Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. каф. ЭВМ

\_\_\_\_\_\_\_\_\_\_ Д. И. Самаль

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

на тему

WEB – ПРИЛОЖЕНИЕ ДЛЯ МОДЕЛИРОВАНИЯ ЛОГИЧЕСКИХ СХЕМ

БГУИР ДП 1–40 02 01 01 012 ПЗ

Студент А. А. Волчанин

Руководитель С. А. Остроухова

Консультанты:

от кафедры ЭВМ С. А. Остроухова

по экономической части В. А. Палицын

Нормоконтролер А. С. Сидорович

Рецензент

МИНСК 2016

РЕФЕРАТ

Дипломный проект предоставлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата А1. Пояснительная записка: 104 страницы, 17 рисунков, 5 таблиц, 16 литературных источников, 3 приложения.

Ключевые слова: web-приложение, логическая схема, моделирование, графический редактор, ASP.NET MVC, HTTP, JSON, JavaScript, MS SQL Server, Entity Framework.

Объектом исследования является процесс построения и моделирования логических схем. Объектом разработки является приложение, предоставляющее инженерам возможности построения и моделирования логических схем, публикации и обсуждения проектов.

Целью проекта является разработка web-приложения для моделирования логических схем, которое позволяет просматривать внутреннее состояние схемы во время процесса моделирования в удобной графической форме.

При разработке web-приложения для моделирования логических схем использовалась среда разработки Visual Studio 2015 с фреймворком для создания веб-приложений ASP.NET MVC 5.

В результате работы над дипломным проектом было разработано приложение, предоставляющее возможности построения и моделирования логических схем, публикации и обсуждения проектов. Данные возможности облегчает и увеличивают эффективность процесса обучения.

Областью практического применения данного приложения является использование при обучении в высших учебных заведениях. Пользователями программы преимущественно будут студенты инженерных специальностей и их преподаватели.

Разработанный программный продукт можно считать экономически эффективным, и он полностью оправдывает вложенные в него средства.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития программы и увеличение её функционала.

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: нет.

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Д. И. Самаль

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2016 г.

ЗАДАНИЕ

по дипломному проекту студента

Волчанина Артёма Александровича

**1** Тема проекта: «Web - приложение для моделирования логических схем» – утверждена приказом по университету 22 февраля 2016 г. № 371-с.

**2** Срок сдачи студентом законченного проекта: 1 июня 2016 г.

**3** Исходные данные к проекту:

**3.1** Языки программирования: C#, JavaScript.

**3.2** Среда разработки: Microsoft Visual Studio 2015.

**3.3** Операционная система: Windows 10.

**3.4** Браузер: Google Chrome

**3.5** База данных: MS SQL SERVER 2014.

**4** Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. 7. Технико-экономическое обоснование дипломного проекта. Заключение. Список использованных источников. Приложения.

**5** Перечень графического материала (с точным указанием обязательных чертежей):

**5.1** Вводный плакат. Плакат.

**5.2** Web-приложение для моделирования логических схем. Схема структурная.

**5.3** Web-приложение для моделирования логических схем. Диаграмма классов.

**5.4** Web-приложение для моделирования логических схем. Модель данных.

**5.5** Web-приложение для моделирования логических схем. Диаграмма последовательности.

**5.6** Web-приложение для моделирования логических схем. Схема программы.

**5.7** Заключительный плакат. Плакат.

**6** Содержание задания по экономической части: «Технико-экономическое обоснование дипломного проекта».

ЗАДАНИЕ ВЫДАЛ В.А. Палицын

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  дипломного проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Введение, обзор литературы | 10 | 15.02 – 28.02 |  |
| Системное проектирование | 10 | 29.02 – 18.03 |  |
| Функциональное проектирование | 20 | 21.03 – 01.04 |  |
| Разработка программных модулей | 20 | 04.04 – 24.04 |  |
| Программа и методика испытаний | 10 | 25.04 – 30.04 |  |
| Выполнение задания по экономической части | 10 | 01.05 – 07.05 |  |
| Завершение оформления пояснительной записки | 10 | 08.05 – 14.05 |  |
| Подготовка документов | 10 | 15.05 – 31.05 |  |

Дата выдачи задания: 2 февраля 2016 г.

Руководитель С.А. Остроухова

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc450936172)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc450936173)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 18](#_Toc450936174)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 23](#_Toc450936175)

[3.1 Классы серверной части приложения 23](#_Toc450936176)

[3.2 Представления, возвращаемые приложением 31](#_Toc450936177)

[3.3 Структура таблиц базы данных 32](#_Toc450936178)

[3.4 Модули клиентской части приложения 33](#_Toc450936179)

[3.5 Модели данных серверной части приложения 45](#_Toc450936180)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 48](#_Toc450936181)

[4.1 Авторизация и аутентификация 48](#_Toc450936182)

[4.2 Модель данных логической схемы 50](#_Toc450936183)

[4.3 Графический редактор схем 54](#_Toc450936184)

[4.4 Работа с комментариями 57](#_Toc450936185)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 59](#_Toc450936186)

[5.1 Модульное тестирование серверной и клиентской частей 59](#_Toc450936187)

[5.2 Обработка исключительных ситуаций 61](#_Toc450936188)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 64](#_Toc450936189)

[6.1 Регистрация и вход в систему 64](#_Toc450936190)

[6.2 Работа с проектами 65](#_Toc450936191)

[6.3 Обсуждения проектов 68](#_Toc450936192)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА 70](#_Toc450936193)

[7.1 Описание проекта 70](#_Toc450936194)

[7.2 Расчёт сметы затрат и цены ПО 70](#_Toc450936195)

[7.3 Расчёт экономического эффекта от применения программного средства у пользователя (заказчика) 76](#_Toc450936196)

[ЗАКЛЮЧЕНИЕ 81](#_Toc450936197)

[СПИСОК ИCПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 82](#_Toc450936198)

[ПРИЛОЖЕНИЕ А Исходный текст модулей клиентской части приложения 83](#_Toc450936199)

[ПРИЛОЖЕНИЕ Б Спецификация дипломного проекта 103](#_Toc450936200)

[ПРИЛОЖЕНИЕ В Ведомость документов 104](#_Toc450936201)

ВВЕДЕНИЕ

Процесс создания аппаратной части любого электронно-вычислительного устройства начинается с разработки схемы на уровне логических вентилей. С увеличением количества элементов схемы возрастает и сложность разработки конечного устройства - инженерам становится всё сложнее учитывать все нюансы работы устройства. Чтобы упростить задачу проектирования и уменьшить временные затраты на разработку, люди стали создавать и использовать специализированные системы проектирования. Эти системы позволяют проводить моделирование процесса работы устройства и избегать ошибок при разработке.

Логическое моделирование представляет собой процедуру проверки функционирования логической схемы с помощью компьютера. Основная цель моделирования состоит в том, чтобы проверить функцию проектируемой логической схемы без ее физической реализации, поскольку после изготовления схемы внесение изменений в нее при современной технологии сделать нелегко и недешево. Верификация выполняется путем сравнения результатов моделирования, полученных для проектируемого устройства, со спецификацией. При этом проверяются как логические функции, так и временные соотношения.

На текущий момент системы проектирования и моделирования играют важную роль при разработке вычислительных машин.

Данный дипломный проект является web-приложением, предназначенным для создания, всестороннего изучения и анализа различных схем на логическом уровне. С помощью разработанного приложения можно описывать схемы графическим и текстовым способом, добавлять новые логические элементы и изменять их характеристики.

Большинство профессиональных систем проектирования в первую очередь предназначены для продвинутых пользователей. В то же время у начинающих разработчиков нет возможности получить наглядную информацию о работе какого-либо устройства.

Современные системы проектирования содержат в себе огромную функциональность, которая не полностью используется начинающими проектировщиками и на начальных этапах работы с конкретной системой. Вследствие этого пользователи вынуждены тратить большие объёмы памяти на своих ПК. Так же возникает привязанность к определённой операционной системе.

Актуальность системы, разработанной в этом дипломном проекте, в первую очередь представляет возможность наблюдения внутренних процессов работы проектируемого устройства. Начинающие проектировщики могут наблюдать процесс работы устройства без необходимости кропотливого изучения сигналов на линиях схемы.

Цель разработки – создать эффективное, быстрое и интуитивно понятное приложение, с помощью которого начинающие проектировщики смогут проанализировать процесс работы устройства.

Для осуществления обозначенной цели служат следующие задачи:

* система не должна зависеть от операционной системы;
* система должна содержать графический редактор схем;
* система должна поддерживать повторное использование уже созданных элементов;
* пользователи должны иметь возможность просмотреть внутреннее состояние схемы во время процесса моделирования;
* пользователи должны иметь возможность просмотреть результаты моделирования схемы на любой входной последовательности сигналов.

Необходимость в разработке описанной системы прежде всего обусловлена увеличением спроса на инженеров аппаратурной отрасли, который в свою очередь ведёт к увеличению студентов, обучающихся на специальностях, связанных с проектированием и разработкой аппаратных средств.

1. ОБЗОР ЛИТЕРАТУРЫ

Самыми популярными системами проектирования с возможностью моделирования логических схем являются зарубежные продукты Xilinx ISE[1], Altera Quartus, Altera MAX+PLUSII [2].

Вышеперечисленные системы используют симулятор ModelSim[3], который стал фактически промышленным стандартом. Он предназначен для симуляции и верификации проектов интегральных схем, созданных на основе их HDL описаний. Можно выделить следующие отличительные особенности симулятора ModelSim:

* полная поддержка основных стандартов языков Verilog, VHDL, SystemC и их расширений;
* единое ядро моделирования пакета, обеспечивающее возможность полной отладки смешанных проектов, которые одновременно содержат модули, написанные на VHDLи Verilog;
* использование принципа оптимизированной прямой компиляции. В соответствии с этим принципом исходные описания компилируются в машинно-независимый объектный код, исполняемый на любой поддерживаемой платформе;
* открытая архитектура программных средств, которая обеспечивает тесную интеграцию с пакетами САПР сторонних компаний. Средства управления предоставляют возможность организации прямого доступа к моделирующему ядру ModelSim, загрузки информации о выполнении процесса моделирования и его результатов в среду используемой САПР и управления работой системы через интерфейс применяемых средств проектирования;
* возможность работы в различных режимах, в том числе и пакетном. Разработав и отладив некоторый сценарий моделирования в интерактивном режиме, можно оформить его для последующего использования в виде пакетного командного файла.

В процессе изучения дисциплины «Контроль и диагностика средств вычислительной техники» очень часто использовалась система VLSI-SIM. Данная система создавалась в Белорусском Государственном Университете и обладает мощными средствами моделирования схем. Система написана на языке Delphi и использует Borland Database Engine для управления данными. Пользователи данной системы сталкиваются с множеством трудностей при установке приложения – необходима установка сторонних утилит, приложение не поддерживает x64 архитектуру.

Отличительно особенностью данных приложений является то, что все они требуют установки на компьютер пользователя. В современном мире наблюдается тенденция роста популярности web-приложений и спада популярности классических (десктопных) приложений.

Поэтому, чтобы упростить процесс работы с разрабатываемой системой, было решено использовать web-технологии. Использование данного подхода избавляет пользователей от многих хлопот и позволяет разработчикам оперативно вносить обновления в продукт.

Для написания серверной части приложения была использована технология ASP.NET MVC. Эта технология является частью Microsoft .NET Framework и предоставляет различные инструменты для создания приложений. В качестве языка для написания серверной части был выбран язык C#.

Написание клиентской части приложения подразумевает использование языка JavaScript и технологии которая появились в HTML5 - Canvas.

.NET Framework – программная платформа, разработанная Microsoft. Существует множество альтернативных технологий, призванных облегчить создание архитектуры и реализацию исходного кода приложений.

Microsoft Foundation Classes (MFC) – технология, применяемая в качестве уровня абстрагирования для языка C++. Используется при создании графического пользовательского интерфейса для операционных систем Windows. Используя MFC, разработчики избавляются от работы с циклами обработки сообщений, оконными процедурами и т.п. [4].

Java и J2EE – объектно-ориентированный межплатформенный язык программирования и платформа на его основе для создания приложений. Java-программы транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор, но с тем отличием, что байтовый код, в отличие от текста, обрабатывается значительно быстрее [5].

Active Server Pages (ASP) – технология, которая служит для абстрагирования при создании активных и динамических web-сайтов с использованием Visual Basic Script или JScript. Эта технология позволила разработчикам абстрагироваться от особенностей сетевых взаимодействий и больше внимания уделять содержанию страниц.

.NET Framework позволяет вести разработку как классических десктопных программ, так и web-приложений. Можно сказать, что данная платформа представляет собой высокопроизводительную виртуальную машину, которая выполняет код промежуточный код, который транслируется из различных языках программирования.

Решение от Microsoft состоит из двух частей: общеязыковой исполняющей среды Common Language Runtime (CLR) и библиотеки классов Framework Class Library (FCL). Основой платформы является [виртуальная машина](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) [CLR](http://ru.wikipedia.org/wiki/Common_Language_Runtime), способная интерпретировать промежуточный код. Одной из основных идей Microsoft .NET является совместимость различных служб, написанных на разных языках. Разработку для .NET Framework можно вести на различных языках программирования, например C#, Visual Basic .NET, Jscript .NET, C++/CLI, F#, J# [4].

Так же, как и технология Java, среда разработки .NET создаёт промежуточный код, предназначенный для исполнения виртуальной машиной. Входной язык этой машины в .NET называется Microsoft Intermediate Language (MSIL), или Common Intermediate Language (CIL). Применение промежуточного кода позволяет получить кроссплатформенность на уровне скомпилированного проекта (в терминах .NET – сборка), а не только на уровне исходного текста, как, например, в С. Перед запуском сборки в среде исполнения CLR, байт-код преобразуется встроенным в среду JIT-компилятором в машинные коды целевого процессора.

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке FCL. Туда входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL).

Base Class Library (BCL) – стандартная библиотека классов платформы .NET Framework. Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL – создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и так далее.

C# – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах в компании Microsoft как один из языков разработки приложений для платформы Microsoft .NET Framework. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Описание базовых принципов функционирования платформы .NET, системы типов .NET и различных инструментальных средств разработки, используемых при создании приложений, базовые возможности языка программирования С#, включая новые синтаксические конструкции, появившиеся с выходом .NET 4.5, а также синтаксис и семантика языка CIL подробно описаны в [1]. В качестве наиболее полного руководства по C#, где описаны базовые конструкции и общие правила языка можно привести [6].

В процессе написания любого программного продукта приходится неоднократно сталкиваться с вопросами использования тех или иных функций. В этом случае незаменима электронная документация к .NET Framework [6], предлагаемая разработчиками компании Microsoft и входящая в пакет Visual Studio. Здесь приведены сведения об использовании классов системных библиотек и поведении конкретных методов.

Наравне с документацией [7], использовалась спецификация языка программирования C# [8], которая по праву считается первоисточником для изучения языка С# и платформы .NET Framework.

ASP.NET – технология создания web-приложений и web-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. На данный момент последней версией этой технологии является ASP.NET 5.

В ASP.NET WebForms разработчики Microsoft попытались сокрыть как протокол HTTP (с его неизбежным отсутствием состояния), так и язык HTML (который на тот момент был незнаком многим разработчикам), моделируя интерфейс пользователя в виде иерархии серверных объектов, представляющих элементы управления. Каждый такой элемент управления отслеживает собственное состояние между запросами (с помощью средства ViewState по мере необходимости визуализируя себя в виде HTML – разметки, и автоматически подключая события клиентской стороны (например, щелчки на кнопках) с соответствующим кодом их обработки на стороне сервера). Фактически WebForms – это гигантский уровень абстракции, разработанный для воссоздания классического, управляемого событиями графического пользовательского интерфейса в web-среде.

Идея состояла в том, чтобы web-разработка выглядела подобно разработке Windows Forms. Отныне разработчикам не нужно иметь дело с сериями независимых запросов и ответов HTTP; теперь можно мыслить терминами сохраняющего свое состояние интерфейса пользователя. Можно забыть о web-среде и её не поддерживающей состояние природе, а вместо этого строить пользовательские интерфейсы с помощью визуального конструктора, использующего технологию перетаскивания, и полагать – или, по меньшей мере, делать вид – что все происходит на сервере.

ASP.NET MVC – это платформа для разработки web-приложений от Microsoft, которая сочетает в себе эффективность и аккуратность архитектуры «модель-представление-контроллер», новейшие идеи и приемы гибкой разработки, а также все лучшее из существующей платформы ASP.NET. Она представляет собой полномасштабную альтернативу традиционной технологии ASP.NET WebForms, предоставляя существенные преимущества для всех проектов web-разработки, кроме наиболее тривиальных. Новая платформа ASP.NET MVC обеспечила радикальный сдвиг в разработке web-приложений на платформе Microsoft. В ней делается упор на ясную архитектуру, шаблоны проектирования и тестируемость, и не предпринимается попыток сокрытия того, как работает web-среда.

Со временем использование ASP.NET WebForms в реальных проектах показало некоторые их недостатки:

ViewState: В результате использования актуального механизма для поддержки состояния между клиентом и сервером передаются большие блоки данных. Эти данные могут достигать сотен килобайт даже для небольших приложений. Данные передаются при каждом запросе, что приводит к увеличению времени отклика и повышению требований к пропускной способности сервера.

Жизненный цикл страницы: Механизм для объединения события со стороны клиента с кодом серверного обработчика события может быть чрезвычайно сложным.

Неправильное разделение задач: Модель выделенного кода (code-behind) ASP.NET предоставляет возможность для того, чтобы вынести код приложения за рамки HTML разметки в отдельный класс выделенного кода. Это широко приветствовалось из-за разделения логики и представления, но, в действительности, разработчики вынуждены смешивать код представления с логикой приложения (например, управлением базами данных) в этих же классах выделенного кода.

Ограниченные возможности с HTML: До версии ASP.NET 4 выходным данным HTML не удавалось соответствовать web стандартам или хорошо работать с каскадными таблицами стилей (CSS). Также серверные элементы управления генерировали непредсказуемые и сложные значения атрибута ID, к которым трудно получить доступ при помощи JavaScript. Эти проблемы во многом решились в ASP.NET 4.

Абстракции: WebForms пытается спрятать HTML и HTTP, где это только возможно.

Слабая тестируемость: Разработчики ASP.NET не могли предположить, что автоматизированное тестирование станет важным компонентом разработки программного обеспечения. Жёсткая архитектура не подходит для модульного тестирования.

