МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет *компьютерных наук*

Кафедра *программирования и информационных технологий*

Автогенератор интерфейса веб-форм по json файлу для фреймворка Vue

Курсовая работа

09.03.02 *Информационные системы и технологии*

Программная инженерия в информационных системах

Допущен к защите

Зав. кафедрой _____*С.Д. Махортов, д.ф.- м.н.* __.__.20__

Обучающийся _____*М.Д. Шеменев, 3 курс, д/о*

Руководитель _____*А.И. Чекмарев, ст. преподаватель*

Воронеж 2023

Содержание

Введение

В современном мире веб-разработки наблюдается стремительное развитие фреймворков и библиотек, которые позволяют разрабатывать высокопроизводительные и масштабируемые приложения. Одним из таких фреймворков является Spring Boot, который предоставляет возможность быстрого и простого создания веб-приложений на языке Java.

Одним из важных аспектов разработки веб-приложений является создание форм для ввода данных. В этом контексте особенно важно обеспечить правильный и удобный пользовательский интерфейс для ввода данных. Для этого часто используется фреймворк Vue.js, который является популярным на сегодняшний день.

В данной работе рассматривается вопрос автоматической генерации форм на основе JSON Schema в Spring Boot REST приложениях с использованием библиотеки FormSchema Native. Такой подход позволяет сократить время на создание форм и обеспечить их согласованность с описанными в JSON Schema требованиями. Создание удобных пользовательских интерфейсов для веб-приложений является неотъемлемой частью разработки программного обеспечения. Одним из способов облегчить этот процесс является использование JSON Schema для автоматической генерации форм.

FormSchema Native – это библиотека, написанная на Vue.js, которая позволяет генерировать пользовательские формы на основе json-схем. Эта библиотека предоставляет разработчикам удобный способ создания динамических форм, которые могут адаптироваться к различным типам данных и удовлетворять потребности самых разных проектов.

Цель данного курсового проекта - изучение основных концепций и принципов генерации форм на основе JSON Schema в Vue.js, а также разработка примеров использования библиотеки FormSchema Native. В рамках работы будет рассмотрен процесс создания JSON Schema, применение

различных типов полей и правил валидации, а также настройка пользовательского интерфейса формы.

В результате выполнения данной работы будет разработан программный модуль, который позволит автоматически генерировать формы для ввода данных на основе JSON Schema в Spring Boot REST приложениях с использованием библиотеки FormSchema Native. Этот модуль позволит разработчикам сократить время на создание форм и обеспечить согласованность форм с требованиями, описанными в JSON Schema.

Программный модуль, который будет разработан в рамках данной работы, может быть полезен при создании различных веб-приложений, в которых необходимо использовать формы для ввода данных. Также он может быть интересен как разработчикам, так и пользователям, так как позволяет ускорить процесс разработки и повысить качество пользовательского интерфейса.

Для успешной реализации данной работы необходимо провести тщательный анализ спецификации JSON Schema и изучить особенности работы библиотеки FormSchema Native. Кроме того, необходимо разработать архитектуру программного модуля, которая обеспечит эффективное использование данных технологий и позволит получить высокое качество генерируемых форм.

В целом, данная работа позволит получить новые знания и навыки в области разработки веб-приложений, а также продемонстрирует важность использования современных технологий для ускорения и упрощения процесса разработки.

1 Постановка задачи

1.1 Постановка задачи

Задача проекта заключается в разработке приложения, состоящего из серверной части на Spring Boot REST и клиентской части на Vue с использованием библиотеки FormSchema Native.

Для реализации данного проекта необходимо создать JSON Schema, которая будет генерироваться на серверной стороне Spring Boot REST. На основе этой схемы будут генерироваться формы, которые будут отображаться на клиентской стороне приложения с помощью библиотеки FormSchema Native.

Кроме того, в проекте необходимо реализовать логику обработки данных, введенных пользователем в формы, и их передачи на серверную сторону для дальнейшей обработки.

Окончательной целью проекта является создание удобного и функционального интерфейса для пользователей, который позволит им легко и быстро заполнять формы и отправлять данные на сервер для дальнейшей обработки.

2 Анализ предметной области

2.1 Терминология (глоссарий) предметной области

Веб-приложение – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.

HTTP – это протокол, позволяющий получать различные ресурсы, например, HTML-документы. Протокол HTTP лежит в основе обмена данными в Интернете.

REST – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. [1]

Backend – логика работы сайта, внутренняя часть продукта, которая находится на сервере и скрыта от пользователя. [2]

Frontend – презентационная часть информационной или программной системы, ее пользовательский интерфейс и связанные с ним компоненты. [3]

HTTPS – расширение протокола HTTP для поддержки шифрования в целях повышения безопасности.

Пользователь – человек, который использует приложение.

API – описание взаимодействия одной компьютерной программы с другой.

JSON (JavaScript Object Notation) – это легкий формат для обмена данными, основанный на синтаксисе объектов JavaScript. Он используется для сериализации и передачи структурированных данных по сети между различными приложениями и является популярным форматом для API-интерфейсов.

JSON Schema – это язык схемы, используемый для проверки и валидации структурированных данных в формате JSON.

Github – это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

2.2 Обзор аналогов

Библиотека FormSchema Native является удобным инструментом для создания динамических форм на основе JSON-схем. Она позволяет создавать формы с помощью простых JSON-схем, которые определяют структуру данных, а также правила валидации и форматирования. Однако, на рынке существует множество альтернативных библиотек, которые также обеспечивают создание форм на основе JSON-схем.

2.2.1 Ncform

Ncform – это Китайский проект, имеет 900 звёзд на github и развивается неспешно. Все комментарии в коде на китайском языке, что усложняет понимание. Имеет встроенный механизм локализации. Основная часть проекта покрыта Unit и функциональными тестами на cypress. В библиотеке используются проприетарные ключи, отступающие от оригинальной JSON Schema, но при этом не ломающие её.

