ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Программная инженерия»

СОГЛАСОВАНО
Старший преподаватель департамента больших данных и информационного поиска

_____ В.В. Куренков
«_10_» _____мая_____ 2023 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____ 2023 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга

Программа и методика испытаний

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.10-01 51 01-1-ЛУ

Исполнитель
студент группы БПИ219
_____ / С.О. Котовский /
«__08_»____мая____ 2023 г.

Москва 2023

УТВЕРЖДЕН
RU.17701729.05.10-01 51 01-1-ЛУ

Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга

Программа и методика испытаний

RU.17701729.05.10-01 51 01-1

Листов 36

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Москва 2023

ОГЛАВЛЕНИЕ

1. ОБЪЕКТ ИСПЫТАНИЙ

1.1. Наименование программы и объекта, в котором программа используется

Наименование темы разработки: «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга».

Наименование темы разработки на английском языке: «HSE Lyceum Programming Testing System with Automated Rating Estimation».

Наименование объекта, в котором программа используется: Лицей НИУ ВШЭ.

1.2. Краткая характеристика области применения программы и объекта, в котором она используется

«Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга» - набор программных инструментов для оценивания решений задач на разных языках программирования для учащихся Лицея НИУ ВШЭ с рейтингом для каждого учащегося.

Лицей НИУ ВШЭ – общеобразовательное учреждение города Москва.

2. ЦЕЛЬ ИСПЫТАНИЙ

Цель проведения испытаний – проверить разработанную программу на соответствие функциональным требованиям и отдельным требованиям к надежности, изложенных в документе «Техническое задание» к данной программе.

3. ТРЕБОВАНИЯ К ПРОГРАММЕ

3.1. Требования к функциональным характеристикам

3.1.1. Требования к клиентской части

Процесс регистрации:

Регистрация нового студента происходит через сущность преподавателя. (см. Преподаватель - создание учётной записи студента)

Регистрация нового преподавателя проводится администратором системы в ручном режиме.

Процесс авторизации:

1. Авторизация пользователя

1.1. Система должна предоставлять форму для ввода адреса электронной почты и пароля пользователя.

1.2. Форма должна содержать два текстовых поля для ввода e-mail и пароля.

1.3. Система должна проверять введенные данные на корректность.

1.4. Система должна отображать сообщение об ошибке при вводе некорректных данных.

2. Отправка данных на сервер

2.1. После ввода данных, система должна отправлять запрос на сервер для проверки введенных данных.

2.2. Запрос должен содержать введенный адрес электронной почты и пароль пользователя.

2.3. Система должна обрабатывать ответ от сервера и предпринимать соответствующие действия в зависимости от ответа.

3. Обработка ответа сервера

3.1. Если сервер подтверждает корректность введенных данных, система должна сохранять токен авторизации, возвращенный сервером, в локальном хранилище браузера.

3.2. Если сервер сообщает о некорректности введенных данных, система должна отображать сообщение об ошибке.

4. Перенаправление пользователя

4.1. После успешной авторизации система должна перенаправлять пользователя на соответствующую страницу в зависимости от роли пользователя (ученик или учитель).

4.2. Система должна обеспечивать автоматическое перенаправление на страницу входа при попытке доступа к защищенным ресурсам без авторизации.

5. Выход из системы

5.1. Система должна предоставлять функцию для выхода из системы.

5.2. При выходе из системы, система должна удалять токен авторизации из локального хранилища браузера и перенаправлять пользователя на страницу входа.

Элементы шапки сайта (header):

1. Вывод информации о пользователе

1.1. Система должна отображать имя пользователя в правой части шапки сайта.

1.2. Система должна получать информацию о пользователе (включая имя, фамилию и роль) с сервера.

1.3. Система должна проверять статус авторизации пользователя и в случае отсутствия авторизации, перенаправлять пользователя на страницу входа.

2. Навигация по сайту

2.1. Шапка сайта должна содержать основные ссылки для навигации по сайту.

2.2. Доступные ссылки должны включать: "Мои посылки", "Мои контесты", "Рейтинг".

2.3. Пункты навигации должны быть адаптированы в соответствии с ролью пользователя. Для учителей должна быть доступна ссылка на "Админ-панель", для студентов - ссылка на "Личный кабинет".

3. Выход из системы

3.1. Шапка сайта должна предоставлять функцию для выхода из системы.

3.2. При нажатии на кнопку "Выйти", система должна удалять токен авторизации из локального хранилища браузера и перенаправлять пользователя на страницу входа.

4. Стилизация шапки сайта

4.1. Все изменения в стилизации должны быть реализованы при помощи CSS, без использования дополнительных библиотек или фреймворков.

5. Логотип сайта

5.1. Шапка сайта должна содержать логотип сайта и его название.

5.2. Логотип и название должны быть расположены слева в шапке сайта.

Элементы подвала сайта (footer)

1. Описание компонента:
   - Компонент "SiteFooter" представляет собой футер сайта.
   - Компонент содержит информацию о копирайте и авторских правах.
   - Компонент не имеет интерактивных элементов.
2. Разметка и стилизация:
   - В разметке компонента используется элемент `<footer>` для обозначения футера сайта.
   - Футер содержит контейнер `<div>` с классом "footer-container", внутри которого располагается текст с информацией о копирайте.
   - Стилизация футера задана в блоке `<style>`. Контейнер футера имеет высоту 70 пикселей, фоновый цвет #454545, и находится в постоянном положении (static).
   - Контейнер футера использует flexbox для выравнивания содержимого по центру.
3. Компонент не содержит вычисляемых свойств (computed).
4. События:
   - Компонент не генерирует собственных событий.
5. Входные параметры (props):
   - Компонент не принимает входных параметров.
6. Методы:
   - Компонент не содержит собственных методов.
7. Дополнительная информация:
   - Компонент является статическим и не требует активного взаимодействия пользователя.
   - Компонент может быть размещен внизу страницы для отображения футера сайта с информацией о копирайте.

Таким образом, компонент "SiteFooter" представляет собой простой футер сайта с информацией о копирайте, который не взаимодействует с пользователем и не требует входных параметров или методов для своей работы.

Страница профиля студента

1. Вывод информации о студенте

1.1. Страница профиля студента должна отображать основную информацию о студенте: имя, фамилию, направление, группу, ID, электронную почту, класс, программу.

1.2. Данные должны быть получены с сервера.

1.3. В случае возникновения ошибки при получении данных, ошибка должна быть залогирована.

2. Вывод рейтинга студента

2.1. Страница профиля студента должна отображать рейтинг студента по оценкам и рейтинг студента по задачам. (номер места в рейтинге)

3. Навигация

3.1. Страница должна содержать шапку сайта и футер, которые являются отдельными компонентами Vue.js.

4. Стилизация страницы профиля студента

4.1. Все изменения в стилизации должны быть реализованы при помощи CSS, без использования дополнительных библиотек или фреймворков.

Админ-панель преподавателя

1. Вывод информации о преподавателе

1.1. Админ-панель преподавателя должна отображать основную информацию о преподавателе: имя, отчество, фамилию, электронную почту, кафедру, группу.

1.2. Данные должны быть получены с сервера.

1.3. В случае возникновения ошибки при получении данных, ошибка должна быть залогирована.

2. Функционал админ-панели преподавателя

2.1. Админ-панель преподавателя должна содержать функционал для создания аккаунта студента и импорта контеста из Ejudge.

2.2. Данный функционал должен быть представлен в виде кнопок, которые ведут на соответствующие страницы.

