|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

*Разработка программно-аппаратной части сервиса для конференции “Студенческая весна”*

Студент ИУ7-73  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Зыкин**  

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.О. Рогозин**  

(Подпись, дата) (И.О.Фамилия)

*2019 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

(Индекс)

И.В.Рудаков

(И.О.Фамилия)

« \_\_\_ » **\_\_\_\_\_\_\_\_\_\_\_\_\_** 2019 г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине Компьютерные сети

Студент группы ИУ7-73

Зыкин Данила Андреевич

(Фамилия, имя, отчество)

Тема курсового проекта Разработка программно-аппаратной части сервиса для конференции “Студенческая весна”

Направленность КП (учебный, исследовательский, практический, производственный, др.)

практический

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 5 нед., 50% к 10 нед., 75% к 14 нед., 100% к 17 нед.

***Задание*** Разработать программно-аппаратную часть портала для конференции по поддержке и развитию студенческого творчества “Российская студенческая весна”.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 15-20 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, структура комплекса программ, интерфейс, характеристики разработанного ПО.

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**Руководитель курсового проекта**    Н.О. Рогозин

(Подпись, дата) (И.О.Фамилия)

**Студент**    Д.А. Зыкин  

(Подпись, дата) (И.О.Фамилия)

**Содержание**

[**Введение**](#_mwzz0krorjsj) **3**

[**Глоссарий**](#_w8f8n7jg5iup) **5**

[**1. Аналитическая часть**](#_lgalm5xvyjkt) **6**

[1.1 Проблемы](#_t1v8x4ze12ck) 7

[1.2 Возможные решения](#_64u3lyjes46d) 8

[**2. Конструкторская часть.**](#_i5v73y6ud7g0) **11**

[2.1 Central Authentication Service(CAS)](#_erudq725hasw) 11

[2.2 Кодогенерация](#_99zirmjrhhfa) 13

[**3. Технологическая часть**](#_6fge6pydtej5) **16**

[3.1 Взаимодействие с базой данных](#_ns6som1uqybx) 16

[3.2 Программно-аппаратная часть сервиса](#_vwwbdlmkkjbs) 19

[3.2.1 Репозитории](#_6kby5vnmwkyk) 20

[3.2.2 Сервисы](#_8bhb7rxrrtyg) 20

[3.2.3 Интеграционные тесты](#_ef5on9c58z8) 23

[3.2.3 Контроллеры](#_2342kluigne4) 24

[3.3 Хранение документов](#_xa9sjcti2vnd) 30

[3.4 Авторизация](#_fv3piyhbtuww) 33

[**4. Исследовательская часть**](#_djkbb6njk6sz) **36**

[4.1 Интерфейс](#_34zajqaden3x) 36

[**Заключение**](#_d7n0pgss1l6r) **37**

[**Список использованной литературы**](#_9nh911cn3wnk) **38**

### Введение

Объектом разработки данной курсовой работы является портал для студентов, участвующих в Студенческой весне. Целью работы является проектирование и реализация сервисов, необходимых для работы портала.

Результатом является разработанный продукт, выполняющий поставленные задачи.

На языке программирования C# и с использованием MSSQL разработана программно-аппаратная части сервиса, позволяющий в дальнейшем приступить к разработке клиентской стороны пользовательского интерфейса.

Результаты работы предполагается использовать как для автоматизации процесса подачи заявок на участие в Студенческой Весне, так и для управления заявками со стороны модераторов. Использование портала будет полезно также для отслеживания статусов заявок со стороны студентов и уменьшение нагрузки на сотрудников МГТУ им. Н.Э. Баумана.

Основной целью работы является автоматизация рутинных процессов при помощи использования программно-аппаратная часть веб-сервиса. Необходимо выявить все потребности сотрудников МГТУ им Н.Э. Баумана, проанализировать их и реализовать.

Работа состоит из двух частей. Первая часть заключалась в проектировании и разработки архитектуры будущего сервиса, создании и наполнении базы данных, а также реализации API для клиентской стороны пользовательского интерфейса. Эта часть была выполнена в ходе выполнения данной курсовой работы.

Вторая часть заключается в разработке клиентской стороны пользовательского интерфейса. Эта часть не входит в данную курсовую работу и будет сделана позже.

### 

### Глоссарий

*Данные* — это совокупность сведений, зафиксированных на определенном носителе в форме, пригодной для постоянного хранения, передачи и обработки. Преобразование и обработка данных позволяет получить информацию.

*Информация* — это результат преобразования и анализа данных. Отличие информации от данных состоит в том, что данные — это фиксированные сведения о событиях и явлениях, которые хранятся на определенных носителях, а информация появляется в результате обработки данных при решении конкретных задач.

*База данных* — это поименованная совокупность структурированных данных, относящихся к определенной предметной области.

*Информационный объект* — описание некоторого реального объекта, явления, процесса, события в виде логически связанных атрибутов.

## 1. Аналитическая часть

Всероссийская конференция Студенческая Весна является традиционным ежегодным научно-техническим мероприятием для студентов МГТУ им. Н.Э.Баумана, на которое приглашаются студенты других Российских технических ВУЗов. Цель конференции — рассказать о самостоятельно выполненных научно-исследовательских проектах, приобрести навыки участия в подобных мероприятиях, обменяться идеями и опытом, соотнести уровень собственных разработок с работами других участников. Конференция имеет конкурсный характер. В ходе работы экспертная комиссия определяет лучшие студенческие работы и распределяет призовые места. Итоги работы экспертная комиссия объявляет в заключительный день работы конференции.