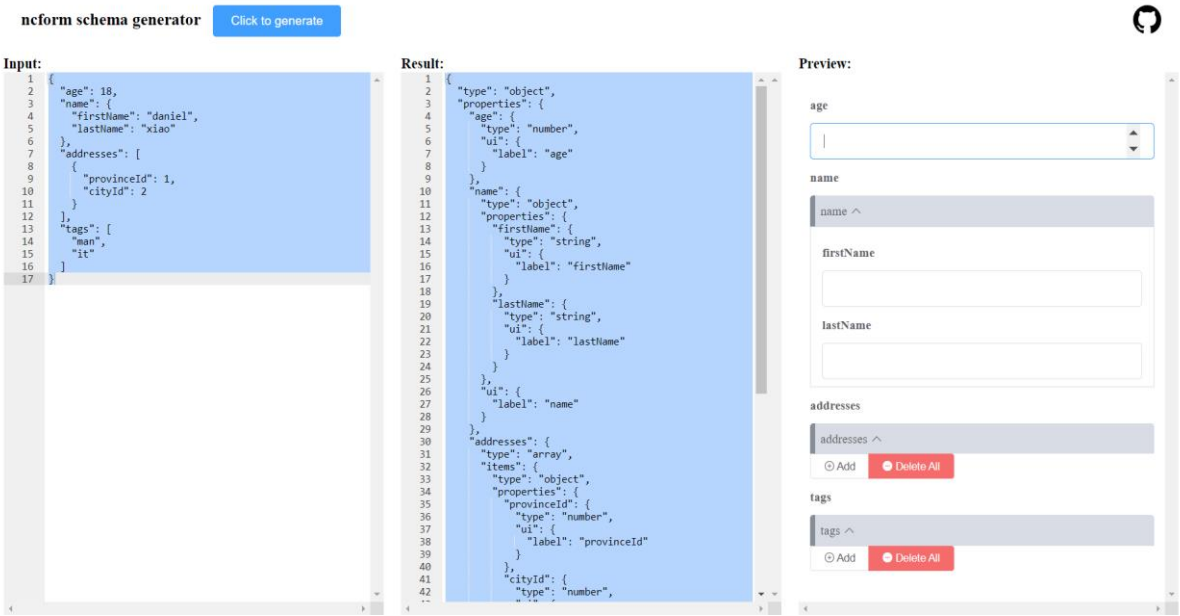

Рисунок 1 - Пример использования Ncform для управления формами

Недостатки:

— Нет возможности загрузки из JSON файла, как минимум валидаторы это объекты;

— Собственный формат схемы для создания формы, несовместим с оригинальной JSON-схемой;

— Не поддерживает пакет i18n и локализацию для валидаторов.

— Необходимо переопределять стили для ошибок валидаторов и объектов форм.

Преимущества:

— Встроенная поддержка локализации;

— Возможность создания собственных валидаторов в JSON-схеме через dx-выражения;

— Минимально необходимая поддержка Element UI.

2.2.2 JSON Forms

JSON Forms – это библиотека JavaScript, которая облегчает создание пользовательских интерфейсов на основе JSON Schema. Она позволяет быстро и легко создавать HTML-формы, которые могут быть связаны с JSON-данными.

JSON Forms работает следующим образом: вы определяете JSON Schema для данных, которые вы хотите использовать, а затем JSON Forms создает HTML-форму на основе этого схемы. Эта форма может быть сконфигурирована для отображения любого числа свойств и элементов управления для этих свойств. Например, если ваша схема JSON включает в себя поле "имя", то вы можете настроить JSON Forms для создания текстового поля для ввода имени.

JSON Forms предоставляет различные виды элементов управления, такие как текстовые поля, выпадающие списки, флажки, кнопки и т.д., которые могут быть связаны с соответствующими свойствами JSON. Кроме того, вы можете определить свои собственные виджеты для более точного управления отображением и поведением формы.

JSON Forms также поддерживает валидацию данных на основе JSON Schema и может выводить сообщения об ошибках, если пользователь вводит некорректные данные.

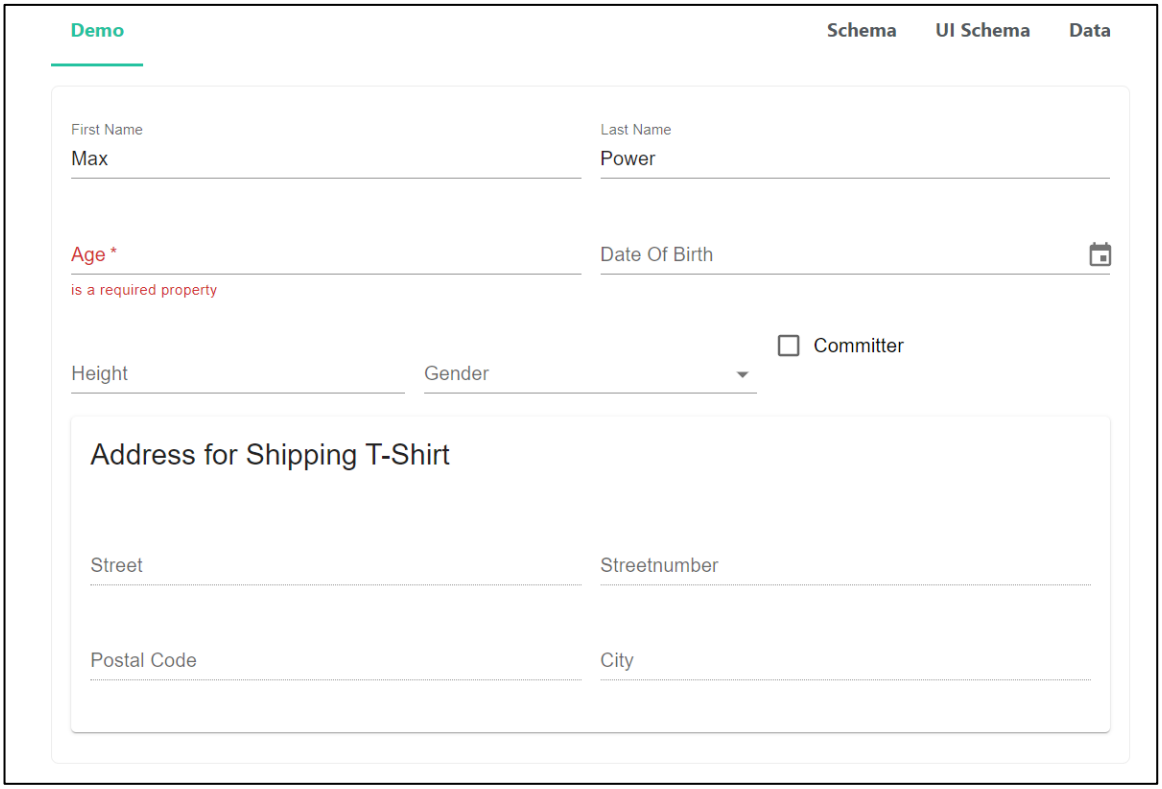

Рисунок 2 - Пример использования JSON Forms для управления формами

Недостатки:

— Сложность в освоении. JSON Forms может быть сложен в использовании для новичков в программировании, особенно если у них нет опыта работы с JSON и JSON Schema;

— Ограниченность в функциональности. Несмотря на то, что JSON Forms предоставляет множество опций для настройки интерфейса, она все же не может заменить полноценный фреймворк для разработки веб-приложений;

— Зависимость от JSON Schema. Если у вас нет готовой JSON Schema, вам придется создавать ее самостоятельно, что может быть трудоемким процессом;

— Ограничения в производительности. JSON Forms может замедлять производительность веб-приложения при работе с большими объемами данных или при использовании более сложных форм.

Преимущества:

— Быстрое и легкое создание интерфейсов на основе JSON данных. Это особенно полезно, если у вас есть большой объем данных, который нужно отобразить в удобном для пользователя виде;

— Возможность быстрой настройки формы, используя JSON Schema. Вы можете определить свойства данных, элементы управления и их связи в одном месте;

— Гибкость. JSON Forms предоставляет множество опций для настройки интерфейса, включая возможность создания пользовательских виджетов и настройку внешнего вида;

— Поддержка валидации данных. JSON Forms позволяет определить правила валидации данных в JSON Schema и автоматически проверять введенные данные на соответствие этим правилам;

— Открытый исходный код и активное сообщество разработчиков. Это означает, что вы можете найти множество ресурсов и поддержку в Интернете при работе с JSON Forms.