Термин модель-представление-контроллер (model-view-controller) используется с конца 70-х годов прошлого столетия. Эта модель явилась результатом проекта Smalltalk в компании Xerox, где она была задумана как способ организации некоторых из ранних приложений графического пользовательского интерфейса. Некоторые из нюансов первоначальной модели MVC были связаны с концепциями, специфичными для Smalltalk, такими как экраны и инструменты, но более глобальные понятия все еще применимы к приложениям, и особенно хорошо они подходят для web-приложений[9].

Концепция паттерна (шаблона) MVC (model-view-controller) предполагает разделение приложения на три компонента [10]:

Контроллер (controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.

Представление (view) – это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт. Модель (model) представляет класс, описывающий логику используемых данных.

Схема паттерна MVC изображена на рисунке 1.1.

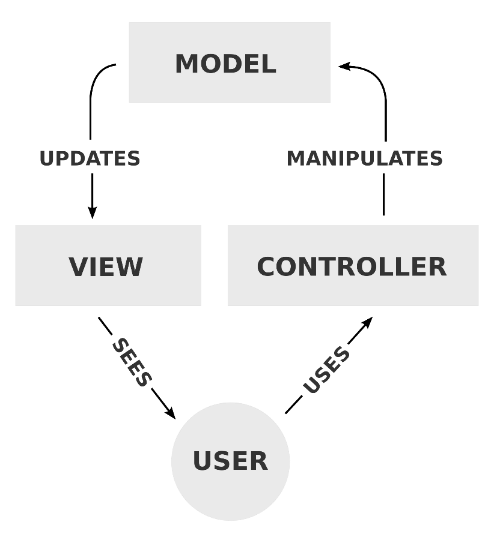


Рисунок 1.1 – Шаблон разработки MVC[11]

В ASP.NET MVC контроллеры являются классами, обычно производными от класса System.Web.Mvc.Controller. Каждый метод public в классе, унаследованном от класса Controller, называется методом действия(Action) и посредством системы маршрутизации ASP.NET связан с конфигурируемым URL. Когда запрос отправляется поURL, связанному с методом действия, в классе контроллера выполняются операторы, чтобы провести некоторую операцию по отношению к модели и затем выбрать представление для отображения клиенту.

В отличие от ASP.NET WebForms платформа ASP.NETMVC имеет следующие преимущества [10]:

* облегчает управление сложными структурами путем разделения приложения на модель, представление и контроллер;
* не использует состояние просмотра и серверные формы, что делает платформу MVC идеальной для разработчиков, которым необходим полный контроль над поведением приложения;
* использует схему основного контроллера, при которой запросы web-приложения обрабатываются через один контроллер, что позволяет создавать приложения, поддерживающие расширенную инфраструктуру маршрутизации;
* обеспечивает расширенную поддержку разработки на основе тестирования;
* хорошо подходит для web-приложений, поддерживаемых крупными коллективами разработчиков, а также web-разработчикам, которым необходим высокий уровень контроля над поведением приложения.

Платформа ASP.NETMVC предоставляет следующие возможности:

1. Разделение задач приложения (логика ввода, бизнес-логика и логика пользовательского интерфейса), широкое возможности тестирования и разработки на основе тестирования. Все основные контракты платформы MVC основаны на интерфейсе и подлежат тестированию с помощью макетов объекта, которые имитируют поведение реальных объектов приложения. Приложение можно подвергать модульному тестированию без запуска контроллеров в процессе ASP.NET, что ускоряет тестирование и делает его более гибким. Для тестирования возможно использование любой платформы модульного тестирования, совместимой с .NET Framework.
2. Расширяемая и дополняемая платформа. Компоненты платформы ASP.NET MVC можно легко заменить или настроить. Разработчик может подключать собственный механизм представлений, политику маршрутизации URL-адресов, сериализацию параметров методов действий и другие компоненты. Платформа ASP.NET MVC также поддерживает использование моделей контейнера внедрения зависимости (DI) и инверсии управления (IOC). Модель внедрения зависимости позволяет внедрять объекты в класс, а не ожидать создания объекта самим классом. Модель инверсии элемента управления указывает на то, что если один объект требует другой объект, то первые объекты должны получить второй объект из внешнего источника (например, из файла конфигурации). Это облегчает тестирование.
3. Расширенная поддержка маршрутизации ASP.NET. Этот мощный компонент сопоставления URL-адресов позволяет создавать приложения с понятными URL-адресами, которые можно использовать в поиске. URL-адреса не должны содержать расширения имен файлов и предназначены для поддержки шаблонов именования URL-адресов, обеспечивающих адресацию, оптимизированную для поисковых систем (SEO) и для передачи репрезентативного состояния (REST).
4. Поддержка использования разметки в существующих файлах страниц ASP.NET (ASPX), элементов управления (ASCX) и главных страниц (MASTER) как шаблонов представлений. Вместе с платформой ASP.NET MVC можно использовать существующие функции ASP.NET, например, вложенные главные страницы, декларативные серверные элементы управления, шаблоны, привязку данных, локализацию и другие.
5. Поддержка существующих функций ASP.NET. ASP.NET MVC позволяет использовать такие функции, как проверка подлинности с помощью форм и Windows, проверка подлинности по URL-адресу, членство и роли, кэширование вывода и данных, управление состоянием сеанса и профиля, наблюдение за работоспособностью, система конфигурации и архитектура поставщика.

Платформа ASP.NET MVC предоставляет поддержку для выбора механизмов визуализации. В более ранних версиях MVC использовался стандартный механизм визуализации ASP.NET, который обрабатывал ASPX‑страницы с применением оптимизированной версии синтаксиса разметки WebForms. В версии платформы MVC 3 был введен механизм визуализации Razor.

При обработке запросов ASP.NET MVC Framework опирается на систему маршрутизации, которая сопоставляет все входящие запросы с определенными в системе маршрутами, которые указывают какой контроллер и метод должен обработать данный запрос. Встроенный маршрут по умолчанию предполагает трехзвенную структуру: контроллер/действие /параметр [12].

Одной из наиболее популярных книг по ASP.NETMVC является [9]. Данная книга даёт подробное описание архитектуры ASP.NETMVC 5, а также инфраструктуры ASP.NET MVC Framework.

В проекте активно используется взаимодействие с базой данных MS SQL (Microsoft SQL Server). Однако использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя писать программное обеспечение, которое должно уметь как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создает трудности при программировании, так как обе формы данных накладывают ограничения друг на друга.

Одним из решений проблемы семантического разрыва при использовании реляционной базы данных является Object-Relational Mapping (ORM) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

В качестве ORM-решения в проекте используется ADO.NET Entity Framework - решение для .NET Framework от Microsoft. Данная библиотека распространяется в виде NuGet пакета и предоставляет возможность взаимодействия с объектами как посредством [LINQ](https://ru.wikipedia.org/wiki/LINQ) в виде LINQ to Entities, так и с использованием Entity SQL[4].

EntityFramework представляет специальную объектно-ориентированную технологию на базе .NET Framework для работы с данными[13]. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то EntityFramework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, то на концептуальном уровне, который нам предлагает EntityFramework, мы уже работает с объектами.

Изначально с самой первой версии Entity Framework поддерживал подход DatabaseFirst, который позволял по готовой базе данных сгенерировать модель edmx. Затем эта модель использовалась для подключения к базе данных. Позже был добавлен подход ModelFirst. Он позволял создать вручную с помощью визуального редактора модель edmx, и по ней создать базу данных. Начиная с 5.0 предпочтительным подходом становится CodeFirst. Его суть - сначала пишется код модели на С#, а затем по нему генерируется база данных. При этом модель edmx уже не используется.

Язык программирования JavaScript[14] разработан фирмой Netscape для создания интерактивных HTML-документов. Это прототипно-ориентированный сценарный язык разработки встраиваемых приложений, выполняющих как на стороне клиента, так и на стороне сервера. Синтаксис языка очень похож на синтаксис языка Java – поэтому его часто называют Java-подобным. Клиентские приложения выполняются браузером просмотра web-документов на машине пользователя, серверные приложения выполняются на сервере.

При разработке обоих типов приложений используется общий компонент языка, называемый ядром и включающий определения стандартных объектов и конструкций (переменные, функции, основные объекты и средство LiveConnect взаимодействия с Java-апплетами), и соответствующие компоненты дополнений языка, содержащие специфические для каждого типа приложений определения объектов.

Клиентские приложения непосредственно встраиваются в HTML-страницы и интерпретируются браузером по мере отображения частей документа в его окне. Серверные приложения для увеличения производительности предварительно компилируются в промежуточный байт-код.

Структурно JavaScript можно представить в виде объединения трёх чётко различимых друг от друга частей:

* ядро (ECMAScript);
* объектная модель браузера (Browser Object Model);
* объектная модель документа (Document Object Model).

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться.

ECMAScript не является браузерным языком и в нём не определяются методы ввода и вывода информации. Это, скорее, основа для построения скриптовых языков. Спецификация ECMAScript описывает типы данных, инструкции, ключевые и зарезервированные слова, операторы, объекты, регулярные выражения, не ограничивая авторов производных языков в расширении их новыми составляющими.

Объектная модель браузера — браузер-специфичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом window, центральным объектом DOM.

Объектная модель документа — интерфейс программирования приложений для HTML и XML-документов. Согласно DOM, документ (например, web-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции.

При разработке клиентской части системы был применён подход AMD (Asynchronous module definition) — это подход к разработке на Javascript, который позволяет создавать модули таким образом, чтобы они и их зависимости могли быть загружены асинхронно[15]. Асинхронная загрузка модулей позволяет улучшить скорость загрузки Web страницы в целом, так как модули загружаются одновременно с остальным контентом сайта.

В качестве модульного загрузчика был выбран RequireJS[16] – один из самых популярных загрузчиков. Он может быть использован не только при разработке браузерного приложения, в качестве окружения может быть использован Rhinoили Node.

Отдельно стоит отметить относительно новый элемент WEB-API, которыйпоявился вместе с HTML5 – Canvas. Это элемент для рисования графики при помощи JavaScript. Для аппаратного ускорения используется WebGL.

Клиентская часть системы построена по архитектуре SPA (Single Page Application). Особенностью данной архитектуры является то, что такое приложение использует единственный html документ как оболочку для всех web-страниц и организует взаимодействие с сервером посредством REST запросов. Приложения такого типа появились достаточно недавно и завоевали популярность у разработчиков.

При таком подходе к разработке, большинство кода выполняется на стороне клиента, что в значительной степени уменьшает требования к аппаратному обеспечению сервера.

Единожды загрузив скрипты, браузер их кэширует, и при следующем запуске приложения отпадает необходимость повторной загрузки кода. Тем самым уменьшается сетевая загрузка и повышается скорость работы приложения в целом.

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Согласно логике работы приложения, пользователю необходимо зарегистрироваться в системе, после чего он сможет начать создавать логическую схему из простейших элементов.

В системе присутствует графический редактор схем, который упрощает процесс проектирования устройства.

Помимо стандартных встроенных логических элементов пользователь может сам создавать и сохранять новые логические элементы, которые в смогут использоваться при построении схемы.

После создания схемы пользователь сможет приступить к её моделированию. Для моделирования схемы необходимо задать входные сигналы и выбрать режим (ручной или автоматический). После этого пользователю будет доступен результат.

При ручном режиме моделирования пользователь сможет наблюдать процессы, которые протекают в схеме.

При использовании автоматического режима пользователю будут доступны только конечные выходные последовательности.

Исходя из вышеперечисленных условий, можно выделить следующие блоки приложения:

1. Блок пользовательского API.
2. Блок авторизации.
3. Блок работы с базой данных.
4. База данных.
5. Блок «Ядро».
6. Блок работы со схемой.
7. Блок работы с элементами.
8. Блок работы с сигналами.
9. Блок моделирования схемы.
10. Блок работы с комментариями.

На каждый логически выделенный блок программы возлагаются определенные задачи. Кроме того, каждый блок программы так или иначе связан с некоторыми остальными блоками, чтобы обеспечить работоспособность всего приложения в целом. Связь, как правило, реализуется посредством обмена данными между блоками.

Согласно выбранной архитектуре – SPA (Single Page Application), основная часть логики приложения будет выполнятся на стороне клиента. Серверная часть лишь предоставляет доступ к ресурсам клиентской части приложения (html страница, JS файлы, таблицы стилей) и предоставляет API для работы базовыми операциями хранения данных приложения (загрузить/сохранить схему, элемент, последовательность сигналов). Также на стороне сервера размещается блок авторизации. Блок авторизации используется для обеспечения авторизации пользователей и ограничения доступа к функциям приложения неавторизованных пользователей.

Приложение использует доступ к базе данных, соответственно выделен блок работы с базой данных. Таким образом, обобщённая структура приложения представлена на чертеже ГУИР.400201.012 C1.

*Блок пользовательского API* является основным блоком серверной части приложения. Данный блок предоставляет конечные точки для HTTP запросов. Также данный блок предоставляет 2 web-страницы: страницу авторизации и собственно страницу пользовательского интерфейса. Также данный блок поддерживает API методы, с помощью которых клиентская часть системы сможет загружать и сохранять данные. Данному блоку в ASP.NET MVC соответствуют контроллеры, которые являются центральными компонентами в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки, например, в виде представлений. Также к каждому данному блоку относятся наборы моделей и соответствующих представлений. Можно выделить основные команды, которые поддерживает данный блок:

* получить страницу (включая html, JavaScript, CSS);
* зарегистрировать пользователя;
* авторизовать пользователя;
* получить список проектов;
* создать/получить/сохранить/удалить проект;
* получить/сохранить данные проекта;
* получить список сохранённых логических элементов;
* создать/получить/сохранить/удалить логический элемент.

*Блок авторизации* используется для ограничения доступа к функциям приложения неавторизованных пользователей. Блок представлен фильтром авторизации AuthorizeAttribute. Фильтр в ASP.NETMVC представляет собой некоторый класс, реализующий логику, которая должна отрабатывать до вызова действий контроллера. Фильтры реализованы как атрибуты, которые могут применять ко всему классу-контроллеру, так и к отдельным его методам, свойствам и полям. Фильтры авторизации срабатывают после фильтров аутентификации и до запуска остальных фильтров и вызова методов действий. Цель фильтров авторизации – разграничить доступ пользователей, чтобы к определенным ресурсам приложения имели доступ только определенные пользователи. Фильтры авторизации реализуют интерфейс IAuthorizationFilter. И если при получении запроса окажется, что к запрашиваемому действию контроллера применяется данный фильтр, то сначала срабатывает метод OnAuthorization данного интерфейса. И если фильтр одобрит запрос, то далее вызывается действие. Иначе действие не будет работать. Данный класс реализует проверку, аутентифицирован и авторизован ли пользователь, с использованием аутентификации форм (FormsAuthentication).

Аутентификация с помощью форм – это система аутентификации ASP.NET общего назначения. Посредством аутентификации с помощью форм платформа ASP.NET создает cookie-набор безопасности для зарегистрированных пользователей, обслуживает их и автоматически поддерживает контекст безопасности для последующих запросов. Аутентификация с помощью форм основана на билетах (также называемых маркерами). Это значит, что, когда пользователь регистрируется, он получает так называемый билет с базовой информацией о себе. Информация сохраняется в зашифрованном cookie-наборе, который присоединяется к ответу, так что автоматически отправляется в каждом последующем запросе.

При первом доступе к приложению пользователю необходимо пройти регистрацию, где необходимо ввести логин и пароль. После этого данный пользователь сохраняется в системе и может использовать функции приложения.

Стоит отметить, что для регистрации и авторизации в приложении выделена отдельная страница, на которой пользователь должен ввести логин и пароль.

В целях повышения безопасности, в приложении реализована поддержка защищённого HTTP соединения – HTTPS (HyperText Transfer Protocol Secure). Данные, передаваемые по протоколу HTTPS упаковываются в криптографический протокол SSL или TLS. Конфигурация защищённого соединения хранится в блоке авторизации.

*Блок работы с базой данных* позволяет обращаться к таблицам в базе данных и выполнять с ними различные операции: чтения и поиск, вставка и удаление объектов. Работа с базой данных производится с использованием технологии Object-Relational Mapping (ORM), которая связывает таблицы базы данных с классами бизнес-объектов приложения. В качестве ORM-решения используется библиотека EntityFramework. В данном блоке описаны все сущности которыми оперирует система. Ключевым понятием является EntityDataModel. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. EntityDataModel содержит в себе три уровнея:

* концептуальный;
* уровень хранилища;
* уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

*База данных* является центральным хранилищем данных в системе. В качестве системы управления базами данных (СУБД) было решено использовать решение от Microsoft. MS SQL Server предназначен для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия, что является одним из факторов при выборе СУБД для использования в приложении.

База данных содержит в себе следующие таблицы:

* таблица пользователей;
* таблица проектов;
* таблица комментариев.

Блоки клиентской части системы выполнены в виде отдельных JavaScript модулей, которые браузер загружает с сервера. Данный подход позволяет выполнять расширение системы без изменения уже существующих модулей.

*Блок работы со схемой* по сути является основным блоком, с которым взаимодействует пользователь. Модуль предоставляет инструменты для создания схем из различных логических элементов. Данный блок отображается в виде отдельного холста, на которой пользователь может добавить логические элементы и составить связи между элементами. Все операции происходят посредством управления мышью. Также данный блок отвечает за отображение состояния схемы во время моделирования в ручном режиме. Связи между элементами подсвечиваются различным цветом в зависимости от логического уровня на выходах элементов. Такой подход позволяет установить неверно работающие части схемы без необходимости постоянного слежения за логическими уровнями сигналов, которые отображаются в отдельной области пользовательского интерфейса.

*Блок работы с элементами* позволяет пользователю просмотреть логические элементы и производить различные манипуляции - создать, добавить на схему, сохранить, удалить. Данный блок отображается в отдельной части страницы. Элементы отображаются в виде прямоугольников, которые пользователь может добавить на схему. Описание поведения нового элемента выполняется в отдельном всплывающем окне. Логика работы логического элемента задаётся посредством языка JavaScript. Пользователю необходимо описать поведение элемента, после чего данный элемент может быть сохранён и использован в проекте.

*Блок работы с сигналами* позволяет пользователю формировать различные входные воздействия для логической схемы. Также этот блок отвечает за отображение выходных сигналов при моделировании схемы. Данный блок отображается в отдельной части страницы. Каждый сигнал отображается в виде поля, содержащего последовательности, состоящие из нулей и единиц. Поля располагаются друг под другом, что позволяет просматривать состояние сигналов в определённый момент времени.

