МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

КУРСОВАЯ РАБОТА

по дисциплине «Интерфейсы ВС»

Тема: “ Создание RESTful веб-сервиса, регистрирующего нарушение техники безопасности”

Автор работы _____ Иванкин К.С.
(подпись) ВТ-42

Руководитель проекта _____ ст. пр.
(подпись) Торопчин Д.А.

Оценка _____

Белгород
2021 г.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2
ВВЕДЕНИЕ 3
1 Анализ предметной области и постановка задачи 3
1.1 Постановка задачи 3
1.2 Модель данных 3
2 Теоретические сведения 9
Веб-сервис 9
2.1.1 Определение 9
2.1.2 Протокол SOAP 10
2.1.3 Протокол XML-RPC 11
2.1.4 REST-архитектура 11
2.1 Веб-сервер 12
2.3 База данных 14
3.1 Apache 17
3.2 IIS 17
3.3 Nginx 17
3.4 Node.js 18
3.5 ASP.NET Core 18
3.6 Kestrel 19
4 Проектирование БД и разработка серверной части веб-сервиса 20
4.1 Программные средства 20
4.2 Проектирование архитектуры БД 20
4.3 Архитектура проекта 21
4.3.1 Слой доступа к данным 21
4.3.2 Слой данных 23
4.3.3 Слой бизнес-логики 24
4.3.4 Слой представления 27
4.4 Микросервис авторизации 29
4.5 Механизм аутентификации и авторизации 30
ЗАКЛЮЧЕНИЕ 33

ВВЕДЕНИЕ

В настоящее время интернет-технологии и цифровые устройства стремительно развиваются. В связи с этим, благодаря автоматизации большинства бизнес-процессов, стало возможным избавиться от рутинной работы, кроме того автоматизация даёт возможность собирать статистику и анализировать различные данные.

Целью курсовой работы является сокращение количества случаев нарушения техники безопасности и повышение эффективности работы на предприятии путем автоматизации процесса подачи и обработки жалоб на нарушение требований техники безопасности в организации.

1 Анализ предметной области и постановка задачи

1.1 Постановка задачи

требуется:

- спроектировать архитектуру базы данных и разработать сервер, который способен работать как с мобильными клиентами различных операционных систем, так и с веб-сайтом;
- разработать мобильное приложение для операционной системы Android.

Актуальность проекта заключается в том, что готового решения для перечисленных задач на данный момент не существует.

1.2 Модель данных

В проекте будут использованы следующие сущности модели данных:

- User - представляет пользователя;
- Auth - используется для аутентификации и авторизации;
- Attachment - вложение, прикрепленное к заявке (фото или видео);
- Application - заявка о нарушении ТБ.

Подробная модель данных, содержащая поля сущностей, их типы данных и описание представлена в таблице 1.

Таблица 1 - Модель данных

Сущность	Поля		
	Название	Тип данных	Описание
User	Id	Integer	Идентификатор
	RoleId	Integer	Идентификатор роли пользователя
	FirstName	String	Имя
	LastName	String	Фамилия
	PersonnelNumber	String	Табельный номер
	Email	String	Адрес электронной почты
	Password	String	Пароль
Role	Id	Integer	Идентификатор
	Name	String	Название роли
Application	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	ReplyId	Integer	Идентификатор ответа
	Theme	String	Тема заявки
	Message	String	Текст заявки
	Status	Integer	Статус заявки: 0 - не рассмотрена; 1 - отвечена;
	CreatedAt	Date	Дата создания
	ModifiedAt	Date	Дата изменения

Сущность	Поля		
	Название	Тип данных	Описание
Attachment	Id	Integer	Идентификатор
	ApplicationId	Integer	Идентификатор заявки
	Type	Integer	Тип: 0 - фото; 1 - видео.
	Url	String	Адрес доступа к вложению
	ThumbnailUrl	String	Адрес доступа к миниатюре
Reply	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	Message	String	Текст ответа
	Danger	Int	Уровень опасности: 0 - легкий; 1 - средний; 2 - тяжелый;
	CreatedAt	Date	Дата создания
	ModifiedAt	Date	Дата изменения
Notification	Id	Integer	Идентификатор
	UserId	Integer	Идентификатор пользователя
	Application^	Integer	Идентификатор заявки
	Type	Integer	Тип: 0 - создано; 1 - обновлено;

В графическом формате модель данных можно посмотреть на рисунке 1.

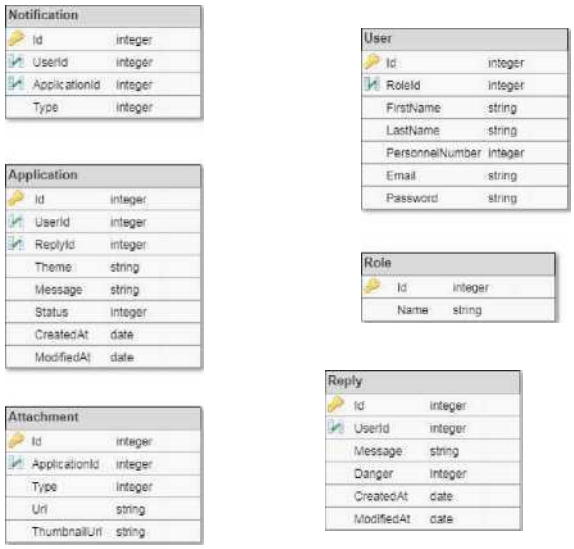

Рисунок 1 - Модель данных

Данные, которые будут передаваться в теле HTTP запроса (body) при обращении к серверу отображены в таблице 2.

Окончание таблицы 3

Таблица 2 - Виды HTTP запросов к серверу

Наименование	Тело запроса	
	Название	Тип данных
SignInRequest	last_name	String
	personnel_number	String
	password (необязательно)	String
	client_id	String
RefreshTokenRequest	refresh_token	String
	client_id	String
UpdatePasswordRequest	password	String
RecoverPasswordRequest	last_name	String
	personnel_number	String
CreateApplicationRequest	theme	String
	message	String
	files	Массив Multipart
UpdateApplicationRequest	id	Integer
	theme	String
	message	String
	files	Массив Multipart
CreateReplyRequest	application_id	Integer
	message	String
	danger	Integer
UpdateReplyRequest	reply_id	Integer
	message	String

Данные, которые будут передаваться в теле HTTP ответа (body) при обращении к серверу отображены в таблице 3.