2.2.3 Angular JSON Schema Form

Angular JSON Schema Form (AngularJSF) – это библиотека для Angular, которая позволяет создавать динамические формы, основанные на JSON-схемах. Она предоставляет компоненты для отображения форм и связанных элементов управления на основе определений JSON-схем.

Это означает, что вы можете использовать JSON-схемы для определения структуры и содержания формы, включая типы полей, валидацию, текст подсказки и многие другие параметры. AngularJSF автоматически создаст форму на основе этих определений, что позволяет сократить время разработки и уменьшить вероятность ошибок.

Кроме того, AngularJSF позволяет настраивать стили формы и элементов управления с помощью CSS и собственных директив Angular. Она также имеет множество функций, таких как возможность создания вложенных форм, динамическое добавление и удаление полей, встроенная валидация и т.д.

AngularJSF является открытым исходным кодом и доступна в репозитории на GitHub. Она активно поддерживается сообществом разработчиков, что означает, что вы можете получить помощь и поддержку, если у вас возникнут проблемы при использовании этой библиотеки.

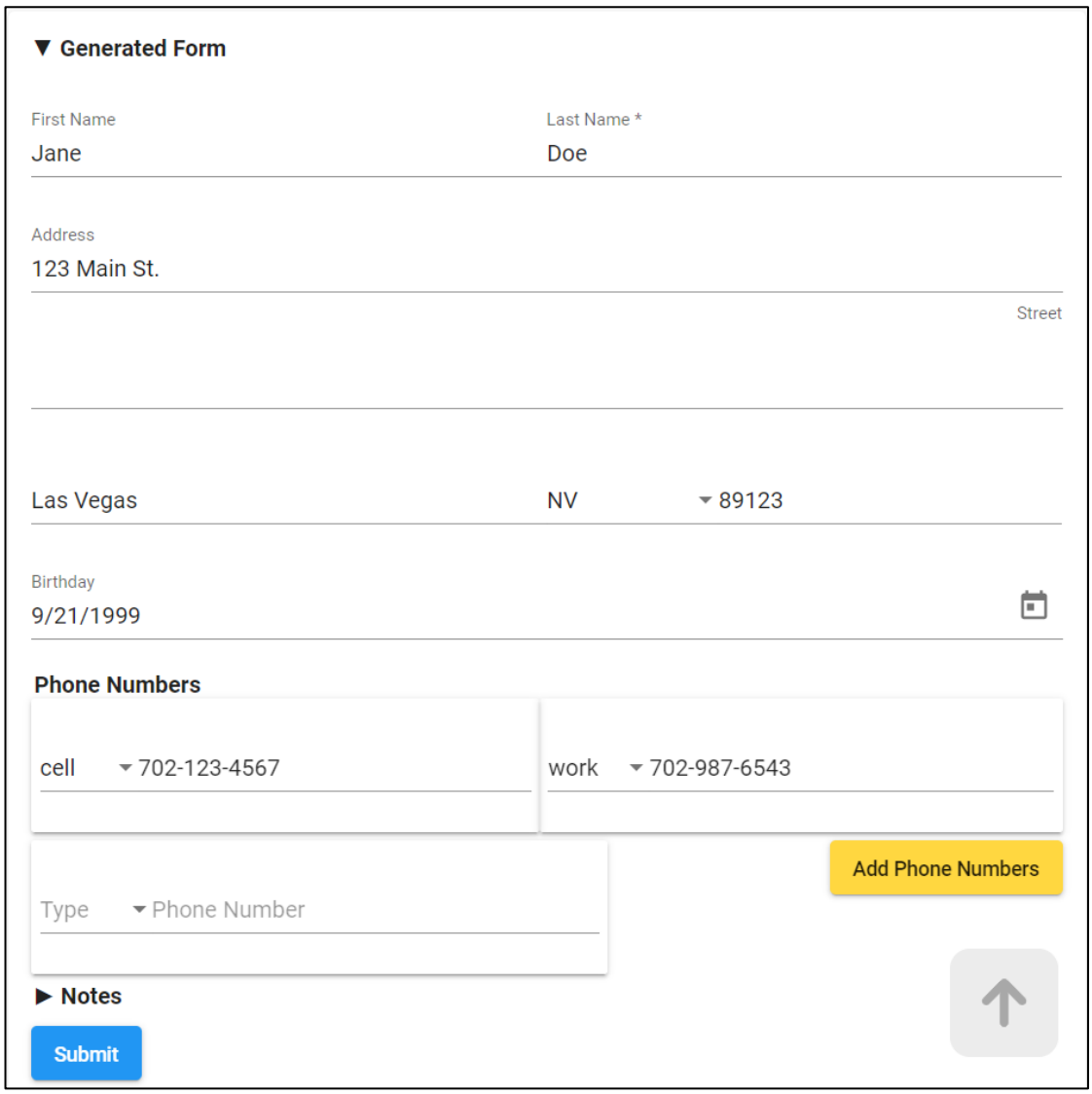

Рисунок 3 - Пример использования AngularJSF для управления формами

Недостатки:

— Ограниченный выбор элементов управления. AngularJSF предоставляет только ограниченный выбор элементов управления для форм, что может ограничивать возможности разработчика в некоторых случаях;

— Сложность для новичков. Некоторые функции и настройки в AngularJSF могут быть сложными для новичков в Angular и JSON-схемах;

— Ограничения JSON-схем. В некоторых случаях JSON-схемы могут оказаться недостаточно гибкими для определения формы, что может привести к ограничениям в функциональности формы.

Преимущества:

— Удобство использования. AngularJSF позволяет быстро создавать формы на основе JSON-схем, что сокращает время разработки и уменьшает вероятность ошибок;

— Расширяемость. Библиотека имеет множество функций и возможностей, таких как валидация, динамическое добавление и удаление полей, вложенные формы и многое другое. Кроме того, вы можете настроить внешний вид формы и элементов управления, используя CSS и собственные директивы Angular;

— Совместимость с Angular. AngularJSF разработана для использования с Angular, что обеспечивает легкую интеграцию и взаимодействие с другими компонентами приложения;

— Открытый исходный код. Библиотека распространяется под лицензией MIT и доступна в репозитории на GitHub, что позволяет разработчикам использовать и изменять ее бесплатно.

2.2.4 Form-create

Form-create – это Китайский проект с 2100 звёздочек на Github, но при этом есть минимальная документация на английском языке. Формы можно

конфигурировать через JSON, но для этого используется собственный формат, отличный от оригинальной JSON-схемы. Присутствует тема для Element UI.

Рисунок 4 - Пример использования Form-create для управления формами

Недостатки:

— Используется отличная от оригинальной JSON-схемы структура описания;

— Плохая локализация на английский, как в документации, так и создаваемом представлении, во многих местах отсутствует перевод с китайского языка.