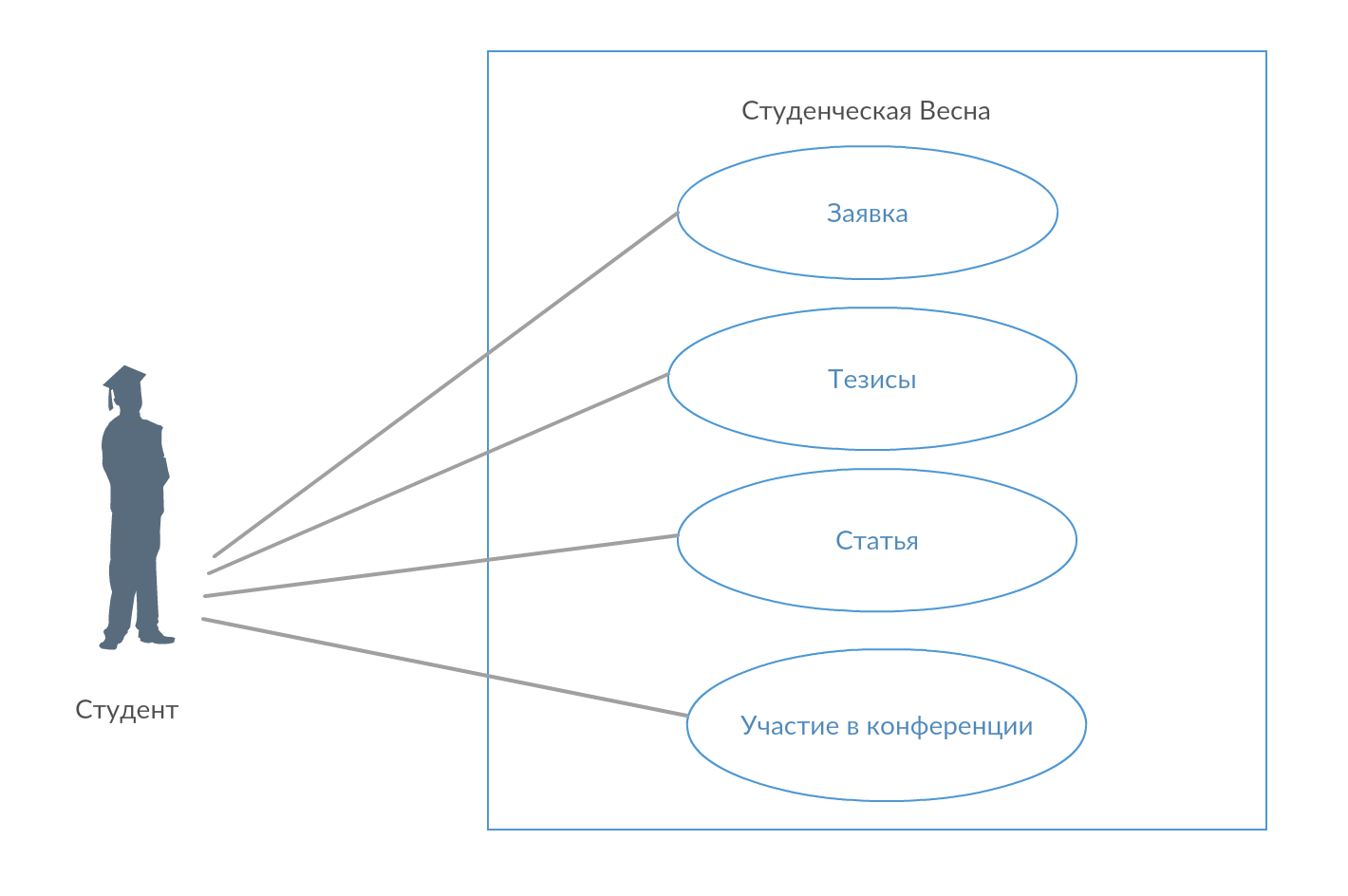


Рис. 1.1 Диаграмма взаимодействия студента с конференцией

### 1.1 Проблемы

Ежегодно количество участников студенческой конференции растет и в связи с этим увеличивается нагрузка на сотрудников МГТУ им Н.Э. Баумана, которые обрабатывают заявки, исправляют тезисы и занимаются публикацией статей. Помимо вышеперечисленных обязанностей, сотрудники отвечают на вопросы студентов и подготавливают документы для проведения конференции. Такая нагрузка порождает множество проблем связанных с проведением данного мероприятия.

### 1.2 Возможные решения

* Увеличение штата сотрудников, что негативно скажется на бюджете отдела и управляемости системы;
* Использование технологий для автоматизации процессов (CRM или собственный портал)

Технологическое решение проблем имеет явное преимущество над остальными. Из уже существующих решений отдел по взаимодействию со студентами использовали Redmine.

Redmine — открытое серверное веб-приложение для управления проектами и задачами (в том числе для отслеживания ошибок). Redmine написан на Ruby и представляет собой приложение на основе широко известного веб-фреймворка Ruby on Rails. Распространяется согласно GNU General Public License[5].

Redmine предоставляет следующие возможности:

* ведение нескольких проектов; гибкая система доступа, основанная на ролях;
* система отслеживания ошибок;
* диаграммы Ганта и календарь;
* ведение новостей проекта, документов и управление файлами;
* оповещение об изменениях с помощью RSS-потоков и электронной почты;
* форумы для каждого проекта;
* учёт временных затрат;
* настраиваемые произвольные поля для инцидентов, временных затрат, проектов и пользователей;
* легкая интеграция с системами управления версиями (SVN, CVS, Git, Mercurial, Bazaar и Darcs);
* создание записей об ошибках на основе полученных писем;
* поддержка множественной аутентификации LDAP;
* возможность самостоятельной регистрации новых пользователей;
* многоязычный интерфейс (в том числе русский);
* поддержка СУБД MySQL, Microsoft SQL Server[2], PostgreSQL, SQLite, Oracle.

Redmine, имея большой функционал, оказался слишком сложен для внедрения в существующую организационную структуру отдела. Доработка данного продукта под требования МГТУ им. Н.Э. Баумана потребует колоссального количества времени разработчиков.

**Вывод**

В рамках данной работы производится анализ требований к порталу с целью определения наиболее подходящего набора технологий на основе которых будут разрабатывается сервисы. Для решения данной задачи требуется декомпозировать задачи и выделить микросервисы.

Была поставлена задача разработать программно-аппаратную часть сервиса, которая могла бы получать на вход запросы с файлами, обрабатывать их и хранить в безопасности.

## 2. Конструкторская часть.

Портал состоит из базы данных, сервисов и авторизации. Часть данных о пользователях синхронизируются из других сервисов МГТУ им Н.Э. Баумана.

### 2.1 Central Authentication Service(CAS)

Существует много реализаций идеи единой авторизации, но , по ряду причин, выбор МГТУ им Н.Э. Баумана сделан в пользу JASIG CAS[4].

**Причины**:

* CAS полностью написан на Java с применением Spring. Для сборки используется Maven;
* Open Source;
* Поддержка нескольких реализаций LDAP;
* Активное сообщество;
* Приятные портал JASIG и вики, есть много статей в сети;
* Клиенты под все популярные платформы;
* Поддержка разных способов аутентификации;
* Частые релизы;
* Относительная простота настройки.

