Здравствуйте.

Я Максим ЛЯННОЙ и темой моего дипломного проекта является веб-чат.

Назначение (общее назначение мессенджера)

Для начала я бы хотел рассказать об общем назначении работы.

Данная работа предназначена для предоставления простого, удобного и кроссплатформенного мессенджера для обеспечения текстовой связи между пользователями.

Цель (определение главной цели при написании работы)

Перед началом любой разработки, как правило, следует определить и четко обозначить цель финального результата.

Лично для меня было важно собрать в единое целое, в некий финальный аккорд, все знания полученные в процессе обучения в Академии. В тоже время, *не было цели* создать некий “конкретный продукт” (который мог бы послужить аналогом существующим лидирующим мессенджерам на рынке). В частности, данная работа также является переосмыслением и полной переписью курсового проекта по веб-программированию.

Принципы, парадигмы, практики

При написании данной работы я старался придерживаться наилучших практик со всей области разработки бизнес-приложений.

Среди них:

- Объектно-ориентированное программирование (OOP)
- Объектно-ориентированный дизайн (OOD - SOLID, GRASP)
- Шаблоны проектирования GOF, а также архитектурные шаблоны (MVC и CQRS)

- Предметно-ориентированный дизайн (DDD)
- Парадигма реактивного программирования
- Принцип “Чистой архитектуры”

Архитектура

[(слайд 6)] Также одной из основополагающих задач для достижения хорошего результата - для того чтобы приложение хорошо масштабировалось, легко подвергалось изменениям и эти изменения не влияли на всю систему целиком (чтобы не пришлось ради одной задачи, вносить изменения в множество несвязанных классов) - этой задачей является выбор корректной архитектуры.

[(слайд 7)] Каждому программисту известна так называемая “классическая” или “трехуровневая” архитектура, при которой мы следуем разделению всей нашей системы на три довольно больших слоя…

[(слайд 8)] Но такое решение, на мой взгляд, сильно раздувает слои и они получаются большими. Разрабатывать и / или поддерживать такую систему начинает становиться дорого. Поэтому стоит обратить внимание на архитектуру, которая предлагает немного другую стратегию.

[(слайд 9)] Архитектурный шаблон CQRS (Command and Query Responsibility Segregation) подразумевает разделение всех внутренних операций приложения на команды и запросы. Тем самым мы получаем больше возможностей с точки зрения обработки поступающих на посредника команд / запросов, отслеживания их, уведомлении о успешном выполнении команд и кешировании ответов запросов.

[(слайд 10)] Вы можете видеть высокоуровневый пример от Microsoft, который наглядно демонстрирует работу приложения построенного по архитектуре CQRS…

[(слайд 11)] Следующий пример от архитектора Jason Taylor, человека который впервые мне продемонстрировал этот шаблон в действии. Справа вверху вы видите QR-код, который перенаправит вас на доклад этого человека и его репозиторий с конкретным примером…

[(слайд 12)] К недостаткам я бы отнес дублирование кода, которое если и решается, то максимально избежать его не получиться из-за того, что по-принципу команды и запросы не должны быть связаны.

[(слайд 13)] Переходя к нашей системе, для большего понимания внутренних процессов - стоит посмотреть на решение, на то как разделены проекты… *[(диаграмма архитектуры серверной стороны)]*

Время ограничено, поэтому стоит лишь отметить что серверная сторона реализована на .NET 5 с применением библиотеки MediatR для имплементации архитектуры CQRS. *[(возвращаемся на презентацию)]*

Архитектура серверной стороны

[(слайд 14-15)] **Хранилище** (диаграмма основной базы данных с объяснением СУБД Microsoft SQL Server 2019 и технологии Entity Framework Core 5)

[(слайд 16-17)] **Аутентификация** (диаграмма Identity базы данных с объяснением СУБД Microsoft SQL Server 2019, технологии Entity Framework Core 5 и ASP.NET Core Identity)

[(слайд 18)] **Проецирование** (преобразование с доменных моделей на бизнес-типы и наоборот) - AutoMapper

[(слайд 19)] **Валидация** (валидация во fluent-нотации) - FluentValidation

[(слайд 20)] **Фильтрация** (построение дерева выражений в простой способ) - LinqKit

[(слайд 21)] **Пейджинг** - нативными средствами на IQueryable

[(слайд 22)] **Кеширование** (уменьшение нагрузки путем минимизации количества запросов к удаленному хранилищу) - Redis

[(слайд 23)] **Логирование** - Serilog

[(слайд 24)] **Документирование** (быстрое документирование нового API) - Swagger

Архитектура клиентского веб-приложения

Архитектура клиентского веб-приложения подразумевает, что приложение написано на Angular 11, с использованием препроцессора Sass, разделением на умные / глупые компоненты, а также компоненты разделены по модулям.

[(слайд 26)] **Автоматизация разработки** - Nx

[(слайд 27)] **Оформление** - Bootstrap (ng-bootstrap), Font Awesome, SweetAlert 2

[(слайд 28)] **Состояние** (хранение состояния) - NgRx

[(слайд 29)] **Реактивное программирование** - RxJS

Протоколы общения

- основное серверное RESTful Web API доступно через REST (протокол HTTPS)
- *[(слайд 31)]* обеспечен хаб для уведомления клиентов в режиме реального времени по протоколу WebSocket (библиотека SignalR)

Развертывание

- хостинг серверной стороны на Azure
- хостинг клиентского веб-приложения на Firebase
- Docker (для RESTful Web API на Azure и полного набора серверной стороны локально)

Аутентификация

Отдельное внимание хочется уделить аутентификации, ведь это SPA-приложение и на нем возможно реализовать только **JWT-аутентификацию**. В отличии от MVC, где еще присутствует ряд других вариантов, в частности классический вариант - cookies-based.

Помимо JWT-аутентификации пользователь также может настроить **двухфакторную аутентификацию**, которая добавляет еще один слой безопасности к учетной записи пользователя.

Демонстрация

(QR)

Репозиторий

Весь код находиться в open-source и размещен в репозитории на GitHub. В нем описана инструкция по локальному развертыванию системы, подключены так называемые “Code Review” сервисы, которые призваны помочь контролировать качество коммитов и в частности всей кодовой базы, а также в репозитории настроены GitHub Actions (CI/CD контейнер который собирает проект при каждом коммите в main-ветку).

Roadmap

- Разработка мобильного приложения на нативном Kotlin
- Разработка десктопного приложения
- Более адаптивная верстка
- Оформление (полностью черная тема)
- Локализация
- Вход через социальные сети (в часности Google)
- ...