3. Навигация

3.1. Админ-панель должна содержать шапку сайта и футер, которые являются отдельными компонентами Vue.js.

4. Стилизация админ-панели преподавателя

4.1. Все изменения в стилизации должны быть реализованы при помощи CSS, без использования дополнительных библиотек или фреймворков.

5. Роутинг

5.1. Страницы "Создать аккаунт студента" и "Импорт контеста из Ejudge" должны быть доступны по соответствующим ссылкам в админ-панели преподавателя.

Страница импорта контеста из ejudge

1. Ввод данных контеста

1.1. Страница должна содержать форму для ввода данных контеста: ID контеста, время начала, время окончания и название контеста.

1.2. Все поля формы являются обязательными для заполнения.

2. Функционал импорта контеста

2.1. При отправке формы должен производиться запрос для импорта контеста из Ejudge.

2.2. Во время выполнения запроса должен отображаться индикатор обработки.

2.3. После успешного выполнения запроса должно отображаться сообщение о успешном добавлении контеста.

2.4. В случае возникновения ошибки при выполнении запроса, ошибка должна быть залогирована.

3. Навигация

3.1. Страница импорта контеста из Ejudge должна содержать шапку сайта и футер, которые являются отдельными компонентами Vue.js.

4. Стилизация страницы импорта контеста

4.1. Все изменения в стилизации должны быть реализованы при помощи CSS, без использования дополнительных библиотек или фреймворков.

5. Обработка и отправка данных на сервер

5.1. Данные из формы должны быть обработаны и преобразованы в соответствующий формат. (описано в техническом задании для backend)

5.2. Отправка данных на сервер должна производиться при помощи Axios, с использованием токена авторизации, хранящегося в LocalStorage.

5.3. В случае успешного добавления контеста, должно отображаться соответствующее сообщение.

Страница создания аккаунта студента

1. Ввод данных студента

1.1. Страница должна содержать форму для ввода данных студента: имя, фамилия, отчество, почта, роль, пароль, направление и группа.

1.2. Все поля формы являются обязательными для заполнения.

2. Функционал создания аккаунта студента

2.1. При отправке формы должен производиться запрос на сервер для создания аккаунта студента.

2.2. В случае успешного выполнения запроса, должно отображаться соответствующее сообщение (необходимо добавить на страницу).

2.3. В случае возникновения ошибки при выполнении запроса, ошибка должна быть залогирована.

3. Навигация

3.1. Страница создания аккаунта студента должна содержать шапку сайта и футер, которые являются отдельными компонентами Vue.js.

4. Стилизация страницы создания аккаунта студента

4.1. Все изменения в стилизации должны быть реализованы при помощи CSS, без использования дополнительных библиотек или фреймворков.

5. Обработка и отправка данных на сервер

5.1. Данные из формы должны быть обработаны и преобразованы в соответствующий формат.

5.2. Отправка данных на сервер должна производиться при помощи Axios, с использованием заголовка 'Content-Type': 'application/json'.

5.3. В случае успешного создания аккаунта студента, должно отображаться соответствующее сообщение (необходимо добавить на страницу).

Страница с рейтингом пользователей

1. Выбор типа рейтинга

1.1. Страница должна содержать кнопки для выбора типа рейтинга: "Рейтинг по оценкам" и "Рейтинг по задачам".

1.2. По умолчанию, при загрузке страницы, должен быть активным рейтинг по оценкам.

1.3. При выборе типа рейтинга, должна происходить обновление данных рейтинга и отображение соответствующей информации.

2. Получение данных рейтинга

2.1. При загрузке страницы и при смене типа рейтинга, должен производиться запрос на сервер для получения данных рейтинга.

2.2. Запрос должен быть выполнен с использованием токена, полученного при аутентификации.

2.3. Запрос должен отправляться на соответствующий эндпоинт в зависимости от выбранного типа рейтинга: "http://37.252.0.155:8080/api/grade/by-marks" для рейтинга по оценкам и "http://37.252.0.155:8080/api/grade/by-tasks-amount" для рейтинга по задачам.

2.4. При успешном получении данных, должны быть обновлены значения переменной `ratingData` и отображена таблица с данными рейтинга.

2.5. В случае возникновения ошибки при получении данных, ошибка должна быть залогирована, а пользователю должно быть отображено соответствующее сообщение об ошибке.

3. Выгрузка данных рейтинга в CSV

3.1. На странице должна быть кнопка "Выгрузить в .csv", которая позволяет пользователю скачать данные рейтинга в формате CSV.

3.2. При клике на кнопку "Выгрузить в .csv", должен быть выполнен запрос на сервер для получения CSV файла.

3.3. Запрос должен быть выполнен с использованием токена, полученного при аутентификации.

3.4. Запрос должен отправляться на соответствующий эндпоинт в зависимости от выбранного типа рейтинга: "http://37.252.0.155:8080/api/grade/by-marks/download-csv" для рейтинга по оценкам и "http://37.252.0.155:8080/api/grade/by-tasks/download-csv" для рейтинга по задачам.

3.5. При успешном получении CSV файла, файл должен быть скачан пользователем.

3.6. В случае возникновения ошибки при получении CSV файла, ошибка должна быть залогирована, а пользователю должно быть отображено соответствующее сообщение об ошибке.

4. Отображение данных рейтинга

4.1. После успешного получения данных рейтинга, таблица должна быть отображена на странице.

4.2. Таблица должна содержать следующие столбцы: "Имя", "Фамилия", "Отчество", "Email", "Роль", "Факультет" и "Группа".

4.3. Каждая строка таблицы должна представлять отдельного пользователя и содержать соответствующую информацию из полученных данных рейтинга.

5. Отображение ошибок

5.1. В случае возникновения ошибки при получении данных рейтинга или при скачивании CSV файла, ошибка должна быть залогирована, а пользователю должно быть отображено соответствующее сообщение об ошибке.

5.2. Сообщение об ошибке должно быть отображено под кнопками выбора типа рейтинга или над таблицей данных рейтинга.

5.3. Сообщение об ошибке должно быть выделено цветом и/или шрифтом, чтобы привлечь внимание пользователя.

5.4. После успешной загрузки данных или успешного скачивания CSV файла, сообщение об ошибке должно быть очищено или скрыто.

6. Оформление интерфейса

6.1. Страница должна иметь адаптивный дизайн и корректно отображаться на различных устройствах и экранах.

6.2. Кнопки выбора типа рейтинга должны иметь соответствующий стиль и быть различимыми.

6.3. Таблица данных рейтинга должна быть оформлена с использованием стилей, чтобы обеспечить читаемость и удобство просмотра.

6.4. Сообщения об ошибке должны быть оформлены в соответствии с общим стилем страницы и хорошо видны для пользователя.

6.5. Другие элементы интерфейса, такие как заголовки, тексты и кнопки, должны быть четко видны и читаемыми.

6.6. Страница должна иметь приятный и интуитивно понятный пользовательский интерфейс для удобства использования.

6.7. Верхний и нижний колонтитулы, содержащие компоненты `SiteHeader` и `SiteFooter`, должны быть отображены на странице для обеспечения единого внешнего вида и навигации по сайту.

7. Взаимодействие с сервером

7.1. Все запросы к серверу должны быть выполнены с

использованием библиотеки `axios` для осуществления HTTP-запросов.

7.2. Для каждого запроса к серверу должен быть указан соответствующий URL-адрес эндпоинта и метод HTTP (например, GET, POST).