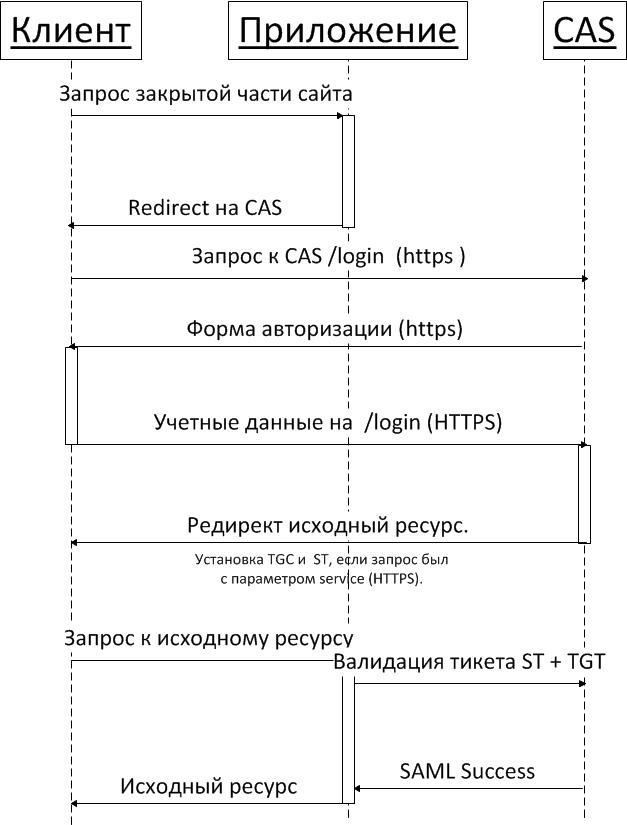


Рис. 2.1 Схема работы CAS

Аналогичную схему использует Яндекс и Google.

На рисунке 2.1 рассмотрена схема работы CAS:

1. Пользователь открывает портал.

2. Запрос получает пока что не CAS, а портал.

3. Портал видит, что у пользователя нет сессии и перенаправляет пользователя на CAS. Для пользователя всё просто — он открыл портал и сразу увидел страницу логина.

4. Пользователь вводит логин/пароль.

5. CAS их валидирует.

6. CAS генерирует случайный набор символов — «тикет». В будущем он идентифицирует пользователя.

7. Запрос перенаправляется на портал, тикет передается как параметр.

8. Портал делает запрос к CAS и проверяет валидность тикета.

9. Если есть, в ответ CAS отправляет логин пользователя и прочие данные.

### 2.2 Кодогенерация

В техническом задании заложена основа для дальнейшего масштабирования портала и упрощения работы с программно-аппаратной частью сервиса.

Для выполнения данных требований портал разделен на сервисы, репозитории и контроллеры. Шаблоны кодогенерации позволяют значительно уменьшить время разработки нового функционала.

На основе базового интерфейса репозитория, базового интерфейса сервиса и других интерфейсов, которые описаны в технологической части данной курсовой работы, генерируется реализация сервиса и контроллера по модели сущности.

Модель сущности генерируется при обращении к таблицам в базе данных. После всех операций разработчику необходимо определить соответствия данных между потенциально различными семантиками одного объекта или разных объектов и реализовать расширенный класс для методов PUT и DELETE.