Таблица 3 - Виды HTTP ответов сервера

Наименование	Тело ответа	
	Название	Тип данных
AuthResponse	access_token	String
	refresh_token	String
	role	Integer

Наименование	Тело ответа	
	Название	Тип данных
UserResponse	id	Integer
	first_name	String
	last_name	String
	email	String
	role	Integer
ApplicationResponse	id	Integer
	user_id	Integer
	reply_id	Integer
	theme	String
	message	String
	status	String
	created_at	Date
	modified_at	Date
AttachmentResponse	id	Integer
	application_id	Integer
	type	Integer
	url	String
	thumbnail_url	String
ReplyResponse	id	Integer
	user_id	Integer
	message	String
	danger	Integer
	created_at	Integer
	modified_at	Integer
NotificationResponse	id	Integer
	application_id	Integer
	type	Integer
StatisticsResponse	applications_number	Integer
	replies_number	Integer
	danger_average	Integer
	date	Date

В таблице 4 представлены возможные запросы к серверу с указанием пути, типа запроса, его краткого описания, необходимости авторизации и видов тел

HTTP запроса и ответа (для подробной информации о них см. предыдущие таблицы).

Окончание таблицы 3

Таблица 4 - Возможные запросы к серверу

Путь	Тип запроса	Авторизация	GET параметры		Тело HTTP запроса	Тело HTTP ответа	Описание запроса
			Параметр	Тип данных			
api/auth/signin	POST	-	-		SignInReques t	AuthResponse	Аутентифицировать пользователя по персональному номеру
api/auth/refreshtoken	POST	-	-		RefreshToken Request	AuthResponse	Подлить токен авторизации
api/auth/password	PUT	+	-		UpdatePassw ordRequest	-	Сменить или установить пароль
api/auth/recoverpassword	POST	-	-		RecoverPass wordRequest	-	Восстановить пароль через электронную почту
api/users/{id}	GET	+	id	Integer	-	UserResponse	Получить данные о пользователе
api/applications	GET	+	page	Integer	-	Массив ApplicationRes ponse	Получить список заявок о нарушении ТБ
			per_page	Integer			
			query	String			
api/applications/{id}	GET	+	id	Integer	-	ApplicationRes ponse	Получить данные заявки о конкретной заявке
api/applications	POST (multipart/form-data)	+	-		CreateApplica tionRequest	ApplicationRes ponse	Создать новую заявку
api/applications	PUT (multipart/form-data)	+	-		UpdateApplic ationRequest	ApplicationRes ponse	Редактировать заявку
api/applications/{id}/attachments	GET	+	id	Integer	-	Массив AttachmentRes ponse	Получить список вложений

api/attachments/{id}	GET	+	id	Integer	-	AttachmentResponse	Получить данные о вложении
api/attachments/{id}	DELETE	+	id	Integer	-	-	Удалить вложение
api/replies/{id}	GET	+	id	Integer	-	ReplyResponse	Получить данные об ответе на заявку
api/replies	POST	+	-		CreateReplyRequest	ReplyResponse	Добавить ответ на заявку
api/replies	PUT	+	-		UpdateReplyRequest	ReplyResponse	Редактировать ответ на заявку
api/notifications	GET	+	-		-	NotificationResponse	Получить список уведомлений
api/statistics/month	GET	+	-		-	Массив StatisticsReponse	Получить статистику за месяц
api/statistics/year	GET	+	-		-	Массив StatisticsResponse	Получить статистику за год

Коды возможных HTTP ответов:

- 200 - возвращается при каждом успешном запросе;
- 400 - возвращается при неверном запросе, сформированным клиентом;
- 401 - возвращается, если пользователь не авторизован;
- 403 - возвращается, если уровень доступа пользователя недостаточен;

2 Теоретические сведения

Веб-сервис

2.1.1 Определение

Веб-сервис - это сетевая технология, обеспечивающая межпрограммное взаимодействие на основе веб-стандартов. [1]

Веб-сервисы независимы от языка и платформы реализации. Они способны взаимодействовать друг с другом, а также с другими приложениями, обмениваясь сообщениями с помощью сетевых протоколов.

Стандартным протоколом для обмена информацией является SOAP, но также существуют и другие. Они будут рассмотрены в следующих пунктах.

Для описания интерфейсов веб-сервиса используется специальный язык описания - WSDL (Web Services Description Language).

Также веб-сервис содержит универсальный интерфейс распознавания, описания и интеграции - UDDI (Universal Discovery, Description and Integration), который используется для поиска существующих веб-сервисов и обеспечения доступа к ним.

Традиционная концепция веб-сервиса представлена на рисунке 2.

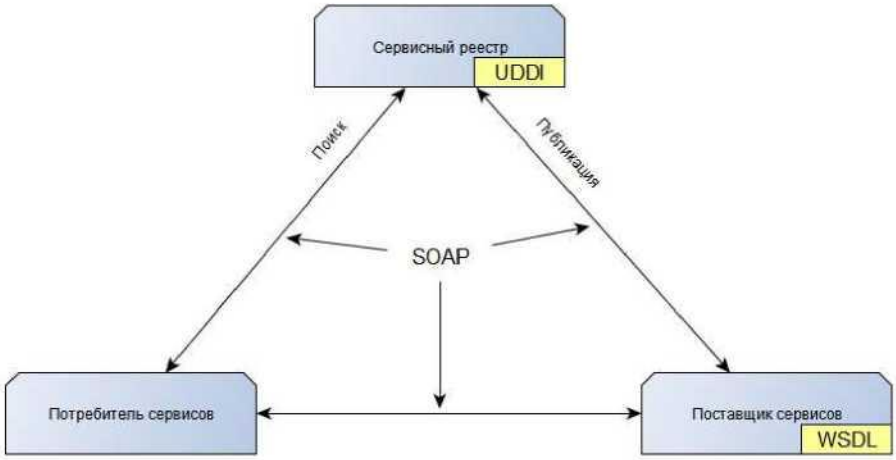

Рисунок 2 - Концепция веб-сервиса

2.1.2 Протокол SOAP

SOAP (Simple Object Access Protocol) —протокол доступа к объектам (компонентам распределенной вычислительной системы), предназначенный для обмена структурированной информацией.

Проще говоря, SOAP - это протокол для взаимодействия с веб-сервисами, созданный на основе XML - расширяемом языке разметки. Можно сказать, что SOAP - это соглашение о строго форматированном XML-документе. Согласно ему, XML-документ должен содержать определенные элементы и пространства имён, а также специальные теги.

Сообщения SOAP оформляются в виде особой структуры (рисунок 3), которая называется конверт (Envelop). Она включает в себя заголовок (Header) - необязательный элемент и тело (Body).

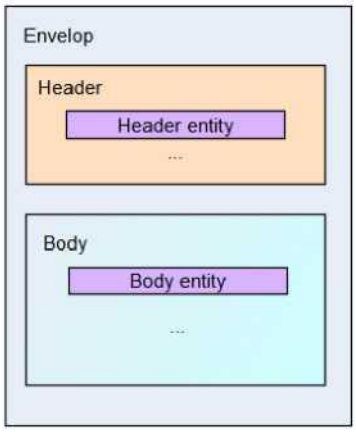

Рисунок 3 - Структура SOAP-сообщения

2.1.3 Протокол XML-RPC

XML-RPC (Extensible Markup Language Remote Procedure Call) - протокол для вызова удаленных процедур. Он очень прост и эффективен в использовании, однако, в отличии от SOAP, не предназначен для решения глобальных задач. Но несмотря на это, данный протокол используется достаточно широко во многих веб-разработках.