*Блок моделирования схемы* позволяет проводить моделирование схемы на различных входных последовательностях. Данный блок получает схему с блока работы со схемой и входную последовательность с блока работы с сигналами. После этого в данном блоке строится машинное представление схемы и подаются на входы подаются сигналы. Данный модуль поддерживает 2 типа моделирования – ручное и автоматическое. Ручной режим позволяет просмотреть поведение схемы в режиме реального времени. Пользователь может увидеть состояние схемы а не только конечный результат моделирования. В автоматическом режиме пользователю будут доступны только выходные сигналы схемы. Результаты ручного моделирования отображаются в блоке работы со схемой, автоматического моделирования – в блоке работы с сигналами.

*Блок “Ядро”* является основной частью клиентской части системы. Данный модуль отвечает за загрузку и совместную работу клиентских модулей. Каждый модуль связан с ядром, через которое происходит передача данных между блоками приложения. Также в данном блоке реализованы операции связи с сервером – здесь формируются HTTP запросы, результаты которых передаются соответствующему модулю приложения. Передача данных между модулями организована через специальный механизм передачи сообщений, основанный на событиях. Разработанная архитектура модуля «ядро» позволяет расширять функциональность системы путём добавления новых функциональных модулей.

*Блок работы с комментариями* отвечает за создание и отображение пользовательских комментариев. Для каждого проекта система создаёт дискуссию, в которой пользователи могут принимать участие. Данный блок взаимодействует с блоком пользовательского API для проведения операций с базой данных.

Таким образом, приложение, построенное по данной архитектуре, обладает следующими преимуществами:

* модульность и расширяемость;
* безопасность;
* высокая производительность;
* низкая требовательность к аппаратным ресурсам сервера.

1. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе мы разберем и произведем подробный анализ компонентов программы, а также покажем зависимости между модулями, что обеспечит полное понимание взаимодействия модулей между собой. В этом разделе мы опишем существующие классы, познакомимся с их методами и свойствами. Помимо этого будет рассмотрено соотношение некоторых классов с соответствующими им таблицами в базе данных. Для выполнения дипломного проекта была выбрана концепция Model-View-Controller, все компоненты которой будут реализованы благодаря фреймворку Spring. В основе написания проекта лежат базовые понятия модуля Spring MVC.

Каждый раз, когда пользователь щелкает по ссылке или отправляет форму в веб-браузере, формируется запрос, который несет в себе информацию о требованиях пользователя. По крайней мере, запрос будет нести в себе запрошенный URL-адрес. Но помимо него, он может нести еще и дополнительные данные, такие как информация из формы, заполненной пользователем. Первой остановкой на пути запроса является DispatcherServlet.

Как и большинство веб-фреймворков на языке Java, фреймворк Spring MVC пропускает все входящие запросы через единственный сервлет входного контроллера. *Входной контроллер* (front controller) является типичным шаблоном проектирования веб-приложений, где единственный сервлет берет на себя ответственность за передачу всех запросов остальным компонентам приложения, выполняющим фактическую их обработку. В Spring MVC входным контроллером является DispatcherServlet.



Задача контроллера DispatcherServlet состоит в том, чтобы передать запрос контроллеру Spring MVC. Контроллер является компонентом Spring, обрабатывающий запрос. Но приложение может иметь несколько контроллеров, и входному контроллеру DispatcherServlet требуется помощь, чтобы определить, какому контроллеру передать запрос. Поэтому контроллер DispatcherServlet консультируется c одним или несколькими механизмами отображения и выясняет, где будет следующая остановка запроса. При принятии решения механизм отображения в первую очередь руководствуется адресом URL в запросе.

Как только будет выбран соответствующий контроллер, Dispatcher-

Servlet отправляет запрос в путь к выбранному контроллеру. Достигнув контроллера, запрос отдаст часть своей информации. В результате работы контроллера часто появляется некоторая информация, которая должна быть передана назад пользователю и отображена в браузере. Эта информация называется *моделью*.

Но отправки обратно необработанной информации недостаточно, перед

отправкой ее следует представить в удобном для пользователя формате, обычно в HTML. Для этого информация должна быть передана в одно из *представлений*, которыми обычно являются JSP. Последнее, что должен сделать контроллер, – упаковать вместе модель и имя представления для отображения результатов в браузере. Затем он отсылает запрос вместе с моделью и именем представления обратно входному контроллеру DispatcherServlet.

Чтобы контроллер не оказался тесно связанным с каким-либоконкретным представлением, имя представления, возвращаемое входному контроллеру DispatcherServlet, не определяет JSP-страницу непосредственно. Фактически оно даже не предполагает, что представление вообще является страницей JSP. Оно является лишь логическим именем представления, используемым затем для поиска фактического представления. Чтобы отобразить логическое имя представления в ссылку на конкретную реализацию, входной контроллер DispatcherServlet обратится к арбитру представлений (view resolver).

Теперь, когда контроллер DispatcherServlet определил, какое представление будет отображать результаты, работа запроса подошла к концу. Его конечная остановка – реализация представления (возможно, страница JSP), куда он доставит модель данных. На этом работа запроса заканчивается. На основе модели данных представление создаст отображение страницы, которое будет отправлено обратно клиенту с другим курьером – объектом ответа.

* 1. Классы серверной части приложения

К данным классам относятся прежде всего классы контроллеров. Контроллеры обрабатывают входящие запросы, вводимые пользователями данные и их действия, а также реализуют необходимую логику web-приложения.

* + 1. Класс AppController

Данный класс контроллера является стартовым для авторизованного пользователя в системе, помечен аннотацией @Controller, что относит его к слою представления. Данная аннотация служит для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework.

Класс StudentController

Данный класс контроллера реализует процесс управления студентами. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяет его роль. Данный класс предназначен для выполнения основных операций со студентами.

Поля:

* studentService – поле типа StudentService, которое нам необходимо для работы со слоем studentDAO(Data Access Object). StudentService является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин studentService, который является реализацией класса StudentServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getStudent (long id) – метод действия контроллера, отправляющий клиенту страницу c персональной информацией выбранного им студента по имеющемуюся у него идентификатору. Персональная информация по студенту включает в себя имя, фамилию, университет, факультет и группу, где учится в настоящий момент студент, а также номер комнаты и этажа, где данный студент проживает. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «student/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addStudent () – метод действия контроллера, принимающий от клиента модель с заполненными данными заселяемого студента: имя, фамилия, университет, факультет, группа, в которой обучается студент в настоящий момент, а также номер комнаты, доступный для проживания. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, студент будет успешно заселен в выбранную комнату, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/student/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST). После успешного добавления студента в систему, он будет отображен, как заселенный в комнату посредством дальнейшего вызова метода «/room/id».
* deleteStudent (long id) – метод действия контроллера, принимающий от клиента идентификатор студента. Проверяет соответствие данного идентификатора записи студента в базе данных, если проверка прошла успешно, удаляет студента из базы данных. Удалить студента может только пользователь с правами заведующей, так данное действие фактически означает выселение студента из общежития. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/student/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE). После успешного выполнения метода действия пользователь перенаправляется по адресу «/student/»;
* updateStudent (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными студента, подлежащими редактированию: имя, фамилия, университет, факультет, группа, в которой обучается студент в настоящий момент, а также номер комнаты, доступный для проживания. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, информация о выбранном студенте будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/student/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT). После успешного редактирования информации о студенте, он будет отображен с уже обновленной информацией посредством дальнейшего вызова метода «/room/{id}».
* listAllStudents () – метод действия контроллера, отправляющий клиенту страницу c персональной информацией по всем студентам, проживающим на данный момент в общежитии. Персональная информация по студенту включает в себя имя, фамилию, университет, факультет и группу, где учится в настоящий момент студент, а также номер комнаты и этажа, где данный студент проживает. Для выполнения этого действия пользователь должен быть просто авторизован в приложении(с любыми правами доступа). Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «student/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.

Класс MarkController

Данный класс контроллера реализует процесс управления оценками, выставленными за санитарное состояние комнаты в общежитии. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяется его ролью. Данный класс предназначен для выполнения основных операций над оценками, выставленными комнатам.

Поля:

* markService – поле типа markService, которое нам необходимо для работы со слоем markDAO. MarkService является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин markService, который является реализацией класса MarkServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getMark (String roomNumber) – метод действия контроллера, отправляющий клиенту страницу c информацией об оценках комнаты, в качестве параметра передается номер выбранной пользователем комнаты. Информация включает в себя номер комнаты, выставленные данной комнате оценки и соответствующие им даты. Права на данное действие предоставлются пользователю с ролью коменданта. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «roomMark/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addMark ( ) – метод действия контроллера, принимающий от клиента модель с заполненными данными добавляемой оценки: балл, номер комнаты, дата добавления оценки. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами коменданта. Если введенные пользователем данные корректны, оценка будет успешно добавлена, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/mark/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST). После успешного добавления оценки в систему, она будет отображена посредством дальнейшего вызова метода «roomMark/{id}».
* deleteMark (long id) – метод действия контроллера, принимающий от клиента идентификатор выставленной оценки. Проверяет соответствие данного идентификатора записи оценки в журнале в базе данных, если проверка прошла успешно, удаляет оценку из базы данных. Удалить запись может только пользователь с правами коменданта. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/mark/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE). После успешного выполнения метода действия пользователь перенаправляется по адресу «/roomMark/».
* updateMark (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными оценки, подлежащими редактированию: выставленный балл, номер комнаты, а также дата выставления оценки. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами коменданта. Если введенные пользователем данные корректны, запись с оценкой будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/mark/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT). После успешного выполнения метода действия пользователь перенаправляется по адресу «/roomMark/».
* listAllMarks () – метод действия контроллера, отправляющий клиенту страницу c персональной информацией по всем оценкам, выставленным на данный момент комнатам в общежитии. Информацию составляют выставленный за санитарное состояние балл, номер оцениваемой комнаты, а также дата проставления оценки комендантом. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «mark/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
  + 1. Класс ResidenceController

Данный класс контроллера реализует процесс управления датами заселения и выселения студентов. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяется его ролью. Данный класс предназначен для выполнения основных операций над датами заселения и выселения студентов из общежития.

Поля:

* residenceService – поле типа residenceService, которое нам необходимо для работы со слоем residenceDAO. ResidenceService является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин residenceService, который является реализацией класса ResidenceServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getResidence (long id) – метод действия контроллера, отправляющий клиенту страницу c информацией о датах заселения и выселения студента, в качестве параметра передается идентификатор студента. Полученная информация включает в себядату заселения студента в общежитие, дату выселения студента из общежития, а также ссылку на студента. Права на данное действие предоставлются пользователю с ролью коменданта или заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «residence/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addResidence ( ) – метод действия контроллера, принимающий от клиента модель с заполненными данными добавляемой записи в журнал заселения студентов: даты заселения и выселения, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, запись будет успешно добавлена в журнал, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/residence/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST).
* deleteResidence (long id) – метод действия контроллера, принимающий от клиента идентификатор студента. Проверяет соответствие данного идентификатора записи студента в журнале, если проверка прошла успешно, удаляет запись из базы данных. Удалить запись может только пользователь с правами заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/residence/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE).
* updateResidence (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными записи, подлежащими редактированию: дата заселения студента, дата выселения студента, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, запись будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/residence/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT).
* listAllResidences () – метод действия контроллера, отправляющий клиенту страницу c информацией по всем студентам и датам их заселения, а также датам их приблизительного выселения из общежития. Для использования этого действия необходимо обладать правами пользователя на уровне заведующей или коменданта.Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «residence/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
  + 1. Класс RebukeController

Данный класс контроллера реализует процесс управления датами получения и снятия выговоров со студентов. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяется его ролью. Данный класс предназначен для выполнения основных операций над датами получения и снятия выговоров со студентов из общежития.

Поля:

* rebukeService – поле типа rebukeService, которое нам необходимо для работы со слоем rebukeDAO. RebukeService является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин rebukeService, который является реализацией класса RebukeServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getRebuke (long id) – метод действия контроллера, отправляющий клиенту страницу c информацией о датах получения и снятия выговоров со студента, в качестве параметра передается идентификатор студента. Полученная информация включает в себя дату получения студентом выговора, предположительную дату снятия выговора со студента, а также ссылку на студента. Права на данное действие предоставлются пользователю с ролью коменданта или заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «rebuke/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addRebuke ( ) – метод действия контроллера, принимающий от клиента модель с заполненными данными добавляемой записи в журнал получения и снятия выговоров со студентов: даты получения и снятия выговора, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, запись будет успешно добавлена в журнал, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/rebuke/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST).
* deleteRebuke (long id) – метод действия контроллера, принимающий от клиента идентификатор студента. Проверяет соответствие данного идентификатора записи студента в журнале выговоров, если проверка прошла успешно, удаляет запись из базы данных. Удалить запись может только пользователь с правами заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/rebuke/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE).
* updateRebuke (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными записи, подлежащими редактированию: дата получения студентом выговора, дата предположительного снятия выговора со студента, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, запись будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/rebuke/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT).
* listAllRebukes () – метод действия контроллера, отправляющий клиенту страницу c информацией по всем студентам и датам получения ими выговоров, а также датам их приблизительного снятия. Для использования этого действия необходимо обладать правами пользователя на уровне заведующей или коменданта.Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «rebuke/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
  + 1. Класс RoomController

Данный класс контроллера реализует процесс управления комнатами общежития. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяется его ролью. Данный класс предназначен для выполнения основных операций комнатами общежития.

Поля:

* roomService – поле типа roomService, которое нам необходимо для работы со слоем markDAO. RoomService является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин roomService, который является реализацией класса RoomServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getRoom (String roomNumber) – метод действия контроллера, отправляющий клиенту страницу c информацией о комнате, в качестве параметра передается идентификатор комнаты. Полученная информация включает в себя количество мест в данной комнате, количество свободных мест и номер этажа, на котором данная комната располагается. Права на данное действие предоставлются пользователю с ролью коменданта или заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «room/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addRoom ( ) – метод действия контроллера, принимающий от клиента модель с заполненными данными добавляемой комнаты: количество мест, количество свободных мест, номер этажа. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами коменданта. Если введенные пользователем данные корректны, запись будет успешно добавлена в журнал, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/Room/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST).
* deleteRebuke (long id) – метод действия контроллера, принимающий от клиента идентификатор студента. Проверяет соответствие данного идентификатора записи студента в журнале выговоров, если проверка прошла успешно, удаляет запись из базы данных. Удалить запись может только пользователь с правами заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/rebuke/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE).
* updateRebuke (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными записи, подлежащими редактированию: дата получения студентом выговора, дата предположительного снятия выговора со студента, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей. Если введенные пользователем данные корректны, запись будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/rebuke/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT).
* listAllRebukes () – метод действия контроллера, отправляющий клиенту страницу c информацией по всем студентам и датам получения ими выговоров, а также датам их приблизительного снятия. Для использования этого действия необходимо обладать правами пользователя на уровне заведующей или коменданта.Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «rebuke/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
  + 1. Класс WorkOurController

Данный класс контроллера реализует процесс управления датами отработки и количеством отработанных общехозяйственных часов студентами. Методы действия данного класса требуют авторизации в системе, а некоторые операции также подразумевают наличие определенных прав у пользователя, что определяется его ролью. Данный класс предназначен для выполнения основных операций над общехозяйственными отработками студентов.

Поля:

* workHourService – поле типа workHourService, которое нам необходимо для работы со слоем workHourDAO. WorkHourService является интерфейсом, содержащим все необходимые методы для работы с общехозяйственными часами, на его основе создается бин workHourService, который является реализацией класса WorkHourServiceImpl. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getWorkHour (long id) – метод действия контроллера, отправляющий клиенту страницу c информацией о датах отработок студента с количеством выполненных часов, в качестве параметра передается идентификатор студента. Полученная информация включает в себя дату отработки, количество отработанных часов, а также ссылку на студента. Права на данное действие предоставлются пользователю с ролью коменданта или заведующей. Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «workHour/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.
* addWorkHour ( ) – метод действия контроллера, принимающий от клиента модель с заполненными данными добавляемой записи в журнал отработок студентов: даты отработок с количеством часов, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей или коменданта. Если введенные пользователем данные корректны, запись будет успешно добавлена в журнал, а соответствующая запись появится в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP POST запроса по адресу «/workHour/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.POST).
* deleteWorkHour (long id) – метод действия контроллера, принимающий от клиента идентификатор студента. Проверяет соответствие данного идентификатора записи студента в журнале отработок, если проверка прошла успешно, удаляет запись из базы данных. Удалить запись может только пользователь с правами заведующей или коменданта. Данный метод действия контроллера вызывается при обработке входящего HTTP DELETE запроса по адресу «/workHour/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.DELETE).
* updateWorkHour (long id) – метод действия контроллера, принимающий от клиента модель(по идентификатору) с заполненными данными записи, подлежащими редактированию: дата отработки студентом общехозяйственных часов, количество отработанных часов, ссылка на студента. Не все пользователи смогут выполнить данное действие в системе. Для выполнения данного действия пользователь должен быть авторизован в приложении с правами заведующей или коменданта. Если введенные пользователем данные корректны, запись будет успешно изменена, как и соответствующая ей запись в базе данных. Данный метод действия контроллера вызывается при обработке входящего HTTP PUT запроса по адресу «/workHour/{id}». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.PUT).
* listAllWorkHours () – метод действия контроллера, отправляющий клиенту страницу c информацией по всем отработанным общехозяйственным часам, а также их датами. Для использования этого действия необходимо обладать правами пользователя на уровне заведующей или коменданта.Данный метод действия контроллера вызывается при обработке входящего HTTP GET запроса по адресу «/workHour/». Метод помечен аннотацией @RequestMapping, параметрами которой являются value и method(RequestMethod.GET). Аннотация используется для маппинга урл-адреса запроса на указанный метод.

Теперь опишем классы сервисов, которые выполняют роль связующего звена между контроллером и слоем доступа к базе данных, по сути классы этого слоя приложения являются ключевыми с серверной стороны и сосредотачивают в себе основную бизнес-логику системы.