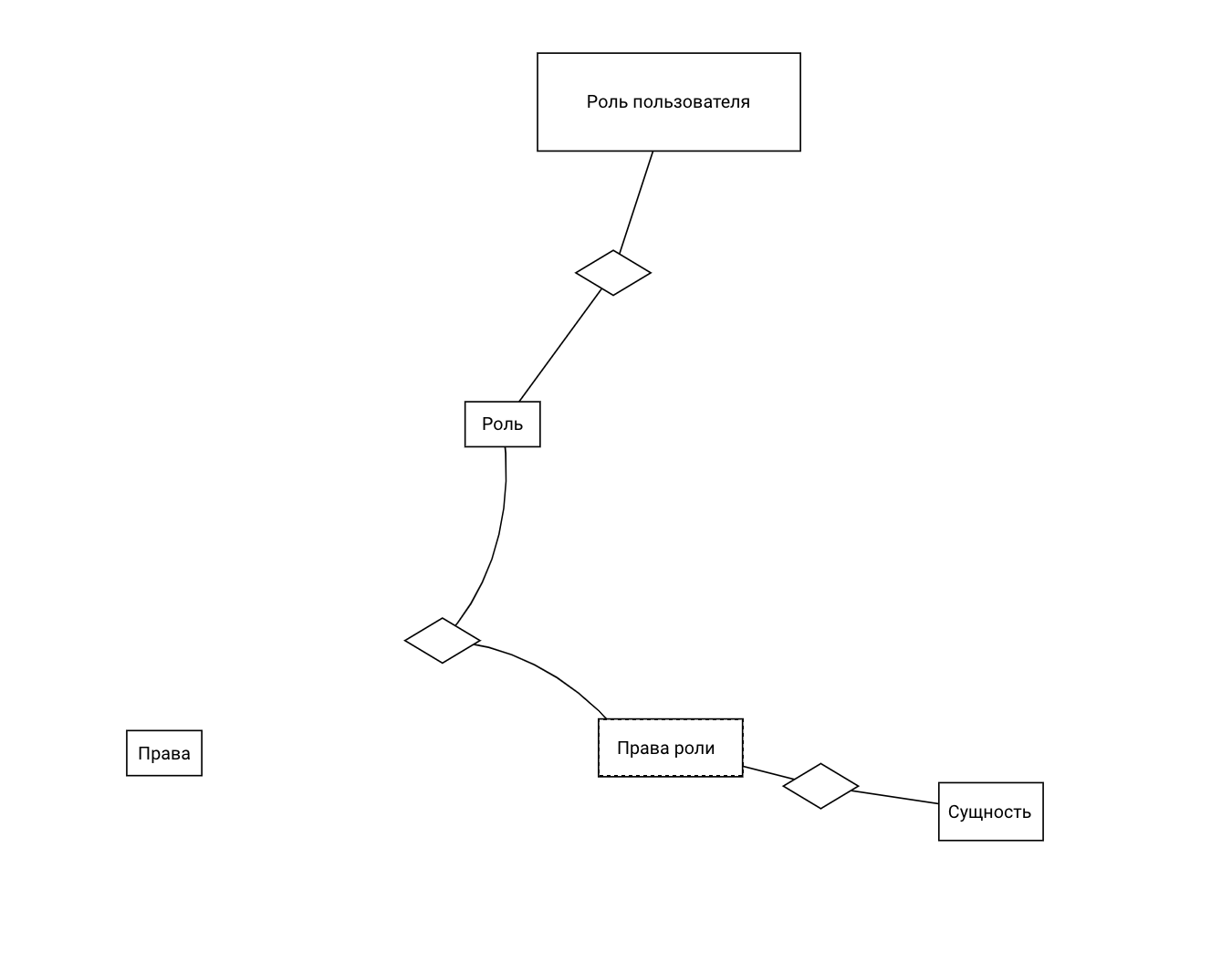


Рис. 2.2 Ролевая схема

На рисунке 2.2 представлена модель “сущность — связь” (ERD) для распределения прав на различные сервисы внутри портала. При помощи реализованной связи можно создавать роли и права по мере необходимости и сделать распределение доступа более гибким к тем или иным сервисам.

## 3. Технологическая часть

В качестве СУБД используется Microsoft SQL Server. Языком программирования был выбран C#, поскольку он предоставляет удобный объектный интерфейс для взаимодействия программы и базы данных.

### 3.1 Взаимодействие с базой данных

Для работы с базой данных используется пакет Entity Framework (EF), разработанный Microsoft. Это объектно-реляционный модуль сопоставления (object-relational mapping, ORM), позволяющий разработчикам C# работать с реляционными данными с помощью LINQ-запросов и объектов, специализированных для доменов.

При использовании разработчиком подхода "Code First", таблицы БД создаются в соответствии с описываемыми в приложении классами данных. Изменения в БД производятся с помощью миграций. На основе изменений в классах данных генерируются специальные классы Code First Migrations, использование которых позволяет производить необходимые модификации структуры базы данных, а также откатывать эти модификации в случае необходимости.

Основу функциональности Entity Framework составляют классы, находящиеся в пространстве имен System.Data.Entity. Среди всего набора классов этого пространства имен следует выделить следующие:

* DbContext: определяет контекст данных, используемый для взаимодействия с базой данных;
* DbModelBuilder: сопоставляет классы на языке C# с сущностями в базе данных;
* DbSet<TEntity>: представляет набор сущностей, хранящихся в базе данных.

В любом приложении, работающим с БД через Entity Framework, используется экземпляр контекста данных DbContext и экземпляры наборов данных DbSet, с помощью которых разработчик взаимодействует с таблицами из БД.

Использование EF избавляет разработчика от необходимости рутинного написания SQL-запросов для доступа к базе, а также позволяет использовать удобные интерфейсы для взаимодействия с данными: фактически, каждая строка в таблице БД может быть представлена в приложении экземпляром соответствующего класса.

Работая с объектом класса, мы можем обращаться к его свойствам, чтобы получить доступ к выделенным связям между документами, вместо того чтобы формировать SQL-запросы, использующие соединения (join) таблиц.

Содержимое документов представляется классом DocContens:

[Table("Documents", Schema = "docs")]

public class DocContents

{

[Key]

public int Id { get; set; }

public byte[] Contents { … }

public Document Document { get; set; }

[NotMapped]

public string ContentsString { … }

[NotMapped]

public string[] ContentsStringArray { … }

}

В данном случае, при работе с содержимым документа мы можем не только получить доступ к "сырому" массиву байт, но и преобразовать его в строку в формате Unicode или массив строк.

Содержимое документов подгружается лениво: не в момент создания объекта Document, а только при обращении к Document.Contents. Обрабатывать непосредственно содержимое документов требуется только на первом этапе анализа, когда происходит поиск списка источников и выделение заголовков статей. При дальнейшей работе с документами, когда их содержимое уже не представляет интереса, его выгрузка из базы данных не происходит, благодаря чему увеличивается быстродействие программы.

Недостатком использования Entity Framework является снижение производительности программы в некоторых случаях. Например, если требуется удалить из таблицы большое количество элементов, SQL-запрос DELETE FROM *table* WHERE *conditions* будет выполнен в несколько раз быстрее, нежели обращение к контексту данных Context.*table*.RemoveRange(*conditions*).

### 3.2 Программно-аппаратная часть сервиса

В проекте используется Clean architecture. Вершиной абстракции является сборка BMSTU.SV.Core содержащая бизнес-логику. Разработка ведётся по методологии TDD.

Используемые технологии:

* .net core 2.2;
* xunit;
* Docker;
* Web API;
* Dapper;
* Swagger;
* FluentMigrator;
* Ms SQL Server.

#### 3.2.1 Репозитории

Для удобной работы с базой данных был создан базовый интерфейс:

public interface IBaseRepository<TDbEntity> where TDbEntity: class, IEntity

{

IQueryable<TDbEntity> GetAll(params Expression<Func<TDbEntity, object>>[] loadWith);

Task<TDbEntity> Get(long id, params Expression<Func<TDbEntity, object>>[] loadWith);

Task<long> Add(TDbEntity newItem);

Task<ICollection<TDbEntity>> AddRange(ICollection<TDbEntity> newItemCollection);

Task<TDbEntity> Update(TDbEntity item);

Task<ICollection<TDbEntity>> UpdateRange(ICollection<TDbEntity> itemCollection);

Task Delete(long id);

Task DeletePhysically(long id); bool CanEdit();

}

Реализация данного репозитория производится в сервисах портала.

#### 3.2.2 Сервисы

Базовый интерфейс сервиса

public interface IBaseService<TRequest, in TDbEntity, TDto> :

IViewService<TDto>,

IAddOrUpdateService<TRequest, TDto>,

IDeleteService<TRequest>

where TRequest : class

where TDbEntity: class, IEntity

where TDto: class, new() { }

Кодогенерация сервиса:

<#+

void AddViewService(TemplateFileManager fileManager, string dbModel) {

fileManager.StartNewFile(dbModel + "Service.generated.cs", "BMSTU.SV.Service", "Implementation\\Services\\Generated");

#>

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated from a template.

//

// Manual changes to this file may cause unexpected behavior in your application.