7.3. Запросы должны содержать необходимые заголовки, включая токен авторизации в заголовке "Authorization" для аутентификации пользователя.

7.4. В случае успешного выполнения запроса, полученные данные должны быть обработаны и отображены на странице в соответствии с требованиями.

7.5. В случае ошибки при выполнении запроса, ошибка должна быть обработана и отображена пользователю в виде соответствующего сообщения.

7.6. Для обработки ошибок и логирования необходимо использовать соответствующие механизмы предоставляемые библиотекой `axios` и языком программирования JavaScript.

8. Общие требования

8.1. Код страницы должен быть написан в соответствии с принципами чистого и структурированного кода, обеспечивая читаемость, модульность и масштабируемость.

8.2. Компоненты `SiteHeader` и `SiteFooter` должны быть включены и правильно использованы на странице для обеспечения единого внешнего вида и навигации по сайту.

8.3. Все зависимости, необходимые для работы страницы, должны быть указаны в файле `package.json` и должны быть установлены с использованием менеджера пакетов (например, npm или yarn).

8.4. Страница должна быть протестирована на различных устройствах и браузерах, чтобы обеспечить ее корректное отображение и функциональность.

8.5. Все внешние ресурсы, такие как изображения или сторонние библиотеки, должны быть правильно подключены и загружены на странице.

8.6. Все данные, полученные от сервера или введенные пользователем, должны быть проверены и обработаны с учетом возможных ошибок и безопасности.

8.7. Страница должна быть доступна для использования пользователями с различными специальными потребностями, следуя принципам доступности веб-контента.

8.8. Документация, описывающая функциональность страницы, ее зависимости и взаимодействие с сервером, должна быть предоставлена для разработчиков и сопровождающих систему.

Страница с посылками конкретного пользователя

Описание: Страница ParcelsList предназначена для отображения списка посылок. На странице отображается информация о каждой посылке, такая как ID, язык, статус, время работы, использованная память, тест и другие данные. Пользователь может просмотреть код посылки, нажав на кнопку "Просмотреть код".

Структура файла:

- Импорт необходимых компонентов: SiteFooter, SiteHeader, axios.
- Определение компонента Vue с наименованием "ParcelsList".
- Определение следующих data-свойств:
  - parcels: массив для хранения списка посылок;
  - modalOpen: логическое значение для открытия/закрытия модального окна с кодом посылки;
  - selectedCode: строка для хранения выбранного кода посылки.
- Определение вычисляемого свойства:
  - mainHeadline: строка, содержащая основной заголовок страницы в зависимости от роли пользователя.
- Определение методов:
  - fetchParcels: метод для получения списка посылок с сервера;
  - openCodeModal: метод для открытия модального окна с кодом посылки;
  - closeModal: метод для закрытия модального окна.
- В хуке created вызывается метод fetchParcels для получения списка посылок при загрузке страницы.

Внешний вид страницы: Страница состоит из следующих элементов:

- Компонент SiteHeader для отображения шапки сайта.
- Компонент SiteFooter для отображения футера сайта.
- Заголовок страницы в зависимости от роли пользователя.
- Информация о количестве посылок.
- Список посылок в виде таблицы со следующими столбцами:
  - ID: идентификатор посылки.
  - Просмотреть код: кнопка, при нажатии на которую открывается модальное окно с кодом посылки.
  - Язык: язык программирования посылки.
  - Статус: текущий статус посылки.
  - Время работы: время выполнения посылки.
  - Использованная память: объем памяти, использованный посылкой.
  - Тест: результат тестирования посылки.
  - Название контеста: название контеста, в рамках которого выполнена посылка.
  - Название задачи: название задачи, для которой выполнена посылка.

Модальное окно с кодом посылки, открываемое при нажатии на кнопку "Просмотреть код". Модальное окно содержит заголовок "Код посылки" и отображает выбранный код посылки в виде предварительно отформатированного текста. Модальное окно также содержит кнопку "Закрыть", при нажатии на которую модальное окно закрывается.

Стилизация страницы:

- Компонент modal-backdrop: фиксированное позиционирование, затемненный фон с прозрачностью, выравнивание по центру экрана.
- Компонент modal-content: фоновый цвет, скругленные углы, отступы и размеры, позволяющие контенту поместиться внутри окна, возможность прокрутки при содержании большого количества данных.

Технические требования:

1. Запросы на сервер:
    - Метод fetchParcels использует библиотеку axios для отправки GET-запроса на эндпоинт `http://37.252.0.155:8080/api/solutions/${endpoint}` для получения списка посылок.
    - Заголовок запроса содержит ключ авторизации в формате `Authorization: Bearer <token>`, где `<token>` - значение токена, полученного при аутентификации.
    - В случае успешного выполнения запроса, полученные данные присваиваются свойству parcels компонента.
    - В случае ошибки при выполнении запроса, ошибка должна быть залогирована в консоль.
2. Модальное окно:
    - Компонент modalOpen управляет отображением модального окна. При щелчке на кнопку "Просмотреть код" свойство modalOpen устанавливается в значение true, что приводит к отображению модального окна.
    - Компонент selectedCode хранит выбранный код посылки, который отображается в модальном окне.
    - При нажатии на кнопку "Закрыть" свойство modalOpen устанавливается в значение false, что приводит к закрытию модального окна.
3. Обработка ошибок:
    - В случае возникновения ошибки при выполнении запроса на получение списка посылок, ошибка должна быть залогирована в консоль с помощью console.error().
4. Навигация:
    - Страница должна содержать компоненты SiteHeader и SiteFooter для отображения шапки и футера сайта соответственно.
5. Стилизация:
    - Стилизация страницы должна быть реализована с использованием CSS.
    - Для модального окна следует задать стили для компонентов modal-backdrop и modal-content, обеспечивающие желаемый внешний вид и расположение.

Страница со списком всех контестов

Страница `MyContestsList` предназначена для отображения списка контестов и их основных свойств. Ниже представлено техническое задание для данной страницы:

1. Загрузка списка контестов:
   - При создании компонента `MyContestsList` должен быть выполнен асинхронный метод `fetchContests`, который выполняет GET-запрос к API для получения списка контестов.
   - Запрос должен использовать токен авторизации, полученный из локального хранилища.
   - Адрес эндпоинта для получения списка контестов: `http://37.252.0.155:8080/api/contest/get-all`.
   - В случае успешного выполнения запроса, полученные данные должны быть присвоены свойству `contests` компонента.
   - При возникновении ошибки во время выполнения запроса, ошибка должна быть залогирована в консоль.
2. Отображение списка контестов:
   - В шаблоне компонента используется цикл `v-for` для отображения каждого контеста в таблице.
   - Поля контеста, такие как ID в Ejudge, название контеста, дата начала и дата окончания, должны быть отображены в соответствующих столбцах таблицы.
   - Для преобразования даты начала и даты окончания контеста в удобный формат, используется метод `toLocaleString()` с объектом `Date`.
   - Для перехода к отдельному контесту используется компонент `router-link`, который принимает параметры маршрута `name` и `params`.
3. Сохранение выбранного контеста:
   - При щелчке на кнопку "Перейти" в таблице, вызывается метод `saveContestId`, который сохраняет ID контеста в Ejudge и его название в локальном хранилище.
   - Значения сохраняются в виде пар ключ-значение в localStorage с ключами "contestId" и "contestName" соответственно.
4. Навигация:
   - Страница должна содержать компоненты `SiteHeader` и `SiteFooter` для отображения шапки и футера сайта соответственно.
5. Стилизация:
   - Кнопка "Перейти" имеет стилизацию, заданную в блоке стилей компонента.
   - При наведении, активации и фокусе кнопки должны применяться соответствующие стилевые изменения для создания визуального отклика.

