Исследование архитектурного решения

Содержание

3.2. Пути улучшения архитектуры

Часть 1 - Проектирование архитектуры

1.1. Тип приложения

Проект предполагает проектирование и создание веб-приложения для прохождения викторин, в ходе которых один или несколько участников отвечают на поставленные им вопросы. Проект предполагает реализацию на языках C#, JavaScript, HTML, CSS с использованием библиотек React и Redux и платформе .NET Core.

1.2 Выбор стратегии развёртывания

Нераспределенное развертывание по шаблону Клиент-Сервер[razv]

1.3. Выбор технологии

Ключевым фактором при выборе технологий являлись тип разрабатываемого продукта (web-

приложение) и наличие навыков по работе с этими технологиями у команды. - Критерии, по которым мы выбрали платформу ASP.NET Core: * Желание работать с данной технологией * Разработка в Visual Studio * Поддержка клиентских фреймворков, таких как AngularJs, ReactJs, React с Redux и т.д. * Скорость разработки * Примеры с кодом на MSDN - Выбор **MongoDB** основывался на преимуществах этой базы данных: * Отсутствие схемы (Данная БД основана на коллекциях различных документов. Количество полей, содержание и размер этих документов может отличаться.) * Крайне понятная структура каждого объекта. * Легко масштабируется * Для хранения используемых в данный момент данных используется внутренняя память, что позволяет получать более быстрый доступ. * Данные хранятся в виде JSON документов * MongoDB поддерживает динамические запросы документов (document-based query) * Отсутствие сложных JOIN запросов - **React** - это библиотека для создания пользовательских интерфейсов. Одной из ее отличительных особенностей является возможность использовать JSX, язык программирования с близким к HTML синтаксисом, который компилируется в JavaScript. Разработчики могут добиваться высокой производительности приложений с помощью Virtual DOM. Компонентно-ориентированный подход, возможность с легкостью изменять имеющиеся компоненты и переиспользовать код превращают React разработку в непрерывный процесс улучшения.

1.4. Показатели качества

1.4.1 Качества дизайна

1.4.1.1 Концептуальная целостность

- Единый стиль написания кода как для Frontend, так и для Backend частей системы

1.4.1.2 Удобство и простота обслуживания

- В приложении реализован принцип единой ответственности, что позволяет добавлять или изменять функциональности отдельных компонент, не влияя на другие компоненты. Интерфейсы системы легко расширяются и не завязаны на реализации.

1.4.2 Качества времени выполнения

1.4.2.1 Производительность

- Приложение должно иметь быстрый отклик при взаимодействии с пользователем, выполнять любое действие за определённый промежуток времени.

1.4.2.2 Безопасность и надёжность

- На сервере будет установлен брандмауэр, чтобы можно было фильтровать входящие транзакции. Целостность данных для критических переменных также будет проверена. Веб-приложение хранит данные пользователей в базе данных и имеет к ней исключительный доступ.

1.4.3 Качества системы

1.4.3.1 Тестируемость

1.4.3.1 Тестируемость

- В системе есть полный контроль над входными и выходными параметрами, что позволяет тестировать код.

1.4.4 Качества взаимодействия с пользователем

1.4.4.1 Удобство и простота использования

- Навигация одинакова на всех страницах сайта;
- Правильное размещение контента на странице: большая часть пользователей привыкла к направлению письма слева направо;
- Все функциональные элементы пользовательского интерфейса имеют названия либо интуитивно указывают на действие, которое произойдет при выборе элемента.

1.5. Пути реализации сквозной функциональности

Помимо общей функциональности в приложении планируется реализовать функции сквозной функциональности: аутентификация, авторизация, валидация.

- Аутентификация:
  - пароли не передаются по сети и не хранятся в БД в открытом виде. Используется хеш пароля;
  - применяется стратегия единой регистрации, используются одни и те же учетные данные;
- Авторизация: защита ресурсов посредством авторизации вызывающей стороны;
- Хэширование: реализация алгоритма SHA-1
- Валидация: проверка данных форм позволяет удостовериться в том, что пользователи заполняют форму в правильном формате, убедиться, что отправленные данные будут успешно работать с нашим приложением.

1.6. Архитектура "To Be"

to_be_arch

Часть 2 - Анализ архитектуры

2.1. Архитектура "As is"

as_is_arch

Часть 3 - Сравнение и рефакторинг

3.1. Сравнение и анализ

По окончанию первого спринта задачи, которые входили в серверную часть, были выполнены в соответствии с ожидаемой архитектурой. Так как в приоритете была серверная часть, клиентская часть пока реализовывалась немного не так, как ожидается. Команда старается придерживаться

ожидаемой архитектуры и созданных мокапов приложения.

3.2. Пути улучшения архитектуры

В качестве рефакторинга клиентской части приложения планируется произвести дробление компонент на подкомпоненты.
Для тех частей приложения, в которых одни и те же данные совместно используются несколькими компонентами, полезно предусмотреть некое централизованное хранилище информации, расположенное на верхнем уровне иерархии. Для этого следует выносить данные в ReduxStore, а не принимать их от сервера на стороне UI.