// Manual changes to this file will be overwritten if the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

using BMSTU.SV.DataCore;

using BMSTU.SV.Service.Contract.Repositories.Base;

using BMSTU.SV.Service.Contract.Services;

using BMSTU.SV.Service.Contract.Services.Base;

using BMSTU.SV.Service.Implementation.Services.Base;

using BMSTU.SV.Common.Models;

using BMSTU.SV.Common.Models.Snap.Requests;

using BMSTU.SV.Common.Models.Snap.Responses;

namespace BMSTU.SV.Service.Contract.Services {

public partial interface I<#=dbModel#>Service : IViewService<<#=dbModel#>Response>

{ }

}

namespace BMSTU.SV.Service.Implementation.Services {

public partial class <#=dbModel#>Service : BaseViewService<<#=dbModel#>, <#=dbModel#>Response>, I<#=dbModel#>Service {

public <#=dbModel#>Service(IBaseRepository<<#=dbModel#>> baseRepository) : base(baseRepository)

{ }

}

}

<#+

fileManager.EndBlock();

}

void AddCrudService(TemplateFileManager fileManager, string dbModel) {

fileManager.StartNewFile(dbModel + "Service.generated.cs", "BMSTU.SV.Service", "Implementation\\Services\\Generated");

#>

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated from a template.

//

// Manual changes to this file may cause unexpected behavior in your application.

// Manual changes to this file will be overwritten if the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

using BMSTU.SV.DataCore;

using BMSTU.SV.Service.Contract.Repositories.Base;

using BMSTU.SV.Service.Contract.Services;

using BMSTU.SV.Service.Contract.Services.Base;

using BMSTU.SV.Service.Implementation.Services.Base;

using BMSTU.SV.Common.Models;

using BMSTU.SV.Common.Models.Snap.Requests;

using BMSTU.SV.Common.Models.Snap.Responses;

namespace BMSTU.SV.Service.Contract.Services {

public partial interface I<#=dbModel#>Service : IBaseCRUDService<<#=dbModel#>Request, <#=dbModel#>, <#=dbModel#>Response>

{ }

}

namespace BMSTU.SV.Service.Implementation.Services {

public partial class <#=dbModel#>Service : BaseCRUDService<<#=dbModel#>Request, <#=dbModel#>, <#=dbModel#>Response>, I<#=dbModel#>Service

{ }

}

<#+

fileManager.EndBlock();

}

#>

#### 3.2.3 Интеграционные тесты

Подход, описанный в п 2.2 конструкторской части, позволяет генерировать интеграционные тесты на основе вышеописанных базовых интерфейсов, но в рамках данной курсовой работы это не было реализовано. В дальнейшем это будет сделано для проверки соответствия проектируемых единиц функциональным, приемным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «черного ящика».

После реализации будет доработана система непрерывной интеграции для автоматизации интеграционного тестирования.

#### 3.2.3 Контроллеры

Для того, чтобы в дальнейшем можно было использовать авто сгенерированные тесты, необходимо унифицировать контроллеры. Это было достигнуто при помощи использования шаблонов для контроллеров и деление портала на контроллеры, сервисы и репозитории. Контроллеры были разделены на два типа:

* чтение
* изменение

Шаблон контроллера на чтение данных:

<#+

void AddViewController(TemplateFileManager fileManager, string dbModel, TypeInstance typeInstance) {

string apiRoute;

string targetPath;

string apiNameSpace;

var typeInstanceName = typeInstance?.Name ?? dbModel;

apiRoute = "api/v{version:apiVersion}/[controller]";

apiNameSpace = "BMSTU.SV.Site.Controllers.Api.V2";

targetPath = "Controllers\\Api\\V2\\Generated";

fileManager.StartNewFile(dbModel + "Controller.generated.cs", "BMSTU.SV.Site", targetPath);

#>

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated from a template.

//

// Manual changes to this file may cause unexpected behavior in your application.

// Manual changes to this file will be overwritten if the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using BMSTU.SV.Common.Infrastructure;

using BMSTU.SV.Site.Infrastructure;

using BMSTU.SV.DataCore;

using BMSTU.SV.Common.Models;

using BMSTU.SV.Common.Models.Snap.Requests;

using BMSTU.SV.Common.Models.Snap.Responses;

using BMSTU.SV.Service.Contract.Services;

using System.Threading.Tasks;

namespace <#=apiNameSpace#> {

/// <summary>

/// <#=typeInstanceName#>

/// </summary>

[Authorize]

[ApiVersion("2.0")]

[Route("<#=apiRoute#>")]

public partial class <#=dbModel#>Controller : BaseController {

private readonly I<#=dbModel#>Service \_service;

public <#=dbModel#>Controller(DataManager dataManager, I<#=dbModel#>Service service) : base(dataManager) {

\_service = service;

}

/// <summary>

/// Получить все элементы сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="limit">Кол-во возвращаемых элементов (по-умолчанию 100)</param>

/// <param name="offset">Кол-во пропускаемых элементов</param>

/// <param name="modifiedFrom">Фильтр по дате последнего изменения</param>

[ProducesResponseType(typeof(ResponseWrapper<List<<#=dbModel#>Response>>), 200)]

[HttpGet]

public async Task<IActionResult> Get(int? limit = 100, int? offset = 0, DateTime? modifiedFrom = null) {

var response = await \_service.GetAll(limit, offset, null, modifiedFrom);

return GetResponseFromResponseWrapper(response);

}

/// <summary>

/// Получить отфильтрованные элементы сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="limit">Кол-во возвращаемых элементов (по-умолчанию 100)</param>

/// <param name="offset">Кол-во пропускаемых элементов</param>

/// <param name="dynamicFilters">Набор динамичеких фильтров</param> [ProducesResponseType(typeof(ResponseWrapper<List<<#=dbModel#>Response>>), 200)]

[HttpPost("ByFullFilter")]

public async Task<IActionResult> GetWithFilters(int? limit = 100, int? offset = 0, [FromBody] Dictionary<string, object[]> dynamicFilters = null) {

var response = await \_service.GetAll(limit, offset, dynamicFilters, null);

return GetResponseFromResponseWrapper(response);

}

/// <summary>

/// Получить элемент '<#=typeInstanceName#>' по Id

/// </summary>

/// <param name="id">Id элемента</param>

[ProducesResponseType(typeof(ResponseWrapper<<#=dbModel#>Response>), 200)]

[HttpGet("{id}")]

public async Task<IActionResult> Get(long id) {

var response = await \_service.GetById(id);

return GetResponse(response);

}

}

}

<#+

fileManager.EndBlock();

}

Шаблон контроллера на изменение данных:

void AddCrudController(TemplateFileManager fileManager, string connectionString, string dbModel, TypeInstance typeInstance)

{

string apiRoute;

string targetPath;

string apiNameSpace;

var typeInstanceName = typeInstance?.Name ?? dbModel;

apiRoute = "api/v{version:apiVersion}/[controller]";

apiNameSpace = "BMSTU.SV.Site.Controllers.Api.V2";

targetPath = "Controllers\\Api\\V2\\Generated";

fileManager.StartNewFile(dbModel + "Controller.generated.cs", "BMSTU.SV.Site", targetPath);

#>

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated from a template.

