­ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc26714267)

[1 Концепт приложения 6](#_Toc26714268)

[1.1 Стек технологий 6](#_Toc26714269)

[1.2 Причины использования стека 7](#_Toc26714270)

[1.3 Архитектура приложения 9](#_Toc26714271)

[1.4 Анализ Целевой аудитории 11](#_Toc26714272)

[2 Результаты разработки приложения 12](#_Toc26714273)

[2.1 Презентация проекта 12](#_Toc26714274)

[2.2 Описание кода 17](#_Toc26714275)

[2.3 Тестирование 18](#_Toc26714276)

[ЗАКЛЮЧЕНИЕ 20](#_Toc26714277)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 21](#_Toc26714278)

# ВВЕДЕНИЕ

Каждый студент рано или поздно сталкивается с задачей написания курсовой работы.

Курсовая работа или курсовой проект — это один из важных этапов обучения. Он заключается в самостоятельном научном исследовании определённой темы профильной дисциплины. Именно курсовая готовит студентов к написанию главной работы, которая служит итогом их обучения. Написание курсовых проектов развивает такие умения обучаемых, как выбор и чёткое формулирование темы исследования, сбор подходящего материала с использованием научной литературы и источников, сортировку и логическую систематизацию собранного материала, написание понятного текста с соблюдением необходимых общепринятых правил, строгих стандартов оформления. Согласно статистике, именно последнее вызывает у студентов наибольшую трудность.

Правильность оформления работы – это одно из важнейших условий успешного выполнения задания. Студент обязан использовать определенный размер, шрифт, разметку для своего текста.

В то же время, нормоконтролер, преподаватель, занимающийся проверкой работ учащихся на правильность оформления, сталкивается с нелегкой задачей — просмотреть огромное количество курсовых и дать каждому ответ о необходимости исправления ошибок оформления. Безусловно, проверка каждой работы занимает крайне большое количество времени.

Количество студентов за последние годы выросло во много раз, из-за этого времени на проверку студенческих работ соответствию нормам стало не хватать. Появилась потребность в автоматизации процесса проверки.

Целью данной работы является разработка приложения для студентов и преподавателей, которое поможет оптимизировать прохождение нормоконтроля.

Актуальность темы исследования. Наше приложение позволит освободить время сотрудников от проверки работ учащихся, а также поможет студентам правильно оформить курсовую. Современные технологии создадут более комфортные условия прохождения нормоконтроля.

Для достижения поставленной цели были поставлены следующие задачи:

1. Проанализировать предметную область;

2. Определить стек технологий:

3. Ознакомиться с правилами оформления курсовых работ;

4. Создать макет приложения;

5. Разработать приложение;

6. Протестировать корректность работы готового продукта.

Решение состоит в том, чтобы создать веб-сервис, через который студент сможет сдать работу, предварительно автоматически проверив ее на соответствие всем нормам, а нормоконтролер сможет окончательно одобрить или написать замечания студенту.

Через сервис студент также сможет ознакомиться с правилами оформления работы, что поможет заранее предотвратить многие ошибки. Таким образом делается все, чтобы выявить или предотвратить максимальное количество ошибок до того, как за проверку возьмется человек. Сокращается количество ошибок, следовательно, сокращается количество перепроверок и в целом, времени на проверку работ тратиться намного меньше.

# Концепт приложения

## Стек технологий

Представленный список является не полным, в нем рассмотрены только основные технологии, составляющие каркас приложения.

1. **C# -** простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C [3].
2. **ASP.NET Core 2.1** (Active Server Pages для .NET) — платформа, предназначенная для разработки различного рода [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5): от небольших веб-сайтов до крупных веб-порталов и веб-сервисов. ASP.NET Core является полностью opensource-фреймворком и может работать на трех самых популярных операционных системах: Windows, Mac OS X, Linux [4].
3. **ADO.NET Entity Framework Core** (EF) — объектно-ориентированная технология доступа к данным, является [object-relational mapping](https://ru.wikipedia.org/wiki/ORM" \o "ORM) (ORM) решением для [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework) от [Microsoft](https://ru.wikipedia.org/wiki/Microsoft" \o "Microsoft). 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 [7].
4. **Microsoft SQL Server** — [система управления реляционными базами данных (РСУБД)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94), разработанная корпорацией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft" \o "Microsoft). Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка [1].
5. **JavaScript** — [мультипарадигменный](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Поддерживает [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [функциональный](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стили. Единственный язык программирования, поддерживаемый браузерами. Является реализацией языка ECMAScript (стандарт ECMA-262) [9]. В нашем проекте мы использовали стандарт – ECMAScript 6 [6].
6. **React** (иногда **React.js** или **ReactJS**) — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для разработки [пользовательских интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F). Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как [Redux](https://ru.wikipedia.org/wiki/Redux" \o "Redux) [8].
7. **Redux** — шаблон для [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) с открытым исходным кодом, предназначенный для управления состоянием приложения. Чаще всего используется в связке с React для разработки клиентской части [2].
8. **Webpack** —это сборщик модулей современных JavaScript приложений (в принципе любых веб приложений, у которых есть JavaScript часть). Когда он обрабатывает ваше приложение, то рекурсивно собирает граф зависимостей, которые требуются каждому модулю вашего приложения. Затем упаковывает все эти модули в небольшое число склеек (bundles), которые и грузятся в браузере.
9. **Препроцессор SaSS** — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей [10].

## Причины использования стека

На данный момент для создания web-сервисов используются самые разные языки программирования, такие как: Python, JavaScript, PHP, Ruby, C# и многие другие. Проанализировав цели нашего проекта, наша команда решила выбрать связку из нескольких языков программирования в качестве основных для разработки. Ими стали JavaScript и C#.

Web-приложение представляет из себя SPA (Single Page Aplication), то есть сайт, который состоит из одной страницы. По мере взаимодействия с сайтом, изменяются только отдельные его компоненты. SPA позволяет добиться высокой скорости приложения, так как создаёт под капотом легковесную копию DOM-дерева (shadow DOM), изменение которого очень быстрое в отличие от настоящего DOM'а [5]. Путем сравнения виртуального DOM'а с настоящим производится оптимальное изменение текущего DOM'а, что позволяет налету генерировать компоненты при изменении состояния и добиваться высокой производительности.