Преимущества:

— Присутствует возможность расширения и сторонние компоненты, такие как: Text, JSON, Code и Markdown редакторы;

— Поддержка Element UI.

2.3 Диаграммы, иллюстрирующие работу системы

2.3.1 Диаграмма прецедентов

Диаграмма прецедентов позволяет визуализировать различные типы ролей в системе и то, как эти роли взаимодействуют с системой. [5]

На диаграмме, изображенной на Рисунке 5, присутствуют 1 актёр: пользователь.

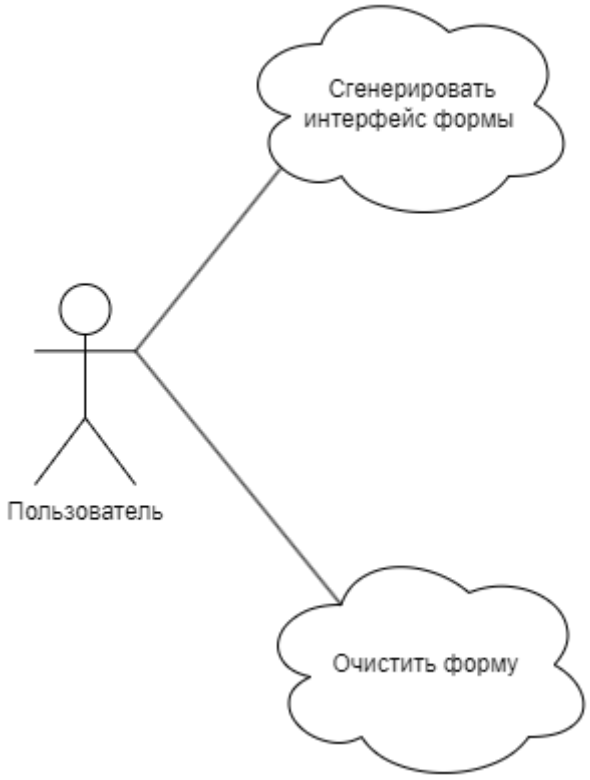

Рисунок 5 - Диаграмма прецедентов

Сценарии пользователя:

— Сгенерировать интерфейс формы;

— Очистить форму.

2.3.2 Диаграмма последовательности

Диаграмма последовательности иллюстрирует, как различные части системы взаимодействуют друг с другом для выполнения функций, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования. [5]

На Рисунке 6 представлена диаграмма последовательности для основного актера системы.

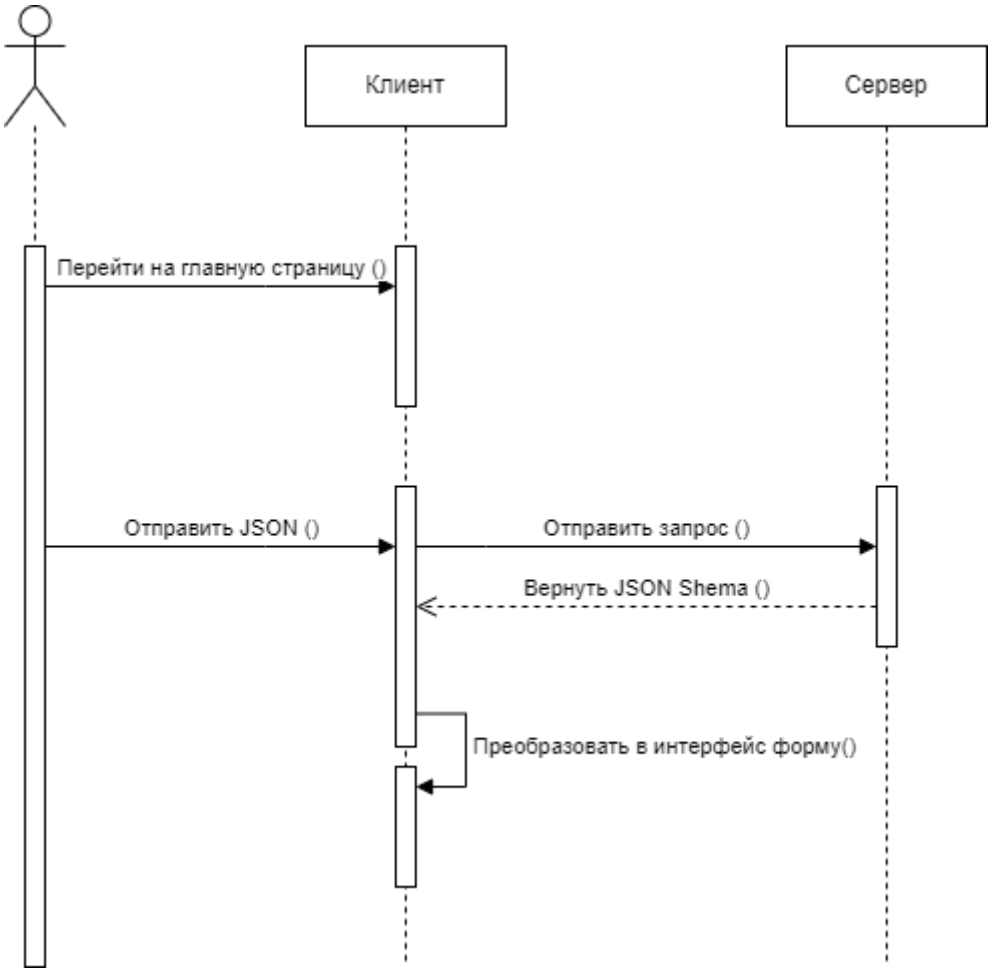

Рисунок 6 - Диаграмма последовательности для неавторизованного пользователя

2.3.3 Диаграмма развертывания

Диаграмма развертывания предназначена для представления общей конфигурации или топологии распределенной программной системы. [5]

На Рисунке 7 изображена топология разрабатываемой системы.