На рисунке 4 представлена концепция веб-сервера, использующего протокол XML-RPC.

Рисунок 4 - Концепция XML-RPC

2.1.4 REST-архитектура

В отличии от SOAP и XML-RPC, REST (Representational State Transfer) нельзя назвать протоколом. Это стиль архитектуры взаимодействия компонентов в распределенной сети.

REST базируется на HTTP протоколе, и следовательно, может использовать все существующие наработки на базе HTTP.

Веб-сервис можно назвать RESTful сервисом, если он не нарушает обязательных требований к REST-архитектуре:

- архитектура сервиса должна быть приведена к модели клиент-сервер;

- на сервере не может храниться никакая информация о состоянии клиента, в период между его запросами;

- клиенты могут кэшировать (сохранять) ответы от сервера. Поэтому ответы сервера должны обозначаться как кэшируемые или некэшируемые, для предотвращения получения устаревших данных;

- интерфейсы REST-сервисов должны быть единообразны [2].

На рисунке 5 представлена модель RESTful сервиса.

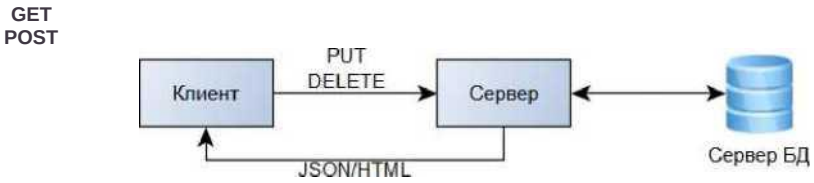

Рисунок 5 - RESTful сервис

2.1 Веб-сервер

Веб-сервер - это сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы [3].

Основные функции веб-сервера:

- управление соединением;

- прием и обработка запроса;

- разделение доступа к нескольким обслуживаемым наборам ресурсов;

- отдача статического содержимого;

- взаимодействие с приложениями для получения и дальнейшей отдачи клиенту динамического содержимого [4].

При запуске веб-сервера ему присваивается номер порта - сетевой идентификатор. При этом говорят, что сервер слушает порт. На этот порт клиент посылает HTTP-запросы.

Алгоритм работы веб-сервера заключается в следующем: на сервер поступает запрос от клиента. Далее запрос анализируется: происходит аутентификация,

обрабатывается адрес, проверяются виртуальные хосты. Затем происходит обработка запроса: если запрашивается статический контент, он считывается из файловой системы; если запрашивается динамический контент, то сервер обращается к программному коду и генерирует новые данные; сервер также может передавать запрос для обработки другому серверу. После этого происходит формирование HTTP-ответа и отправка его клиенту (см. рисунок 6).

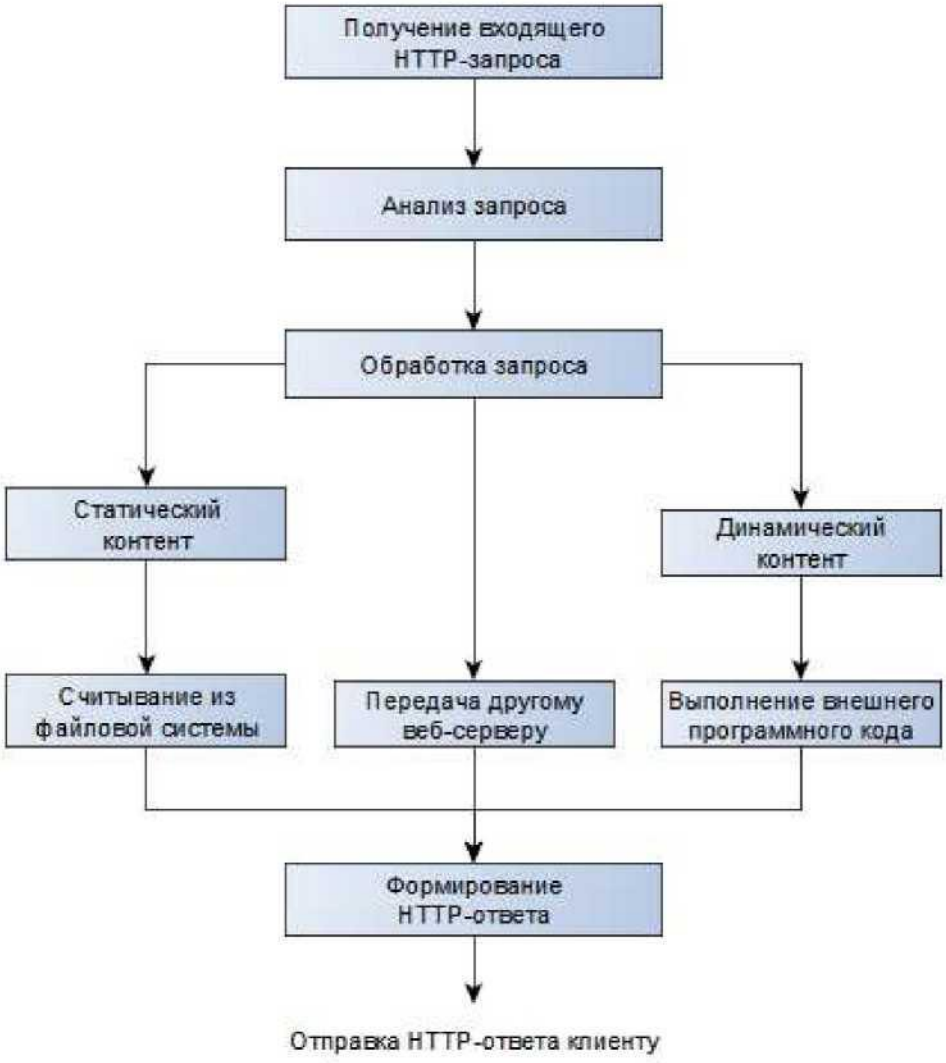

Рисунок 6 - Алгоритм работы веб-сервера

2.3 База данных

База данных - совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений [5].

Базы данных можно разделить на две большие группы: реляционные базы данных и нереляционные базы данных. Основные различия между ними заключаются в том, как базы данных хранят информацию, как они спроектированы и какие типы данных поддерживают.

Реляционные базы данных хранят строго структурированные данные. Они

основаны на реляционной модели представления данных и поддерживают язык структурированных запросов (SQL).

Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида.