* + 1. Класс StudentService

Данный класс является прослойкой между контроллером StudentController и слоем доступа к базе данных StudentDao. На уровне сервиса реализуется основная бизнес-логика работы со студентами: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* studentDao – поле типа StudentDao, которое нам необходимо для работы со базой данных. StudentDao является интерфейсом, содержащим все необходимые методы для работы со студентами, на его основе создается бин studentDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getStudent (long id) – метод действия сервиса, отвечающий за получение студента, в качестве параметра передается его идентификатор. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addStudent (Student student) – метод действия сервиса, отвечающий за добавление студента, в качестве параметра передается объект типа Student c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteStudent (long id) – метод действия сервиса, отвечающий за добавление студента, в качестве параметра передается его идентификатор. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateStudent (Student student) – метод действия сервиса, отвечающий за обновление информации о студенте, в качестве параметра передается его объект типа Student. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllStudents () – метод действия сервиса, отвечающий за получение студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  + 1. Класс MarkService

Данный класс является прослойкой между контроллером MarkController и слоем доступа к базе данных MarkDao. На уровне сервиса реализуется основная бизнес-логика работы с оценками за санитарное состояние комнат общежития: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* markDao – поле типа MarkDao, которое нам необходимо для работы со базой данных. MarkDao является интерфейсом, содержащим все необходимые методы для работы с оценками комнат общежития, на его основе создается бин markDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getMark (String roomNumber) – метод действия сервиса, отвечающий за получение последней оценки за комнату, в качестве параметра передается идентификатор комнаты. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addMark(Mark mark) – метод действия сервиса, отвечающий за добавление оценки за комнату, в качестве параметра передается объект типа Mark c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteMark (long id) – метод действия сервиса, отвечающий за улаление выставленной оценки за комнату, в качестве параметра передается идентификатор оценки. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateMark (long id) – метод действия сервиса, отвечающий за обновление информации о студенте, в качестве параметра передается его объект типа Student. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllStudents () – метод действия сервиса, отвечающий за получение всех оценок, выставленных за комнаты. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  + 1. Класс ResidenceService

Данный класс является прослойкой между контроллером ResidenceController и слоем доступа к базе данных ResidenceDao. На уровне сервиса реализуется основная бизнес-логика работы с датами заселения и выселения студентов: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* residenceDao – поле типа ResidenceDao, которое нам необходимо для работы со базой данных. ResidenceDao является интерфейсом, содержащим все необходимые методы для работы с датами заселения и выселения студентов, на его основе создается бин residenceDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getResidence (long id) – метод действия сервиса, отвечающий за получение даты заселения и даты выселения студента из общежития, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addResidence (Residence residence) – метод действия сервиса, отвечающий за добавление записи заселения студента в общежитие, в качестве параметра передается объект типа Residence c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteResidence (long id) – метод действия сервиса, отвечающий за удаление записи о заселении студента в общежитие, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateResidence (long id) – метод действия сервиса, отвечающий за обновление информации о датах заселения и выселения студента, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllResidence () – метод действия сервиса, отвечающий за получение всех дат заселений и выселений студентов данного общежития. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  + 1. Класс RebukeService

Данный класс является прослойкой между контроллером RebukeController и слоем доступа к базе данных RebukeDao. На уровне сервиса реализуется основная бизнес-логика работы с датами получения студентами выговоров: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* rebukeDao – поле типа RebukeDao, которое нам необходимо для работы со базой данных. RebukeDao является интерфейсом, содержащим все необходимые методы для работы с датами получения студентами дисциплинарных взысканий, на его основе создается бин rebukeDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getRebuke (long id) – метод действия сервиса, отвечающий за получение даты объявления и даты снятия выговора со студента, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addRebuke (Rebuke rebuke) – метод действия сервиса, отвечающий за добавление записи о получении выговора студентом, в качестве параметра передается объект типа Rebuke c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteRebuke (long id) – метод действия сервиса, отвечающий за удаление записи о получении студентом выговора, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateRebuke (long id) – метод действия сервиса, отвечающий за обновление информации о датах получения и снятия выговора со студента, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllRebukes () – метод действия сервиса, отвечающий за получение всех дат объявлений и снятий выговоров со студентов данного общежития. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  + 1. Класс RoomService

Данный класс является прослойкой между контроллером RoomController и слоем доступа к базе данных RoomDao. На уровне сервиса реализуется основная бизнес-логика работы с комнатами общежития: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* roomDao – поле типа RoomDao, которое нам необходимо для работы со базой данных. RoomDao является интерфейсом, содержащим все необходимые методы для работы с комнатами общежития, на его основе создается бин roomDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getRoom (String roomNumber) – метод действия сервиса, отвечающий за получение информации о выбранной комнате, в качестве параметра передается ее номер. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addRoom (Room room) – метод действия сервиса, отвечающий за добавление комнаты в общежитие, в качестве параметра передается объект типа Room c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteRoom (String roomNumber) – метод действия сервиса, отвечающий за удаление комнаты из общежития, в качестве параметра передается ее номер. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateRoom (String roomNumber) – метод действия сервиса, отвечающий за обновление информации о комнате, в качестве параметра передается ее номер. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllRooms () – метод действия сервиса, отвечающий за получение всех комнат данного общежития. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  + 1. Класс WorkHourService

Данный класс является прослойкой между контроллером WorkHourController и слоем доступа к базе данных WorkHourDao. На уровне сервиса реализуется основная бизнес-логика работы с датами получения отработок общехозяйственных часов студентами: добавление, удаление, редактирование информации, получение по запросу. Именно сервис получает данные от контроллера и выбирает нужный метод из слоя DAO, который уже непосредственно обращается к базе данных.

Поля:

* workHourDao – поле типа WorkHourDao, которое нам необходимо для работы со базой данных. WorkHourDao является интерфейсом, содержащим все необходимые методы для работы с датами отработок студентами общехозяйственных часов, на его основе создается бин workHourDao. Поле помечено аннотацией @Autowired. Свойства класса с аннотацией @Autowired заполняются соответствующими значениями сразу после создания бина и перед тем, как любой из методов класса будет вызван.

Методы:

* getWorkHour (long id) – метод действия сервиса, отвечающий за получение даты отработки общехозяйственных часов студентом с указанием их количества, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
* addWorkHour (WorkHour workHour) – метод действия сервиса, отвечающий за добавление записи об отработки студентом общехозяйственных часов с указанием их количества, в качестве параметра передается объект типа WorkHour c заполненными свойствами. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* deleteWorkHour (long id) – метод действия сервиса, отвечающий за удаление записи об отработке студентом общехозяйственных часов, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* updateWorkHour (long id) – метод действия сервиса, отвечающий за обновление информации о дате отработки студентом общехозяйственных часов, в качестве параметра передается идентификатор студента. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.REQUIRED и readonly = false. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение REQUIRED у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе создавать новую. Параметр readonly установлен в false, так происходит модификация данных.
* listAllWorkHours () – метод действия сервиса, отвечающий за получение всех дат отработок студентами общехозяйственных часов с указанием их количества данного общежития. Метод помечен аннотацией @Transactional, параметрами которой являются propagation = Propagation.SUPPORTS и readonly = true. Перед выполнением данного метода начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается. Значение SUPPORTS у propagation указывает на то, что метод будет выполняться в существующей транзакции, если она есть, иначе выполняться вне транзакции. Параметр readonly установлен в true, это означает, что метод выполняет только чтение данных, без их модификации.
  1. Структура таблиц базы данных

В проекте используется обращение к реляционной базе данных, работающей под управлением системы управления базами данных (СУБД) MySQL. Каждой таблице в базе данных соответствует класс доменной модели в приложении, который инкапсулирует всю необходимую для программы информацию об объекте предметной области. Для отображения объектов классов доменных моделей на таблицы базы данных используется JDBCTemplate.

* + 1. Таблица USERS

Таблица содержит сведения о зарегистрированных в системе пользователях. Поле первичного ключа: us\_name – содержит уникальный идентификатор зарегистрированного пользователя.

Поля:

* us\_name – содержит имя зарегистрированного пользователя;
* us\_password– содержит пароль пользователя.

Поля внешних ключей:

* us\_role – определяет права пользователя в системе. Поле ссылается на первичный ключ role\_rank таблицы role.

Данной таблице соответствует класс доменной модели User.

* + 1. Таблица role

Таблица содержит роли пользователей в системе, что определяет их права. Поле первичного ключа: role\_rank – содержит уникальный идентификатор каждой из возможных ролей пользователя в системе.

Поля:

* role\_rank – содержит идентификатор приоритета роли;
* role\_name– содержит наименование роли.
  + 1. Таблица Student

Таблица содержит записи студентов, существующие в системе. Поле первичного ключа: ID – содержит уникальный идентификатор студента, который хранится системе.

Поля внешних ключей:

* st\_room – идентификатор комнаты, в которую заселен студент. Поле ссылается на первичный ключ ro\_number таблицы room.

Поля:

* id – содержит идентификатор студента;
* st\_name – содержит имя студента;
* st\_surname – содержит фамилию студента;
* st\_university – содержит университет, в котором проходит обучение студент;
* st\_faculty – содержит название факультета, на котором обучается студент;
* st\_group – содержит название учебной группы студента;

Данной таблице соответствует класс доменной модели Student.

* + 1. Таблица Room

Таблица содержит записи комнат, существующие в системе. Поле первичного ключа: ro\_number – содержит уникальный идентификатор комнаты, который хранится системе.

Поля:

* Ro\_number – содержит номер комнаты;
* Ro\_places – содержит общее количество мест в комнате;
* Ro\_free\_places – содержит колчество свободных мест в комнате;
* Ro\_floor – содержит номер этажа, на котором располагается комната;

Данной таблице соответствует класс доменной модели Room.

* + 1. Таблица residence\_history

Таблица содержит записи даты заселения и выселения студентов из общежития, существующие в системе. Поле первичного ключа: ID – содержит уникальный идентификатор записи, которая хранится системе.

Поля внешних ключей:

* resh\_student – идентификатор студента, которому соответствует данная запись. Поле ссылается на первичный ключ id таблицы student.

Поля:

* id – содержит идентификатор записи;
* resh\_set\_date – содержит дату заселения студента;
* resh\_eject\_date – содержит дату выселения студента;

Данной таблице соответствует класс Residence доменной модели .

* + 1. Таблица rebuke\_history

Таблица содержит записи даты получения и снятия выговоров со студентов общежития, существующие в системе. Поле первичного ключа: ID – содержит уникальный идентификатор записи, которая хранится системе.

Поля внешних ключей:

* rebh\_student – идентификатор студента, которому соответствует данная запись. Поле ссылается на первичный ключ id таблицы student.

Поля:

* id – содержит идентификатор записи;
* rebh\_name – содержит наименование выговора;
* rebh\_rank – содержит дату строгость выговора;
* rebh\_start\_date – содержит дату получения выговора;
* rebh\_end\_date – содержит дату предположительного снятия выговора;

Данной таблице соответствует класс Rebuke доменной модели .

* + 1. Таблица mark\_history

Таблица содержит записи оценок, выставленных за санитраное состояние комнат в общежитии, существующие в системе. Поле первичного ключа: ID – содержит уникальный идентификатор записи, которая хранится системе.

Поля внешних ключей:

* Mh\_room – идентификатор комнаты, которой выставлена соответствующая оценка. Поле ссылается на первичный ключ ro\_number таблицы room.

Поля:

* id – содержит идентификатор записи;
* mh\_date – содержит дату выставления оценки;
* mh\_value – содержит балл;

Данной таблице соответствует класс Mark доменной модели .

* + 1. Таблица work\_hours\_history

Таблица содержит записи даты отработок студентами общехозяйственных часов, существующие в системе. Поле первичного ключа: ID – содержит уникальный идентификатор записи, которая хранится системе.

Поля внешних ключей:

* wh\_student – идентификатор студента, которому соответствует данная запись. Поле ссылается на первичный ключ id таблицы student.

Поля:

* id – содержит идентификатор записи;
* wh\_date – содержит дату отработки общехозяйственных часов;
* wh\_number – содержит количество отработанных общехозяйственных часов;

Данной таблице соответствует класс WorkHour доменной модели .

* 1. Модели данных серверной части приложения

Модели являются классами, необходимыми для работы с данным в приложении. Каждая модель не должна содержать избыточной информации и использоваться только по прямому назначению.

* + 1. Класс User

Данный класс используется для работы с данными таблицы USER базы данных приложения, процессов регистрации и авторизации пользователя в системе.

Свойства:

* String name – содержит имя пользователя;
* String password – содержит пароль пользователя;
* int role – содержит приоритет роли пользователя в системе.
  + 1. Класс Student

Данный класс используется для работы с данными таблицы Student базы данных приложения, процессов создания и редактирования записи студента в системе.

Свойства:

* int id – содержит уникальный идентификатор студента;
* String name – содержит имя студента;
* String surname – содержит фамилию студента;
* String university –содержит университет, в котором проходит обучение студент;
* String faculty – содержит факультет, на котором обучается студент;
* String group – содержит учебную группу студента;
* String roomNumber– содержит номер комнаты, в которой проживает студент в общежитии.
  + 1. Класс Room

Данный класс используется для работы с данными таблицы Room базы данных приложения, процессов создания и комнат в системе.Свойства:

* String roomNumber – номер комнаты;
* int places – количество мест комнаты;
* int free\_places – содержит количество свободных мест комнаты;
* int floor – содержит номер этажа, на котором находится комната.
  + 1. Класс Residence

Данный класс используется для работы с данными таблицы residence\_history базы данных приложения, процессов создания и редактирования записи заселения студента в системе.

Свойства:

* int id – содержит уникальный идентификатор записи;
* Date setDate – содержит дату заселения студента в общежитие;
* Date ejectDate – содержит дату выселения студента из общежития.
* int studentId – содержит идентификатор студента;
  + 1. Класс Rebuke

Данный класс используется для работы с данными таблицы rebuke\_history базы данных приложения, процессов создания и редактирования записи получения студентом выговора.

Свойства:

* int id – содержит уникальный идентификатор записи;
* int rank – содержит критерий строгости полученного выговора;
* String name – содержит наименование выговора;
* Date startDate – содержит дату получения студентом выговора.
* Date endDate – содержит предположительную дату снятия выговора со студента;
* int studentId – содержит идентификатор студента;
  + 1. Класс Mark

Данный класс используется для работы с данными таблицы mark\_history базы данных приложения, процессов создания и редактирования записи выставления оценок за санитарное состояние комнат в системе.

Свойства:

* int id – содержит уникальный идентификатор записи;
* Date date – содержит дату выставления оценки;
* String roomNumber – содержит номер комнаты, которой выставлена оценка.
* int value – содержит балл, выставляемый за санитарное состояние комнаты;
  + 1. Класс WorkHour

Данный класс используется для работы с данными таблицы work\_hour\_history базы данных приложения, процессов создания и редактирования записи отработок студентами общехозяйственных часов в системе.

Свойства:

* int id – содержит уникальный идентификатор записи;
* Date date – содержит дату отработки студентом общехозяйственных часов;
* int number – содержит количество отработанных студентом часов;
* int studentId – содержит идентификатор студента;

1. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Разработанная система является web-приложением, что подразумевает под собой клиент-серверную архитектуру и разбиение кода на 2 части: серверную и клиентскую. Задача серверной части состоит в хранении и выдаче данных, разграничении пользователей и их прав доступа к различным ресурсам и частям приложения.

* 1. Модуль авторизации

Так как приложение работает в web среде, то оно одновременно обрабатывает запросы от множества пользователей. В таких системах очень важно иметь разграничение между данными разных пользователей. Если пользователей много и каждому доступны права на добавление студентов, выговоров, оценок за санитарное состояние комнаты проживания, то неизбежно это приведет к ошибочным записям в рамках системы. Поэтому если каждый пользователь будет иметь возможность производить действия в системе в строгом соответствии с выдаденными ему правами, то это наилучшим способом скажется на удобстве пользования системой.

Важной частью приложения является блок авторизации, который реализуется в приложении с использованием модуля Spring Security. Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework.

#### Ключевые объекты контекста Spring Security:

SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder использует ThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy).

SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

Authentication представляет пользователя (Principal) с точки зрения Spring Security.

GrantedAuthority отражает разрешения, выданные пользователю в масштабе всего приложения.

UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetails содержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса

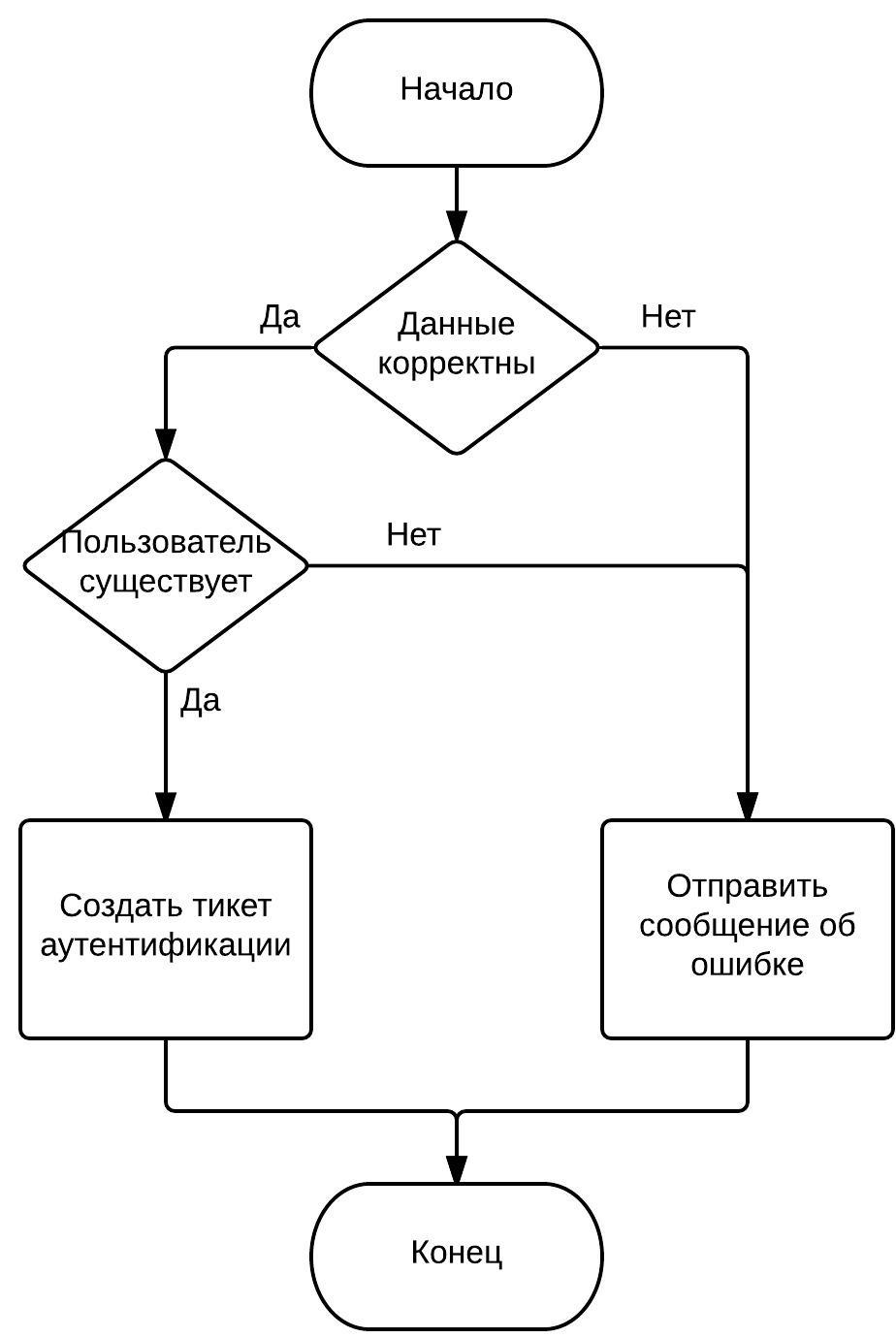


Рисунок 4.1 – Алгоритм работы аутентификаци

Рассмотрим валидацию входных данных сервером. Основной задачей является проверить, содержится ли в базе данных пользователь с введёнными параметрами.

