            "from grade " +
            "       join student s on grade.student_id = s.id " +
            "       join person p on s.person_id = p.id " +
```

```
            "      join lecturer_and_discipline lad on grade.lecturer_discipline_id = lad.id " +
            "      join discipline d on lad.discipline_id = d.id " +
            " where p.id = ${filter.personId} " +
            "  and grade.year = ${filter.year} " +
            "  and grade.semester = ${filter.semester} ")
          List<Discipline> loadDisciplinesForStudent(@Param("filter") GradeChartFilter filter);
          @Select(" select grade.grade " +
            " from grade" +
            "  join student s on grade.student_id = s.id " +
            "  join person p on s.person_id = p.id " +
          "  join lecturer_and_discipline lad on grade.lecturer_discipline_id = lad.id " +
            "  join discipline d on lad.discipline_id = d.id " +
            "where d.name = #{discipline} " +
            "  and p.id = ${filter.personId} " +
            "  and grade.year = ${filter.year} " +
            "  and grade.semester = ${filter.semester} " +
            "  and grade.grade_type_id not in (2, 3, 4) ")
          List<Double> loadSrGradeByDiscipline(@Param("discipline") String discipline,
@Param("filter") GradeChartFilter filter)
          @Select(" select grade.grade " +
            " from grade" +
            "  join student s on grade.student_id = s.id " +
            "  join person p on s.person_id = p.id " +
            "  join lecturer_and_discipline lad on grade.lecturer_discipline_id = lad.id " +
            "  join discipline d on lad.discipline_id = d.id " +
            "where d.name = #{discipline} " +
            "  and p.id = ${filter.personId} " +
            "  and grade.year = ${filter.year} " +
            "  and grade.semester = ${filter.semester} " +
            "  and grade.grade_type_id = 2 ")
          Double loadRKFGradeByDiscipline(@Param("discipline") String discipline,
@Param("filter") GradeChartFilter filter);
          @Select(" select grade.grade " +
            " from grade" +
            "  join student s on grade.student_id = s.id " +
            "  join person p on s.person_id = p.id " +
            "  join lecturer_and_discipline lad on grade.lecturer_discipline_id = lad.id " +
            "  join discipline d on lad.discipline_id = d.id " +
            "where d.name = #{discipline} " +
            "  and p.id = ${filter.personId} " +
            "  and grade.year = ${filter.year} " +
            "  and grade.semester = ${filter.semester} " +
            "  and grade.grade_type_id = 3 ")
          Double loadRKSGradeByDiscipline(@Param("discipline") String discipline,
@Param("filter") GradeChartFilter filter);
          @Select(" select grade.grade " +
            " from grade" +
            "  join student s on grade.student_id = s.id " +
            "  join person p on s.person_id = p.id " +
```

```
            "  join lecturer_and_discipline lad on grade.lecturer_discipline_id = lad.id " +
              "  join discipline d on lad.discipline_id = d.id " +
             "where d.name = #{discipline} " +
              "  and p.id = ${filter.personId} " +
              "  and grade.year = ${filter.year} " +
              "  and grade.semester = ${filter.semester} " +
              "  and grade.grade_type_id = 4 ")
          Double loadRKEXAMGradeByDiscipline(@Param("discipline") String discipline,
@Param("filter") GradeChartFilter filter);
}
```