Такие базы данных соответствуют требованиям ACID, это значит, что они атомарные, непротиворечивы, изолированы и долговечны. Из этого следует основное преимущество реляционных баз данных: они надежны - вероятность неожиданного поведения системы достаточно мала; а также данные требования обеспечивают целостность базы данных.

Однако существенным недостатком является медленный доступ к данным и ограниченная масштабируемость базы данных.

Для решения проблемы быстродействия и масштабируемости данных, были придуманы новые подходы к построению баз данных. Эти подходы обозначают как нереляционные базы данных.

Различают четыре основных типа нереляционных хранилищ:

- хранилище вида ключ-значение - это простейший вид базы данных, где каждому значению соответствует уникальный ключ. Такая простота обеспечивает неограниченную масштабируемость базы. Однако из-за несвязности отдельных записей в базе, становится невозможен быстрый анализ имеющейся информации в базе данных. База данных такого вида нашла свое применение в качестве кэшей для объектов;

- документно-ориентированные базы данных хранят информацию в виде иерархической структуры данных. Это позволяет осуществлять быстрый поиск даже при достаточно сложной общей структуре хранилища. Такие базы удобно применять, когда требуется упорядоченное хранение информации, но нет множества связей между данными;

- графовые базы данных используют сетевую модель представления данных. Чаще всего их используют, когда данные имеют естественную графовую структуру.

- bigtable-подобные базы данных содержат данные, упорядоченные в виде разряженной матрицы, строки и столбцы которой используются в качестве ключей.

Такой тип базы данных находит применение в задачах, предполагающих огромные объемы данных.

3 Обзор актуальных технологий для разработки серверной части веб-сервиса

3.1 Apache

Apache - веб-сервер с открытым исходным кодом. Это значит, что пользователь может редактировать и адаптировать платформу под свои нужды. Apache работает на всех популярных операционных системах. Основным достоинством данного веб-сервера является его гибкость. Он включает в себя обширный функционал, при этом его можно расширить, добавив дополнительные модули.

Данный веб-сервер надежный и достаточно мощный веб-сервер, однако имеет и недостатки. Функционала настолько много, что возможно, он даже избыточен. В основном пользователь задействует всего лишь 10% от этих возможностей. Также, с точки зрения архитектуры, Apache работает по модели «процессов». Это означает, что для каждого соединения Apache выделяет отдельный поток данных, что вызывает большую нагрузку [6].

3.2 IIS

IIS (Internet Information Services) - веб-сервер от компании Microsoft, и поэтому, данная платформа работает только с операционными системами Windows. Но несмотря на это, IIS является вторым по полярности на рынке, после Apache. Он обладает достаточно широким функционалом, надёжен, и безопасен, так как его поддержкой официально занимается Microsoft. Однако он уступает в производительности своему конкуренту.

3.3 Nginx

Nginx - достаточно популярный веб-сервер, который часто используют в качестве обратного прокси-сервера. Он способен обрабатывать большое количество соединений, при этом расходуя минимум системных ресурсов. Nginx 22

можно назвать одним из самых производительных веб-серверов, он лучше всех отдаёт статические данные среди популярных серверов. Однако он не имеет возможности самостоятельно обрабатывать запросы к динамическому контенту.

3.4 Node.js

Node.js - платформа для разработки распределенных сетевых приложений, таких как веб-сервер. Данная платформа предназначена для использования на всех типах операционных систем; она достаточно проста, доступна и стабильна.

Node.js основан на движке V8, а он является одним из самых производительных для JavaScript на данный момент. За счёт использования данного движка, код выполняется намного быстрее. Кроме того, еще одно преимущество V8 - это эффективное управление памятью.

Также для Node.js существует очень много библиотек и инструментов, которые активно развиваются. Однако это можно отнести и к недостаткам - так как нет каких-либо основных библиотек или инструментов - у каждого есть множество альтернатив, свои плюсы и минусы, которые не всегда можно понять из документации. Как итог - не просто сделать выбор в сторону той или иной библиотеки, и не всегда этот выбор оказывается правильным.

Node.js - относительная молодая платформа, поэтому для многих стандартных задач нет какого-то готового и законченного решения.

3.5 ASP.NET Core

Платформа ASP.NET Core - разработана корпорацией Microsoft и предназначена для разработки различных веб-приложений.

Основным преимуществом данной технологии является её кроссплатформенность. Так как она основана на .NET Core, то приложение, разработанное на этой платформе, может быть развёрнуто не только на операционных системах семейства Windows, но и на всех других популярных операционных системах.

ASP.NET Core состоит из нескольких компонентов, благодаря чему можно легко

расширять базовый функционал фреймворка.

Главный недостаток данной платформы состоит в том, что она была выпущена не так давно. Она активно развивается, и дополняется в настоящее время. Поэтому многие привычные фреймворки и пакеты еще не умеют работать с ASP.NET Core.

3.6 Kestrel

Kestrel - это кроссплатформенный веб-сервер который обычно используется с приложениями на основе ASP.Net core. Его можно использовать как отдельный веб-сервер, так и с обратным прокси-сервером, таким как IIS, Apache или Nginx. Использование обратного прокси-сервера не обязательно, но всё же желательно. Так как он получает HTTP запросы из сети и направляет их в Kestrel после первоначальной обработки. Это позволяет облегчить распределен ие нагрузки, обеспечить безопасность и поддержку SSL.

4 Проектирование БД и разработка серверной части веб-сервиса

4.1 Программные средства

Для разработки веб-сервера было решено использовать платформу ASP.NET Core, сервер Kestrel, и язык программирования C#. Эта платформа появилась относительно недавно, однако уже успела стать достаточно популярной в IT-сообществах. Одним из главных достоинств платформы является её кроссплатформенность, а это значит, что приложение может быть развёрнуто на всех популярных операционных системах. Также преимуществом ASP.NET является встроенный IOC-контейнер.

В качестве IDE была выбрана Visual Studio 2017 Community от Microsoft. Так как на сегодняшний день она предлагает самый широкий спектр функциональных возможностей для разработки на языке C#.

В качестве СУБД была выбрана MS SQL. Она хорошо работает в связке с выбранной платформой и позволяет полностью удовлетворить требования к построению структуры базы данных для приложения.

4.2 Проектирование архитектуры БД

Для создания структуры базы данных, в приложении используется подход code-first. Это значит, что таблицы базы данных генерируются на основе C# кода. Сущности реализованы в слое данных.

База данных приложения содержит следующие сущности: Заявка (Application), Пользователь (User), Ответ (Reply), Приложение (Attachment), Роль (Role), Уведомление (Notification). Схема данных представлена в техническом задании в п. 1.3.4 на рисунке 1.