Есть еще одно преимущество в пользу SPA: это глобальное отделение клиентской части от серверной, что позволяет как можно больше отделить серверную разработку от фронтенда. Остается только одна точка соприкосновения — это API.

Таким образом, в нашем приложении клиентская и серверная часть разделяются так: на сервере располагается API и база данных, API получает данные с базы данных и отдаёт их клиенту. На клиенте же происходит прием этих данных, проброска к компонентам и красивое отображение. Так же на клиенте происходит роутинг (это еще одно отличие от классических сайтов, где роутинг осуществляет сервер).

Для построения таких приложений на данный момент на рынке существуют три самых популярных технологии: ReactJS, Angular, VueJS. Мы выбрали React, потому что он не ограничивает так, как Angular (всё-таки React библиотека, а не framework), но у него есть большое сообщество разработчиков, в отличии от Vue. Благодаря этому, искать информацию в интернете в разы проще.

Redux используется в связке с React практически всегда, так как в “нативном” React управлять потоками данных в больших приложениях очень сложно. Написание чего-то чуть больше to-do - листа превращается в пытку с пробрасыванием всех данных через props'ы. Благодаря этому корневой компонент становится очень большим.

Redux же позволяет избежать этого благодаря глобальному состоянию приложения (Store), которое может быть изменено только reducer'ами (которые являются “чистыми” функциями) в ответ на срабатывание action'ов. Таким образом обеспечивается иммутабельность state'а. То есть всем ходом внедряется функциональное программирование в продукт. Это позволяет обеспечить линейную масштабируемость приложения, что очень важно при последующей поддержке проекта и добавлении новых фич.

Хороший UI должен не только хорошо работать, но и хорошо выглядеть. Для того, чтобы удобно писать стили, мы использовали препроцессор SaSS и применили слегка модифицированную методологию БЭМ.

Осталось определится с серверной частью. Выбор был между NodeJS и [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core. Обе эти технологии прекрасно подходят для тех задач, которые на них планировалось возложить. Выбор пал на [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core из-за того, что много чего там реализовано "из коробки" (например, генерация JWT, на NodeJS пришлось бы устанавливать стороннюю библиотеку). Выбору поспособствовало и то, что в среде [ASP.NET](https://vk.com/away.php?utf=1&to=http%3A%2F%2FASP.NET) Core есть удобная функциональность для работы с базой данных, а именно ORM Entity Framework, которая реализует Code First, а также сама база данных MS Server.

Для сборки проекта, использовался Webpack, он не только собирал все файлы компонентов и стилей в сборки (бандлы), но также позволял поднимать отдельный Dev-сервер для более удобной разработки с использованием “горячей” перезагрузки приложения при изменении кода в клиентской части.

## Архитектура приложения

В нашем приложении реализована классическая клиент-серверная архитектура. Сервер в нашем случае роль API. То есть стороннего ПО, которое, во-первых, по запросу от клиента предоставляет различные данное из базы данных, во-вторых, сохраняет данные в этой же базе данных.

Как уже говорилось выше, в качестве основной технологии для создания API мы выбрали ASP.NET Core 2.1, а в качестве базы данных MS Server, которую мы будем использовать вместе с ORM Entity Framework Core, применяя подход Code First.

В качестве основного паттерна проектирования мы использовали классический MVC. Где “M” – модели данных, “C” – контроллеры, основная часть API, а “V” – это вся клиентская часть приложения.

В нашем приложении необходимо было реализовать авторизацию и регистрацию пользователей. Это требует повышенной защиты информации, ведь люди оставляют на сайте свои личные данные, а также пароли, которые в большинстве случаев одинаковые на разных ресурсах. Для того, чтобы обезопасить пользователей надо:

* Обеспечить надежную систему авторизации.
* Защитить пароли, если база данных попадет в чужие руки.

В качестве системы авторизации мы выбрали систему, основанную на Json Web токенах (JWT). На сегодняшний день она является лучшим решением для API из-за своей расширяемости. В отличии от классической системы, основанной на Cookie, она не привязана к конкретному домену, этот факт дает возможность размещать разные части приложения на разных серверах с разными адресами. Но при этом система не теряет своей надежности.

Защитить пароли при краже базы данных, поможет обычное хэширование. В нашем случае, мы сначала хэшируем пароль алгоритмом SHA-256, потом солим получившийся хэш с помощью секретного слова. Таким образом злоумышленникам, чтобы расшифровать пароль придётся перебрать все существующие комбинации, которых крайне большое количество. Можно сказать, что перебрать такое количество вариантов в разумное время не получится.

Из-за того, что приложение представляет из себя SPA, на клиентскую часть приложения возложено много функций. Это:

* Роутинг (или маршрутизация) по различным частям приложения.
* Запрос и получение данных с API.
* Управление потоками данных и доставка их в нужные компоненты.
* Отображение UI с нужными данными.
* Обновление UI при изменении состояния приложения.

Реализация всего этого функционала “нативными” средствами JS сделать очень тяжело. К счастью, решить эту проблему помогает React в связке с Redux.

Роутинг реализуется в нашем приложении библиотекой React-Router-DOM. Она очень проста в использовании, но при этом обладает богатым функционалом, позволяющий реализовать даже сложные вещи.

Для того, чтобы делать запросы к API максимально просто мы использовали очень легковесную библиотеку “axios”. Несмотря на ее минималистичность, она обладает большим функционалом и хорошей документацией.

Как уже говорилось выше Redux нужен для управления потоками данных. Он реализует подходы функционального программирования для того, чтобы писать большие расширяемые приложения. Также он автоматически обновляет UI при изменении состояния приложения.

React в свою очередь отображает данные полученные из глобального состояния.

## Анализ Целевой аудитории

Целевой аудиторией нашего приложения являются студенты и преподаватели.

Вне зависимости от года обучения, любой студент может пользоваться нашим приложением. Те учащиеся, которым необходимо пройти нормоконтроль, получают возможность прочитать легко воспринимаемую информацию о стандартах написания отчета, узнать о допущенных ошибках в оформлении курсовой работы. Им больше не потребуется долго ждать, чтобы узнать результат прохождения нормокнотроля.

Также преподавателю, занимающемуся нормокнонтролем, используя наше приложение, больше не потребуется вручную проверять огромное число курсовых работ. Результаты прохождения студентами нормоконтроля будут отправлены преподавателю.