//

// Manual changes to this file may cause unexpected behavior in your application.

// Manual changes to this file will be overwritten if the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using BMSTU.SV.Common.Infrastructure;

using BMSTU.SV.Site.Infrastructure;

using BMSTU.SV.DataCore;

using BMSTU.SV.Common.Models;

using BMSTU.SV.Common.Models.Snap.Requests;

using BMSTU.SV.Common.Models.Snap.Responses;

using BMSTU.SV.Service.Contract.Services;

using System.Threading.Tasks;

namespace <#=apiNameSpace#> {

/// <summary>

/// <#=typeInstanceName#>

/// </summary>

[Authorize]

[ApiVersion("2.0")]

[Route("<#=apiRoute#>")]

public partial class <#=dbModel#>Controller : BaseController

{

private readonly I<#=dbModel#>Service \_service;

public <#=dbModel#>Controller(DataManager dataManager, I<#=dbModel#>Service service) : base(dataManager) {

\_service = service;

}

/// <summary>

/// Получить все элементы сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="limit">Кол-во возвращаемых элементов (по-умолчанию 100)</param>

/// <param name="offset">Кол-во пропускаемых элементов</param>

/// <param name="modifiedFrom">Фильтр по дате последнего изменения</param>

[ProducesResponseType(typeof(ResponseWrapper<List<<#=dbModel#>Response>>), 200)]

[HttpGet]

public async Task<IActionResult> Get(int? limit = 100, int? offset = 0, DateTime? modifiedFrom = null) {

var response = await \_service.GetAll(limit, offset, null, modifiedFrom);

return GetResponseFromResponseWrapper(response);

}

/// <summary>

/// Получить отфильтрованные элементы сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="limit">Кол-во возвращаемых элементов (по-умолчанию 100)</param>

/// <param name="offset">Кол-во пропускаемых элементов</param>

/// <param name="dynamicFilters">Набор динамичеких фильтров</param>

[ProducesResponseType(typeof(ResponseWrapper<List<<#=dbModel#>Response>>), 200)]

[HttpPost("ByFullFilter")]

public async Task<IActionResult> GetWithFilters(int? limit = 100, int? offset = 0, [FromBody] Dictionary<string, object[]> dynamicFilters = null) {

var response = await \_service.GetAll(limit, offset, dynamicFilters, null);

return GetResponseFromResponseWrapper(response);

}

/// <summary>

/// Добавить или изменить элемент сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="item">Элемент для добавления или обновления</param>

[ProducesResponseType(typeof(ResponseWrapper<<#=dbModel#>Response>), 200)]

[HttpPut]

public async Task<IActionResult> Update([FromBody]<#=dbModel#>Request item) {

return GetResponse(await \_service.AddOrUpdate(item));

}

/// <summary>

/// Удалить элемент сущности '<#=typeInstanceName#>'

/// </summary>

/// <param name="item">Элемент для удаления</param>

[ProducesResponseType(typeof(ResponseWrapper<bool>), 200)]

[HttpDelete]

public async Task<IActionResult> Delete([FromBody]<#=dbModel#>Request item) {

await \_service.Delete(item);

return GetResponse(true);

}

}

}

<#+

fileManager.EndBlock();

}

#>

<#+

string GenerateDeleteMethod(string dbModel, string connectionString, TypeInstance typeInstance){

switch (CheckTypeInstanceFields(connectionString, typeInstance)) {

case Entity.IEntityGuid1C:

return "await \_service.DeleteByGuid(item.Guid1C);";

case Entity.IEntityGuid:

return "await \_service.DeleteByGuid(item.Guid);";

default:

return "await \_service.Delete(item);";

}

}

Entity CheckTypeInstanceFields(string connectionString, TypeInstance typeInstance) {

if (typeInstance==null)

return Entity.IEntity;

using (var connection = new SqlConnection(connectionString)) {

connection.Open();

var sql = "SELECT TOP 0 \* FROM " + typeInstance.Table;

using (SqlCommand command = new SqlCommand(sql, connection)) {

using (SqlDataReader reader = command.ExecuteReader()) {

List<string> fields = Enumerable.Range(0, reader.FieldCount) .Select(reader.GetName).ToList();

if (fields.Contains("Guid1C"))

return Entity.IEntityGuid1C;

else if (fields.Contains("Guid"))

return Entity.IEntityGuid;

else

return Entity.IEntity;

}

}

}

return Entity.IEntity;

}

enum Entity {

IEntityGuid1C = 1,

IEntityGuid = 2,

IEntity = 3

}

#>

### 3.3 Хранение документов

Порталу необходимо хранить по несколько документов на каждого пользователя. Ограничения по размеру документов техническим заданием не предусмотрено, поэтому было проанализированы несколько вариантов хранения документов.

Классический подход — документы хранятся в файловой системе, а в базе данных хранятся пути к этим документам.

**Недостатки**[1]:

* документы не удаляются при удалении соответствующей записи в базе данных;
* проблемы при одновременной попытке обновления файла;
* нарушение синхронизации между базой данных и файловой системой при откате транзакции;
* при резервном копировании и восстановлении информации в базе данных может возникнуть рассинхронизация с файловой системой;
* документы не подчиняются ограничениям доступа, наложенным с помощью базы данных.

Хранение документов в базе данных — в данном подходе нужно проанализировать ситуация и принять взвешенное решение для каждого конкретного случая. Ведь при таком способе хранения файлов программно-аппаратная часть сервиса должна при каждом запросе вызывать скрипт, который будет извлекать файл из базы данных, что отрицательно скажется на производительности.