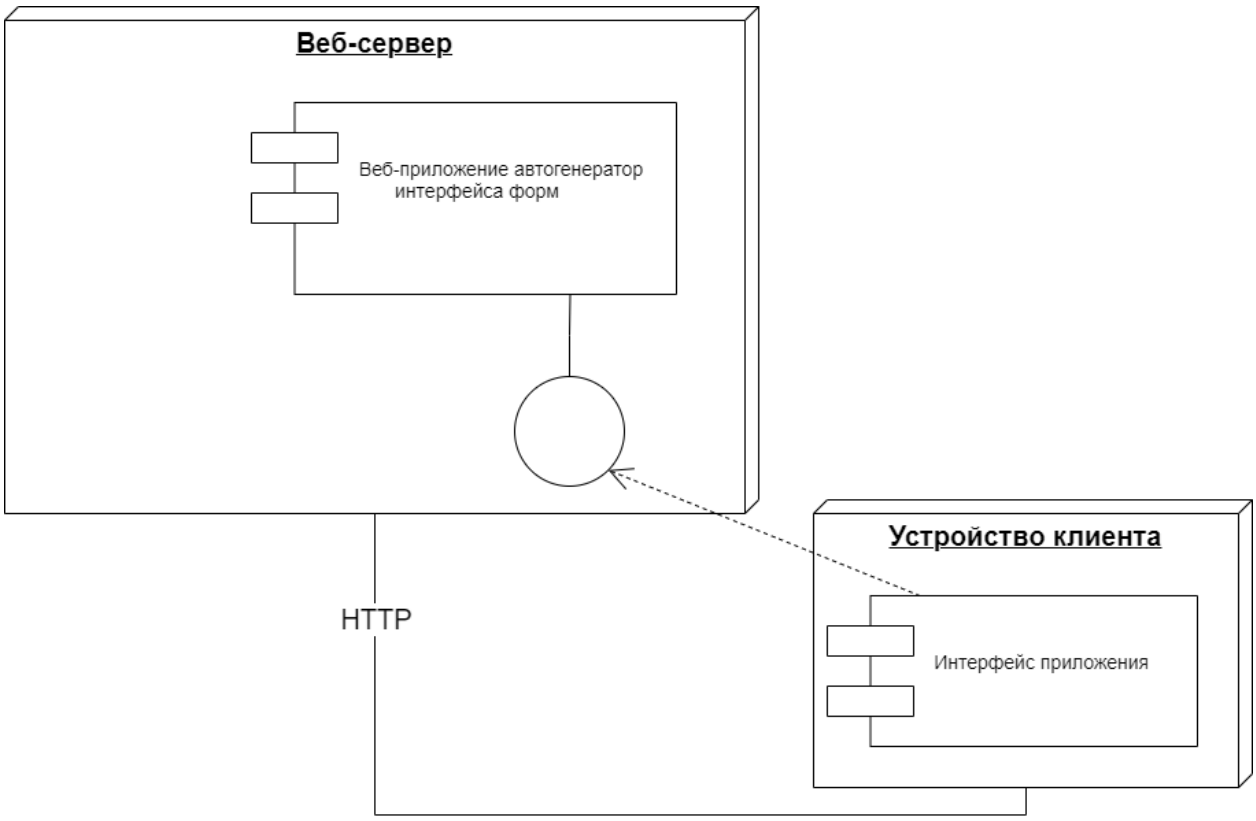

Рисунок 7 - Диаграмма развертывания

3 Реализация

3.1 Средства реализации

Для разработки приложения был выбран следующий стек технологий:

— Java – строго типизированный объектно-ориентированный язык программирования. Был выбран в качестве основного, так как он остается очень популярным языком программирования в этой области благодаря своим мощным возможностям и широкому спектру инструментов для разработки. К тому же существует огромное количество фреймворков и библиотек, написанных на Java, которые в перспективе можно легко интегрировать в проект;

— Spring Boot Framework – универсальный фреймворк с открытым исходным кодом для Java-платформы. Был выбран, так как он совместим с большим количеством библиотек и фреймворков, что позволяет использовать его в различных проектах и на различных платформах. Так же он позволяет разработчикам быстро создавать приложения без необходимости тратить много времени на конфигурацию; [6]

— HTML – язык разметки, используемый для создания структуры и содержимого веб-страниц. Он обладает следующими преимуществами: читаемость для разработчиков, возможность создания структурированного контента с использованием семантических элементов, доступность и адаптивности для различных устройств и браузеров. HTML является основным языком для создания веб-страниц и работает в сочетании с CSS для определения внешнего вида и JavaScript для добавления интерактивности;

— Git – распределённая система управления версиями;

— GitHub – платформа разработки программного обеспечения с открытым исходным кодом, представляющая систему управления репозиториями кода для Git;

— Vue.js – это прогрессивный JavaScript фреймворк для создания пользовательских интерфейсов. Он позволяет создавать масштабируемые и быстрые приложения с помощью компонентной архитектуры, реактивных свойств и директив. Vue.js также имеет удобный API и поддерживает двустороннюю привязку данных, что упрощает разработку и поддержку приложений. Vue.js также легко интегрируется с другими библиотеками и фреймворками, что делает его гибким и универсальным инструментом для разработки фронтенда. [4]

3.2 Реализация backend

Серверная (backend) часть приложения была написана на языке Java с использованием фреймворка Spring Boot. Spring Boot подразумевает разделение приложения на модули (прослойки), которые будут описаны далее. Каждый модуль организован в виде отдельного пакета. Структура приложения и иерархия пакетов с модулями изображена на Рисунке 8.

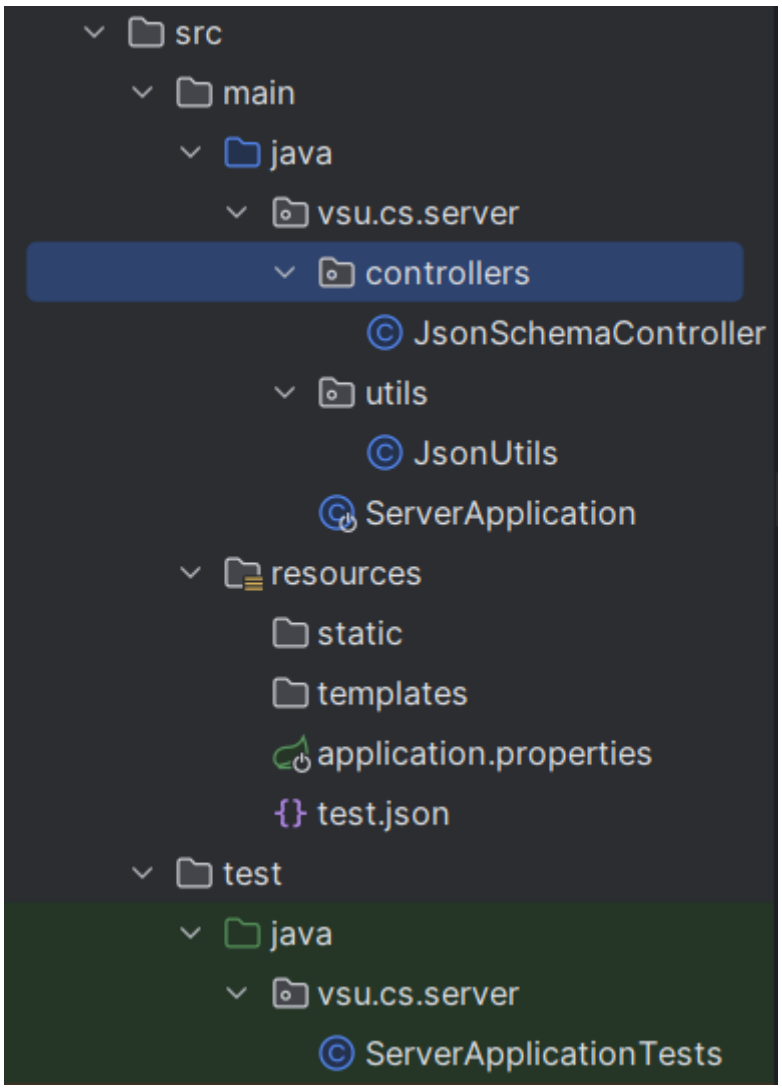

Рисунок 8 - Структура серверной части приложения

Приложение разделено на следующие модули:

— Модуль приложения (Application module). Это основной модуль, который содержит классы и настройки, связанные с запуском и конфигурацией приложения Spring Boot. В этом модуле находится главный класс приложения с аннотацией @SpringBootApplication и файл конфигурации application.properties, содержащий параметры подключения к базе данных. Для защиты базы данных эти параметры не объявляются явно в коде, а передаются в программу через переменные окружения;

— Модуль контроллеров (Controller Module). В этом модуле находятся классы контроллеров, отвечающие за обработку запросов и

взаимодействие с клиентом (браузером). Контроллеры содержат аннотации, такие как @RestController и @RequestMapping, для определения эндпоинтов и обработки запросов.