# Результаты разработки приложения

## Презентация проекта

На нашем сервисе студенты могут без труда сдать свою работу на проверку и максимально быстро и понятно получить ответ. Для этого нужно пройти регистрацию в сервисе, указав все необходимые данные.

На сайте реализована система регистрации (*см. Рисунок 1*) и авторизации (*см. Рисунок 2*) пользователя, для того чтобы нормоконтролер знал, кто отправил работу.

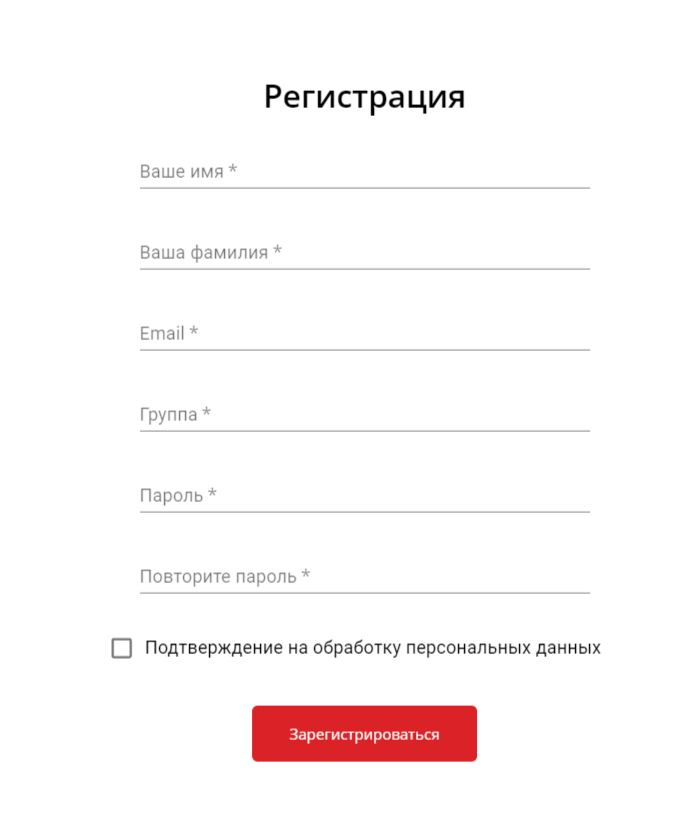


Рисунок 1 – Окно регистрации

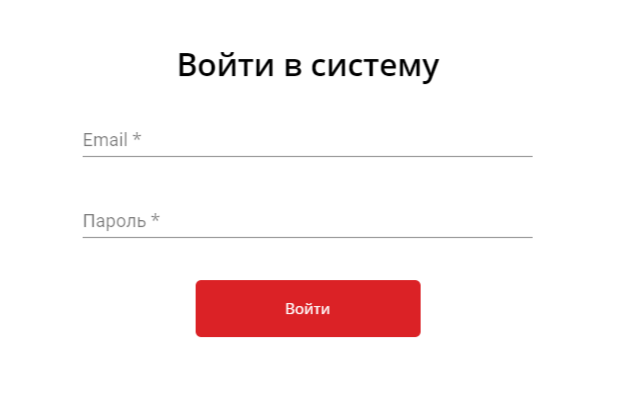


Рисунок 2 – Окно входа

Без регистрации пользователю доступна страница, где можно скачать правила оформления студенческих работ. (*см. Рисунок 3*)

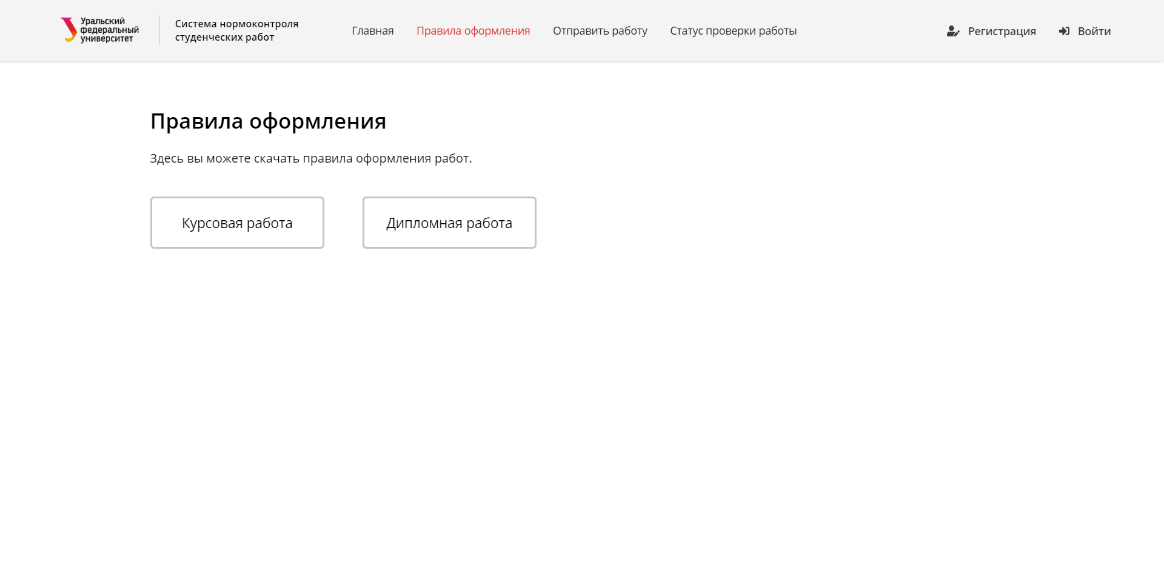


Рисунок 3 – Правила оформления

Для того, чтобы отправить работу нужна авторизоваться, перейти на страницу отправки работы (*см. Рисунок 4)* и пройти четыре простых шага:

* Выбор шаблона
* Загрузка работы
* Автоматическая проверка
* Отправка на проверку нормоконтролеру

Первым шагом надо выбрать шаблон оформления, который будет использоваться при автоматической проверке и для сортировки работ при проверке уже нормоконтролером.