Данный процесс состоит из следующих шагов:

1. Открыть соединение с базой данных.
2. Получить пользователя с необходимым именем и паролем.
3. Закрыть соединение с базой данных.
4. Если пользователь существует – вернуть true, иначе – false.
   1. Модуль обработки данных

Система хранит и обрабатывает записи студентов, а также дополнительно вносимую информацию о выговорах, комнатах проживания и т.д.

Общение клиента с сервером реализуется благодаря RESTful-архитектуре. Это значит, что клиент формирует необходимый http-запрос в соответствии с информацией, которую он хочет получить от сервера либо информацией, которую он хочет добавить. Рассмотрим это на примере добавления модели студента. Клиентская и серверная части обмениваются данными в формате json. Следующая модель студента по сформированному http.post-запросу будет добавлена в базу данных:

{

"id": 3,

"name": "Alexey",

"surname": "Sedlietsky",

"faculty": "FITR",

"group": 250501,

"room": "147a",

"univercity": "BNTU"

}

Всем этим занимается StudentController клиентской части приложения, который, который обрабатывает данные, пришедшие с представления и делигирует дальнейшую работу StudentService клиентской части приложения, где и будет сформирован запрос к серверу.

StudentController клиентской части представляет собой модуль, написанный на angular.js:

angular.module('crudApp').controller('StudentController',

['StudentService', '$scope', function( StudentService, $scope) {

var self = this;

Переменная, в которую запишется модель студента с принадлежащими ей полями:

self.student = {};

Переменная, в которую запишется массив из объектов, каждый из которых представляет собой модель студента:

self.students=[];

Метод, отвечающий за выбор между созданием или редактированием выбранного студента на основании пришедшего идентификатора:

function submit() {

console.log('Submitting');

if (self.student.id === undefined || self.student.id === null) {

console.log('Saving New Student', self.student);

createStudent(self.student);

} else {

updateStudent(self.student, self.student.id);

console.log('Student updated with id ', self.student.id);

}

}

Метод, отвечающий за создание студента, делегирующий его выполнение методу сервиса клиентской части:

function createStudent(student) {

console.log('About to create Student');

StudentService.createStudent(student)

.then(

function (response) {

console.log('Student created successfully');

self.successMessage = 'Student created successfully';

self.errorMessage='';

self.done = true;

self.student={};

$scope.myForm.$setPristine();

},

function (errResponse) {

console.error('Error while creating Student');

self.errorMessage = 'Error while creating Student: ' + errResponse.data.errorMessage;

self.successMessage='';

}

);

}

Метод, отвечающий за обновление информации о студенте, делегирующий его выполнение методу сервиса клиентской части:

function updateStudent(student, id){

console.log('About to update Student');

StudentService.updateStudent(student, id)

.then(

function (response){

console.log('Student updated successfully');

self.successMessage='Student updated successfully';

self.errorMessage='';

self.done = true;

$scope.myForm.$setPristine();

},

function(errResponse){

console.error('Error while updating Student');

self.errorMessage='Error while updating Student '+errResponse.data;

self.successMessage='';

}

);

}

Метод, отвечающий за удаление студента, делегирующий его выполнение методу сервиса клиентской части:

function removeStudent(id){

console.log('About to remove Student with id '+id);

StudentService.removeStudent(id)

.then(

function(){

console.log('Student '+id + ' removed successfully');

},

function(errResponse){

console.error('Error while removing Student '+id +', Error :'+errResponse.data);

}

);

}

Метод, отвечающий за извлечение всех студентов, записи которых зарегестрированы в системе, делегирующий его выполнение методу сервиса клиентской части:

function getAllStudents(){

return StudentService.getAllStudents();

}

function editStudent(id) {

self.successMessage='';

self.errorMessage='';

StudentService.getStudent(id).then(

function (student) {

self.student = student;

},

function (errResponse) {

console.error('Error while removing student ' + id + ', Error :' + errResponse.data);

}

);

}

Метод, отвечающий за сброс формы, в случае ее неправильного заполнения со стороны пользователя:

function reset(){

self.successMessage='';

self.errorMessage='';

self.student={};

$scope.myForm.$setPristine(); //reset Form

}

}

]);

Вышеприведенный блок является частью модуля по обработке данных. По аналогии с ним функционируют другие блоки, ответственные за работы с другими моделями системы, такими как Rebuke, Residence, Mark и т.д.

* 1. Модуль работы с данными

Модуль работы с данными, представляющими персональную информацию о студенте, представлен клиентским модулем StudentService:

angular.module('crudApp').factory('StudentService',

['$localStorage', '$http', '$q', 'urls',

function ($localStorage, $http, $q, urls) {

$http — это сервис, который используется для вызова REST сервисов. $http сервис это основа Angular для коммуникации с удаленным HTTP сервером через XMLHttpRequest объект или через JSONP. $http сервис это функция, которая принимает на вход только один аргумент — объект конфигурации, который используется для создания HTTP запроса и возврата ответа.

$q - cервис для управления асинхронными операциями (HTTP запросами или другими действиями).

var factory = {

loadAllStudents: loadAllStudents,

getAllStudents: getAllStudents,

getStudent: getStudent,

createStudent: createStudent,

updateStudent: updateStudent,

removeStudent: removeStudent,

test: test

};

return factory;

Метод, позволяющий получить все записи студентов, имеющихся в системе, реализуется формированием http запроса к серверу и последующей обработкой запроса:

function loadAllStudents() {

console.log('Fetching all students');

var deferred = $q.defer();

$http.get(urls.USER\_SERVICE\_API)

.then(

function (response) {

console.log('Fetched successfully all Students');

$localStorage.students = response.data;

deferred.resolve(response);

},

function (errResponse) {

console.error('Error while loading Students');

deferred.reject(errResponse);

}

);

return deferred.promise;

}

Метод, позволяющий получить запись студента по передаваемому идентификатору, реализуется формированием http запроса к серверу и последующей обработкой запроса:

function getStudent(id) {

console.log('Fetching Student with id :'+id);

var deferred = $q.defer();

$http.get(urls.USER\_SERVICE\_API + id)

.then(

function (response) {

console.log('Fetched successfully Student with id :'+id);

deferred.resolve(response.data);

},

function (errResponse) {

console.error('Error while loading users with id :'+id);

deferred.reject(errResponse);

}

);

return deferred.promise;

}

Метод, позволяющий создать запись студента по передаваемому объекту типа Student, поля которого считываются с формы, реализуется формированием http запроса к серверу и последующей обработкой запроса:

function createStudent(student) {

console.log('Creating Student');

var deferred = $q.defer();

$http.post(urls.USER\_SERVICE\_API, student)

.then(

function (response) {

loadAllStudents();

deferred.resolve(response.data);

},

function (errResponse) {

console.error('Error while creating Student : '+errResponse.data.errorMessage);

deferred.reject(errResponse);

}

);

return deferred.promise;

}

Метод, позволяющий обновить запись студента по передаваемому идентификатору, реализуется формированием http запроса к серверу и последующей обработкой запроса:

function updateStudent(student, id) {

console.log('Updating Student with id '+id);

var deferred = $q.defer();

$http.put(urls.USER\_SERVICE\_API + id, student)

.then(

function (response) {

loadAllStudents();

deferred.resolve(response.data);

},

function (errResponse) {

console.error('Error while updating Student with id :'+id);

deferred.reject(errResponse);

}

);

return deferred.promise;

}

}

]);

Вышеприведенный блок является частью модуля работы с данными. По аналогии с ним функционируют другие блоки, ответственные за работы с другими моделями системы, такими как Rebuke, Residence, Mark и т.д.

* 1. Модуль «Ядро»

В качестве данного модуля высупает app.js:

var app = angular.module('crudApp',['ui.router','ngStorage']);

Описываем константы урлов, необходимых для доступа к различным ресурсам системы:

app.constant('urls', {

BASE: 'http://localhost:8080/SpringBootApp',

STUDENT\_SERVICE\_API: 'http://localhost:8080/SpringBootApp/api/student/',

MARK\_SERVICE\_API: 'http://localhost:8080/SpringBootApp/api/mark/',

RESIDENCE\_SERVICE\_API: 'http://localhost:8080/SpringBootApp/api/residence/',

REBUKE\_SERVICE\_API: 'http://localhost:8080/SpringBootApp/api/rebuke/',

ROOM\_SERVICE\_API: 'http://localhost:8080/SpringBootApp/api/room/',

});

app.config(['$stateProvider', '$urlRouterProvider',

function($stateProvider, $urlRouterProvider) {

$stateProvider

.state('home', {

url: '/',

templateUrl: 'partials/list',

controller:'StudentController',

controllerAs:'ctrl',

resolve: {

students: function ($q, StudentService) {

console.log('Load all students');

var deferred = $q.defer();

StudentService.loadAllStudents().then(deferred.resolve, deferred.resolve);

return deferred.promise;

}

}

})

Для каждого state имеется соответствующее ему описание урла, контроллера, его обрабатывающего и т.д.

}]);

1. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ
   1. Модульное тестирование серверной и клиентской частей

Платформа Java предоставляет богатые возможности по тестированию всех частей web-приложения на всех уровнях разработки (от отдельной части программы до приложения в целом). Система тестирования включает в себя следующие возможности:

* юнит-тесты, позволяющие быстро и автоматически протестировать отдельные участки кода независимо от остальной части программы. При надлежащем составлении юнит-тесты вполне могут покрыть большую часть кода приложения;
* интеграционные тесты. Даже если охватить юнит-тестами практически всю логику приложения, все равно могут быть моменты, которые будут работать не так, как надо. Кроме того, сложно протестировать представления. Подобные вещи тестируются с помощью интеграционных тестов, выполняющихся на уровне web-браузера;
* нагрузочные тесты, которые призваны протестировать работу сайта в условиях высокой нагрузки. Подобные тесты позволяют выявить узкие места при работе с базой данных или при обращении к диску и ряд других проблем, которые сложно выявить другими способами.

Под модульным тестом понимается код, который тестирует работу некоторого компонента (модуля) в изолированной среде. Под интеграционным тестом понимается код, который тестирует совместную работу нескольких компонентов. Чтобы протестировать модуль в изолированной среде в случае, когда он зависит от других модулей, применяются «дублёры» (test doubles).

Дублёры бывают следующих видов:

* stub – это вспомогательный объект, выходные значения для которого задаются заранее. Он используется для того, что имитировать интерфейс зависимого компонента;
* mock – это вспомогательный объект, поведение которого задаётся заранее. Он используется для того, что имитировать интерфейс зависимого компонента и проверить в ходе теста, правильно ли он используется;
* spy – это вспомогательный объект для инспектирования вызываемых методов и передаваемых им параметров в ходе теста;
* fake – это вспомогательный объект, реализующий интерфейс зависимого компонента в упрощённом виде. Например, для целей модульного тестирования можно завести базу данных в памяти вместо реляционной базы данных, которая используется в рабочей версии продукта;
* dummy – это вспомогательный объект, указание или передача которого требуется сигнатурой метода или любым другим контрактом, но реальное значение никогда не используется.

Разница между Stub и Mock заключается в способе проверки результатов работы теста. В случае со Stub в конце теста проверяется состояние объекта. В случае с Mock в ходе теста проверяется то, что объект используется именно так, как было описано при регистрации.

Тестирование серверной части системы проводилось с использованием предсозданных моделей Student, Mark, Residence, Rebuke и т.д.

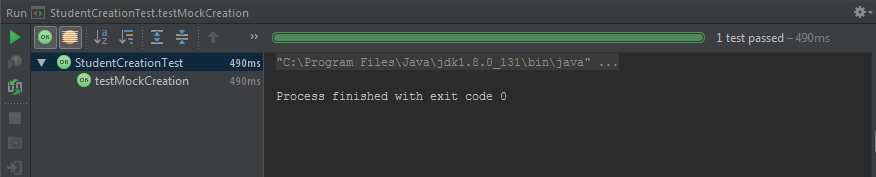


Рисунок 5.1 – Тестирование создание студента

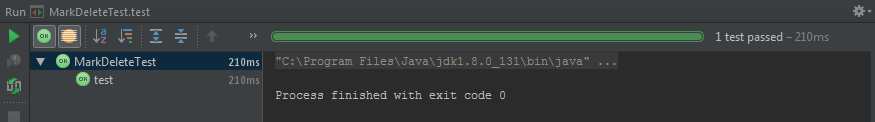


Рисунок 5.2 – Тестирование удаления оценки

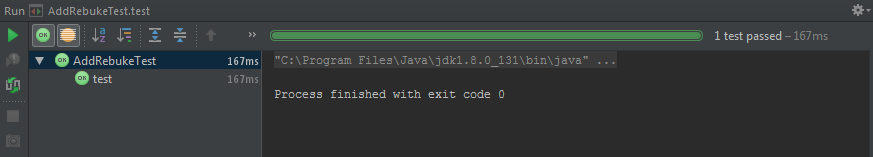


Рисунок 5.3 – Тестирование добавления выговора

Тестирование клиентской части системы производилось на всех современных наиболее популярных браузерах: Firefox, Google Chrome, Internet Explorer, Opera. Проверка производилась в том числе на минимально поддерживаемых версиях данных браузеров. Стоит отметить, что при использовании неподдерживаемого браузера пользователю предлагается обновить браузер до последней версии. Это избавляет от необходимости тестировать возможные непредвиденные ситуации при использовании устаревших браузеров.

Автоматические модульные и интеграционные тесты рекомендуется создавать для выполненного функционала для того, чтобы уменьшить риск регрессионных ошибок при изменении кода в будущем. В случае с JavaScript подобные тесты могут существенно упростить проверку работоспособности системы в различных браузерах путем автоматизации действий по обеспечению такой проверки.

В качестве фреймворка для проведения JavaScript юнит-тестов была выбрана библиотека QUnit, разработанная командой jQuery.

Дублёры для тестов системы создаются библиотекой Sinon.js. Эта библиотека может быть использована с любым тестовым фреймворком. Она предоставляет функции для эмуляции и проверки требуемого поведения в JavaScript. В библиотеке представлено три вида дублёров: Spy, Stub и Mock.

QUnit позволяет запускать модульные тесты в прямо в браузере. Для проведения тестирования фреймворком создаётся специальная интерактивная страница, на которой отображаются результаты проведения тестирования.

В результате тестирования были выявлены некоторые ошибки, которые были сразу же исправлены. Таким образом, необходимость модульного тестирования была подтверждена опытным путём.

При расширении приложения разработчикам необходимо проверять результаты тестов, чтобы случайно не повредить уже существующую функциональность.

* 1. Обработка исключительных ситуаций

Важную роль при тестировании играет информирование о необработанных исключительных ситуациях и их протоколирование. Вся логика выполнения метода помещается в блок try...catch, где и отслеживается исключение.

Отдельно стоит упомянуть про обработку ошибок в клиентской части приложения. Самая распространённый вариант – ошибки в описании добавляемой в систему модели студента. В таких случаях пользователь увидит всплывающее окно с предупреждением об ошибке.

Использование модульных тестов необходимо и экономически обосновано в крупных проектах, в которых задействовано большое количество разработчиков. Проверка результатов модульных тестов перед сохранением изменений позволяет снизить риск возникновения ошибок, связанных с повторным использованием кода.

В то же время, создание модульных тестов к проекту является достаточно ресурсоёмкой задачей, которая требует больших финансовых затрат. Именно поэтому большинство средних и малых проектов обходятся без модульного тестирования.

Применительно к данному дипломному проекту, модульное тестирование не принесло большой выгоды. Хоть тестами были выявлены некоторые ошибки, но время, затраченное на написание тестов, было неоправданно велико.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для работы с данным веб-приложением необходим любой современный браузер с включёнными JavaScript и cookie. Минимальные поддерживаемые версии браузеров: Internet Explorer 9.0, Firefox 15, Opera 12.

Разработанная система размещена на ресурсах Windows Azure (webmodeling.azurewebsites.net).

* 1. Регистрация и вход в систему

Прежде чем начать работу с системой, пользователю необходимо пройти процедуру регистрации. По умолчанию система перенаправляет пользователя на страницу входа в систему. Страница регистрации доступна по ссылке «Register» в верхней панели интерфейса (рисунок 6.1).

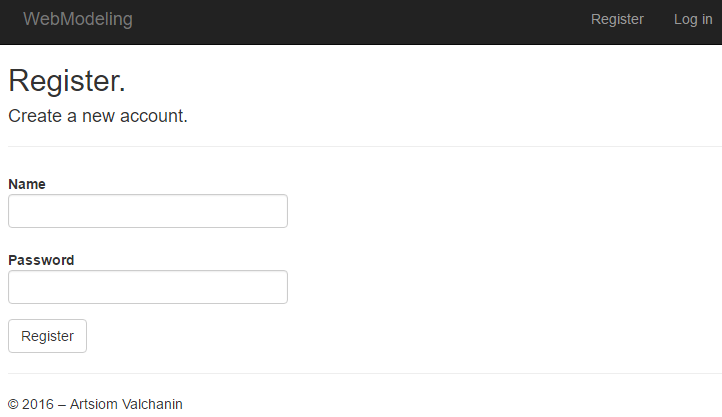


Рисунок 6.1 – Страница регистрации

Для регистрации в системе и создания нового аккаунта пользователь должен задать имя и пароль, которые будут использоваться для входа в приложение. Имя пользователя должно быть уникальным. Если пользователь введёт уже существующее имя, то получит сообщение об ошибке, которое изображено на рисунке 6.2.

После того, как пользователь прошёл процедуру регистрации, он может войти в систему, введя свои данные на странице авторизации (рисунок 6.3).

Опция «Remember me» позволяет пользователю оставаться в системе продолжительное время без необходимости повторного ввода данных учётной записи.