На Рисунке 9 представлены классы серверной части системы, их методы и атрибуты с типами данных.

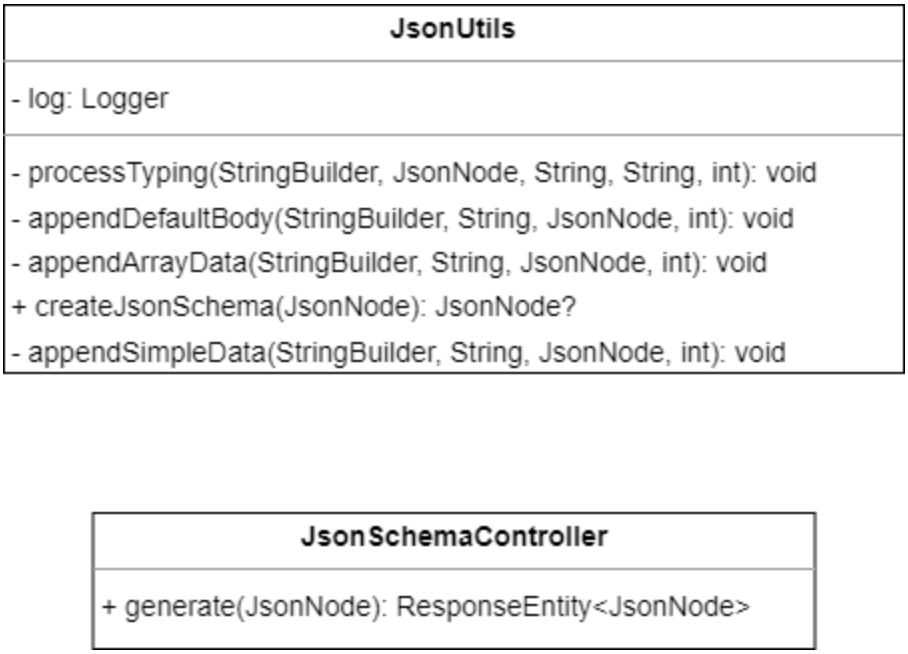

Рисунок 9 - Схематичное представление классов серверной части приложения

3.3 Реализация frontend

Клиентская часть приложения (frontend) разработана на языке JavaScript с использованием Фреймворка Vue.js. Для взаимодействия с серверной частью приложения на клиентской части используется библиотека axios, которая предоставляет интерфейс для выполнения HTTP-запросов из браузера.

Разрабатываемое приложение на Vue представляет собой SPA (single page application), то есть одностраничное приложение. Приложение загружает одну HTML страницу и динамически обновляет её без необходимости перезагрузки страницы при взаимодействии пользователя с приложением. Вместо традиционного подхода, когда каждый переход на новую страницу вызывает полную загрузку HTML, CSS и JavaScript, SPA загружает и инициализирует приложение один раз, а затем динамически обновляет только

необходимую часть страницы при переходе между разделами или выполнении определенных действий.

Особенностью разработки приложения на Vue является разбиение на компоненты. Все приложение разбито на компоненты, которые в необходимый момент загружаются на страницу. Компоненты представляют собой функции, возвращающие определённый HTML код. Эти функции могут хранить в себе переменные или другие функции.

Разработанное Vue-приложение имеет базовую структуру: директорию public, в которой хранятся статические ресурсы (HTML файлы), и директорию src (source), в которой хранятся все исходные файлы приложения. Директория src хранит в себе поддиректорию components (для хранения всех компонентов приложения). Кроме того, в директории src находятся основные .js файлы: index.js и App.js.

Описанная структура приложения изображена на Рисунке 10.

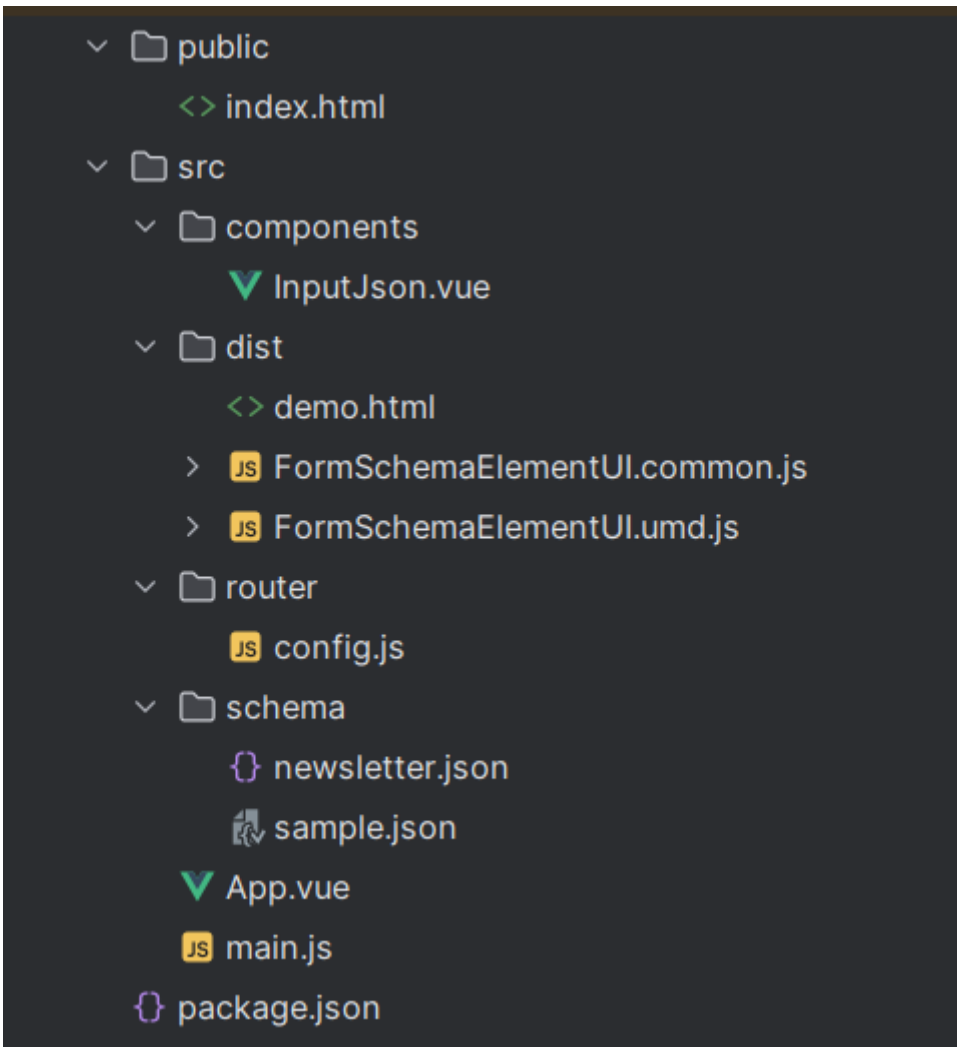

Рисунок 10 - Структура клиентской части приложения

3.4 Навигация по приложению

При запуске приложения пользователя встречает главная страница, представленная на Рисунке 11.