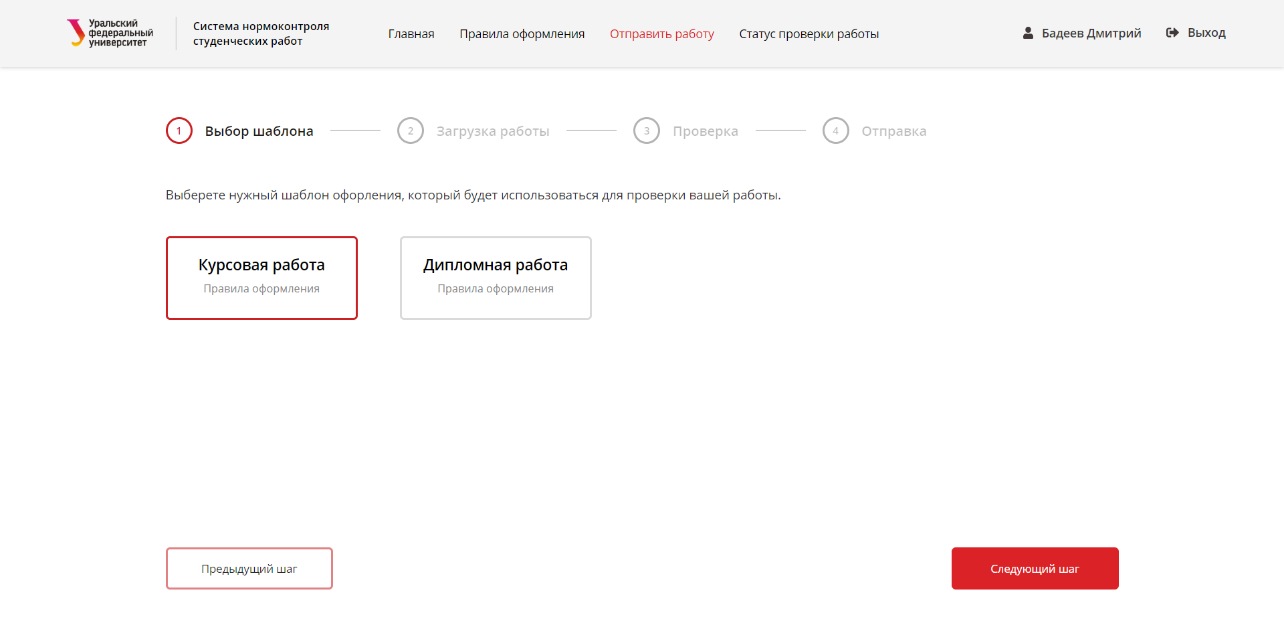


Рисунок 4 – Отправка работы

Вторым шагом нужно загрузить работу в формате docx. Можно выбрать файл вручную или перетащить его в обведенную область. (*см. Рисунок 5*)

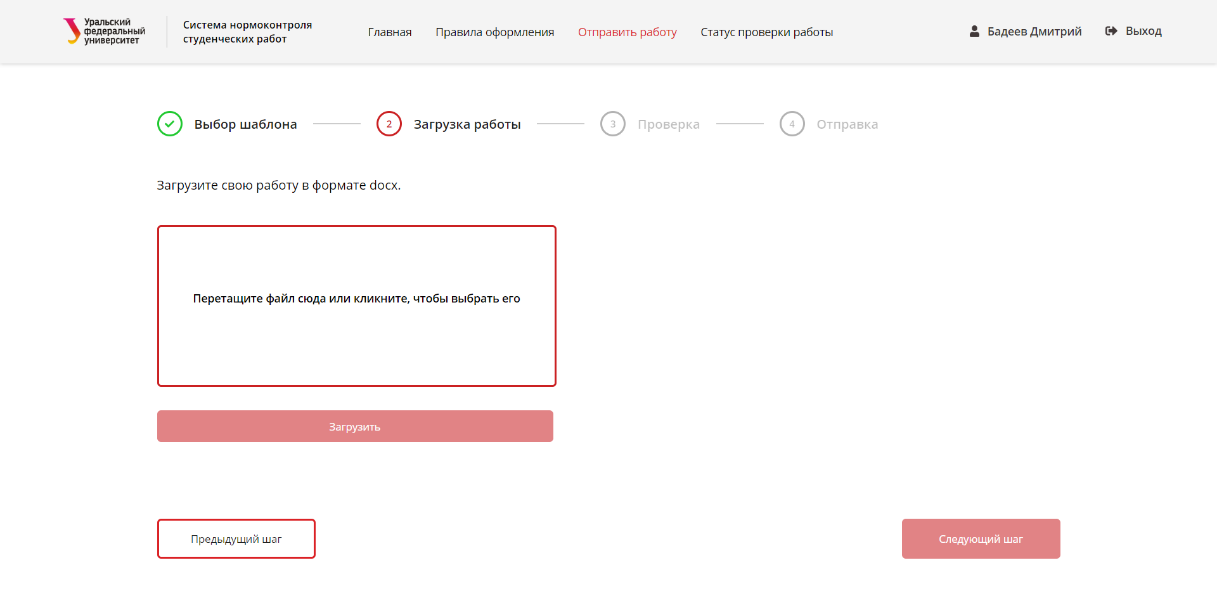


Рисунок 5 – Загрузка работы

Следующим шагом нужно пройти автоматическую проверку (*см. Рисунок 6)*. Система укажет на ошибки в работе если они есть. В случае ошибки система укажет на параметр, где она произошла и ожидаемое значение (*см. Рисунок 7).*