В таблице Users хранятся данные пользователя. Пользователь может создать или редактировать заявку, и в зависимости от роли, которые содержатся в таблице Roles, он может создать или редактировать ответ. Созданные заявки хранятся в таблице Applications, созданные ответы - в таблице Replies. Кроме 25

того, пользователь может прикрепить приложение к заявке. Приложения хранятся в таблице Attachments. Таблица Notifications содержит информацию о созданных, не рассмотренных или обновленных заявках для того чтобы своевременно уведомлять пользователя о изменениях.

4.3 Архитектура проекта

Структура приложения состоит из четырех основных слоев: слой доступа к данным, слой данных, слой бизнес-логики и слой представления. Авторизация в приложении реализована отдельным микросервисом. Для обеспечения слабой связанности в проекте использован паттерн Dependency Injection. Слои и связи между ними показаны на рисунке 7.

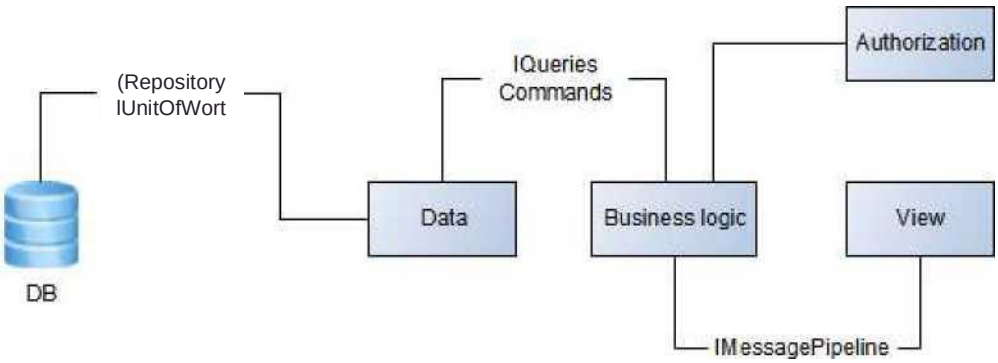

Рисунок 7 - Структура приложения

4.3.1 Слой доступа к данным

В этом слое производится подключение к базе данных, а также реализованы паттерны Repository и Unit of work.

В данном слое также хранятся миграции, которые служат для обновления структуры базы данных, не теряя содержащейся в ней информации.

В качестве базы данных используется MSSQL, а для удобной работы с ней используется фреймворк Entity Framework Core (EF). Таким образом мы не работаем напрямую с SQL запросами в базу данных, а используем для этого обёртку, которую предоставляет нам EF. Таким образом значительно упрощается работа с базой из

приложения, например, на рисунке 8 представлен SQL запрос для получения одной заявки из базы данных, и как это реализовано с помощью фреймоворка.

```
SELECT * FROM [dbo] [Apps] WHERE Id = 2

van application = await _repository.FirstOrDefaultAsyncfapp => app.Id == id);
```

Рисунок 8 - Запрос в базу с помощью Entity Framework Core

Однако приложение не завязано на одной базе данных, и по необходимости её можно заменить на любую другую. Для обеспечения такой взаимозаменяемости, в приложении реализованы паттерны Repository и Unit of work.

Применяя паттерн Repository, мы можем легко изменить источник данных, не меняя ничего в остальных слоях приложения. Таким образом слой бизнес логики и слой данных ничего не знают о том, откуда берутся данные, и используют для работы виртуальный репозиторий.

Для реализации паттерна был создан интерфейс IRepository, а также его реализация Repository. Интерфейс IRepository содержит определение сигнатуры общих для всех репозиториев функций, таких как добавить в репозиторий и удалить. Кроме того, в нем определена сигнатура метода Query для получения экземпляра типа IQueryble. Repository содержит имплементацию этих методов.

Репозиторий создаётся отдельно для каждой сущности, следовательно, в крупном проекте возможно достаточно большое количество репозиториев, для удобства работы с ними применяется паттерн Unit of work.

Паттерн Unit of work дословно переводится как единица работы, он содержит в себе экземпляры репозиториев для каждой сущности. Это очень удобно, ведь нам не нужно подключать к проекту множество репозиториев, достаточно подключить один Unit of work. Также данный шаблон позволяет сначала произвести все нужные действия с репозиториями, а затем сохранить изменения в базу. За счёт того, что нет необходимости делать запрос в базу после каждого небольшого изменения, существенно снижается нагрузка на базу данных.

Реализация Unit of work в приложении состоит из интерфейса IUnitOfWork и его

имплементации UnitOfWork. Интерфейс содержит определения сигнатуры репозиториев для заявки, ответа, пользователя, приложения и роли, методов для миграции базы данных и сохранения изменений, а также их асинхронные аналоги. В UnitOfWork реализуются данные методы и создаются репозитории, используя паттерн Singletone. Этот шаблон обеспечивает создание репозитория один раз за все время жизни приложения.

Для того чтобы использовать асинхронные методы при работе с данными, были созданы интерфейс фабрики IAsyncQuerybleFactory и его реализация, для создания экземпляра типа IAsyncQueryble. А также интерфейс IAsyncQueryble, который содержит сигнатуру асинхронных методов, с имплементацией AsyncQueryble, которая содержит реализацию этих методов.

4.3.2 Слой данных

В слое данных описаны сущности в виде классов, которые используются для генерации структуры базы данных.

Так как в проекте используется подход CQRS, в слое данных также хранятся команды и интерфейсы запросов для каждой сущности и DTO-модели.

Команды представляют собой набор данных, которые передаются серверу для совершения какого-либо действия. Пример команды для создания заявки представлен на рисунке 9.

```
public class CreateAppCommand {
    [Required]
    [ 1axLength{64)]|
     public string Theme ■[ get; set; }

    [Required]
    [ 1axLei>gth(4№6) ]
    public string Message { get; set; } }
```

Рисунок 9 - Пример команды

В квадратных скобках над определением свойства перечисляются атрибуты для валидации данных, например, Required - обозначает обязательное свойство, а MaxLength - максимально возможную длину строки.

Интерфейсы запросов содержат определение сигнатуры методов для получения данных из репозитория.

DTO-модели (Data Transfer Object) - состоят из набора свойств, которые используются для передачи данных между приложениями (в нашем случае между сервером и клиентом).

4.3.3 Слой бизнес-логики

В данном слое хранятся обработчики команд. Они используют данные из команды и содержат в себе логику выполнения какого-либо действия.

В нашем приложении реализован подход CQRS (Command Query Responsibility Segregation). Он предполагает разделение операций считывания данных и их обновления. Это означает, что для запросов и обновлений используются разные модели данных.

Модель запросов для чтения данных и модель для обновления данных чаще всего обращаются к разным физическим хранилищам, это обеспечивает повышение производительности, масштабируемости и безопасности (рисунок 10). Однако данные модели могут обращаться и к одному физическому хранилищу.