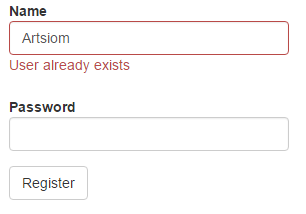


Рисунок 6.2 – Сообщение об ошибке

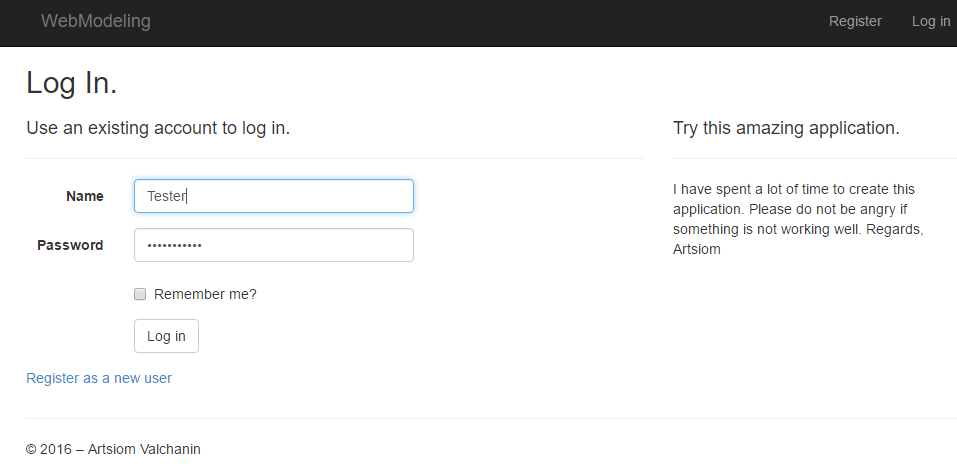


Рисунок 6.3 – Страница входа в систему

* 1. Работа с проектами

При входе в приложение по умолчанию открыта страница «My Projects» (рисунок 6.4). На данной странице отображается список проектов пользователя Также на странице присутствуют следующие элементы управления:

* кнопка «Create New» – позволяет создать новый проект;
* кнопка «Delete» – позволяет удалить проект;
* кнопка «Edit» – позволяет редактировать проект;
* кнопка «Share/Unshare» – позволяет поместить проект в общий доступ или изъять из общего доступа.

Прежде всего пользователю необходимо создать проект. Для этого необходимо нажать кнопку «Create New» и ввести имя проекта и указать его статус (общий/частный). После создания проекта пользователь будет опять перенаправлен на страницу «My Projects».

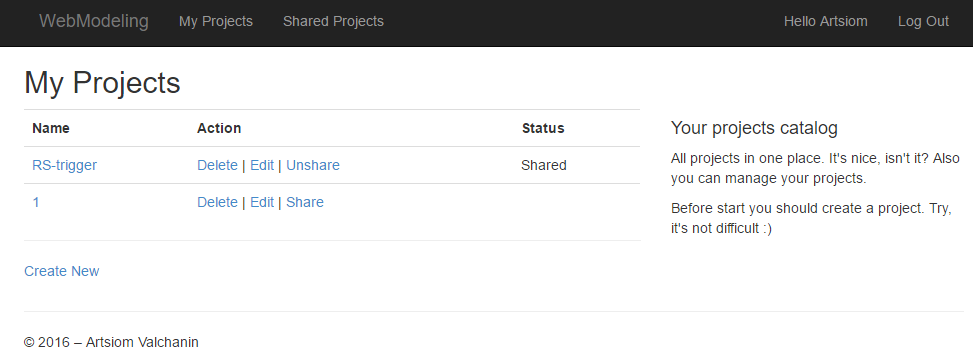


Рисунок 6.4 – Список проектов пользователя

Чтобы перейти к процессу редактирования схемы, пользователь должен нажать на имя проекта в списке. После этого система открывает редактор схем, который изображён на рисунке 6.5.

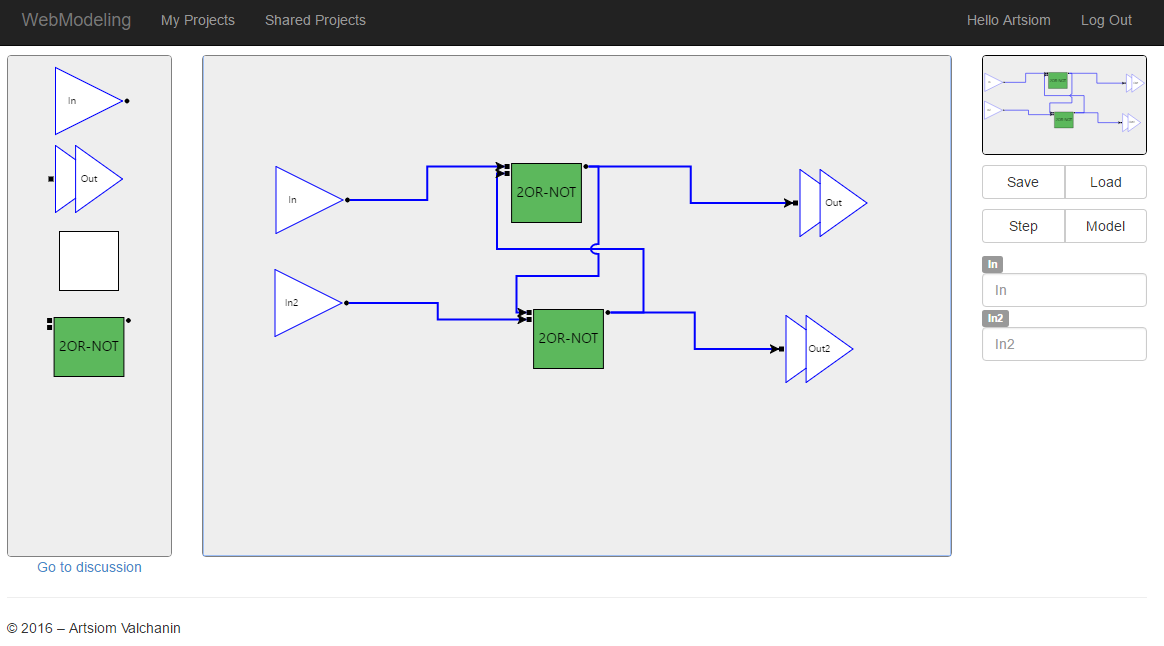


Рисунок 6.5 – Страница работы со схемой

С левой стороны страницы находится панель доступных элементов. По умолчанию доступны только 3 элемента – вход схемы, выход схемы, пустой элемент. Если проект является общим, то под панелью доступных элементов присутствует ссылка на страницу обсуждения проекта.

С правой стороны располагаются мини карта, кнопки управления, входные и выходные последовательности.

В центре страницы находится интерактивная панель редактора схем.

Панель доступных элементов хранит объекты, которые могут быть использованы при построении схемы. Чтобы использовать элемент, необходимо поместить его на редактор схем. Для добавления элемента в панель необходимо нажать кнопку «Add as template» в контекстном меню (рисунок 6.6).

Чтобы создать логический элемент необходимо добавить пустой элемент в редактор схем, добавить необходимые входные и выходные порты, используя соответствующие пункты контекстного меню, добавить поведение объекта.

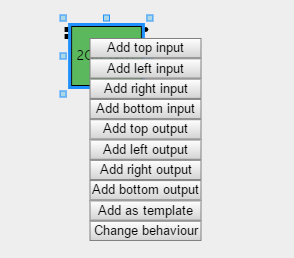


Рисунок 6.6 – Контекстное меню логического элемента

Чтобы описать поведение логического элемента, необходимо выбрать опцию «Change behaviour» в контекстном меню. В всплывающем окне необходимо описать поведение элемента используя язык JavaScript.

При описании поведения элемента стоит учитывать следующие особенности:

1. Объект node. Данный объект представляет текущий логический элемент в рамках процесса моделирования. Все операции должны выполняться только с этим объектом.
2. Входы и выходы элемента представлены массивами leftInputs, leftOutputs, rightInputs, rightOutputs, topInputs, topOutputs, bottomInputs, bottomOutputs, которые соответствуют входам и выходам конкретной стороны элемента.
3. Для обращения к состоянию конкретного входа или выхода стоит использовать свойство state.

Для проведения ручного моделирования необходимо выставить состояния входов схемы. Для этого используется опция «Change state» контекстного меню входных элементов. После этого стоит нажать на кнопку «Step» для проведения одного шага моделирования.

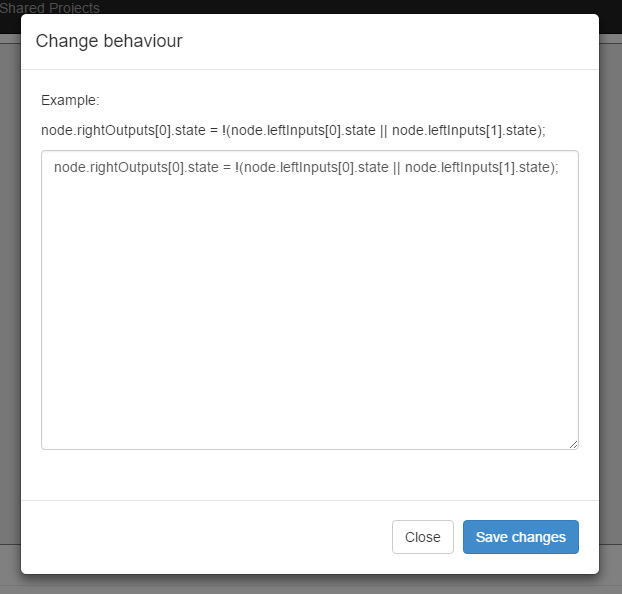


Рисунок 6.7 – Окно изменения поведения элемента

При проведении моделирования в ручном режиме необходимо задать входные последовательности для каждого входа используя цифры 1 и 0. Например 011010. Здесь 0 соответствует низкому логическому уровню.

* 1. Обсуждения проектов

Для того чтобы приступить к обсуждению проекта необходимо поместить его в общий доступ. После этого можно перейти на страницу обсуждения (рисунок 6.8).

На странице обсуждения отображаются все комментарии, оставленные пользователями к проекту. Под списком комментариев находится форма добавления нового комментария.

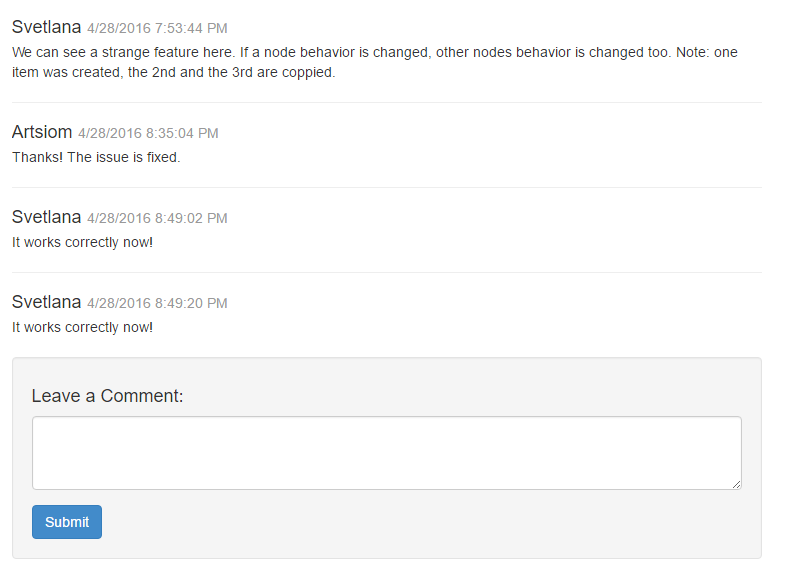


Рисунок 6.8 – Обсуждение проекта

Система обладает удобным пользовательским интерфейсом, который позволяет быстро перейти на необходимую страницу приложения.

Стоит отметить, что пользовательский интерфейс системы отображается по разному, но одинаково удобно на больших мониторах и на устройствах с небольшим экраном, например планшетах или смартфонах.

# **7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

## **7.1** Характеристика программного продукта

Web-приложение для учета и контроля студентов в общежитии предназначено для ведения документации, связанной с проживанием студентов, а также ведения различного рода учета(оценки за состояние комнаты, в которой проживает студент, дисциплинарные взыскания студента и т.д.)

Основная цель данной системы – упрощение ведения документооборота, связанного с учетом студентов.

## **7.2** Экономический эффект у разработчика

Данный программный модуль относится к 1 категории сложности за счет функционирования в режиме реального времени. При этом дополнительный коэффициент сложности равняется 1,12. По степени новизны относится к категории Б с коэффициентом новизны , так как он является развитием существующего ПО и не подразумевает в себе использование принципиально нового типа ЭВМ или ОС.

При разработке программного средства используются существующие технологии и средства разработки, которые охватывают около 20 – 30% реализуемых функций, поэтому коэффициент использования стандартных модулей принимается равным 0,8.

### **7.2.1** Определение объема и трудоемкости ПО

Для того, чтобы рассчитать плановую смету затрат на разработку ПО, требуется определить общий объем ПС (). В качестве единицы измерения возьмем количество строк исходного кода (LinesofCode, LOC). Объем разрабатываемых функций определяется по каталогу функций (см. таблицу 7.1).

Таблица 7.1 – Каталог функций ПО

|  |  |  |  |
| --- | --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функций по каталогу | Объем функций уточнённый |
| 1 | 2 | 3 | 4 |
| 101 | Организация ввода информации | 290 | 180 |
| 102 | Контроль, предварительная обработка информации | 420 | 400 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 4100 | 3720 |
| 111 | Управление вводом/выводом | 740 | 900 |
| 203 | Формирование баз данных | 2180 | 2360 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 530 |
| 507 | Обеспечение интерфейса между компонентами | 970 | 680 |
| 707 | Графический вывод результатов | 580 | 590 |
|  | Итого | 9690 | 9360 |

На основе общего объема ПС рассчитывается нормативная трудоемкость ПО (). Так как общий объем ПС = 9360, категория сложности ПО – 1, нормативная трудоемкость = 487. Тогда общая трудоемкость разработки определяется по формуле:

|  |  |
| --- | --- |
|  | (7.1) |

где – дополнительный коэффициент сложности;

– коэффициент, учитывающий использование типовых программ и модулей;

– коэффициент новизны.

Подставляя значения соответствующих коэффициентов в формулу 7.1, получим:

(человеко-дней)

На основе общей трудоемкости определяется плановое число разработчиков по формуле:

|  |  |
| --- | --- |
|  | (7.2) |

где – численность исполнителей проекта;

– общая трудоемкость разработки проекта (чел./дн.);

– срок разработки проекта (лет);

– эффективный фонд времени работы одного работника в течение года (дн).

Срок разработки проекта составляет полгода ( = 0,5 год). Эффективный фонд времени определяется по формуле:

|  |  |
| --- | --- |
| , | (7.3) |

где – количество дней в году;

– количество праздничных дней в году;

– количество выходных дней в году;

– количество дней отпуска.

Подставляя значения в формулу 7.3, получим:

(дней).

При решении сложных задач с длительным периодом разработки ПО трудоемкость определяется по стадиям разработки: техническое задание (ТЗ), эскизный проект (ЭП), технический проект (ТП), рабочий проект (РП), внедрение (ВН). При этом трудоемкость разработки ПО отличается в зависимости от стадий. Общий вид формулы, по которой рассчитывается трудоемкость изготовления ПС:

|  |  |
| --- | --- |
| , | (7.4) |

где – трудоемкость изготовления ПО на данной стадии;

– нормативная трудоемкость;

– удельный вес трудоемкости выбранной стадии разработки ПО в общей трудоемкости разработки ПО.

При этом для стадии «Рабочий проект» полученное значение трудоемкости изготовления ПО требуется умножить на . Общая трудоемкость определяется как сумма трудоемкости изготовления ПО на каждой из стадий разработки. Результаты расчетов трудоемкости по стадиям сведены в таблицу 7.2.

Таблица 7.2 – Расчет общей трудоемкости разработки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1. Коэффициенты удельных весов трудоемкости стадии разработки ПО () | 0,10 | 0,08 | 0,09 | 0,58 | 0,15 | 1,00 |
| 2. Коэффициент сложности ПО | 1,12 | 1,12 | 1,12 | 1,12 | 1,12 |  |
| 3. Коэффициент, учитывающий использование стандартных модулей |  |  |  | 0,80 |  |  |
| 4. Коэффициент, учитывающий новизну ПО () | 0,90 | 0,90 | 0,90 | 0,90 | 0,90 |  |
| 5. Общая трудоемкость ПО (), чел./дн. | 49,09 | 39,27 | 44,18 | 284,72 | 73,63 | 490,89 |

На основе уточненной трудоемкости разработки ПО с использованием формулы 7.2 найдем общую численность разработчиков, которые требуются, чтобы вложиться в заданные сроки:

### **7.2.2** Расчет сметы затрат и цены заказного ПО

Основной статьей расходов на создание ПО является заработная плата разработчиков проекта, то есть людей, непосредственно занимающихся разработкой.

Месячная тарифная ставка каждого специалиста () определяется по формуле:

|  |  |
| --- | --- |
| , | (7.5) |

где – месячная тарифная ставка первого разряда (рублей);

– тарифный коэффициент, соответствующий установленному тарифному разряду.

Часовая тарифная ставка :

|  |  |
| --- | --- |
|  | (7.6) |

где – часовая тарифная ставка (рублей);

– среднемесячная норма рабочего времени в часах (составляет 170 часов).

Расчет месячных и часовых тарифных ставок сведен в таблицу 7.3 с учётом того, что месячная тарифная ставка первого разряда на апрель 01.01.2017 г. года составляет 31 рубль.

Таблица 7.3 – Расчет месячных и почасовых тарифных ставок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Должность | Количе-ство ставок | Тариф-ный разряд | Тарифный коэффициент | Месячная тарифная ставка (руб.) | Часовая тарифная ставка (руб.) |
| Ведущий инженер-программист | 1 | 16 | 3,720 | 115,32 | 0,694 |
| Инженер-программист 1-ой категории | 4 | 12 | 2,840 | 88,04 | 0,53 |

Основная заработная плата исполнителей рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (7.7) |

где *n* – количество исполнителей;

– часовая тарифная ставка i-го исполнителя (ден.ед.);

– количество часов работы в день, ч;

– коэффициент премирования;

– плановый фонд рабочего времени i-го исполнителя (дн.).

(руб.).

Дополнительная заработная плата () включает в себя оплаты отпусков и другие выплаты, предусмотренные законодательством, и определяется по формуле:

|  |  |
| --- | --- |
|  | (7.8) |

где – норматив дополнительной заработной платы (10-20%).

|  |  |
| --- | --- |
|  |  |

Отчисления в фонд социальной защиты определяется по формуле:

|  |  |
| --- | --- |
|  | (7.9) |

где – норматив отчислений в фонд социальной защиты наделения.