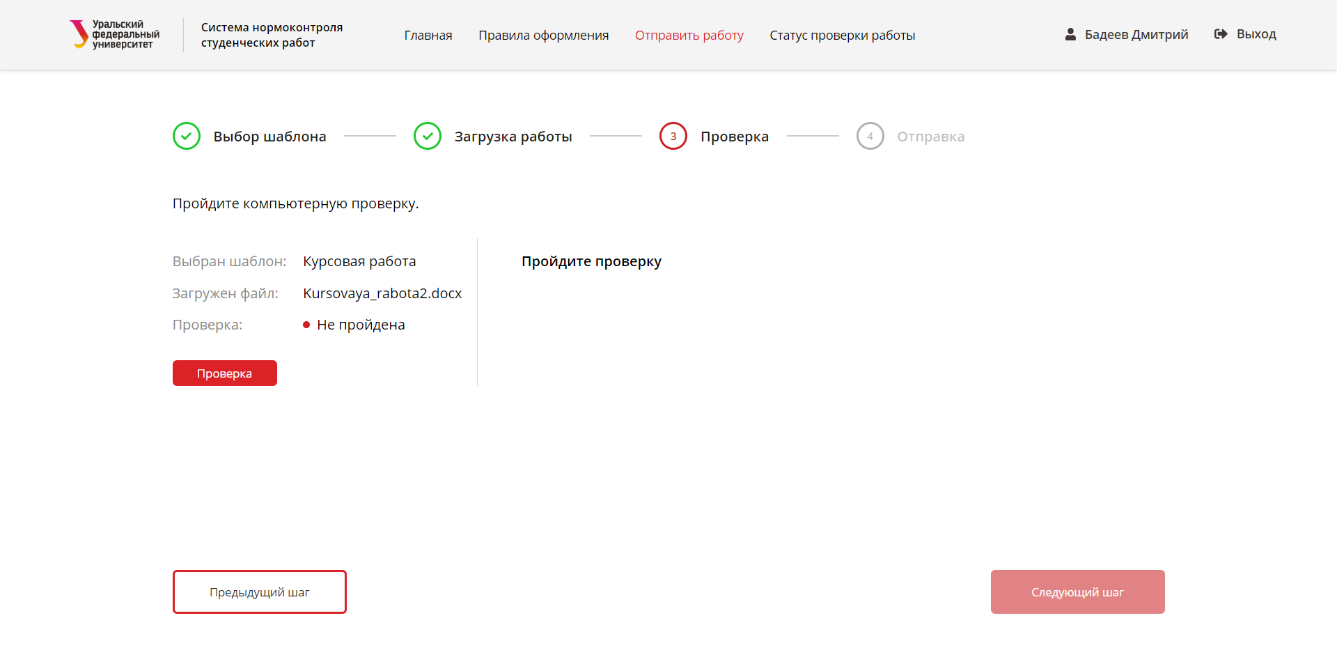


Рисунок 6 – Автоматическая проверка работы

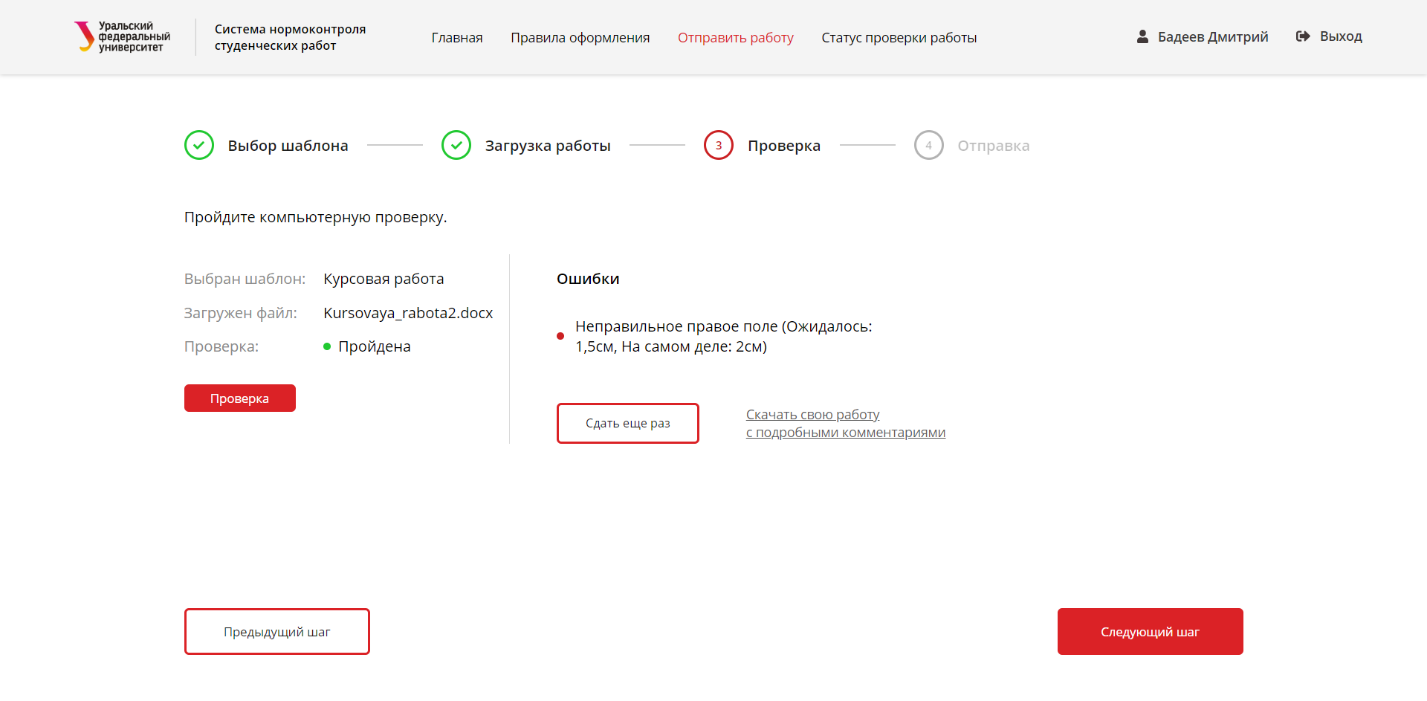


Рисунок 7 – Пройденная проверка

Последним шагом надо отправить работу на проверку нормоконтролеру. Можно включить уведомления по почте, чтобы не пропустить изменение статуса работы (*см. Рисунок 8)*.