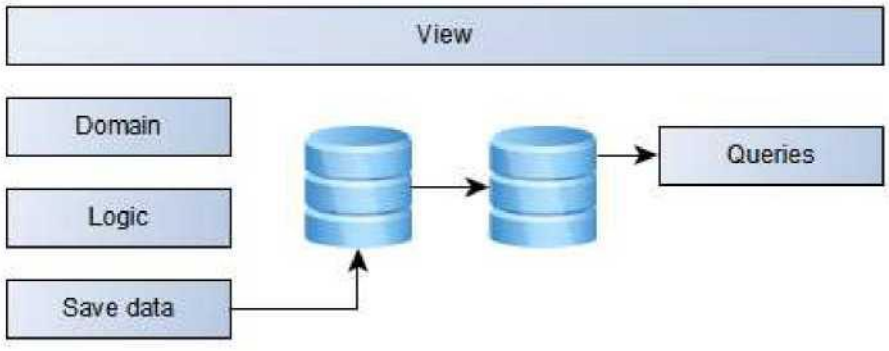

Рисунок 10 - Обращение к разным физическим хранилищам

Обычно в качестве хранилища для чтения данных используют копию хранилища для записи с доступом только на чтение.

Таким образом, основная идея принципа CQRS состоит в том, что для изменения состояния системы используется класс Command, а для выборки данных класс Query.

Недостатком такого подхода является невозможность автоматической генерации кода на основе шаблонов.

В нашем проекте при реализации данного подхода была использована сторонняя библиотека SaritasaTools. Выполнение команды происходит с помощью нескольких промежуточных обработчиков - middlewares.

Конвейер обработки команды по умолчанию состоит из следующих обработчиков:

- CommandHandlerLocatorMiddleware - находит расположение классов-обработчиков;

- CommandHandlerResolverMiddleware - сопоставляет команду с её обработчиком;

- CommandHandlerExecutorMiddleware - выполняет найденный обработчик;

Также был добавлен дополнительный промежуточный обработчик - CommandValidationMiddleware, который служит для валидации данных команды (схема обработки команды представлена на рисунке 11).

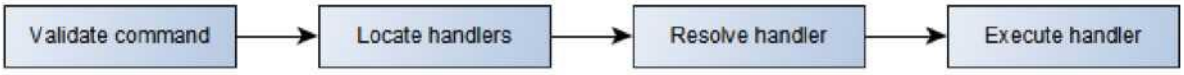

Рисунок 11 - Конвейер обработки команды

Пример конфигурации контейнера, содержащего промежуточные обработчики представлен на рисунке 12.

```
var pipelinecontainer = new DefaultMessagePipelineContainer();

pipelineContainer.AddCommandPipelinef)
      .AddHiddleware(new CommandValidation'Jiiddle.-.dre())
.AddStandardMiddlewares(options => { options.InternalResolver.UseInternalObjectResolver = false;
options.UseExceptionDispatchInfo = true;
```

```
pipelineContainer.AddQueryPipeline()
    .AddStandardMiddlewares(options => { options.InternalResolver.UseInternalObJectResolver = false;
    options.UseExceptionDispatchInfo = true;
```

Рисунок 12 - Конфигурация pipeline контейнера

В нашем приложении реализовано 2 класса-обработчика, которые содержат методы для обработки команд: ApplicationHandlers и ReplyHandlers. А также 4 класса, содержащие обработку запросов: ApplicationQueries, RepliesQueries, NotificationQueries и StatisticQueries. В качестве примера обработки запроса на рисунке 13 представлен метод, который собирает и отдаёт статистику. Также на рисунке 14 представлен пример обработки команды создания заявки.

```
puslic async Fas < .sfiStatistic Response» Create5tatistic(int ? month, int year) <
    List<StatistscResponse> response = new List<StatisticResponse>();

    if (month 1= null)
    <
        war apps = await _factory.CreateAsyncQueryable(_uow.Applications.Query(app => ape,Reply} .Hhere(app =>
                app.CreateDate.Year == year &S app.CreateDate.Mcntti == month) .GroupSy(app =>
                app.CreateDate))
            .ToDictionaryAsync(group => group,. Key .Date, group => group.ToList());
        fbreach (var date in apps.Keys)
        {
            response.Add(new Statist i cResponse
            {
                A.pplicationsNumber = apps[date] .Count,
                RepliesNumber = apps[date].Count(a => a.Reply != null), Date = date
            });
        }
    }
    else
    {
        var apps = await _factory.CreateAsyncQueryable(_uow,Applications.Query(app => app.Reply) .Where(app =>
                app.CreateDate.Vear == year) .GroupSy(app = > app.CreeteDate))
            .ToDictionaryAsync(group => group.Key.Month, group => group.ToListf));
        foreact (var iron in apps.Keys)
        <
            response.Add(new StaciscicResoonse {
                App lie at ionsNum.be r = appsFmon].Count,
                RepliesNumber = apps[mon].Count(a => a.Reply .= null), Date = new Dateline(year, гаол, 1)
            });
        }
    }

    return response;
```

Рисунок 13 - Метод для создания статистики

```
public async Task HandleCreateApplicationCommandl(CreateAppComriand command) {
    var app = _mapper.HapCCreateAppCommand, Application>(command);
    var user = await _pipelineService.Quiery<IllserQueries>()
```

```
            .WithAsync(q => q.GetUserAsync(conmand.Jserld));
        app.User = user;

        _uow.Applications.Add(ap p);
        _uow.Not ificat ions.Add(new Not ification {
            User = user,
            Application = app,
            Type = Type.Created
        });
        await _uow.CcmmitAsync();
    }
```

Рисунок 14 - Метод обработки команды создания заявки

Также слой бизнес-логики включает в себя сборку Infrastructure, которая содержит middlewares, api-клиент, и другие вспомогательные классы (рисунок 12).

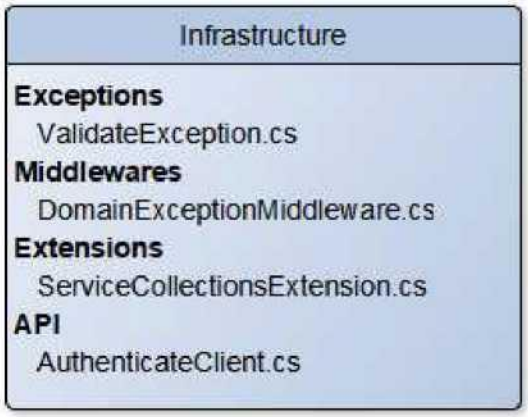

Рисунок 12 - Структура сборки Infrastructure

Exceptions содержит написанные нами исключения, которые вызываются при ошибке выполнения кода.

В Middlewares содержатся промежуточные обработчики, такие как DomainExceptionsMiddleware, который отлавливает и обрабатывает различные виды исключений.