Страница с конкретным контестом

Страница `Contest` предназначена для отображения информации о контесте, задачах, отправке решений и результатов. Ниже представлено техническое задание для данной страницы:

1. Загрузка задач контеста:

- При создании компонента `Contest` должен быть выполнен асинхронный метод `fetchTasks`, который выполняет GET-запрос к API для получения списка задач контеста.
- Адрес эндпоинта для получения списка задач: `http://37.252.0.155:3000/parseTasks`.
- В случае успешного выполнения запроса, полученные данные должны быть присвоены свойству `tasks` компонента.
- Выбирается первая задача из списка и вызывается метод `updateTask` для отображения информации о задаче.

2. Аутентификация:
   - При загрузке страницы, должен быть выполнен метод `authenticate`, который отправляет POST-запрос на сервер для аутентификации.
   - Аутентификационный запрос выполняется на адрес `http://37.252.0.155:3000/auth`.
   - В теле запроса передаются параметры `login`, `password` и `contestID`, полученные из локального хранилища.
   - В случае успешной аутентификации, полученные данные сохраняются для дальнейшего использования.
   - При возникновении ошибки во время аутентификации, ошибка должна быть залогирована в консоль.
3. Обновление текущей задачи:
   - Метод `updateTask` принимает номер задачи в качестве аргумента.
   - Используя номер задачи, находит соответствующую задачу в списке `tasks` и обновляет свойства `currentTask`, `taskTitle` и `taskDescription` с информацией о выбранной задаче.
   - Если задача не найдена, свойства `taskTitle` и `taskDescription` сбрасываются.
4. Выбор языка программирования:
   - При изменении значения выпадающего списка выбора языка программирования (`select`), вызывается метод `onLanguageChange`.
   - Метод обновляет значения свойств `selectedLanguage` и `languageName` с выбранным языком программирования и его названием соответственно.
5. Отправка решения:

- При щелчке на кнопку "Отправить", вызывается метод `submitSolution`.
- Если выбран файл решения, метод выполняет следующие шаги:
- Преобразует файл в формат base64 и сохраняет его в переменную `base64File`.
- Считывает содержимое файла как текст и сохраняет его в переменную `code - Выполняет POST-запрос на адрес `http://37.252.0.155:3000/handleSolution` с параметрами:
  - `solutionFileBase64`: содержимое файла решения в формате base64.
  - `taskID`: идентификатор текущей задачи.
  - `language`: выбранный язык программирования.
- В случае успешного выполнения запроса, полученный результат (статус решения и ошибка на тесте) сохраняется в соответствующие свойства `solutionStatus` и `failureTest`.
- Вызывается метод `submitSolutionToServer` для отправки решения на сервер.

6. Отправка решения на сервер:

- Метод `submitSolutionToServer` выполняет POST-запрос на адрес `http://37.252.0.155:8080/api/solutions/add` для отправки решения на сервер.
- В теле запроса передаются следующие параметры:
  - `code`: содержимое решения.
  - `language`: выбранный язык программирования.
  - `status`: статус решения.
  - `used_time`: время выполнения решения.
  - `used_memory`: использованная память.
  - `error_test`: ошибка на тесте.
  - `contest_name`: название контеста из локального хранилища.
  - `task_name`: название текущей задачи.
- Заголовок запроса должен содержать авторизационный токен из локального хранилища и тип контента `"Content-Type": "application/json"`.
- В случае успешного выполнения запроса, полученный ответ логируется в консоль, а также добавляется новая запись о посылке в массив `submissions` (содержащий информацию о последних 4 посылках).
- Если возникает ошибка при отправке решения на сервер, ошибка логируется в консоль.

7. Отображение результатов:
   - В разделе "Последняя посылка" отображается статус решения из свойства `solutionStatus`.
   - В разделе "Ошибка на тесте" отображается ошибка на тесте из свойства `failureTest`.
   - В таблице "Последние 4 посылки" отображается информация о последних 4 посылках из массива `submissions`.

3.1.2. Требования к взаимодействию клиентской и серверной частей

Клиентская часть (веб-приложение) взаимодействует с сервером по протоколу HTTPS с использованием GET и POST запросов. Из веб-приложения отправляются следующие запросы:

Процесс авторизации (Отправка на сервер данных из формы авторизации):

- POST Request: `http://37.252.0.155:8080/api/auth/login`

```
headers: {
  'Content-Type': 'application/json'
}
```

Получение информации о преподавателе:

GET Request: http://37.252.0.155:8080/api/profile/get-teacher-info

```
headers: {
  Authorization: `Bearer ${token}`
}
```

Получение информации о студенте:

GET Request: http://37.252.0.155:8080/api/profile/get-student-info

```
headers: {
  Authorization: `Bearer ${token}`
}
```

Получение списка всех посылок:

GET Request: http://37.252.0.155:8080/api/solutions/get-all

```
headers: {
  Authorization: `Bearer ${token}`
}
```

Получение списка всех посылок:

GET Request: http://37.252.0.155:8080/api/solutions/get-all-user

```
headers: {
  Authorization: `Bearer ${token}`
}
```

Получение списка всех контестов:

GET Request: http://37.252.0.155:8080/api/contest/get-all

```
headers: {
  Authorization: `Bearer ${token}`
}
```

Авторизация в ejudge через puppeteer:

POST Request: http://37.252.0.155:3000/auth

```
{
  login: "ejudge",
  password: "ejudge",
  contestID: this.contestId,
},
{
  headers: {
    "Content-Type": "application/json",
  },
```

Получение задач с Ejudge с помощью puppeteer:

GET Request: http://37.252.0.155:3000/parseTasks

Добавление контеста в базу данных:

POST Request: http://37.252.0.155:8080/api/contest/add

```
headers: {
  "Content-Type": "application/json",
  "Authorization": `Bearer ${localStorage.getItem('token')}`
}
```

Получение рейтинга по оценкам:

GET Request: http://37.252.0.155:8080/api/grade/by-marks

```
headers: {"Authorization": `Bearer ${token}`}
```

Получение рейтинга по задачам:

GET Request: `http://37.252.0.155:8080/api/grade/by-tasks`
```
headers: {"Authorization": `Bearer ${token}`}
```

Выгрузка рейтинга по оценкам:

GET Request: `http://37.252.0.155:8080/api/grade/by-marks/download-csv`
```
headers: {"Authorization": `Bearer ${token}`}
```

Выгрузка рейтинга по задачам:

GET Request: `http://37.252.0.155:8080/api/grade/by-tasks/download-csv`
```
headers: {"Authorization": `Bearer ${token}`}
```

Создание аккаунта студента:

GET Request: `http://37.252.0.155:8080/api/auth/register`

```
headers: {
  'Content-Type': 'application/json',
},
```

Обработка посылки через Ejudge используя puppeteer:

POST Request: `http://37.252.0.155:3000/handleSolution`
```
{
  solutionFileBase64: base64File,
  taskID: this.currentTask.probId,
  language: this.selectedLanguage,
});
```

Добавление посылки в базу данных:

POST Request: `http://37.252.0.155:8080/api/solutions/add`
```
{
  {
  code: code,
  language: this.languageName,
  status: this.solutionStatus,
  used_time: 0.2,
  used_memory: 2.8,
  error_test: this.failureTest,
  contest_name: localStorage.getItem('contestName'),
  task_name: this.taskTitle,
},
{
  headers: {
    Authorization: `Bearer ${localStorage.getItem("token")}`,
    "Content-Type": "application/json",
  },
}
```

Получение результата используя puppeteer:

GET Request: http://37.252.0.155:3000/getResult

3.1.3. Требования к серверу для парсинга Ejudge.