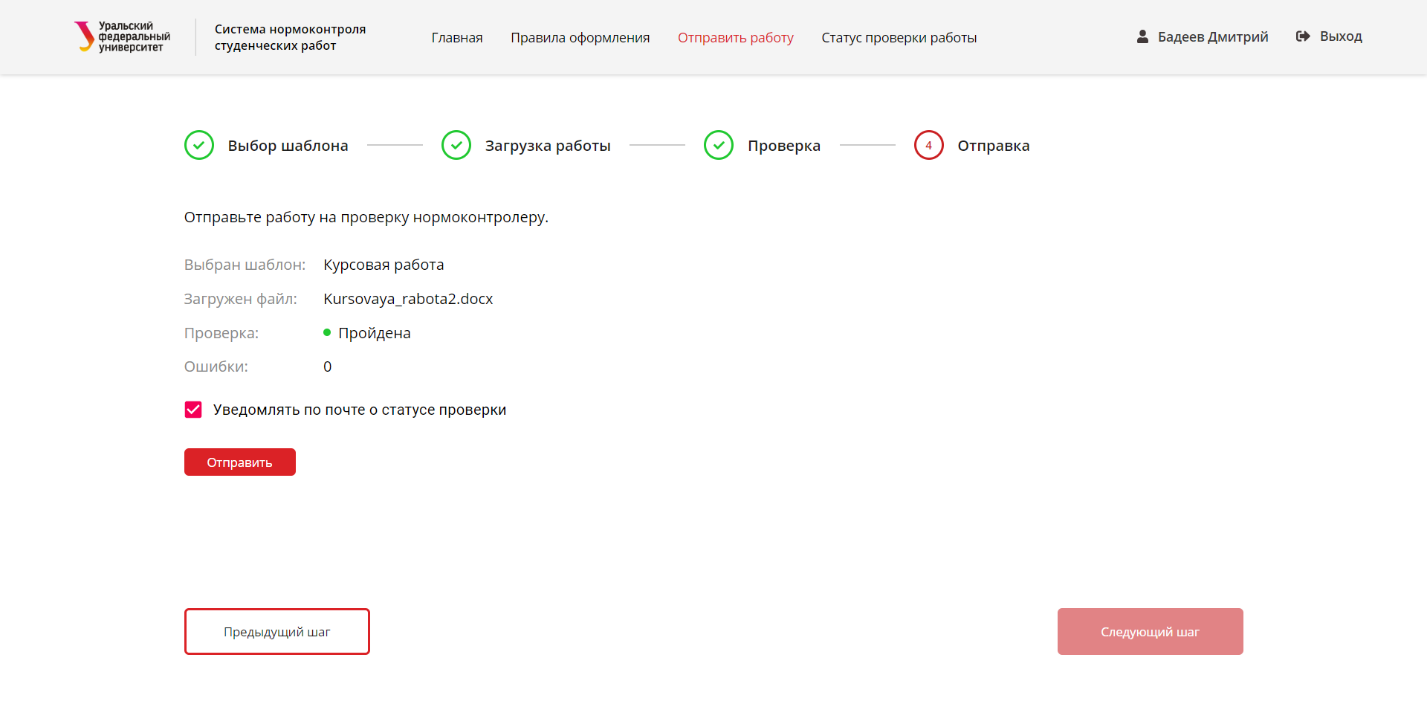


Рисунок 8 – Отправка работы

После отправки работу она появится на странице статусов проверки работ (*см. Рисунок 9)*. Там можно узнать ход проверки работы нормоконтролером.

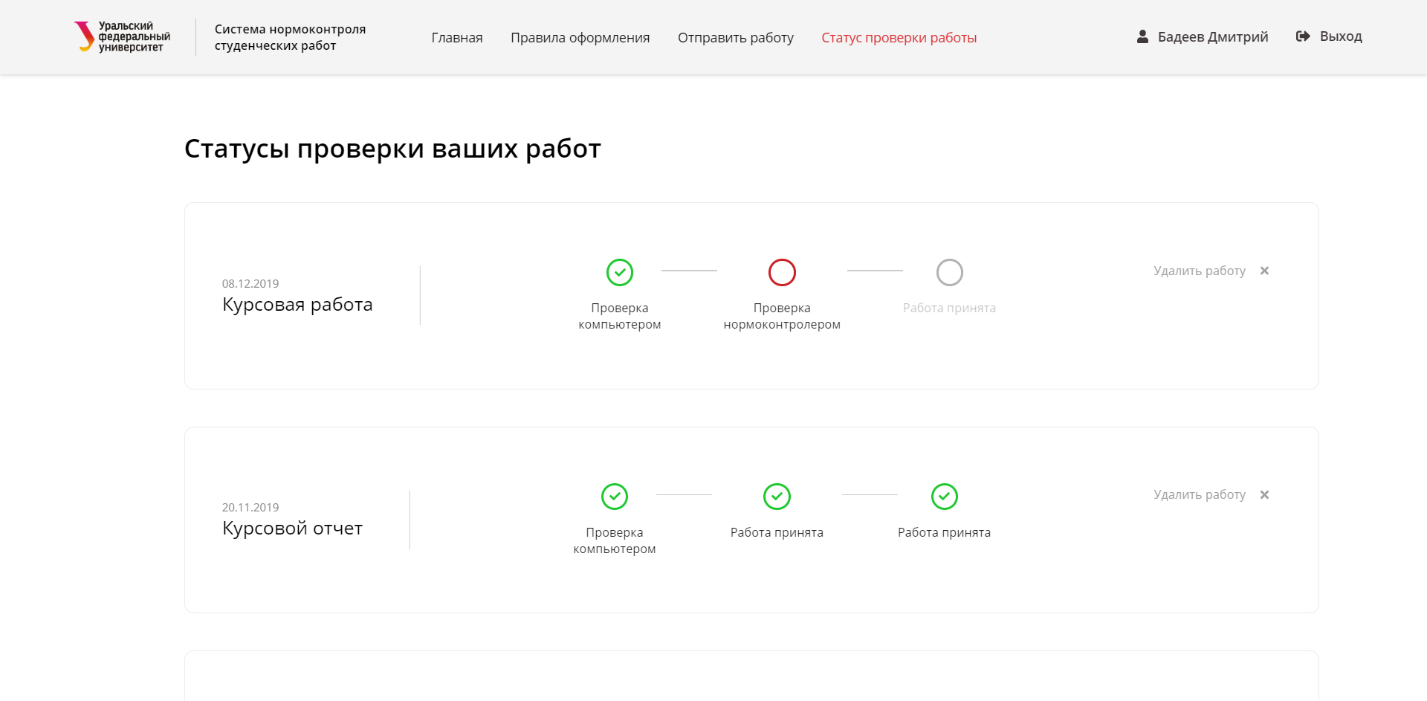


Рисунок 9 – Статусы проверки работ

В сервисе реализована закрытая часть для нормоконтролера. Там доступны все работы, которые были отправлены на проверку, можно написать замечания для исправления и отправить на доработку, или если все хорошо, то принять работу (*см. Рисунок 10)*.