Рисунок 11 - Главная страница приложения

На данной веб-странице пользователь может ввести текст в формате JSON и создать форму на основе введенных данных. Однако, при некорректном форматировании JSON или неисправной работе сервера, возможно возникновение исключения, которое будет отображено на главной странице. Пример неправильного формата JSON представлен на Рисунке 12.

Для корректной работы системы необходимо убедиться в правильности введенных данных и исправности сервера. В случае возникновения ошибок, необходимо провести диагностику и устранить проблему. Только таким образом можно обеспечить бесперебойную работу системы и удовлетворить потребности пользователей.

Рисунок 12 - Пример ошибки неправильного формата JSON

В случае, если сервер выключен или недоступен, пользователь будет уведомлен об этом. Пример такой ситуации изображен на Рисунке 13.

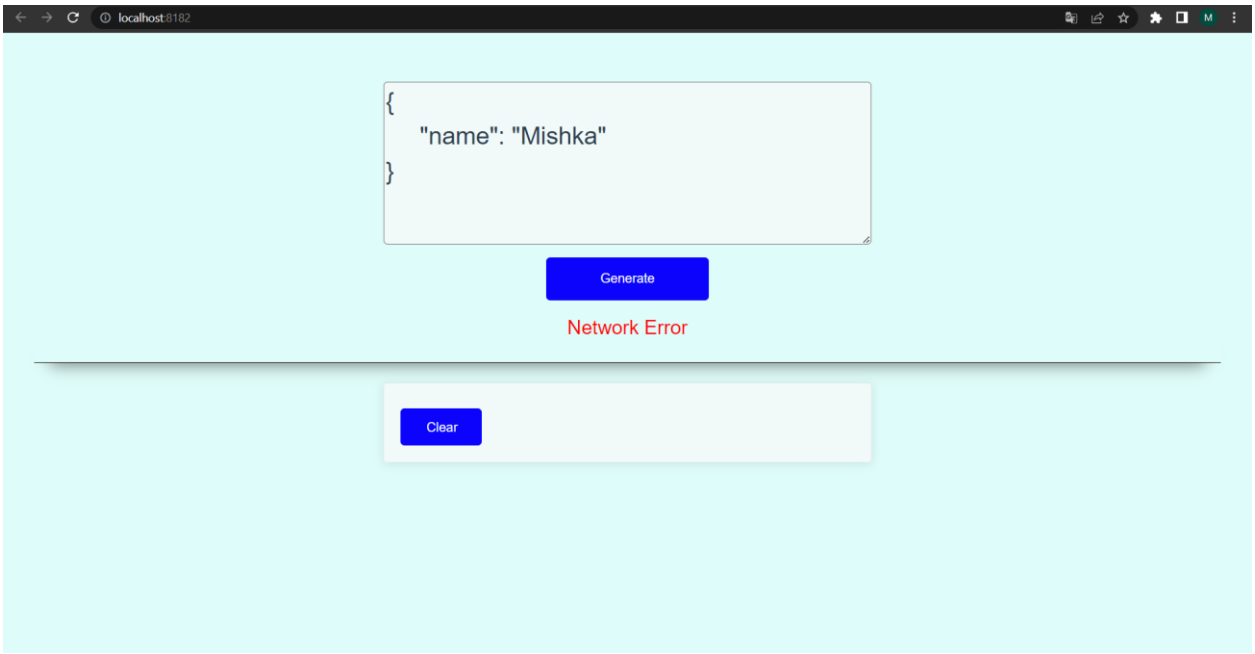

Рисунок 13 - Пример ошибки работы сервера

При успешной валидации JSON и корректной работе сервера, будет отправлен POST запрос на серверную часть. При успешной обработке запроса, будет сгенерирована JSON Schema, по которой будет сгенерирован

графический интерфейс формы. Пример такой формы представлен на Рисунке 14.

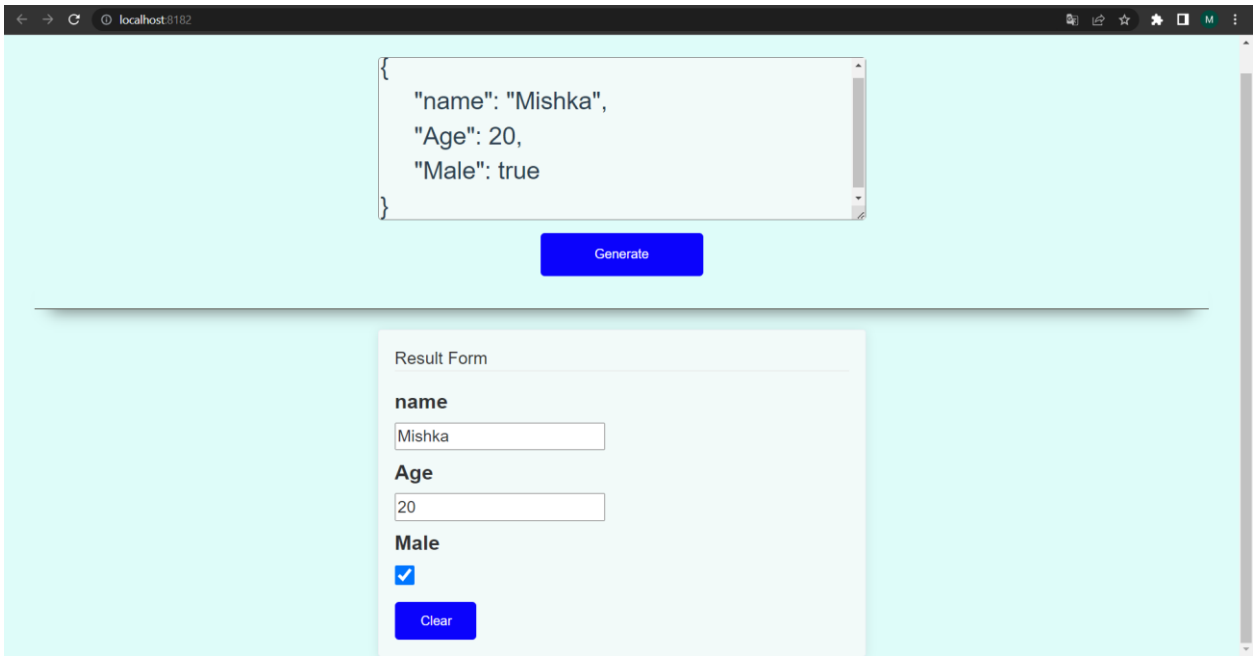

Рисунок 14 - Результат работы приложения

4 Тестирование

Для тестирования разработанной системы были проведены тесты основных типов:

— Модульное тестирование (unit-тесты)

— Тесты, связанные с изменениями (дымовое тестирование).