Отчисления в фонд социальной защиты – 34%, отчисления в фонд социального страхования – 0,7%. Исходя из этого, получаем:

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Материалы» отражают расходы на магнитные носители, бумагу, тонер и прочие вещи, необходимые для разработки ПО. Нормы расхода материалов в суммарном выражении () определяется в расчете на 100 строк исходного кода. Сумма затрат на расходные материалы определяется по формуле:

|  |  |
| --- | --- |
|  | (7.10) |

где – норма расхода материалов в расчете на 100 строк исходного кода ПО (руб.);

– общий объем ПО (строк исходного кода).

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Машинное время» включает оплату машинного времени, необходимого для разработки и отладки ПО. Норматив на 100 строк исходного кода () зависит от характера решаемых задачи и типа приложений. Расходы по этой статье определяются по формуле:

|  |  |
| --- | --- |
| , | (7.11) |

где – цена одного машино-часа (руб.);

– общее время работы над проектом (часов).

(руб.).

Расходы по статье «Научные командировки» () определяются по формуле:

|  |  |
| --- | --- |
|  | (7.12) |

где – норматив расходов на командировки в целом по организации (%).

|  |  |
| --- | --- |
|  |  |

Расходы по статье «Прочие затраты» () включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяется по формуле:

|  |  |
| --- | --- |
|  | (7.13) |

где – норматив прочих затрат в целом по организации.

|  |  |
| --- | --- |
|  |  |

Затраты по статье «Накладные расходы» () связаны с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных производств. Определяются по формуле:

|  |  |
| --- | --- |
|  | (7.14) |

где – норматив накладных расходов в целом по организации.

Общая сумма расходов по смете ():

(руб.).

Затраты на сопровождение и адаптацию ПО ():

|  |  |
| --- | --- |
|  | (7.15) |

где – норматив расходов на сопровождение (%).

Затраты на производство и продвижение программы определяется по формуле:

|  |  |
| --- | --- |
| . | (7.16) |

Подставляя значения в формулу 7.16, получим:

(руб.).

Прибыль от создаваемого ПО определяется по формуле:

|  |  |
| --- | --- |
|  | (7.17) |

где – уровень рентабельности ПО (%).

Прогнозируемая цена ПО без налогов () определяется по формуле:

|  |  |
| --- | --- |
|  | (7.18) |

Подставляя значения в формулу 7.18, получим:

(руб.).

Таблица 7.4 – Результаты и формулы расчетов

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статей | Усл. обозн. | Значение (руб.) | Методика расчета |
| 1 | 2 | 3 | 4 |
| Основная заработная плата исполнителей | Зо |  | Определяются на основании расчетов |
| Дополнительная заработная плата исполнителей | Зд |  |  |
| Отчисления в фонд социальной защиты населения | Рсоц |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Машинное время | Рм |  | Определяются на основании расчета. Цена 1 машино-часа рыночная |
| Расходы на материалы | Mi |  |  |
| Расходы на научные командировки | Рнк |  |  |
| Прочие прямые расходы | Пз |  |  |
| Накладные расходы | Рн |  |  |
| Полная себестоимость | Сп |  | Сп = Рм + Зо + Зд + Рсоц +  + Рм + Рнп + Рпр + Рн |
| Затраты сопровождения и адаптации | Рс |  |  |
| Затраты на производство и продвижение | Спп |  |  |
| Прогнозируемая прибыль | По |  |  |
| Прогнозируемая цена без налогов (цена предприятия) | Цп |  |  |
| Налог на добавленную стоимость (НДС) | НДС |  |  |
| Прогнозируемая отпускная цена | Цо |  |  |
| Месячная тарифная ставка 1-го разряда | Тм1 |  |  |
| Ставка налога на прибыль | Нп | Нп=18% |  |

Налог на добавленную стоимость ():

|  |  |
| --- | --- |
|  | (7.19) |

где – норматив НДС (%).

Прогнозируемая отпускная цена ():

|  |  |
| --- | --- |
| . | (7.20) |

Подставляя значения в формулу 7.20, получим:

(руб.).

Прибыль за вычетом налога рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.21) |

где Нп – ставка налога на прибыль (Нп = 18%).

Подставляя значения в формулу 7.21, получим:

(руб.).

Данная сумма остается при реализации собственнику разработки и представляет собой экономический эффект у собственника.

Все расчеты себестоимости и прибыли представлены в таблице 7.4.

Таким образом, в результате оценки экономического эффекта у разработчика были получены следующие результаты:

* себестоимость проекта составила 11,168 тысяч рублей;
* прогнозируемая отпускная цена – 15,1 тысяч рублей;
* прибыль за вычетом налога составит 2,857 тысяч рублей.

## **7.3** Расчет экономического эффекта от применения программного средства у пользователя

Результатом использования программного продукта в своей сфере является прирост чистой прибыли и амортизационных отчислений.

Прирост чистой прибыли представляет собой экономию затрат на заработную плату и начислений на заработную плату, материальных затрат.

Прирост прибыли за счет экономии расходов на заработную плату,  
связанный с высвобождением работников с повременной оплатой труда, определяется по формуле:

|  |  |
| --- | --- |
|  | (7.22) |

где Кпр – коэффициент премий за выполнение плановых заданий (1,5);

∆Чi – абсолютное высвобождение работников i-й категории, человек;

*i* – годовая заработная плата высвобождаемых работников i-й кате­гории, рублей;

Нд – процент дополнительной заработной платы (17%);

Нно – ставка отчислений от заработной платы, включаемых в себестоимость продукции (0,6%);

*n* – категории высвобождаемых работников.

Таким образом, полученный прирост прибыли за счет экономии расходов на заработную плату равен:

(руб.).

Прирост чистой прибыли рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (7.23) |

где *n* – виды затрат, по которым получена экономия;

Э*i* – сумма экономии, полученная за счет снижения i-го вида затрат, рублей;

Нп – ставка налога на прибыль (18%).

Следовательно,

(руб.).

Прирост чистой прибыли по годам эксплуатации имеет одинаковое значение.

Амортизационные отчисления являются источником погашения инвестиций в приобретение программного продукта. Расчет амортизационных отчислений осуществляется по формуле:

|  |  |
| --- | --- |
|  | (7.24) |

где На – норма амортизации программного продукта (20%);

Ипс – стоимость программного продукта, рублей.

Таким образом, амортизационные отчисления составят:

(руб.).

При оценке эффективности инвестиционных проектов необходимо осуществить приведение затрат и результатов, полученных в разные периоды времени, к расчетному году, путем умножения затрат и результатов на коэффициент дисконтирования , который определяется следующим образом:

|  |  |
| --- | --- |
|  | (7.25) |

где Ен – требуемая норма дисконта;

*t* – порядковый номер года, затраты и результаты которого приводятся к расчетному году;

*tр* – расчетный год, в качестве расчетного года принимается год вложения инвестиций, *tp*= 1.

Следовательно,

|  |  |
| --- | --- |
|  |  |

Расчет чистого дисконтированного дохода за пять лет реализации проекта и срока окупаемости инвестиций представлены в таблице 7.5. Нормативная ставка дисконта составляет Е = 32%.

Таблица 7.5 – Экономические результаты работы предприятия

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование показателей | Един. изм. | По годам производства | | | | |
| 2017 | 2018 | 2019 | 2020 | 2021 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Результат |  |  |  |  |  |  |
| Прирост чистой прибыли | рубль | - | 9904,199 | 9904,199 | 9904,199 | 9904,199 |
| Прирост амортизационных отчислений | рубль | - | 3020,077 | 3020,077 | 3020,077 | 3020,077 |

Продолжение таблицы 7.5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Прирост результата | рубль | - | 6884,122 | 6884,122 | 6884,122 | 6884,122 |
| Коэффициент дисконтирования |  | 1 | 0,85 | 0,73 | 0,62 | 0,53 |
| Результат с учетом фактора времени | рубль | - | 5851,503 | 5025,409 | 4268,155 | 3648,584 |
| Затраты (инвестиции) |  |  |  |  |  |  |
| Инвестиции в приобретение программного продукта | рубль | 15100,38 |  |  |  |  |
| Инвестиции с учетом фактора времени | рубль | 15100,38 | - | - | - | - |
| Чистый дисконтирован-ный доход по годам | рубль | 15100,38 | 5851,503 | 5025,409 | 4268,155 | 3648,584 |
| Чистый дисконтирован-ный доход нарастающим итогом | рубль | -15100,38 | -9248,877 | -4223,468 | 44,687 | 3693,271 |

В качестве результата выступает чистая прибыль, остающаяся в распоряжении предприятия.

Рассчитаем рентабельность инвестиций (РИ) по формуле:

|  |  |
| --- | --- |
|  | (7.26) |

где Пчср – среднегодовая величина чистой прибыли за расчетный период в рублях, которая определяется по формуле:

|  |  |
| --- | --- |
|  | (7.27) |

где Пчi - чистая прибыль, полученная в году i в рублях.

С учетом того, что

Пчср = (9904,199 + 9904,199 + 9904,199 + 9904,199) / 5 =

= 7923,359 (руб.).

Следовательно, рентабельность инвестиций будет Ри = 0,20.

В результате технико-экономического обоснования инвестиций по производству нового изделия были получены следующие значения показателей их эффективности:

* чистый дисконтированный доход нарастающим итогом в 2021-м году составит 3,63 тысячи рублей;
* инвестиции окупаются на четвертый год;
* рентабельность инвестиций составляет 20%.

Таким образом, разработка и применение программного продукта являются эффективными.

ЗАКЛЮЧЕНИЕ

Начинающие инженеры в процессе обучения проектируют множество различных электронных схем. Логическое моделирование необходимо для проверки функционирования созданной схемы на логическом уровне без её физической реализации.

Разработанная система во многом способствует эффективному обучению инженеров, предоставляя возможность построения и проверки функционирования схем.

Основной целевой аудиторией проекта являются студенты высших учебных заведений, обучающиеся на специальностях, тесно связанных с компьютерной инженерией. Во многом обучению способствуют социальные части системы – возможность выставление проектов на общее обозрение, возможность обсуждения разработанных схем.

Проекты, предоставленные в общем доступе, могут стать примерами решения тех или иных задач, с которыми сталкиваются обучающиеся. Очень важно иметь возможность поделиться идеей с единомышленниками.

Возможность обсуждения проектов позволяет найти оптимальные варианты решения задач, обнаружить ошибки в работе проектируемого устройства, поделиться идеями развития проекта.

В качестве усовершенствования системы в дальнейшем планируется:

* организация групп пользователей, которая позволит публиковать проекты ограниченному числу людей;
* реализация различных ролей пользователей с различными правами. Один из вариантов использования данной функциональности – включение в работу с системой преподавателей, которые смогут курировать группы и помогать остальным пользователям.
* добавление графической составляющей блока работы с сигналами, которая позволит задавать и просматривать последовательности сигналов в виде графика.

СПИСОК ИCПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Xilinx ISE [Электронный ресурс]. – Режим доступа: http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html.
2. Altera [Электронный ресурс]. – Режим доступа: https://www.altera.com.
3. ModelSim [Электронный ресурс]. – Режим доступа: https://www.mentor.com/products/fpga/model.
4. Рихтер, Д. CLRviaC#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. – СПб. : Питер, 2013. – 896 с.
5. Троелсен, Эндрю. С# 2008 и платформа .NET 3.5 Framework. 4-е изд. / Э. Троелсен – М.: Вильямс, 2009. – 1168 с.
6. Шилдт, Г. C# 4.0. Полное руководство / Г. Шилдт. – М. : Вильямс. – 1056 с.
7. Библиотека MSDN [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/default.aspx.
8. C# Version 5.0 Specification [Электронныйресурс] : Specification / Microsoft Corporation. – Режим доступа: CSharp Language Specification.docx.
9. Фримен, А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов / А. Фримен. – М. : Вильямс. – 736 с.
10. Руководство по ASP.NET MVC 5 [Электронный ресурс]. – Режим доступа: http://metanit.com/sharp/mvc5.
11. Шаблон MVC [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Model-view-controller.
12. Общие сведения о ASP.NET MVC [Электронный ресурс]. – Режим доступа: http://msdn.microsoft.com/ru-ru/library/dd3412%28v=vs.108%29.aspx.
13. Введение в Entity Framework [Электронный ресурс]. – Режим доступа: http://metanit.com/sharp/entityframework/1.1.php.
14. JavaScript [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/JavaScript.
15. Асинхронное определение модуля [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Асинхронное\_определение\_модуля.
16. RequireJs [Электронный ресурс]. – Режим доступа: http://requirejs.org.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный текст модулей клиентской части приложения

$(webModeling.modelCourt.config.stepButton).on("click", {

diagram: webModeling.diagram,

modelCourt: webModeling.modelCourt

}, stepModel);

$(webModeling.modelCourt.config.modelButton).on("click", {

diagram: webModeling.diagram,

modelCourt: webModeling.modelCourt

}, model);

webModeling.diagram.events.on("IOStateChanged", function () {

webModeling.modelCourt.stepModel = {};

});

webModeling.diagram.events.on("DataLoaded", function (event) {

var inputs = getInputs(event.target);

setInputs(webModeling.modelCourt.config.sequencesArea, inputs); cleanResults(webModeling.modelCourt.config.resultsArea);

});

webModeling.diagram.addDiagramListener("ExternalObjectsDropped", function (event) {

var inputs = getInputs(event.diagram);

setInputs(webModeling.modelCourt.config.sequencesArea, inputs); cleanResults(webModeling.modelCourt.config.resultsArea);

});

webModeling.diagram.addDiagramListener("SelectionDeleted", function (event) {

var inputs = getInputs(event.diagram);

setInputs(webModeling.modelCourt.config.sequencesArea, inputs); cleanResults(webModeling.modelCourt.config.resultsArea);

});

// signals

function setInputs(area, inputs) {

var sequencesArea = $(area);

sequencesArea.empty();

\_.each(inputs, function (input) {

var template = '<span class="label label-pill label-default">' + input.key + '</span>' +

'<input type="text" class="form-control" id="' + input.key + '" placeholder="' + input.key + '"></input>';

sequencesArea.append(template);

});

}

function readConf(inputs, modelCourt) {

\_.each(inputs, function (input) {

var textInput = $("#" + input.key);

var sequence = textInput.val().split('');

modelCourt.input[input.key] = \_.map(sequence, function (e) {

return e === "1";

});

if (!modelCourt.input.length || (modelCourt.input.length < sequence.length)) {

modelCourt.input.length = sequence.length;

}

});

}

function cleanResults(area) {

var resultsArea = $(area);

resultsArea.empty();

}

function showResults(outputs, area, modelCourt) {

cleanResults(area);

var resultsArea = $(area);

resultsArea.append('<hr/>');

\_.each(outputs, function (output) {

var result = \_.map(modelCourt.output[output.key], function (r) {

return r ? '1' : '0';

}).join('');

var template = '<span class="label label-pill label-default">' + output.key + '</span>' +

'<input type="text" value="' + result + '"class="form-control" id="' + output.key + '" placeholder="' + output.key + '"></input>';

resultsArea.append(template);

});

}

// modeling

function model(event) {

var diagram = event.data.diagram;

var modelCourt = event.data.modelCourt;

modelCourt.output = {};

var inputs = getInputs(diagram);

readConf(inputs, modelCourt);

var query = getQuery(diagram);

for (var i = 0; i < modelCourt.input.length; i++) {

\_.each(getInputs(diagram), function(input){

setIOState(diagram, input, modelCourt.input[input.key][i]);

});

for (var j = 0; j < modelCourt.config.step; j++) {

\_.each(query, function(element) {

processNode(diagram, element);

});

}

\_.each(getOutputs(diagram), function(output){

if (!modelCourt.output[output.key]) modelCourt.output[output.key] = [];

modelCourt.output[output.key].push(output.state);

});

}

var outputs = getOutputs(diagram);

showResults(outputs, modelCourt.config.resultsArea, modelCourt);

}

function stepModel(event) {

var diagram = event.data.diagram;

var modelCourt = event.data.modelCourt;

if (\_.isEmpty(modelCourt.stepModel)) {

modelCourt.stepModel.query = getQuery(diagram);

modelCourt.stepModel.index = 0;

}

var query = modelCourt.stepModel.query;

var i = modelCourt.stepModel.index;

processNode(diagram, query[i]);

modelCourt.stepModel.index++;

if (modelCourt.stepModel.index === query.length) {

modelCourt.stepModel = {};

}

}

function getQuery(diagram) {

var query = getInputs(diagram);

var i = 0;

while (i < query.length) {

var children = getChildren(diagram, query[i]);

query = \_.union(query, children);

i++;

}

return query;

}

function getInputs(diagram) {

var inputs = \_.filter(diagram.model.nodeDataArray, function (node) {

return node.category === "INPUT";

});

return inputs;

}

function getChildren(diagram, node) {

var childrenLinks = getChildLinks(diagram, node);

var children = \_.filter(diagram.model.nodeDataArray, function (node) {

return \_.findWhere(childrenLinks, { to: node.key });

});

return children;

}

function getChildLinks(diagram, node) {

var childLinks = \_.filter(diagram.model.linkDataArray, function (link) {

return link.from === node.key;

});

return childLinks;

}

function setIOState(diagram, node, state) {

diagram.model.startTransaction("setIOState");

diagram.model.setDataProperty(node, "state", !state);

diagram.model.setDataProperty(node, "state", state);

diagram.model.commitTransaction("setIOState");

}

function processNode(diagram, node){

if(node.category === "NODE"){

if(node.behaviour) execNode(node);

}

if(node.category === "OUTPUT"){

setIOState(diagram, node, node.state);

}

var childLinks = getChildLinks(diagram, node);

var outputMap = getOutputPortMap(node);

\_.each(childLinks, function(link){

var state = outputMap[link.fromPort].state;

setLinkState(diagram, link, state);

var child = getNode(diagram, link.to);

var childInputs = getInputPortMap(child);

childInputs[link.toPort].state = state;

});

}

function execNode(node) {

var code = node.behaviour;

var func = new Function("node", code);

func(node);

}

function getOutputs(diagram) {

var outputs = \_.filter(diagram.model.nodeDataArray, function (node) {

return node.category === "OUTPUT";

});

return outputs;

}

function getOutputPortMap(node) {

var outputs = getOutputPorts(node);

var map = {};

\_.each(outputs, function (output) {

map[output.portId] = output;

});

return map;

}

function setLinkState(diagram, link, state) {

diagram.model.startTransaction("setLinkState");

diagram.model.setDataProperty(link, "state", state);

diagram.model.commitTransaction("setLinkState");

}

function getNode(diagram, nodeKey) {

var node = \_.findWhere(diagram.model.nodeDataArray, { key: nodeKey });

return node;

}

function getInputPortMap(node) {

var inputs = getInputPorts(node);

var map = {};