Для поиска решения данной проблемы можно проанализировать несколько вариантов реализации:

1. Перед удалением записи делать SELECT с тем же условием и получать имена файлов, которые надо удалить. Проблема в том, что если удаляемых файлов много, эта операция может занять некоторое время и на это время надо блокировать таблицу на чтение и запись, а во многих случаях это недопустимо.
2. Перед удалением устанавливать у удаляемых записей метку «подлежит удалению», получить все записи с этой меткой и удалить файлы, связанные с этими записями, и наконец удалить все записи с этой меткой. Запросы, работающие с этой таблицей следует доработать, чтобы они не выбирали записи с установленным флагом. Недостатки — необходимость правки множества запросов, к тому же у нас в проекте записи на удаление отбираются достаточно сложным SELECT, которые нельзя переделать в один UPDATE.
3. Использовать триггеры. К сожалению, MySQL не имеет в своем языке поддержки команд работы с файлами, такие команды пришлось бы реализовывать самостоятельно, при помощи изучения исходников MySQL. Из минусов — необходимость доработки MySQL и файлы должны храниться на том же устройстве, предоставляющее сервисы формата «клиент-сервер» в режиме сервера по API, что и база данных.
4. Хранить файлы в базе данных, но отдавать их напрямую программно-аппаратной часть сервиса, без участия клиентской стороны пользовательского интерфейса. Реализовать это можно, написав модуль к веб-серверу который позволял бы отдавать файлы напрямую из MySQL или применив драйвер файловой системы MySQLfs. Такой подход решает все перечисленные выше проблемы, но его недостаток — дополнительные накладные расходы на хранение файлов в MySQL.
5. Специализированный Storage Engine для MySQL, хранящий записи как файлы.

После анализа всего вышеперечисленного можно сделать вывод, что миграция базы данных между различными системами управления базами данных будет крайне затратна по времени.

Наиболее благоприятный подход для поставленных задач — использование облачных решений(Google Cloud Platform).

**Преимущества**:

* мощная инфраструктура;
* большие данные и аналитика;
* полностью управляемые вычисления без серверов;
* функций резервного копирования и восстановления данных внутри облачного хранилища.

### 3.4 Авторизация

В соответствии с требования авторизация должна быть реализована по технология единого входа — технология, при использовании которой пользователь переходит из одного раздела портала в другой без повторной аутентификации. Например, если на веб-портале существует несколько обширных независимых разделов (форум, чат, блог и т. д.) то, пройдя процедуру аутентификации в одном из сервисов, пользователь автоматически получает доступ ко всем остальным, что избавляет его от многократного ввода данных своей учётной записи.

**Преимущества данного подхода:**

* уменьшение парольного хаоса между различными комбинациями имени пользователя и пароля
* уменьшение времени на повторный ввод пароля для одной и той же учётной записи
* поддержка традиционных механизмов аутентификации, таких как имя пользователя и пароль
* снижение расходов на IT-службу за счёт уменьшения количества запросов по восстановлению забытых паролей
* обеспечение безопасности на каждом уровне входа/выхода/доступа к системе без причинения неудобств пользователям

**Недостатки**

Некоторыми экспертами в качестве главного недостатка технологии единого входа отмечается увеличивающаяся важность единственного пароля, при получении которого злоумышленник обретает доступ ко всем ресурсам пользователя, использующего единый вход. Поставщики менеджеров паролей также отмечают использование различных паролей к разным информационным ресурсам как более надежное решение по сравнению с технологией единого входа.

Несмотря на недостатки для работы с авторизацией была выбран выбрана технология единого входа.

**Вывод**

С помощью данной реализации и выбранного стека технологий можно значительно сократить время разработки и уменьшить сложность масштабирования портала.

В результате курсового проекта был реализован необходимой функционал и разработана концепция дальнейшей разработки системы.