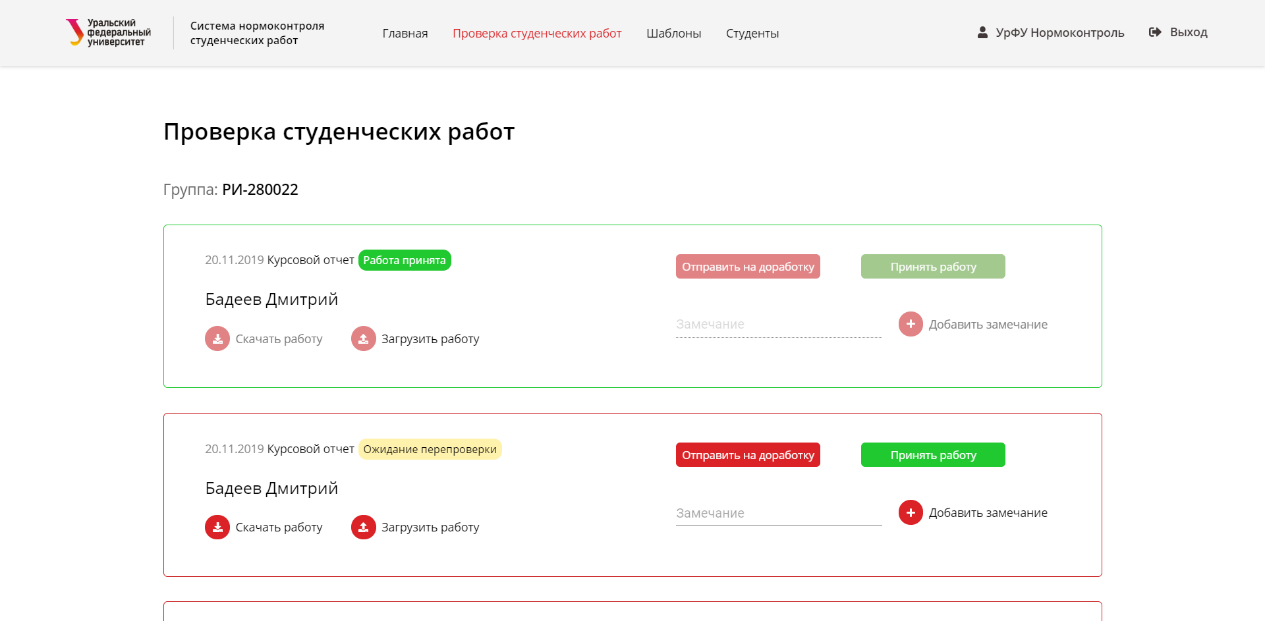


Рисунок 10 – Проверка работ

Также реализована функция добавления шаблонов (*см. Рисунок 11)*, которые будут доступны для всех студентов на странице “Правила оформления”.

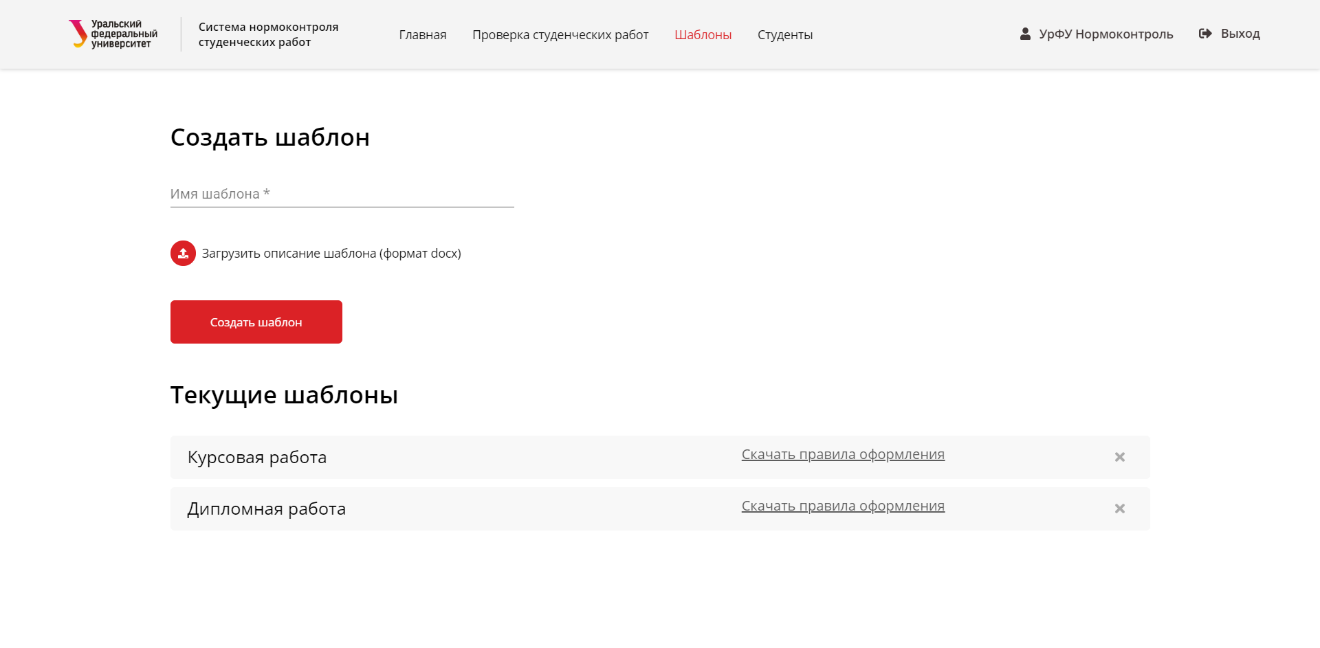


Рисунок 11 – Добавление шаблонов оформления

## Описание кода

Структура нашего приложения выглядит следующим образом (*см. Рисунок 12*)