Extensions содержит в себе расширения для классов. Например, ServiceCollectionExtensions - расширение для класса IServiceCollection, он предназначен для регистрации IOC-контейнера и зависимостей в нем.

API включает в себя классы предоставляющие доступ к сторонним сервисам, в нашем случае к сервису авторизации.

4.3.4 Слой представления

В слое представления находятся контроллеры, которые предназначены для

взаимодействия с клиентом. Клиент отправляет команду или запрос, в контроллере происходит обработка и обращение к слою бизнес-логики. По завершению запроса, контроллер отдаёт данные клиенту в формате JSON.

Полная схема взаимодействия слоёв при получении запроса показана на рисунке 13.

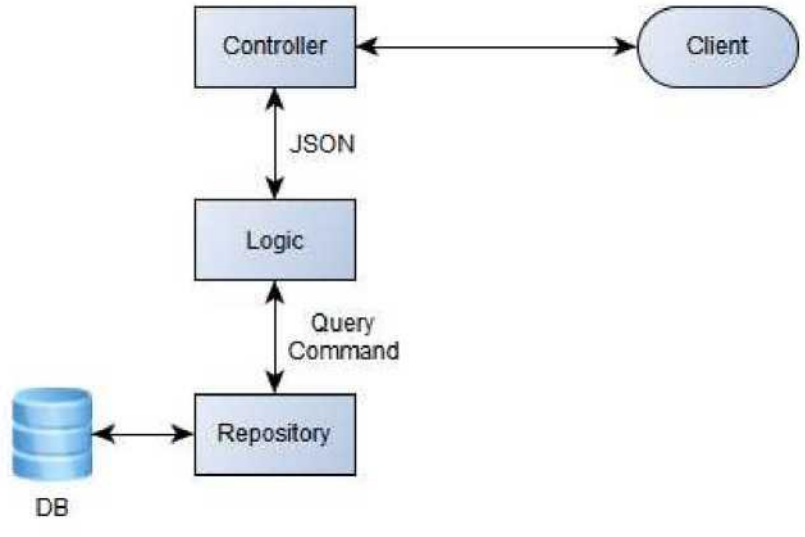

Рисунок 13 - Схема взаимодействия слоёв при запросе

В нашем приложении реализовано шесть контроллеров:

- UserController - для выполнения действий с пользователем, таких как получить пользователя или список пользователей.
- ApplicationsController - для выполнения различных действий с заявками, таких как создание заявки, её получение и изменение, получение списка заявок.
- RepliesController - для создания ответа и его модификации.
- AttachmentsController - для получения медиа-файлов и их удаления.
- StatisticsController - для получения статистики за месяц и за год.
- AuthController - для авторизации пользователя.

После поступления команды или запроса в метод контроллера она обрабатывается с помощью четырех промежуточных обработчиков, описанных в предыдущем пункте. Пример обработки запроса и команды представлен на рисунке 14.

```
[HttpPost]
[ProducesResponseType( 201, Type = typeof(ApplicationResponse))] [ProducesResponseType(4GG)]
public async Task<IActionResult> CreateAppAsync([FromBody]CreateAppCommand command)
```

```
{ ...
        await _messagePipeline.Hand1eCommandAsync(command);

        var userid = ((ClaimsIdentity) User.Identity).FindFirst("Id").Value;
        var response = await _messagePipeline.Query<IApplicationQueries>() .WithAsync(q => q
              .GetApplicationAsync(userId_, command.Id));

        return Created(UrlRouteUrlfnew { response. Id })j response);
} ' '
```

Рисунок 14 - Пример обработки запроса и команды

Также в данном слое, в классе Startup, происходит основное конфигурирование приложения. Регистрируются зависимости, подключается авторизация и промежуточные обработчики (middlewares).

4.4 Микросервис авторизации

Для реализации авторизации в приложении было принято решение использовать отдельный микросервис, так как данные для авторизации пользователя берутся из главной базы данных предприятия.

Основное преимущество создания микросервиса, это его повторное использование в других приложениях. Это значит, что когда компании потребуется создать еще одно внутреннее приложение, то ей не придется писать логику авторизации повторно. К минусам такого подхода можно отнести дополнительное время на обработку запроса за счёт его пересылки его на микросервис.

Микросервис содержит слой доступа к данным, где происходит подключение к базе данных, слой бизнес-логики, включающий в себя логику авторизации пользователя и слой представления, который содержит контроллер для получения и обработки запросов.

Для того, чтобы связать наше основное приложение и микросервис, использовался пакет программ NSwag. После некоторой конфигурации (см.

рисунок 15), NSwag генерирует интерфейс и его реализацию на основе кода из контроллера микросервиса. Полученный авто-сгенерированный код используется для доступа к коду микросервиса в основном приложении.

```
"codeGenerators" : {
  "swaggerToC3h.arpClient" : {
    "generateClientClasses" : true, "generateClientInterfaces": true,
    "generateDtoTypes":      false,      "injectHttpClient":      true,
    "dispcseHttpClient":   false,   "generateExceptionClasses":   true,
    "except!onClass": "SwaggerException", "wrapDtoExceptioив": true, "us
    e HttpCli enter eat ionMetliod" : false, "httpCl lent Type" :
    "System.Net.         Http         .         HttpClient"         ,
    "useHttpRequestMessageCreationMethod": false, "useBaseUrl": true,
    "generateSyncMethods": false, "clientClassAccessModifier": "public",
    "generateContractsQutput": false, "parameterDateTimeFormat": "s",
    "generateUpdateJson.3erializerSettingsMeth.od" : true "className":
    "Authenticationclient",      "namespace":      "Web.Infrastructure",
    "additionalNamespaceUsages": [ ] ,
```

Рисунок 15 - Часть конфигурации Swagger

4.5 Механизм аутентификации и авторизации

Аутентификация в приложении основана на использовании JWT токенов.

JWT (JSON Web Token) - содержит три блока, разделенных точками: заголовок (header), набор полей (payload) и сигнатуру. Первые два блока представлены в JSON-формате и дополнительно закодированы в формате base64. Сигнатура может генерироваться при помощи и симметричных алгоритмов шифрования, и асимметричных.

Схема аутентификации выглядит следующим образом: клиент посылает запрос аутентификации на сервер, сервер пересылает запрос микросервису. Если в микросервисе аутентификация прошла успешно, сервер генерирует токен и отправляет его клиенту. При всех последующих обращениях к серверу клиент 36

обязать присылать этот токен в заголовке запроса. Если же аутентификация не прошла, сервер отправляет клиенту статус ошибки и сообщение. Блок-схема аутентификации в основном приложении представлена на рисунке 16.