Техническое задание для разработки Puppeteer-API для парсинга сайта Ejudge

1. Цель проекта Разработка API для парсинга сайта Ejudge с использованием Puppeteer.
2. Общие требования 2.1. Использование Node.js и Puppeteer. 2.2. Реализация функций аутентификации, отправки решений, получения результатов и парсинга задач. 2.3. Ограничение скорости запросов для защиты от абьюза сервера. 2.4. Использование CORS и Express.js для обработки HTTP-запросов.
3. Функциональные требования 3.1. Аутентификация (auth)

- Принимает логин, пароль и номер контеста.
- Возвращает сообщение об успешной аутентификации или ошибке.

3.2. Обработка решений (handleSolution)

- Принимает base64 кодированный файл решения, номер задачи и язык программирования.
- Отправляет решение на сервер Ejudge.
- Возвращает сообщение об успешной обработке решения или ошибке.

3.3. Получение результатов (getResult)

- Возвращает статус решения и сообщение об ошибке (если есть) или сообщение об успешном получении результатов.

3.4. Парсинг задач (parseTasks)

- Возвращает список задач с информацией о каждой задаче (ID, ограничения по времени и памяти, заголовок, описание, примеры входных и выходных данных).
- В случае ошибки возвращает сообщение об ошибке.

4. Нестандартные условия 4.1. Защита от абьюза сервера (DDoS). На 5 минут максимум 500 запросов.
5. Тестирование 5.1. Проверить аутентификацию с корректными и некорректными данными. 5.2. Проверить отправку решения и получение результатов для разных задач и языков программирования. 5.3. Проверить парсинг задач и корректность полученной информации. 5.4. Протестировать работу API с ограничением скорости запросов.
6. Документация 6.1. Описание функций и параметров API. 6.2. Примеры использования API.
7. Безопасность 7.1. Убедиться, что конфиденциальные данные пользователей (логин и пароль) защищены и не могут быть перехвачены. 7.2. Обеспечить обработку ошибок, чтобы избежать падения сервера или открытия уязвимостей.
8. Доступность 8.1. Поддержка CORS для доступности API на разных доменах. 8.2. Обработка HTTP-запросов и ответов должна быть стабильной и надежной, чтобы поддерживать непрерывную работу сервиса.
9. Обслуживание и поддержка 9.1. Предусмотреть возможность легкого масштабирования и обновления API. 9.2. Документация должна содержать все необходимые инструкции для установки, настройки и использования API. 9.3. Осуществлять регулярное обновление и поддержку кода в соответствии с изменениями на сайте Ejudge и обновлениями используемых технологий.

10. План разработки

10.1. Определение требований и планирование.

10.2. Разработка функциональности и написание кода.

10.3. Тестирование и отладка.

10.4. Подготовка документации.

10.5. Запуск и поддержка.

3.1.4. Требования к входным данным

Входные данные от пользователя поступают в виде нажатий на кнопки, переходов по ссылкам, заполнения текстовых полей, меню выбора времени, слайдеров и чекбоксов. Описание вводимых данных представлено в проекте дизайна на платформе Figma, ссылка на который указана в разделе 4.1.4.

Все входные данные отправляются на сервер в формате json.

3.1.5. Требования к выходным данным

При первом запуске пользователю после загрузки отображается экран авторизации, экран авторизации должен включать в себя текстовые поля для ввода логина и пароля, кнопку авторизации. Возле текстовых полей при некорректном вводе должна отображаться ошибка

После успешной авторизации у пользователя открывается личный кабинет учащегося.

Экран личного кабинета учащегося должен включать в себя надписи с указанием имени, фамилии, отчества, ID пользователя, номер и буква класса, направление, номер группы, количество решённых задач, рейтинг по задачам и рейтинг по оценкам. В правом верхнем углу находится фиктивная фотография профиля (содержит инициалы). Также из учётной записи должна быть возможность выйти с помощью выпадающего меню в верхней части сайта в блоке с именем и фотографией профиля (на которой отображаются инициалы).

Примерный состав реализуемых функций и расположения элементов представлен в виде проекта на платформе Figma.
https://www.figma.com/file/ZL37p1tu12ka5hM0BP0xQB/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0?node-id=0%3A1&t=YIWjZwkeNxVJLXZ9-1

3.2. Требования к интерфейсу

3.2.1. **Интуитивность и понятность**: Интерфейс должен быть понятным и интуитивно-ясным для пользователя. Элементы управления и их функции должны быть очевидными, а иконки и символы должны соответствовать общепринятым стандартам и не вызывать путаницы.

3.2.2. **Консистентность и стандарты**: Все элементы интерфейса, включая кнопки, меню, формы, иконки и цвета, должны быть консистентными на протяжении всего приложения. Это означает, что они должны выглядеть и функционировать одинаково во всех разделах системы.

3.2.3. **Обратная связь**: Система должна предоставлять обратную связь пользователю о действиях, которые он выполняет. Это может включать в себя сообщения об успехе или ошибках, индикаторы загрузки и другие виды уведомлений.

3.2.4. **Доступность**: Интерфейс должен быть доступным для всех пользователей, включая тех, кто имеет проблемы со зрением, слухом или другие ограничения.

3.2.5. **Эстетический дизайн**: Интерфейс должен быть эстетически приятным и современным.

3.2.6. **Отзывчивость**: Интерфейс должен быть отзывчивым, то есть корректно отображаться и функционировать на различных устройствах и разрешениях экрана.

3.2.7. **Навигация**: Навигация по системе должна быть простой и понятной. Пользователь должен легко найти нужную ему информацию или функцию.

3.2.8. **Минимум действий пользователя**: Интерфейс должен быть разработан так, чтобы пользователю требовалось минимальное количество действий для выполнения задачи.

3.2.9. **Языковая поддержка**: Интерфейс должен поддерживать необходимые языки для целевой аудитории системы.

3.2.10. **Безопасность**: Интерфейс должен быть разработан с учетом безопасности, защищая данные пользователя и предотвращая нежелательные действия.

3.3. Требования к надёжности

3.3.1. Обеспечение устойчивого функционирования

Система должна обеспечивать непрерывную работу в течение всего периода проведения соревнований, не допускать падения или зависания во время работы с системой.

3.3.2. Контроль входной информации

Система должна проверять входные данные и уведомлять пользователя об ошибках, если введены некорректные данные.

3.3.3. Контроль выходной информации

Система должна генерировать корректные результаты, проверять их и уведомлять пользователя об ошибках, если необходимо.

3.3.4. Время восстановления после отказа

Система должна иметь возможность восстановления в случае сбоев или отказов в работе в кратчайшие сроки (не более 5 минут).

4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

4.1. Состав программной документации

«Реализация «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга». Техническое задание (ГОСТ 19.201-78);

«Реализация «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга». Программа и методика испытаний (ГОСТ 19.301-78);

«Реализация «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга». Пояснительная записка (ГОСТ 19.404-79);

«Реализация «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга». Руководство оператора (ГОСТ 19.505-79);

«Реализация «Система проверки решений по программированию для Лицея НИУ ВШЭ с автоматизированным подсчётом рейтинга». Текст программы (ГОСТ 19.401-78);

4.2. Специальные требования к программной документации

Документы к программе должны быть выполнены в соответствии с ГОСТ 19.106-78 и ГОСТами к каждому виду документа (см. п. 5.1.).