## 

## 4. Исследовательская часть

### 4.1 Интерфейс

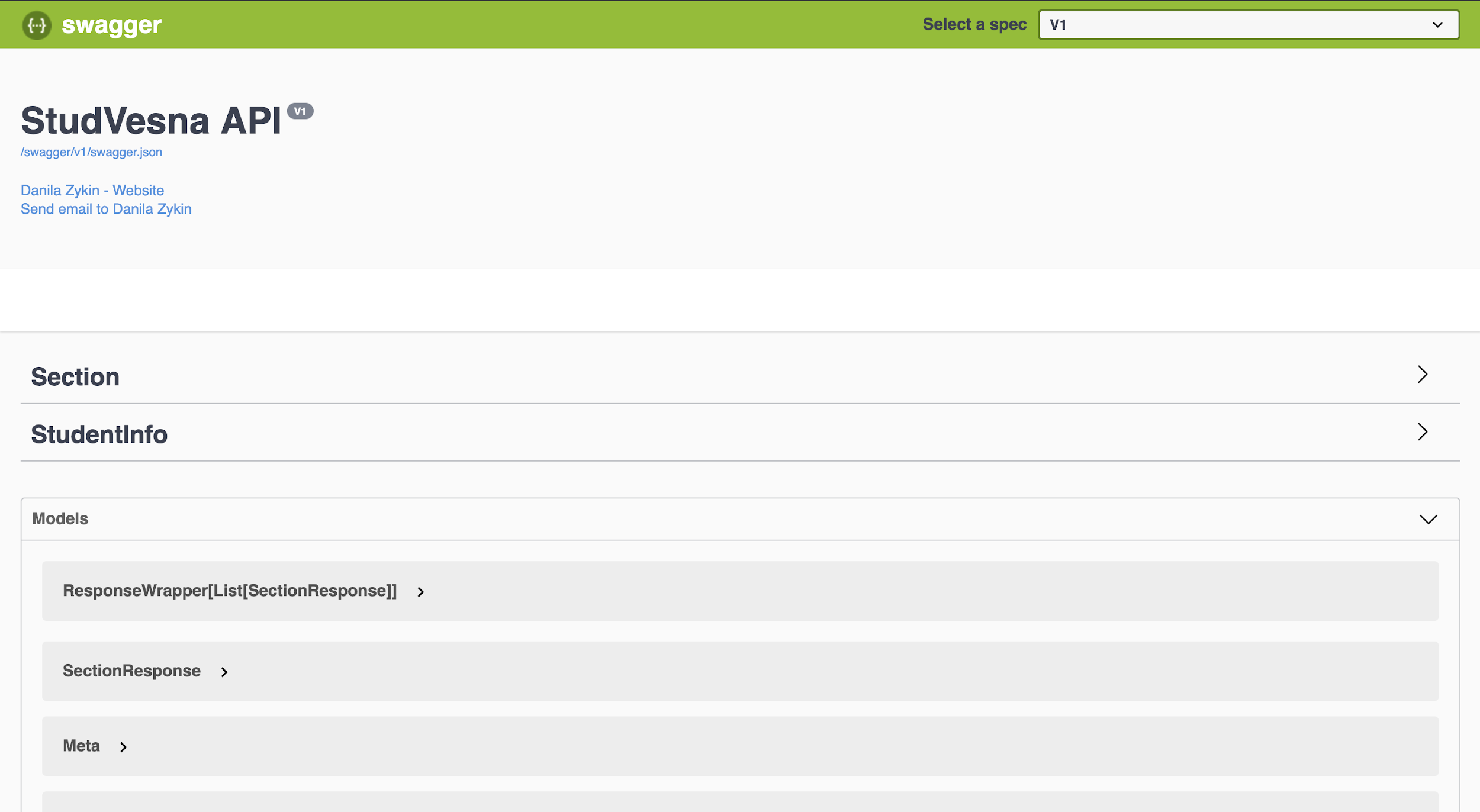


Рис. 4.1 Программно-аппаратная часть сервиса

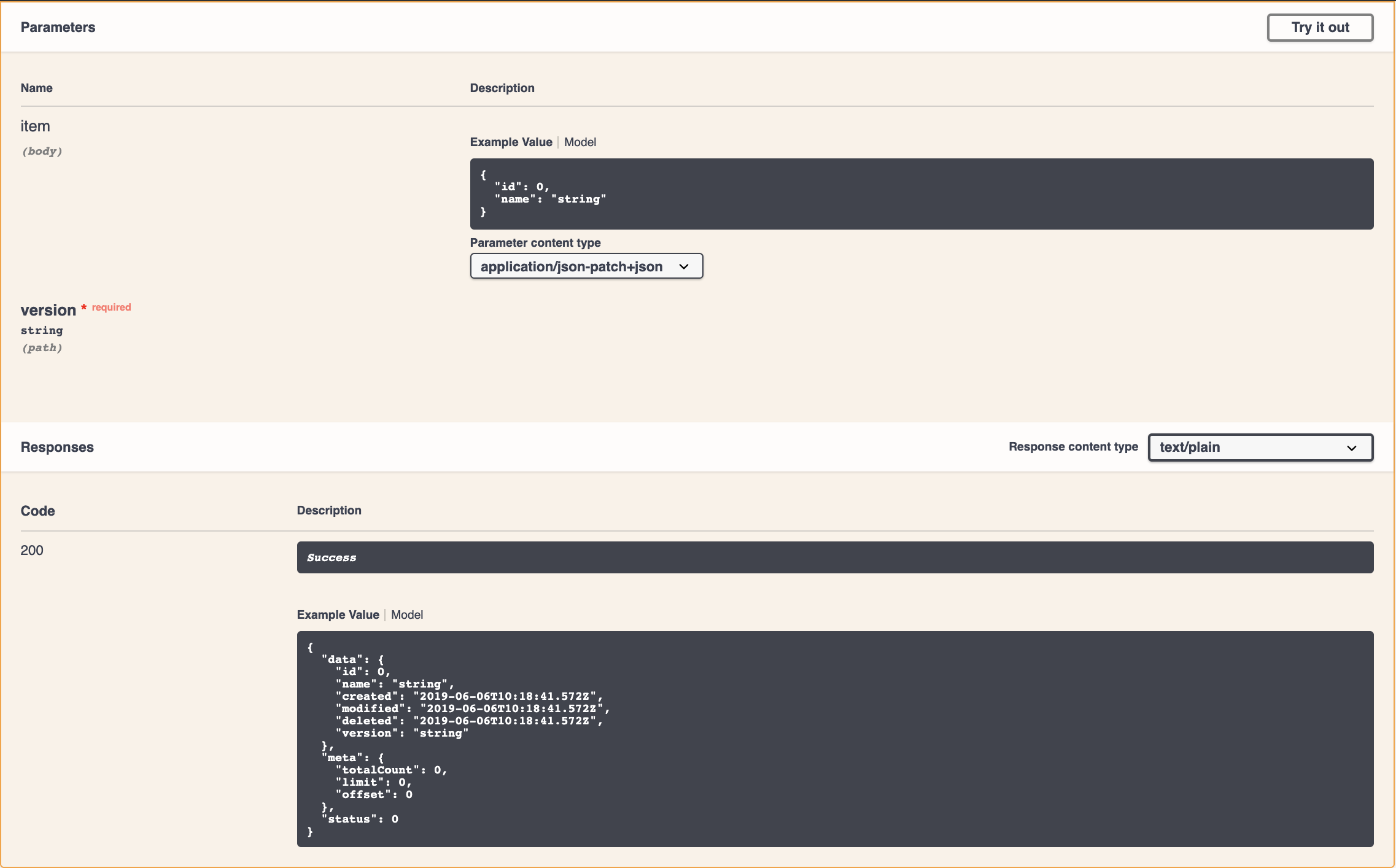


Рис. 4.2 Пример запроса и ответа

## Заключение

В рамках данного курсового проекта было разработано программный интерфейс приложения для портала Студенческой Весны.

Разработанный продукт представляет удобный API с простым интерфейсом, который позволит продолжить разработку портала.

В планах по доработке:

* интеграционные тесты для всех контроллеров API
* завершение работ по автогенерации контроллеров API
* нагрузочное тестирование каждого контроллера и сервиса в целом
* разработка клиентской стороны пользовательского интерфейса

## Список использованной литературы

[1] Bill Karwin: SQL Antipatterns Tables and Queries That Don’t Work.

[2] Fluent migrations framework for .NET, <https://fluentmigrator.github.io>

[3] Jeffrey Richter: CLR via C#

[4] Jasig CAS: <https://www.apereo.org/>

[5] Redmine: <http://www.redmine.org/>

[6] MSSQL: <https://docs.microsoft.com/ru-ru/sql/?view=sql-server-2017>

[7] Net Core: <https://docs.microsoft.com/ru-ru/dotnet/>

[8] Docker: <https://docs.docker.com/>

[9] Dapper: <https://dapper-tutorial.net/>

[10] Swagger: <https://swagger.io/docs/>