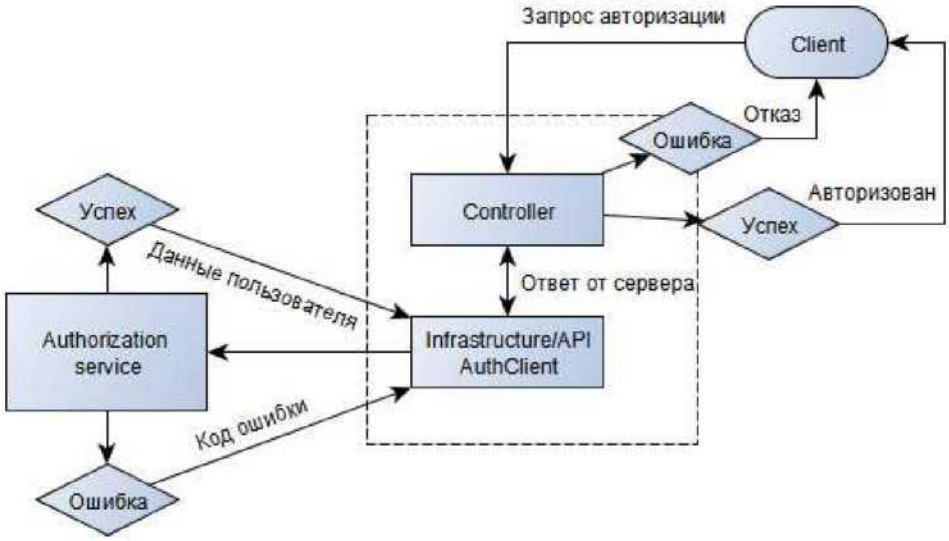

Рисунок 16 - Схема авторизации пользователя

При генерации JWT токена, в него также записываются данные о пользователе. Чтобы хранить данные о пользователе и иметь возможность управлять ими, используется библиотека AspNet.Identity. Для этого в слое представления конфигурируется политика безопасности, а также реализуются интерфейсы из этой библиотеки UserStore и RoleStore. Данные реализации содержат в себе методы получения и добавления пользователей и ролей.

После положительного ответа от микросервиса мы сообщаем нашему приложению, с помощью метода из библиотеки AspNet.Identity, что данный пользователь прошел стороннюю аутентификацию. После чего происходит последовательный вызов методов из UserStore и RoleStore и аутентифицируемому пользователю ставится в соответствие запись из нашей базы данных. Полная схема аутентификации с AspNet.Identity представлена на рисунке 17.

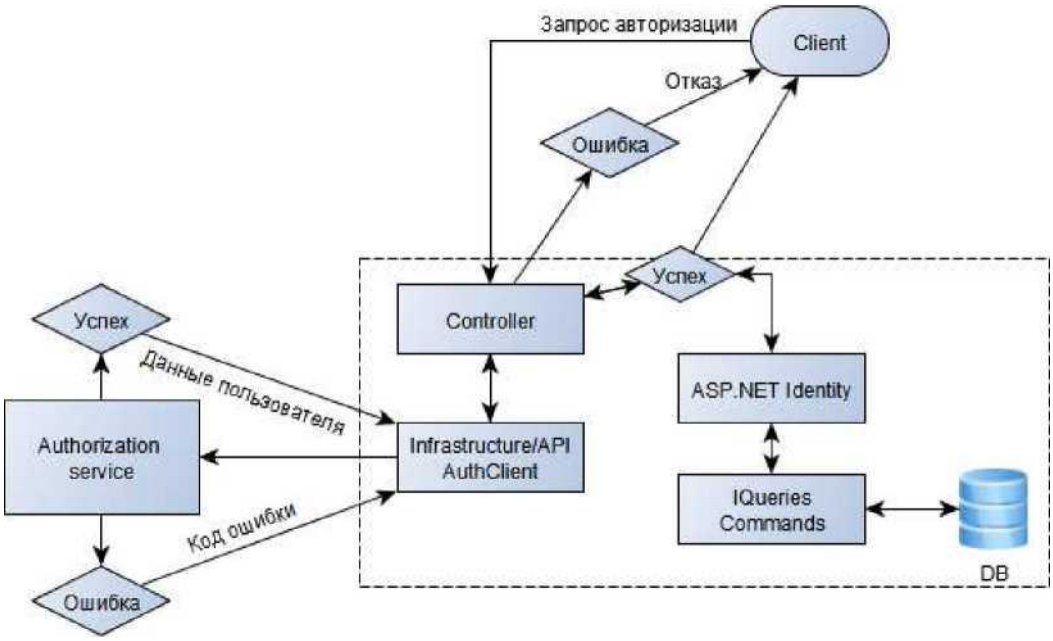

Рисунок 17 - Полная схема авторизации

Авторизация в приложении создана на основе ролей пользователей. Каждой роли соответствует некоторый набор разрешений. Например, User может просматривать только свои заявки и ответы на них, а Head может просматривать заявки от всех пользователей и добавлять ответы. При каждом запросе пользователя проверяется его роль, и, если у этой роли отсутствует разрешение на выполнение запрашиваемого действия, клиенту возвращается ошибка со статусом 403 - отказано в доступе.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были получены практические навыки по разработки веб-сервисов, познакомились с архитектурой Rest/

Был разработан веб-сервис автоматизирующий и упрощающий регистрацию нарушений техники безопасности

СПИСОК ИСПОЛЬЗОВАННЫХ источников

1 Паттерн CQRS [Электронный ресурс] : документация Microsoft - Режим доступа: https://docs.microsoft.com/ru-ru/azure/architecture/patterns/cqrs.

2 Общие архитектуры веб-приложений [Электронный ресурс] : документация Microsoft - Режим доступа: https://docs.microsoft.com/ru- ru/dotnet/standard/modern-web-apps-azure-architecture/common-web-application- architectures.

3 Библиотека Saritasa Tools [Электронный ресурс] : страница на github.com - Режим доступа: https: //github .com/Saritasa/SaritasaT ools.

4 Документация библиотеки Saritasa Tools [Электронный ресурс] : сайт saritasa-tools.readthedocs.io - Режим доступа: http://saritasa-tools.readthedocs.io/en/latest/.

5 ASP.NET Identity и системы аутентификации [Электронный ресурс] : сайт professorweb.ru - Режим доступа: https://professorweb.ru/my/ASP NET/identity/level 1/.

6 REST [Электронный ресурс] : Википедия - Режим доступа: https://ru.wikipedia.org/wiki/REST.

7 Что такое веб-сервер? [Электронный ресурс] : VPS Хостинг - Режим доступа: https: //vmlab .ru/Articles/What is web server.

8 Преимущества и недостатки нереляционных баз данных [Электронный ресурс] : Сайт компании VEESP - Режим доступа: https://veesp-.com/ru/blog/sql- or-nosql.

9 Apache или IIS сравнение и преимущества [Электронный ресурс] : ИТ База знаний - Режим доступа: http://wiki.merionet.ru/servemye-reshemya/3Zapache- ili-iis/.