Пояснительная записка должна быть загружена в систему Антиплагиат через LMS «НИУ ВШЭ».

Техническое задание и пояснительная записка, титульные листы других документов должны быть подписаны руководителем разработки и исполнителем.

Документация и программа сдаётся в электронном виде в формате .pdf или .docx. в архиве формата .zip или .rar.

За три дня до защиты комиссии все материалы курсового проекта: программная документация, программный проект, исполняемый файл, отзыв руководителя, отчёт системы Антиплагиат должны быть загружены одним или несколькими архивами в проект дисциплины «Курсовой проект» в личном кабинете в информационной образовательной среде SmartLMS НИУ ВШЭ.

5. СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ

5.1. Технические средства, используемые во время испытаний

Для выполнения тестов использовался производительный компьютер со следующими характеристиками:

Operating System: Windows 11 Pro 64-bit (10.0, Build 22621) (22621.ni_release.220506-1250)

Processor: Intel(R) Core (TM) i9-9900K CPU @ 3.60GHz (16 CPUs), ~3.6GHz

Memory: 32768MB RAM

Available OS Memory: 32698MB RAM

Page File: 14453MB used, 20293MB available

Card name: NVIDIA GeForce RTX 2070

Manufacturer: NVIDIA

Chip type: NVIDIA GeForce RTX 2070

5.2. Программные средства, используемые во время испытаний

5.2.1. **Apache JMeter**: Это открытое программное обеспечение для тестирования нагрузки и производительности, которое можно использовать для анализа и измерения производительности приложения.

5.2.2. **Puppeteer**: Это библиотека Node.js, которую я использовал для автоматизации действий в браузере Chrome или Chromium. С помощью Puppeteer я мог эмулировать действия реального пользователя и следить за реакцией скрипта.

5.2.3. **Node.js и npm**: Node.js - это среда выполнения JavaScript, которая позволяет запускать JavaScript-код на сервере. Npm (Node Package Manager) - это пакетный менеджер Node.js, который я использовал для управления зависимостями проекта.

5.2.4. **JetBrains WebStorm**: Это мощная среда разработки, которую я использовал для написания и отладки кода.

5.2.5. **Git и GitHub**: Я использовал Git для управления версиями кода и GitHub для хранения и совместной работы над кодом.

5.2.6. **Postman**: Приложение для тестирования API, которое я использовал для отправки запросов на сервер и проверки ответов.

5.2.7. **Express.js**: Это фреймворк для веб-приложений Node.js, который я использовал для создания HTTP-запросов к серверу.

5.2.8. **Multer**: Это middleware Node.js для обработки multipart/form-data, который используется для загрузки файлов.

5.3. Порядок проведения испытаний

(текст)

6. МЕТОДЫ ИСПЫТАНИЙ

6.1. Испытание выполнения требований к программной документации

6.1.1. **Проверка полноты и точности описаний:** Осуществлена оценка полноты и точности описания функциональности проекта в программной документации. Убедительно подтверждено, что все функции, классы, методы и переменные имеют уместные и конкретные описания. Кроме того, проведена проверка на наличие примеров использования для функций и методов повышенной сложности.

6.1.2. **Оценка структуры и организации:** Осуществлена оценка структуры и организации программной документации. Установлено, что все разделы и подразделы организованы логично и обеспечивают удобную навигацию, способствующую эффективному поиску информации.

6.1.3. **Проверка соответствия стандартам:** Осуществлена проверка соответствия программной документации установленным стандартам. Убедительно подтверждено, что используются правильные форматы и шаблоны, а также соблюдаются соглашения об именовании и стилистические правила.

6.1.4. **Оценка актуальности:** Осуществлена оценка актуальности программной документации. Установлено, что все описания соответствуют текущему состоянию кода и все изменения и обновления кода своевременно отражены в документации.

Все вышеуказанные проверки позволили удостовериться в том, что программная документация полностью соответствует всем предъявляемым требованиям и может быть эффективно использована как разработчиками, так и конечными пользователями.

6.2. Испытание требований к интерфейсу

6.2.1. **Тестирование удобства использования (Usability Testing):** Проведено тестирование использования, чтобы убедиться, что все элементы интерфейса интуитивно понятны и легко доступны для пользователей. Это включает в себя проверку всех кнопок, ссылок, меню и других элементов управления на всех страницах.

6.2.2. **Тестирование отзывчивости (Responsiveness Testing):** Проведено тестирование на отзывчивость интерфейса, меняя размер окна браузера и проверяя сайт на различных разрешениях экрана.

6.2.3. **Тестирование производительности (Performance Testing):** Проведено тестирование на производительность, чтобы убедиться, что интерфейс загружается и работает быстро даже при высокой нагрузке.

6.2.4. **Тестирование доступности (Accessibility Testing):** Проведено тестирование доступности, чтобы убедиться, что интерфейс доступен для людей с различными ограничениями, включая тех, кто использует технологии помощи, такие как экранные дикторы.

6.3. Испытание выполнения требований к функциональным характеристикам

6.3.1 Тестирование страницы “LoginPage.vue”

6.3.2.1 **Тестирование пользовательского интерфейса:**

- Проверка отображения всех элементов интерфейса (логотип, поля ввода, кнопка входа, ссылка для связи с преподавателем и т.д.).
- Проверка корректности работы ссылок и кнопок.
- Проверка соответствия стилей дизайну.

6.3.2.2 **Тестирование функциональности:**

- Проверка ввода данных в поля "Электронная почта" и "Пароль".
  - Ввод правильных данных.
  - Ввод неправильных данных (неправильный формат электронной почты, неправильный пароль).
  - Ввод данных с пробелами в начале и конце.
  - Попытка отправки формы с пустыми полями.
- Проверка работы кнопки "Войти".
  - Кнопка должна быть неактивной, пока все поля не заполнены.
  - При нажатии на кнопку происходит авторизация пользователя.
- Проверка отображения сообщений об ошибках.
  - При вводе неправильных данных.
  - При проблемах с сетью или сервером.
- Проверка корректного перехода на другие страницы после успешной авторизации.

6.3.2.3 **Тестирование производительности и безопасности:**

- Тестирование времени ответа сервера при авторизации.
- Проверка на утечку информации при передаче данных на сервер.
- Проверка на возможность SQL-инъекций или XSS-атак через поля ввода.

6.3.2.4 **Примеры действий при тестировании:**

- Вводим в поле "Электронная почта" `test@edu.hse.ru` и в поле "Пароль" `123456`, затем нажимаем на кнопку "Войти". Результат: пользователь успешно авторизован, произошел переход на соответствующую страницу профиля.
- Вводим в поле "Электронная почта" `test@edu.hse.ru` и в поле "Пароль" неправильный пароль `abcdef`, затем нажимаем на кнопку "Войти". Результат: отображается сообщение об ошибке "Неверный адрес электронной почты или пароль".
- Оставляем поля "Электронная почта" и "Пароль" пустыми, затем нажимаем на кнопку "Войти". Результат: кнопка "Войти" остается неактивной, пока поля не заполнены.
- Вводим в поле "Электронная почта" некорректный адрес `test@edu`, затем пытаемся нажать на кнопку "Войти". Результат: отображается сообщение об ошибке, указывающее на некорректный формат электронной почты.
- Проверяем, что при недоступности сервера или проблемах с сетью отображается соответствующее сообщение об ошибке.

- Проверяем, что при успешной авторизации происходит корректное сохранение токена в localStorage и корректная навигация на страницу профиля.
- Проверяем корректность работы функции `fetchUserRole` и наличие правильных данных в localStorage после успешной авторизации пользователя.
- Проверяем, что при вводе SQL-инъекций или XSS-атак через поля ввода система не позволяет такие действия.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.2 Тестирование страницы “SiteHeader.vue”

6.3.2.1 **Функциональность загрузки профиля:** Проведены испытания для проверки корректной работы функций fetchTeacherInfo и fetchStudentInfo. Было установлено, что данные функции корректно выполняют свои задачи при наличии соответствующих ролей у пользователя. Проверка работы этих функций включала в себя анализ обработки ошибок и корректности заполнения данных пользователя.

6.3.2.2 **Функциональность навигационной панели:** Было проведено тестирование функций навигации, включая проверку работы переходов по ссылкам и их соответствия роли пользователя. Было подтверждено, что все ссылки функционируют должным образом и перенаправляют пользователя на соответствующие страницы.

6.3.2.3 **Функциональность кнопки "Выход":** Была проверена работоспособность кнопки "Выход". Подтверждено, что при нажатии на кнопку выполняется функция handleLogout и происходит корректный выход пользователя из системы.

6.3.2.4 **Визуальная проверка:** Было проведено тестирование визуальных элементов страницы, включая верстку и стилизацию. Убедительно подтверждено, что все элементы выглядят и функционируют в соответствии с заданными стилями, включая изменение цветов при наведении и нажатии.

6.3.2.5 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функций fetchTeacherInfo и fetchStudentInfo, они корректно логируются в консоли.

6.3.2.6 **Проверка аутентификации:** Было проведено тестирование функции checkAuth для подтверждения корректности аутентификации пользователя перед доступом к странице.

Все тесты были проведены успешно и не выявили критических ошибок.

6.3.3 Тестирование страницы “StudentProfile.vue”

6.3.3.1 **Функциональность загрузки профиля студента:** Испытания проводились для проверки корректной работы функции getStudentInfo. Было установлено, что данная функция корректно выполняет свои задачи при наличии валидного токена. Проверка работы этой функции включала в себя анализ обработки ошибок и корректности заполнения данных студента.

6.3.3.2 **Визуализация данных профиля:** Проведено тестирование отображения данных профиля студента на странице. Было установлено, что данные пользователя корректно отображаются в соответствующих полях страницы.

6.3.3.3 **Тестирование визуальных элементов:** Проведено тестирование визуальных элементов страницы, включая верстку и стилизацию. Убедительно подтверждено, что все элементы выглядят и функционируют в соответствии с заданными стилями.

6.3.3.4 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице профиля студента.

6.3.3.5 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функции getStudentInfo, они корректно логируются в консоли с соответствующим сообщением об ошибке.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.4 Тестирование страницы “AdminPanel.vue”

6.3.4.1 **Функциональность загрузки информации о преподавателе:** Испытания проводились для проверки корректной работы функции fetchTeacherInfo. Было установлено, что данная функция корректно выполняет свои задачи при наличии валидного токена. Проверка работы этой функции включала в себя анализ обработки ошибок и корректности заполнения данных преподавателя.

6.3.4.2 **Визуализация данных преподавателя:** Проведено тестирование отображения данных преподавателя на странице. Было установлено, что данные преподавателя корректно отображаются в соответствующих полях страницы.

6.3.4.3 **Тестирование визуальных элементов:** Проведено тестирование визуальных элементов страницы, включая верстку и стилизацию. Убедительно подтверждено, что все элементы выглядят и функционируют в соответствии с заданными стилями.

6.3.4.4 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице администратора.

6.3.4.5 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функции fetchTeacherInfo, они корректно логируются в консоли с соответствующим сообщением об ошибке.

6.3.4.6 **Тестирование навигационных ссылок:** Проведено тестирование функционала навигационных ссылок "Создать аккаунт студента" и "Импорт контеста из Ejudge". Было установлено, что при нажатии на эти ссылки осуществляется корректный переход на соответствующие страницы.

6.3.4.7 **Тестирование стилей навигационных кнопок:** Проведено тестирование стилей навигационных кнопок. Подтверждено, что кнопки корректно отображаются и реагируют на взаимодействие с пользователем (наведение мыши).

6.3.4.8 **Тестирование заголовка админ-панели:** Проведено тестирование заголовка страницы. Установлено, что заголовок корректно отображается и соответствует заданному стилю.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.5 Тестирование страницы “CreateStudentAccount.vue”

6.3.5.1 **Функция регистрации:** Функция register была проверена на корректность работы. Она корректно собирает данные из полей формы и отправляет их на

сервер. Проверка включала в себя тестирование корректности обработки ответа от сервера и обработку возможных ошибок.

6.3.5.2 **Тестирование формы регистрации:** Было проверено, что все поля формы корректно связаны с переменными в компоненте и корректно обрабатывают пользовательский ввод.

6.3.5.3 **Тестирование обязательности полей формы:** Было проверено, что все поля формы помечены как обязательные и форма не отправляется без заполнения этих полей.

6.3.5.4 **Тестирование стилей формы:** Стили формы были тщательно проверены для обеспечения соответствия макету. Включая стили для состояния фокуса на полях ввода и стили кнопки отправки формы.

6.3.5.5 **Тестирование функциональности кнопки отправки формы:** Проверено, что при нажатии на кнопку "Сохранить" вызывается функция register и форма отправляется.

6.3.5.6 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице создания учетной записи студента.

6.3.5.7 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функции register, они корректно логируются в консоли с соответствующим сообщением об ошибке.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.6 Тестирование страницы “ImportContest.vue”

6.3.6.1 **Импорт контеста:** Функция importContest была проверена на корректность работы. Она должна корректно выполнять авторизацию, запрашивать и обрабатывать задачи, а затем отправлять данные о контесте на сервер. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.6.2 **Отправка контеста на сервер:** Функция sendContestToServer была проверена на корректность работы. Она должна преобразовывать данные задач в нужный формат и отправлять данные о контесте на сервер. Проверка включала тестирование обработки ответа от сервера и обработку возможных ошибок.

6.3.6.3 **Тестирование формы импорта контеста:** Было проверено, что все поля формы корректно связаны с переменными в компоненте и корректно обрабатывают пользовательский ввод. Проверка включала тестирование обработки пользовательского ввода для каждого поля формы.

6.3.6.4 **Тестирование обязательности полей формы:** Было проверено, что все поля формы помечены как обязательные и форма не отправляется без заполнения этих полей.

6.3.6.5 **Тестирование стилей формы:** Стили формы были тщательно проверены для обеспечения соответствия макету. Включая стили для состояния фокуса на полях ввода и стили кнопки отправки формы.

6.3.6.6 **Тестирование функциональности кнопки отправки формы:** Проверено, что при нажатии на кнопку "Импортировать" вызывается функция importContest и форма отправляется.

6.3.6.7 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице импорта контеста.

6.3.6.8 **Тестирование отображения статуса обработки и добавления контеста:** Было проверено, что сообщения "Обработка..." и "Контест успешно добавлен!" корректно отображаются в зависимости от состояния обработки и добавления контеста.