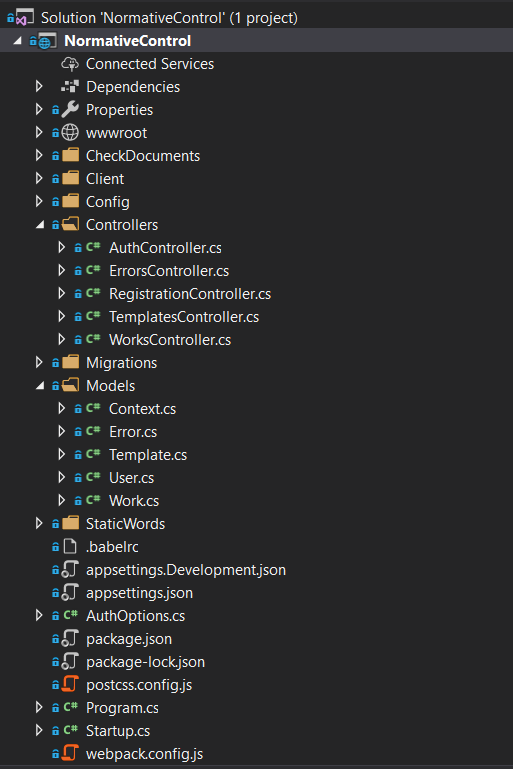


Рисунок 12 – Структура приложения

Запуск проекта начинается с выполнения метода Main класса Program. Код этого метода настраивает и запускает веб-хост, в рамках которого разворачивается приложение. Он же запускает методы из класса Startup, которые настраивают конвейер обработки запросов и подключают необходимые сервисы.

В папке Models лежат модели данных, из которых Entity Framework будет парсить таблицы в базе данных, а также контекст, присоединяющий модели к Entity Framework’у.

В папке Controllers лежат контроллеры, обрабатывающие запрос, приходящий на сервер.

В папке Client находится вся клиентская часть приложения. Она представляет из себя отдельный проект Node.js, точкой входа в который является package.json. В нем прописаны все зависимости приложения, которые хранятся в папке node\_modules.

На клиенте реализован Code Splitting, то есть ленивая загрузка компонентов, по мере их надобности, что позволяет уменьшить размер файла подгружаемого изначально. В index.js “загружаются” компоненты если в пользователь перешел по соответствующему URL.

Основным файлом, который настраивает сборку проекта является webpack.config.js. В нем прописана точка входа для приложения (файл, начиная с которого нужно собирать проект), параметры сборки и то, куда надо складывать готовый проект.

И наконец, в папке CheckDocuments находится все логика для проверки работы в соответствии с шаблоном. Все файлы для проверки загружаются пользователем в папку StaticWords.

## Тестирование

В ходе реализации нашего проекта, мы много времени уделяли тестированию отдельных частей нашего приложения. Нам требовались проверять на работоспособность практически каждую реализованную фичу.

Мы применяли три основных вида тестирования:

* Unit-тесты
* Тестирование с помощью стороннего ПО “Postman”
* Функциональное тестирование

Unit-тесты применялись преимущественно на серверной части в ходе тестирования отдельных методов. Например, проверка корректности хеш-функции.

С помощью “Postman” тестировалось API. В связи со сложностью написания тестов для проверки корректной работы сервера, пришлось отказаться от unit-тестирования в пользу использования стороннего ПО. Остановились на “Postman” из-за простоты и удобности в использовании.

Функциональное тестирование применялось к UI части приложения. Это показалось нам наиболее оптимальным выбором для тестирования интерфейса. Это не обременяет и без того тяжелую фронтенд часть лишним кодом, но позволяет качественно протестировать наш UI.

# ЗАКЛЮЧЕНИЕ

Целью данной работы была разработка приложения для студентов и преподавателей, которое поможет оптимизировать прохождение нормоконтроля.

В ходе работы мы проанализировали предметную область. Были выполнены следующие задачи:

1. определен стек технологий;

2. создан макет приложения;

3. разработано работающее приложение;

4. протестирована корректность работы приложения.

Наш продукт позволяет студентам и преподавателям сократить время похождения нормоконтроля. Вместо ручной проверки пользователи за короткое время получают результат о соответствии документа необходимому шаблону оформления. В результате проверки, автор документа получает список замечаний, подлежащих исправлению.

В дальнейшем мы планируем развивать наше приложение, добавляя новые функции. В частности, увеличить число шаблонов работ для возможности проверки не только курсовых, но и других студенческих работ, например, дипломных. Также мы планируем усовершенствовать приложение, добавив функцию автоматического исправления ошибок оформления работы.

Таким образом, наше приложение модернизирует стандартную процедуру прохождения нормоконтроля, позволяя пользователям минимизировать временные затраты.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Всё о MS SQL [Электронный ресурс] /. Электрон. текстовые дан. http://www.xserver.ru/computer/database/mysql/10/ **—** (дата обращения 29.10.2019)
2. Документация к Redux [Электронный ресурс] /. Электрон. текстовые дан. **—** https://rajdee.gitbooks.io/redux-in-russian/content/ **—** (дата обращения 29.10.2019)
3. Курс по C# [Электронный ресурс] /. Электрон. текстовые дан. **—** https://ulearn.me **—** (дата обращения 03.11.2018)
4. Справочник по ASP.NET Core 2.1 [Электронный ресурс] /. Электрон. текстовые дан. — https://metanit.com/sharp/aspnet5/1.1.php **—** (дата обращения 20.11.2018)
5. Справка HTML5 [Электронный ресурс] /. Электрон. текстовые дан. **—** https://developer.mozilla.org/ru/docs/Web/HTML/Ссылки **—** (дата обращения 29.11.2019)
6. Учебник по JS [Электронный ресурс] /. Электрон. текстовые дан. — https://learn.javascript.ru/intro **—** (дата обращения 29.11.2019)
7. Руководство по Entity Framework [Электронный ресурс] /. Электрон. Текстовые дан. **—** http://www.xserver.ru/computer/database/mysql/10/ **—** (дата обращения 29.11.2019)

## Руководство по ReactJS [Электронный ресурс] /. Электрон. текстовые дан. — https://learn-reactjs.ru/home — (дата обращения 29.11.2019)

1. Руководство ECMAScript module [Электронный ресурс] /. Электрон. тестовые дан. **—** https://o7planning.org/ru/12181/ecmascript-modules-tutorial **—** (дата обращения 29.11.2019)
2. Руководство CSS3 [Электронный ресурс] /. Электрон. текстовые дан. **—** https://www.w3bai.com/ru/css/default.html **—** (дата обращения 29.11.2019)