4.1 Модульное тестирование

Unit-тесты являются частью процесса разработки программного обеспечения и используются для проверки отдельных модулей или компонентов программы. Они предназначены для тестирования функциональности отдельных единиц кода, таких как отдельные классы, методы или функции.

Целью unit-тестов является проверка, что каждая единица программного кода работает правильно в изоляции от других компонентов системы. Unit-тесты проверяют, что методы возвращают ожидаемые результаты при заданных входных данных, а также что ошибочные ситуации обрабатываются корректно.

На Рисунке 15 представлены результаты unit-тестов серверной части приложения. Проверялась работа контроллеров. Тесты были написаны в классе с аннотацией @WebMvcTest.

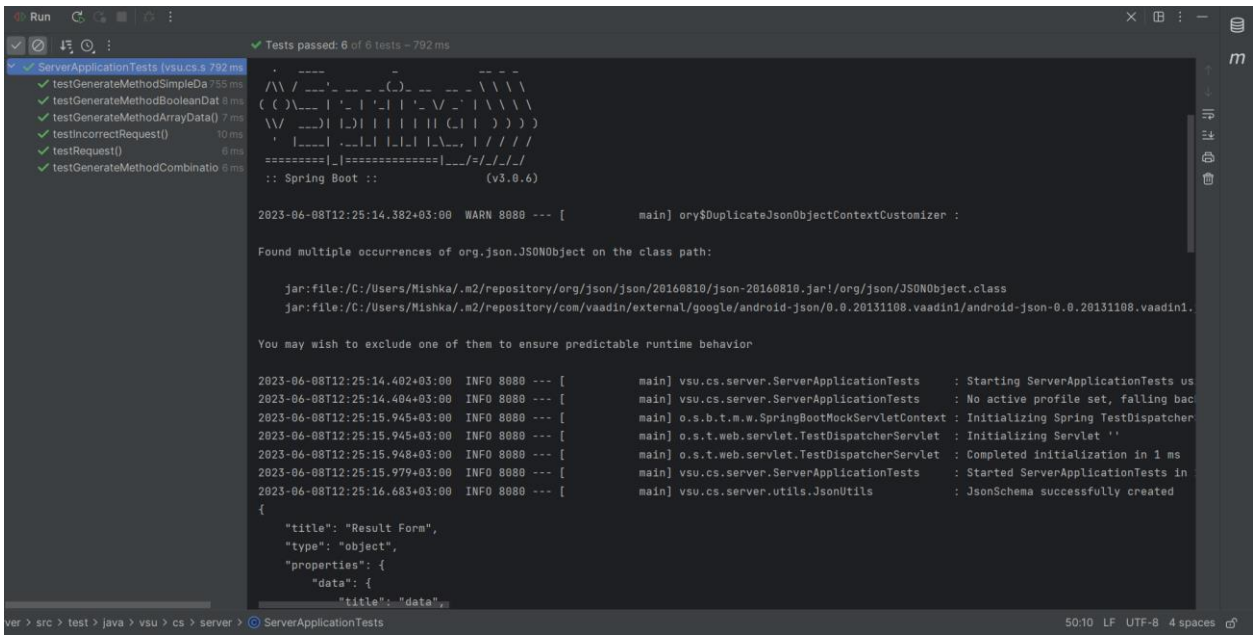

Рисунок 15 - Результаты unit-тестов

4.2 Дымовое тестирование

Дымовое тестирование (smoke testing) – это вид тестирования программного обеспечения, который выполняется для быстрой проверки основной функциональности системы или компонента после внесения изменений или внедрения новой версии. Основная цель дымового тестирования – обнаружить критические ошибки или проблемы, которые могут повлиять на работоспособность системы в целом.

В ходе дымового тестирования выполняются базовые операции или сценарии, которые предполагаются как наиболее важные и часто используемые пользователем.

В Таблице 1 представлены результаты дымового тестирования для сценария пользователя.

Таблица 1 - Результаты дымового тестирования для пользователя

Тестовый сценарий	Результат теста
Регистрация	Пройден
Ошибка неверного формата JSON	Пройден
Ошибка выключенного сервера	Пройден
Генерация формы с форматом String	Пройден
Генерация формы с форматом Boolean	Пройден
Генерация формы с форматом Integer	Пройден
Генерация формы с форматом Array	Пройден
Генерация формы с форматом Object	Пройден
Генерация формы с комбинированным форматом	Пройден
Очистка формы	Пройден

Заключение

В результате успешного выполнения проекта было разработано полнофункциональное веб-приложение, состоящее из серверной части на Spring Boot REST и клиентской части на Vue с использованием библиотеки FormSchema Native.

В ходе работы была создана JSON Schema, которая генерируется на серверной стороне. Эта схема успешно используется для динамической генерации форм на клиентской стороне, что позволяет удобно отображать и заполнять формы в приложении.

Логика обработки данных, введенных пользователем в формы, была успешно реализована. Пользовательские данные передаются на серверную сторону для дальнейшей обработки согласно требованиям бизнес-логики приложения. Весь процесс обработки данных проходит без ошибок и соблюдением необходимых проверок и валидаций.

В целом, выполнение данного проекта привело к разработке надежного и удобного приложения, которое обеспечивает динамическую генерацию форм на основе JSON Schema и эффективную обработку пользовательских данных на серверной стороне. Приложение успешно достигло своей окончательной цели, предоставляя пользователям удобный и функциональный интерфейс для работы с формами и отправки данных на сервер.

Список использованных источников

1. Rest, что это такое? [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/590679/. – Заглавие с экрана. – (Дата обращения: 18.04.2023).

2. Backend разработка [Электронный ресурс]. – Режим доступа: https://mobile-erp.ru/backend-razrabotka/. – Заглавие с экрана. – (Дата обращения: 26.04.2023).

3. Фронтенд [Электронный ресурс]. – Режим доступа: https://www.gorkilib.ru/events/frontend. – Заглавие с экрана. – (Дата обращения: 21.03.2023).

4. Vue [Электронный ресурс]. – Режим доступа: https://ru.vuejs.org/index.html. – Заглавие с экрана. – (Дата обращения: 15.05.2023).

5. UML.indd [Электронный ресурс]. – Режим доступа: https://csharpcooking.github.io/theory/UnifiedModelingLanguageUsereGuide.pdf. – Заглавие с экрана. – (Дата обращения: 27.04.2023).

6. Spring Framework Documentation [Электронный ресурс]. – Режим доступа: https://docs.spring.io/spring-framework/reference/. – Заглавие с экрана. – (Дата обращения: 01.05.2023).