6.3.6.9 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функций importContest и sendContestToServer, они корректно логируются в консоли с соответствующим сообщением об ошибке.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.7 Тестирование страницы “MyContestsList.vue”

6.3.7.1 **Получение списка контестов:** Функция fetchContests была проверена на корректность работы. Она должна корректно отправлять запрос GET на сервер для получения списка контестов и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.7.2 **Сохранение ID контеста и названия:** Функция saveContestId была проверена на корректность работы. Она должна корректно сохранять ID контеста и название в localStorage. Тестирование включало проверку сохранения и чтения данных из localStorage.

6.3.7.3 **Отображение списка контестов:** Было проверено, что список контестов корректно отображается на странице. Это включало проверку отображения каждого элемента списка, включая ID в Ejudge, название контеста, дату начала и окончания, а также ссылку на контест.

6.3.7.4 **Переход по ссылке на контест:** Было проверено, что при нажатии на кнопку "Перейти" осуществляется переход на страницу контеста с соответствующим ID.

6.3.7.5 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице списка контестов.

6.3.7.6 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функции fetchContests, они корректно логируются в консоли с соответствующим сообщением об ошибке.

6.3.7.7 **Проверка стилей:** Было проведено тестирование стилей кнопки и таблицы для соответствия макету.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.8 Тестирование страницы “Contest.vue”

6.3.8.1 **Получение списка задач:** Функция `fetchTasks` была проверена на корректность работы. Она должна корректно отправлять запрос GET на сервер для получения списка задач и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.8.2 **Аутентификация:** Функция `authenticate` была проверена на корректность работы. Она должна корректно отправлять запрос POST на сервер для

аутентификации и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.8.3 **Отправка решения:** Функция `submitSolution` была проверена на корректность работы. Она должна корректно отправлять решение на сервер и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.8.4 **Получение результата решения:** Функция `fetchResult` была проверена на корректность работы. Она должна корректно отправлять запрос GET на сервер для получения результата решения и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.8.5 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице контеста.

6.3.8.6 **Отображение списка задач:** Было проверено, что список задач корректно отображается на странице. Это включало проверку отображения каждого элемента списка, включая ID задачи, название задачи, описание и примеры ввода/вывода.

6.3.8.7 **Отображение результатов решения:** Было проверено, что результаты решения корректно отображаются на странице. Это включало проверку отображения статуса решения, ошибки на тесте и последних посылок.

6.3.8.8 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функций `fetchTasks`, `authenticate`, `submitSolution` и `fetchResult`, они корректно логируются в консоли с соответствующим сообщением об ошибке.

6.3.8.9 **Проверка стилей:** Было проведено тестирование стилей кнопок, таблиц и других элементов для соответствия макету.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.9 Тестирование страницы “ParcelsList.vue”

6.3.9.1 **Получение списка посылок:** Функция `fetchParcels` была проверена на корректность работы. Она должна корректно отправлять запрос GET на сервер для получения списка посылок и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.9.2 **Отображение списка посылок:** Было проверено, что список посылок корректно отображается на странице. Это включало проверку отображения каждого элемента списка, включая ID посылки, код, язык, статус, время работы, использованную память, тест, название контеста и название задачи.

6.3.9.3 **Открытие и закрытие модального окна с кодом посылки:** Были проверены функции `openCodeModal` и `closeModal`. Они должны корректно открывать и закрывать модальное окно с кодом посылки. Тестирование включало проверку работы модального окна и отображения кода посылки в нем.

6.3.9.4 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице списка посылок.

6.3.9.5 **Проверка обработки ошибок:** Было установлено, что в случае возникновения ошибок при выполнении функции `fetchParcels`, они корректно логируются в консоли с соответствующим сообщением об ошибке.

6.3.9.6 **Проверка стилей:** Было проведено тестирование стилей кнопок, таблиц, модального окна и других элементов для соответствия макету.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.3.10 Тестирование страницы “GlobalRating.vue”

6.3.10.1 **Получение данных рейтинга:** Функция `getRatingData` была проверена на корректность работы. Она должна корректно отправлять запрос GET на сервер для получения данных рейтинга и обрабатывать ответ. Тестирование включало проверку обработки ответов от сервера и обработку возможных ошибок.

6.3.10.2 **Смена типа рейтинга:** Была проверена функция `changeRating`. Она должна корректно изменять тип рейтинга и вызывать функцию `getRatingData` для получения соответствующих данных. Тестирование включало проверку смены типа рейтинга и обновления данных рейтинга.

6.3.10.3 **Скачивание данных рейтинга в формате CSV:** Была проверена функция `downloadCSV`. Она должна корректно отправлять запрос GET на сервер для получения данных рейтинга в формате CSV и обрабатывать ответ. Тестирование включало проверку корректного скачивания файла и обработку возможных ошибок.

6.3.10.4 **Отображение данных рейтинга:** Было проверено, что данные рейтинга корректно отображаются на странице. Это включало проверку отображения каждого элемента данных, включая имя, фамилию, отчество, email, роль, факультет и группу.

6.3.10.5 **Тестирование компонентов заголовка и подвала:** Было проведено тестирование компонентов SiteHeader и SiteFooter на предмет их корректного отображения и функционирования на странице.

6.3.10.6 **Проверка обработки ошибок**: Было установлено, что в случае возникновения ошибок при выполнении функций `getRatingData` и `downloadCSV`, они корректно логируются в консоли с соответствующим сообщением об ошибке и отображаются на странице.

6.3.10.7 **Проверка стилей:** Было проведено тестирование стилей кнопок, таблицы и других элементов для соответствия макету.

Все проведенные тесты успешно завершены и не выявили наличие критических ошибок.

6.4. Испытание выполнения требований к надёжности

6.4.1. Обеспечение устойчивого функционирования

Я провёл стресс-тестирование и нагрузочное тестирование, используя инструменты, такие как Apache JMeter. Я создал большое количество параллельных пользовательских сессий и проверил, как система справляется с этим. Также я проверил, как система ведёт себя при внезапном отключении и последующем включении сервера. В результате система показала стабильную работу даже при больших нагрузках.

6.4.2. Контроль входной информации

Для тестирования контроля входной информации я провёл функциональное тестирование и тестирование на корректность вводимых данных. На странице авторизации я проверил реакцию системы на ввод некорректных данных, таких как неправильный адрес электронной почты или пароль. Я также проверил, как система реагирует на попытку ввода вредоносных строк (SQL-инъекции, XSS-атаки и т.д.). Система корректно уведомляла пользователя об ошибках и предотвращала ввод некорректных данных.

6.4.3. Контроль выходной информации

Я провёл тестирование на основе контракта, чтобы проверить, что система возвращает ожидаемые данные и отображает их корректно. После успешной авторизации я убедился, что производится редирект на соответствующую страницу профиля, и что информация о пользователе корректно сохраняется в localStorage.

6.4.4. Время восстановления после отказа

Я имитировал различные сбои в работе системы и замерял время, за которое система восстанавливается. Это включало внезапное отключение и включение сервера, прерывание сетевого соединения, а также симуляцию отказа в работе базы данных. Я также проверил, что при восстановлении после сбоя не теряются данные пользователей и состояние системы восстанавливается корректно. Во всех случаях время восстановления не превышало 5 минут, что соответствует требованиям к системе.

RU.17701729.05.10-01 51 01–1

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.10-01				
Инв. № подл.	Подп. и	Взам. инв.	Инв. №	Подп. и