\_.each(inputs, function (input) {

map[input.portId] = input;

});

return map;

}

function getInputPorts(node){

switch(node.category){

case "NODE":

return getInputsForNode(node);

case "OUTPUT":

return getInputsForOutput(node);

default:

return [node];

}

}

function getInputsForOutput(node) {

return [node];

}

function getInputsForNode(node) {

var inputs = [].concat(node.leftInputs, node.rightInputs, node.topInputs, node.bottomInputs);

return inputs;

}

function getOutputPorts(node){

switch(node.category){

case "NODE":

return getOutputsForNode(node);

case "INPUT":

return getOutputsForInput(node);

default:

return [node];

}

}

function getOutputsForInput(node)

{

return [node];

}

function getOutputsForNode(node){

var outputs = [].concat(node.leftOutputs,node.rightOutputs,node.topOutputs,node.bottomOutputs);

return outputs;

}

// editor

var $$ = go.GraphObject.make;

var webModeling = {

config : {

portSize: new go.Size(5, 5),

diagramContainer: "myDiagramDiv",

paletteContainer: "myPaletteDiv",

saveNodeBehaviourButton: "#saveNodeBehaviour",

behaviourPopup: "#behaviourPopup",

},

modelCourt : {

input:{},

output:{},

config: {

stepButton: "#stepButton",

modelButton: "#modelButton",

sequencesArea: '#inputSequences',

resultsArea: '#outputSequences',

step:5

},

stepModel:{

}

},

diagram: {},

palette: {},

overview: {}

}

function init() {

initializeDiagram(webModeling);

intializeOverview(webModeling);

initializePalette(webModeling);

load();

}

function save() {

$.ajax({

type: "POST",

url: "/Scheme/Save/" + document.projectId,

data: {

diagramData: webModeling.diagram.model.toJson(),

paletteData: webModeling.palette.model.toJson()

}

});

webModeling.diagram.isModified = false;

}

function load() {

$.get("/Scheme/Get/" + document.projectId, function (data) {

var diagram = data.diagramData;

var palette = data.paletteData;

if (diagram) {

webModeling.diagram.model = go.Model.fromJson(diagram);

webModeling.palette.model = go.Model.fromJson(palette);

webModeling.diagram.events.trigger("DataLoaded");

} else {

webModeling.diagram.model = go.Model.fromJson({

"class": "go.GraphLinksModel",

"copiesArrays": true,

"copiesArrayObjects": true,

"linkFromPortIdProperty": "fromPort",

"linkToPortIdProperty": "toPort",

"nodeDataArray": [],

"linkDataArray": []

});

}

});

}

// shared

function getOutputTemplate(nodeContextMenu) {

var outputTemplate = $$(go.Node, "Table", {

locationObjectName: "BODY",

selectionObjectName: "BODY",

locationSpot: go.Spot.Center

},

$$(go.RowColumnDefinition, { column: 0, width: 8 }),

new go.Binding("location", "loc", go.Point.parse).makeTwoWay(go.Point.stringify),

new go.Binding("state", "state").makeTwoWay(),

$$(go.Panel, "Auto", {

row: 0,

column: 1,

name: "BODY",

stretch: go.GraphObject.Fill,

minSize: new go.Size(68, 68),

maxSize: new go.Size(68, 68)

},

$$(go.Shape, "DoubleArrow", {

fill: "#FFFFFF",

strokeWidth: 1

},

new go.Binding("stroke", "state", function (state) {

return state ? "red" : "blue";

})

),

$$(go.TextBlock, {

margin: 5,

textAlign: "center",

font: "10px Segoe UI,sans-serif",

stroke: "black",

editable: true,

isMultiline: false

},

new go.Binding("text", "key")

)

),

$$(go.Panel, {

row: 0,

column: 0,

fromSpot: go.Spot.Left,

toSpot: go.Spot.Left,

cursor: "pointer",

toLinkable: true,

portId: "output",

toMaxLinks: 1

},

$$(go.Shape, "Rectangle", {

stroke: null,

strokeWidth: 0,

desiredSize: webModeling.config.portSize,

fill: "black"

}

)

)

);

if (nodeContextMenu) {

outputTemplate.contextMenu = nodeContextMenu;

}

return outputTemplate;

}

function getInputTemplate(nodeContextMenu) {

var inputTemplate = $$(go.Node, "Table", {

locationObjectName: "BODY",

selectionObjectName: "BODY",

locationSpot: go.Spot.Center

},

$$(go.RowColumnDefinition, { column: 1, width: 8 }),

new go.Binding("location", "loc", go.Point.parse).makeTwoWay(go.Point.stringify),

new go.Binding("state", "state").makeTwoWay(),

$$(go.Panel, "Auto", {

row: 0,

column: 0,

name: "BODY",

stretch: go.GraphObject.Fill,

minSize: new go.Size(68, 68),

maxSize: new go.Size(68, 68)

},

$$(go.Shape, "TriangleRight", {

fill: "#FFFFFF",

strokeWidth: 1

},

new go.Binding("stroke", "state", function (state) {

return state ? "red" : "blue";

})

),

$$(go.TextBlock, {

margin: 5,

textAlign: "center",

font: "10px Segoe UI,sans-serif",

stroke: "black",

editable: false,

isMultiline: false

},

new go.Binding("text", "key")

)

),

$$(go.Panel, {

row: 0,

column: 1,

fromSpot: go.Spot.Right,

toSpot: go.Spot.Right,

cursor: "pointer",

fromLinkable: true,

portId: "input"

},

new go.Binding("portId", "portId").makeTwoWay(),

$$(go.Shape, "Ellipse", {

stroke: null,

strokeWidth: 0,

desiredSize: webModeling.config.portSize,

fill: "black"

}

)

)

);

if (nodeContextMenu) {

inputTemplate.contextMenu = nodeContextMenu;

}

return inputTemplate;

}

function getNodeTemplate(nodeContextMenu, portContextMenu) {

var nodeTemplate = $$(go.Node, "Table", {

locationObjectName: "BODY",

locationSpot: go.Spot.Center,

selectionObjectName: "BODY",

contextMenu: nodeContextMenu,

resizable: true,

minSize: new go.Size(76, 76)

},

$$(go.RowColumnDefinition, { column: 0, width: 8 }),

$$(go.RowColumnDefinition, { column: 2, width: 8 }),

$$(go.RowColumnDefinition, { row: 0, width: 8 }),

$$(go.RowColumnDefinition, { row: 2, width: 8 }),

new go.Binding("location", "loc", go.Point.parse).makeTwoWay(go.Point.stringify),

new go.Binding("width", "width").makeTwoWay(),

new go.Binding("height", "height").makeTwoWay(),

new go.Binding("key", "key").makeTwoWay(),

$$(go.Panel, "Auto", {

row: 1,

column: 1,

name: "BODY",

stretch: go.GraphObject.Fill

},

$$(go.Shape, "Rectangle", {

fill: "#FFFFFF",

stroke: "black",

strokeWidth: 1,

minSize: new go.Size(60, 60)

},

new go.Binding("behaviour", "behaviour"),

new go.Binding("fill", "behaviour", function (behaviour) {

return behaviour ? '#5cb85c' : 'white';

})

),

$$(go.TextBlock, {

margin: 5,

textAlign: "center",

font: "14px Segoe UI,sans-serif",

stroke: "black",

editable: true,

isMultiline: false

},

new go.Binding("text", "displayText").makeTwoWay()

)

),

$$(go.Panel, "Vertical", {

alignment: go.Spot.Top

},

new go.Binding("itemArray", "leftInputs"), {

row: 1,

column: 0,

itemTemplate: getPortTemplate(portContextMenu, "left", go.Spot.Left, "Rectangle")

}

),

$$(go.Panel, "Vertical", {

alignment: go.Spot.Bottom

},

new go.Binding("itemArray", "leftOutputs"), {

row: 1,

column: 0,

itemTemplate: getPortTemplate(portContextMenu, "left", go.Spot.Left, "Ellipse")

}

),

$$(go.Panel, "Horizontal", {

alignment: go.Spot.Right

},

new go.Binding("itemArray", "topInputs"), {

row: 0,

column: 1,

itemTemplate: getPortTemplate(portContextMenu, "top", go.Spot.Top, "Rectangle")

}

),

$$(go.Panel, "Horizontal", {

alignment: go.Spot.Left

},

new go.Binding("itemArray", "topOutputs"), {

row: 0,

column: 1,

itemTemplate: getPortTemplate(portContextMenu, "top", go.Spot.Top, "Ellipse")

}

),

$$(go.Panel, "Vertical", {

alignment: go.Spot.Bottom

},

new go.Binding("itemArray", "rightInputs"), {

row: 1,

column: 2,

itemTemplate: getPortTemplate(portContextMenu, "right", go.Spot.Right, "Rectangle")

}

),

$$(go.Panel, "Vertical", {

alignment: go.Spot.Top

},

new go.Binding("itemArray", "rightOutputs"), {

row: 1,

column: 2,

itemTemplate: getPortTemplate(portContextMenu, "right", go.Spot.Right, "Ellipse")

}

),

$$(go.Panel, "Horizontal", {

alignment: go.Spot.Left

},

new go.Binding("itemArray", "bottomInputs"), {

row: 2,

column: 1,

itemTemplate: getPortTemplate(portContextMenu, "bottom", go.Spot.Bottom, "Rectangle")

}

),

$$(go.Panel, "Horizontal", {

alignment: go.Spot.Right

},

new go.Binding("itemArray", "bottomOutputs"), {

row: 2,

column: 1,

itemTemplate: getPortTemplate(portContextMenu, "bottom", go.Spot.Bottom, "Ellipse")

}

)

);

return nodeTemplate;

}

function getPortTemplate(contextMenu, side, spot, shape) {

return $$(go.Panel, {

\_side: side,

fromSpot: spot,

toSpot: spot,

cursor: "pointer",

contextMenu: contextMenu,

toMaxLinks: 1

},

new go.Binding("portId", "portId"),

new go.Binding("toLinkable", "toLinkable"),

new go.Binding("fromLinkable", "fromLinkable"),

new go.Binding("state", "state").makeTwoWay(),

$$(go.Shape, shape, {

stroke: null,

strokeWidth: 0,

desiredSize: webModeling.config.portSize,

margin: new go.Margin(1, 0)

},

new go.Binding("fill", "portColor")

)

);

}

function getPortMenu(diagram) {

var portMenu = $$(go.Adornment, "Vertical",

$$("ContextMenuButton", $$(go.TextBlock, "Remove port"), {

click: function (e, obj) {

removePort(diagram, obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Remove side ports"), {

click: function (e, obj) {

var nodedata = obj.part.adornedObject.part.data;

var side = obj.part.adornedObject.\_side;

removeAllSidePorts(diagram, nodedata, side);

}

})

);

return portMenu;

}

function removeAllSidePorts(diagram, node, side) {

diagram.startTransaction("removeAllSidePorts");

diagram.model.setDataProperty(node, side + "Inputs", []);

diagram.model.setDataProperty(node, side + "Outputs", []);

diagram.commitTransaction("removeAllSidePorts");

}

function removePort(diagram, port) {

diagram.startTransaction("removePort");

var pid = port.portId;

var arr = port.panel.itemArray;

for (var i = 0; i < arr.length; i++) {

if (arr[i].portId === pid) {

diagram.model.removeArrayItem(arr, i);

break;

}

}

diagram.commitTransaction("removePort");

}

// diagram

function initializeDiagram(context) {

var diagram = $$(go.Diagram, context.config.diagramContainer, {

initialContentAlignment: go.Spot.Center,

"undoManager.isEnabled": true,

"animationManager.duration": 800,

allowDrop: true,

linkTemplate: getLinkTemplate()

});

diagram.events = $(diagram);

var nodeMenu = getNodeMenu(diagram);

var portMenu = getPortMenu(diagram);

var ioMenu = getIOMenu(diagram);

diagram.nodeTemplateMap.add("NODE", getNodeTemplate(nodeMenu, portMenu));

diagram.nodeTemplateMap.add("INPUT", getInputTemplate(ioMenu));

diagram.nodeTemplateMap.add("OUTPUT", getOutputTemplate());

diagram.addDiagramListener("Modified", function () {

var button = document.getElementById("SaveButton");

if (button) button.disabled = !context.diagram.isModified;

var idx = document.title.indexOf("\*");

if (context.diagram.isModified) {

if (idx < 0) document.title += "\*";

} else {

if (idx >= 0) document.title = document.title.substr(0, idx);

}

});

context.diagram = diagram;

}

function getLinkTemplate() {

var linkTemplate = $$(go.Link, {

routing: go.Link.AvoidsNodes,

curve: go.Link.JumpOver,

reshapable: true,

resegmentable: true,

relinkableFrom: true,

relinkableTo: true

},

new go.Binding("points").makeTwoWay(),

new go.Binding("state", "state").makeTwoWay(),

$$(go.Shape, {

stroke: "blue",

strokeWidth: 2

},

new go.Binding("stroke", "state", function (state) {

return state ? "red" : "blue";

})

),

$$(go.Shape, { toArrow: "Standard" })

);

return linkTemplate;

}

function getNodeMenu(diagram) {

var nodeMenu = $$(go.Adornment, "Vertical",

$$("ContextMenuButton", $$(go.TextBlock, "Add top input"), {

click: function (e, obj) {

addPort(diagram, "top", "Inputs", obj.part.adornedObject);

}

$$("ContextMenuButton", $$(go.TextBlock, "Add left input"), {

click: function (e, obj) {addPort(diagram, "left", "Inputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add right input"), {

click: function (e, obj) {

addPort(diagram, "right", "Inputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add bottom input"), {

click: function (e, obj) {

addPort(diagram, "bottom", "Inputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add top output"), {

click: function (e, obj) {

addPort(diagram, "top", "Outputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add left output"), {

click: function (e, obj) {

addPort(diagram, "left", "Outputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add right output"), {

click: function (e, obj) {

addPort(diagram, "right", "Outputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add bottom output"), {

click: function (e, obj) {

addPort(diagram, "bottom", "Outputs", obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Add as template"), {

click: function (e, obj) {

addAsTemplates(webModeling.palette, obj.part.adornedObject);

}

}),

$$("ContextMenuButton", $$(go.TextBlock, "Change behaviour"), {

click: function (e, obj) {

var node = obj.part.adornedObject;

getNodeBehaviour(node);

$(webModeling.config.saveNodeBehaviourButton).off();

$(webModeling.config.behaviourPopup).modal("show");

$(webModeling.config.saveNodeBehaviourButton).on("click", function () {

var code = document.getElementById("myNodeBehaviour").value;

setNodeBehaviour(diagram, node, code);

$(webModeling.config.behaviourPopup).modal("hide");

});

}

})

);

return nodeMenu;

}

function setNodeBehaviour(diagram, node, code) {

diagram.model.startTransaction("setNodeBehaviour");

diagram.model.setDataProperty(node.data, "behaviour", code);

diagram.model.commitTransaction("setNodeBehaviour");

}

function getNodeBehaviour(node) {

var behaviour = node.data.behaviour;

if (behaviour != undefined) {

document.getElementById("myNodeBehaviour").value = behaviour;

} else {

document.getElementById("myNodeBehaviour").value = "";

}

}

function addAsTemplates(palette, element) {

palette.startTransaction("addTemplate");

palette.model.addNodeData(\_.clone(element.data));

palette.commitTransaction("addTemplate");

}

function addPort(diagram, side, type, node) {

diagram.startTransaction("addPort");

if (!(node instanceof go.Node)) return;

var i = 0;

while (node.findPort(side + i.toString()) !== node) i++;

var name = side + i.toString();

var arr = node.data[side + type];

if (arr) {

var inputDirection = (type === "Inputs");

var newportdata = {

portId: name,

portColor: 'black',

fromLinkable: !inputDirection,

toLinkable: inputDirection,

state: false

};

diagram.model.insertArrayItem(arr, -1, newportdata);

}

diagram.commitTransaction("addPort");

}

function getIOMenu(diagram) {

var iOMenu = $$(go.Adornment, "Vertical",

$$("ContextMenuButton", $$(go.TextBlock, "Change state"), {

click: function (e, obj) {

changeIOState(diagram, obj.part.adornedObject);

}

})

);

return iOMenu;

}

function changeIOState(diagram, node) {

diagram.model.startTransaction("invert");

diagram.model.setDataProperty(node.data, "state", !node.data.state);

diagram.events.trigger("IOStateChanged");

diagram.model.commitTransaction("invert");

}

// overview

function intializeOverview(context) {

context.overview = $$(go.Overview, "myOverviewDiv", {

observed: context.diagram,

contentAlignment: go.Spot.Center

});

}

// palette

function initializePalette(context) {

var palette = $$(go.Palette, context.config.paletteContainer, {

initialContentAlignment: go.Spot.Center,

"animationManager.duration": 800,

allowDelete: true

});

var nodemenu = getPaletteNodeMenu(palette);

var portMenu = getPortMenu(palette);

palette.nodeTemplateMap.add("NODE", getNodeTemplate(nodemenu, portMenu));

palette.nodeTemplateMap.add("INPUT", getInputTemplate());

palette.nodeTemplateMap.add("OUTPUT", getOutputTemplate());

palette.model = new go.GraphLinksModel([

getEmptyNodeData(),

getInputNodeData(false),

getOutputNodeData(false),

getEmptyNodeData()]

);

palette.addDiagramListener("Modified", function () {

var button = document.getElementById("SaveButton");

if (button) button.disabled = !palette.isModified;

var idx = document.title.indexOf("\*");

if (palette.isModified) {

if (idx < 0) document.title += "\*";

} else {

if (idx >= 0) document.title = document.title.substr(0, idx);

}

});

context.palette = palette;

}

function getPaletteNodeMenu(palette) {

var nodeMenu = $$(go.Adornment, "Vertical",

$$("ContextMenuButton", $$(go.TextBlock, "Remove"), {

click: function (e, obj) {

removeElement(palette, obj.part.adornedObject);

}

})

);

return nodeMenu;

}

function removeElement(diagram, element) {

diagram.startTransaction("removeElement");

diagram.model.removeNodeData(element.data);

diagram.commitTransaction("removeElement");

}

function getEmptyNodeData() {

return {

category: "NODE",

key: "Unit",

leftInputs: [],

leftOutputs: [],

rightInputs: [],

rightOutputs: [],

topInputs: [],

topOutputs: [],

bottomInputs: [],

bottomOutputs: []

};

}

function getInputNodeData(inputState) {

return {

category: "INPUT",

key: "In",

state: inputState,

portId: "input"

}

}

function getOutputNodeData(outputState) {

return {

category: "OUTPUT",

key: "Out",

state: outputState,

portId: "output"

}

}

ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация дипломного проекта

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов