Санкт-Петербургский государственный университет
Математическое обеспечение и администрирование информационных систем

Отчет по учебной практике

Создание платформой для трейдинга «Trading Station»

Выполнял студент 244 группы Лямин В.А.
Научный руководитель к. т. н.,
Литвинов Ю.В.
Консультант Рябцев С.А.

Санкт-Петербруг 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время разработка более или менее крупного проекта проходит через несколько стадий [1]. Например: выяснение требований, планирование и т. д.. Поэтому проект «Trading Station» призван ознакомить и научить студентов процессу коммерческой разработки софта.

В том числе каждый участник команды должен был попробовать себя в разных сферах. Например все мы были две недели лидами команды. Лид должен был распределять новые задачи между членами команды, а также смотреть, чтобы задачи выполнялись. В конце недели он предоставлял отчет консультанту о проделанной работе. Также обязательно надо было посещать совещаний по проекту два раза в неделю.

Тема трейдинга была выбрана, так как она позволяет максимально затронуть наибольшее количество инструментов. Также нельзя не отметить большой спрос на такие платформы в мире [2], поэтому есть большая вероятность того, что опыт полученный при разработке можно будет применить в будущем.

1 Постановка задачи

Проект был реализован группой под руководством консультанта из Ланит-Терком Рябцева Спартака Александровича. Общая цель проекта: создать приложение, которое содержит инструменты для создания и запуска работающих торговых ботов. Для достижения поставленной цели были поставлены следующие общие задачи:

1. проектирование архитектуры приложения;
2. создание логики:
    1) хранения и управления данными о пользователях;
    2) взаимодействия с API брокера;
    3) хранения и управления действиями пользователей с финансовыми инструментами;
    4) работы ботов;
    5) логирование;
    6) валидация данных;
    7) Unit тестирование;
3. проведение интеграционного тестирования системы;
4. деплой платформы на prod сервер.

Перед автором стояли следующие частные задачи:

а) логика деактивации / изменения пользователя;

б) получение курса валют из ЦБ;

в) тестирование работы валидаторов;

г) часть GUI компонентов;

д) проведение интеграционного тестирования;

е) провести апробацию.

2 Обзор аналогов

Так как наша команда изначально делала нежизнеспособный проект, из-за того что получать реальные курсы валют нам бы никто не дал. Поэтому мы брали данные из «песочницы». Но при разработке мы обратили внимание на существующие аналоги и реализовали наиболее важные детали, которые должны быть в приложении для трейдига.

Одним из таких аналогов стал «MetaTrader». Одним из отличительных особенностей таких приложений — это таблица курсов валют и график стоимости каждого инструмента (рисунок 1).

Рисунок 1 — Интерфейс приложения «MetaTrader»

3 Диаграмма компонентов

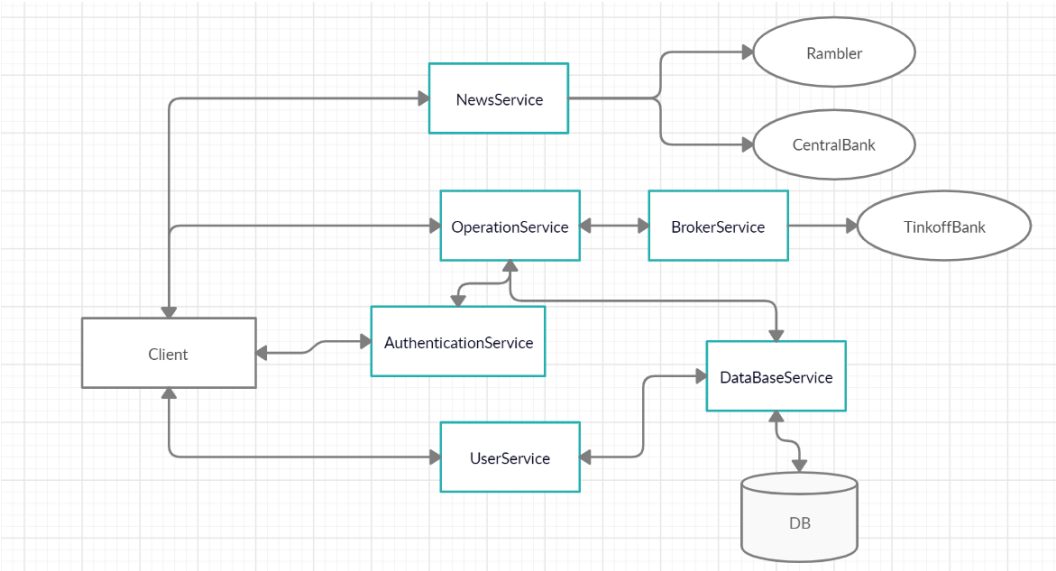

Рисунок 2 — Диаграмма компонентов «Trading Station»

Для архитектуры приложения мы выбрали микросервисную архитектуру, вместо монолита, так как она дает ряд следующих преимуществ [3]: масштабируемость, изолируемость, позволяет разделить разработку на несколько команд, также позволяет обновлять сервисы независимо друг от друга. Также мы основывались на современные решения команд разработчиков, которые все чаще выбирают микросервисную архитектуру. Наша схема показана на римунке 2.

«NewsService» мы использовали для получения новостей и курса валют. Его источниками данных выступают «Rambler» и «Центробанк». Данные поступают в формате RSS (формат для передачи новостей) [4].

Основная цель «AuthenticationService» — это инкапсулировать логику подтверждения личности пользователя. Сервис производит проверку и выдачу индивидуального токена, а также выполняет вход и выход пользователя из системы.

«UserService» управляет данными о пользователях системы. Он может зарегистрировать пользователя, изменить или получить информацию о нем. Также он нужен для активации аккаунта и для подтверждения регистрации пользователя при помощи отправки email на почту.

«DatabaseService» представляет единый интерфейса для работы с базой данных для всех сервисов. Его основные функции:

- исполнение CRUD операции с пользователями, брокерами и ботами;

- запись логов;

- миграция.

«BrokerService» предназначен для инкапсуляции и взаимодействии с биржевыми брокерами. В его функции входят: создание соединения с нужным брокером, получение списка инструментов, предоставляемого брокером, отслеживание информации об инструментах в реальном времени, реализация торговых сделок.

И последний сервис — это «OperationService». Его предназначение - это управление финансовыми операциями пользователей. В рамках этого сервиса осуществляется контроль за операциями покупок и продаж инструментов, контроль баланса пользователя, взаимодействие с портфелем пользователя и взаимодействие с торговыми ботами.

4 Обзор решения

4.1 Обзор решения деактивации и изменение пользователей

Моя работа над этой задачей осуществлялась в сервисе под названием «UserService». Для единообразия кода, мы решили использовать «модели». Например «EditUserRequest» состоит из «UserInfoRequest», «PasswordChangeRequest», «AvatarChangeRequest». А каждая из этих 3 частей состоит из полей со значениями простых типов или строк. Так «PasswordChangeRequest» состоит из полей: «OldPassword», «NewPassword» строкового типа.

Для решения задачи изменения пользователя был создан класс «EditUserCommand», наследуемый от интерфейса «IEditUserCommand». При инициализации класса, в его конструктор передается различные запросы, например ILogger, который передают результат об удавшемся или неудавшемся изменение пользователя.

Когда пользователь хочет изменить информацию о себе вызывается метод «execute». Единственный параметр — это модель изменения пользователя. Метод проверяет является ли старый пароль пустым. Если да, то он понимает, что изменять его не надо и передает в качестве параметра (UserPasswords) запроса к сервису, отвечающему за базу данных пустое множество. Если нет, заполняет его данными. Параметры «User» и «UserAvatar» будут заполнены в любом случае. После этого вызывается метод «EditUser», который уже передает управление «DataBaseService». Он возвращает булевое значение: получилось изменить пользователя или нет.

Тестирование «EditUserCommand», «DeleteUserCommand» Unit тестами представляет некоторые проблемы, так как методы ждут ответа от базы данных. Для этого мне понадобились mock-объекты [5]. Это объект «заглушка» и ее задачей является имитация действий другого сервиса.

Перед началом тестирования, я в отдельном методе инициализировал mock-объеты для разных запросов, таких как «UserInfoRequest», «PasswordChangeRequest» и др. Дальше нужно сделать setup, который говорит, что возвращает объект в зависимости от того, что он принимает. В моем случает метод должен получать «true», если все прошло хорошо, или «false», если произошло исключение В этом же методе я инициализировал и «UserInfoRequest», так как корректную работу этого запроса проверяют другие тесты, а значит я могу считать, что он работает правильно и самому не тестировать случаи с неправильным заполнением полей. После выполнения этих действий остается только написать тесты обычным образом.

4.2 Интеграционное тестирование

В мою задачу входило интеграционное тестирование backend нашего приложения. Для этого выбрали инструмент тестировщик Postman [6]. Это инструмент для тестирования сервисов. Там мы создавали http запросы и таким образом тестировали их, к которым может обратиться клиент через REST.

Приведем пример создания теста на удаление пользователя в Postman. Для начала нужно создать коллекцию. В моем случае она называлась «UserService». Дальше создавались запросы для тестирования разных методов. Для выбора типа запроса надо посмотреть в контроллер сервиса. Например для удаления пользователя тип http запроса будет называться «DELETE». А также надо указать адрес на который будет посылаться запрос. В «Params» мы указываем id пользователя, которого хотим удалить, а в «Headers» мы должны положить токен пользователя. «Body» оставляем пустым, в отличие теста на изменение пользователя. Там мы помещаем новую информацию о нем.

Переходим к заключительной части. Нам надо написать, что должно прийти от сервиса в качестве ответа (код, что все хорошо или наоборот произошла ошибка). Это делается во кладке «Tests». На рисунке три показан пример работы теста в Postman.

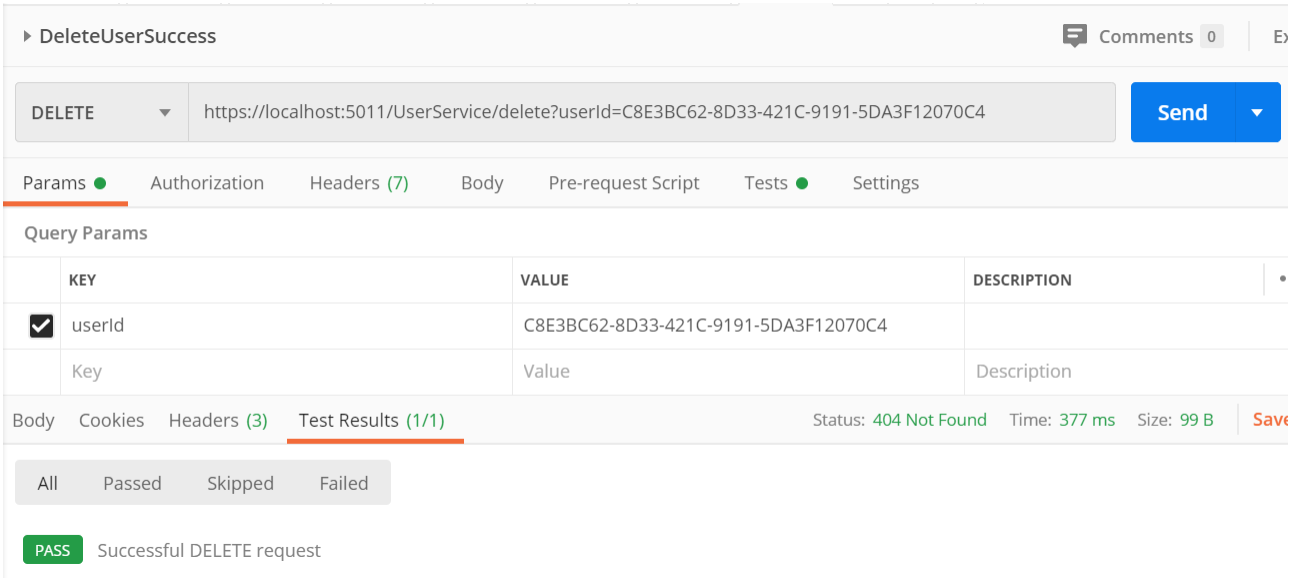

Рисунок 3 — Успешный Postman тест

4.3 Часть GUI компонентов

В процессе разработки приложения понадобилось разработать дизайн нашего сайта. Мне нужно было сделать основные компоненты главного окна: header в котором будет аватар пользователя, эмблема «Trading Station», и кнопки для входа/выхода из приложения, а также переход на другие страницы (рисунок 4). Также мне нужно было сделать всплывающее окно для входа пользователя (рисунок 5).

Для удобства создания интерфейса мы использовали «Blazor» [7]. Blazor представляет UI-фреймворк для создания интерактивных приложений, которые могут работать как на стороне сервера, так и на стороне клиента, на платформе .NET.

Все элементы заголовка я прописал в файле «Header.razor». Так как у нас должно быть несколько групп кнопок, то из лучше разбить на группы. Например: «header-right», где находились кнопки: «Sign Up», «Sign In», «Sign Out», «User Info». Как выглядят эти элементы (ширина заголовка, расположение кнопок и т.д.) описаны в файле «Style.css». После этого остается сказать программе, что у нас есть заголовок. Это делается в файле «MainLayout.razor».

Настройка модального окна входа пользователя в систему делается в файле «SignUpWizzard.razor». Единственным отличием от разработки заголовка или размещения окна валют является команда «modal.Show<>».

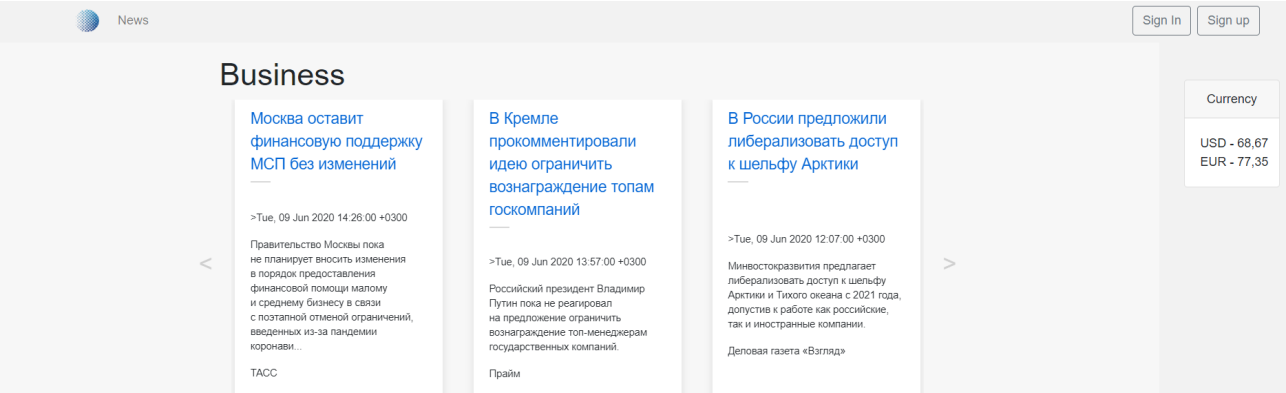

Рисунок 4 — Главная страница приложения

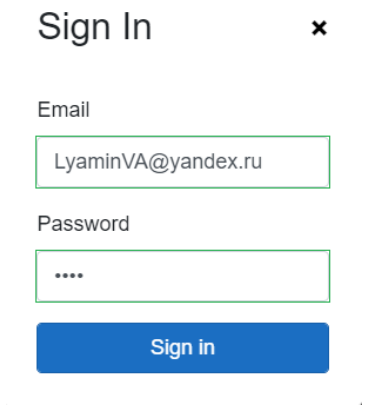

Рисунок 5 — Вводное окно входа пользователя

4.4 Тестирование работы валидаторов

В разработке приложения мы использовали валидаторы [8]. Валидатор - компьютерная программа, которая проверяет соответствие какого-либо документа, потока данных, или фрагмента кода определённому формату, проверяет синтаксическую корректность документа или файла.

Мной были протестированы следующие валидаторы: «PasswordChangeRequestValidator», «UserInfoRequestValidatorTests». В первом тесте я протестировал следующие моменты: все поля пустые, старый паспорт пустой, новый паспорт пустой, оба паспорта одинаковы и тест, в котором все хорошо. Во втором тесте затронуты следующие моменты: все поля пустые, потерян id и email пользователя, фамилия и имя слишком длинные или содержат цифры, возраст меньше 18 лет и тест в котором все хорошо.

4.5 Получение курса валют из ЦБ

В мою задачу входило получение курса валют из ЦБ. Главная работа над этой задачей проходило в файле «RussianCentralBankInfo.cs». Для начала нам надо узнать URI ЦБ [8]. По нему мы будем получать наши данные. При инициализации класса «RussianCBInfo», происходит вызов метода «», который присоединяется с помощью URI к ЦБ и получает наиболее популярные значения. Затем при вызове метода «GetCurrencies», данные кладутся в список. В каждом узле этого списка есть код значения и оно само.

4.6 Апробация

Для принятие PL нужно четыре апрува, поэтому работу моего кода протестили и приняли четыре человека. Притом если хоть один не согласен, PL не может быть принятым. Также в конце проект прошел защиту демонстрацией работы. Также достаточно много человек захотели лично проверить работоспособность приложения.

5 Запуск приложения

Когда пользователь входит на сайт он видит перед собой перед собой главную страницу с новостями и курсом валют. Если пользователь захочет почитать новость подробнее, то он переход автоматически на сайт СМИ. Для работы с ботами нужно зарегистрироваться. Для большей безопасности, личность подтверждается по email.

В инструментах пользователь может купить акции нужной ему компании, а также при нажатие на компанию, может посмотреть график цен на ее акции (рисунок 6). Пополнить баланс пользователь может во кладке «Increase balance».

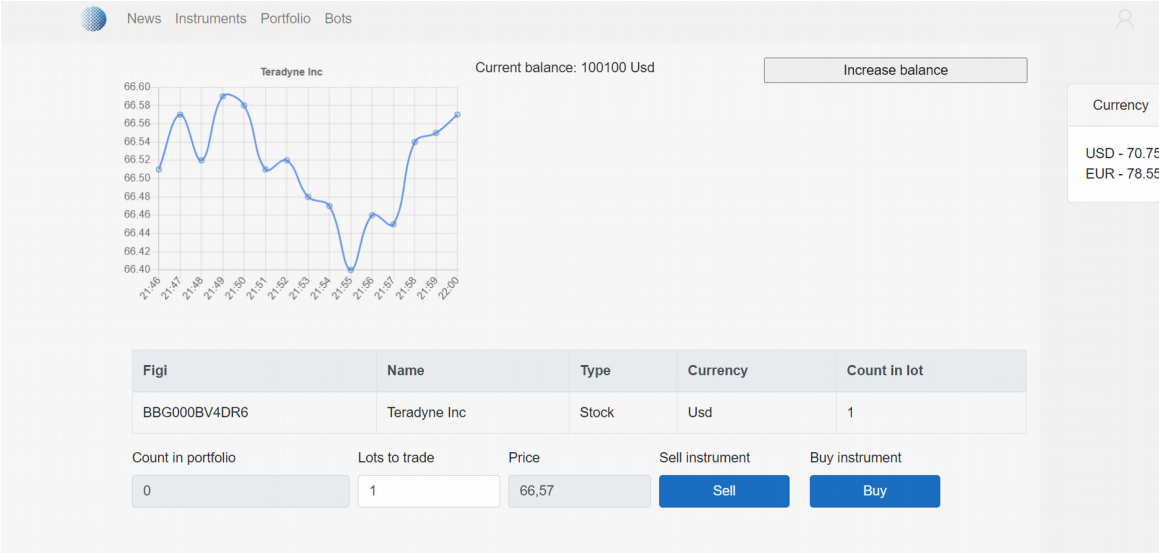

Рисунок 6 — График цен на компанию

ЗАКЛЮЧЕНИЕ

В ходе работы над данным проектом были достигнуты следующие результаты:

- реализована логика деактивации и изменения пользователя;
- реализовано получение курса валют из ЦБ;
- произведено тестирование работы валидаторов;
- реализована часть GUI компонентов;
- проведено интеграционное тестирование;
- проведена апробация.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 8 основных этапов разработки IT проекта, URL: https://www.ituniver.com/article?id=5646392177459200 (дата обращения 2020-06-08)
2. Посещаемость сайтов онлайн-трейдинга резко выросла из-за кризиса, URL: https://habr.com/ru/post/294872 (дата обращения 2020-06-08)
3. Просто о микросервисах, URL: https://habr.com/ru/company/raiffeisenbank/blog/346380 (дата обращения 2020-06-08)
4. RSS - в массы, а массы…, URL: https://www.opennet.ru/docs/RUS/rss_naklon (дата обращения 2020-06-09)
5. Написание Unit-тестов. Mocking объектов, URL: https://habr.com/ru/post/267255 (дата обращения 2020-06-09)
6. Как тестировать API, или Postman для чайников, URL: https://geekbrains.ru/posts/kak-testirovat-api-ili-postman-dlya-chajnikov (дата обращения 2020-06-09)
7. Введение в ASP.NET Core Blazor, URL: https://docs.microsoft.com/ru-ru/aspnet/core/blazor/?view=aspnetcore-3.1 (дата обращения 2020-06-09)
8. V for Validator, URL: https://habr.com/ru/post/348530 (дата обращения 2020-06-09)