МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский технологический университет «МИСиС»**

Институт ИТКН

Квалификация (степень): магистр

Уч.год: 2020-2021

Семестр: 2

Кафедра инженерной кибернетики

Группа: МПИ-20-4-2

**Курсовая работа**

**по дисциплине**

**«Современные инструментальные средства разработки»**

**на тему:**

**«Создание веб-сайта для обучения финансовой грамотности»**

**Студенты** Головатских Марк  
 Гужва Никита

Денисова Наталья  
 Добрынин Владислав

**Преподаватель** / доцент, к.т.н. Тарханов И.А. /

подпись должность, уч. степ., Фамилия И.О.

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва 2021**

Оглавление

[Введение 3](#_Toc75278429)

[1 Выбранная тема работы 4](#_Toc75278430)

[1. 1 Название системы и краткое описание 4](#_Toc75278431)

[1.2 Цель создание проекта 4](#_Toc75278432)

[1.3 Задачи проекта 4](#_Toc75278433)

[2 Разработка проекта 5](#_Toc75278434)

[2.1 Функциональные требования 5](#_Toc75278435)

[2.2 Нефункциональные требования 6](#_Toc75278444)

[2.3 Сценарии использования 8](#_Toc75278452)

[2.4 Сценарии тестирования 12](#_Toc75278471)

[2.5 Прототип основных окон проекта 15](#_Toc75278477)

[Заключение 19](#_Toc75278478)

[Список литературы 20](#_Toc75278479)

[Приложение 1. Листинг кода 21](#_Toc75278480)

Введение

В современном мире каждому человеку необходимы базовые навыки финансовой грамотности. В школьной программе не предусмотрены уроки по ведению бюджета, финансированию и инвестированию. Люди старшего поколения не знакомы с новшествами на рынке, поэтому часто становятся жертвами мошенников.

Веб-сайт по обучению финансовой грамотности поможет повысить общий уровень знаний у пользователей, для молодого поколения будет интересно попробовать свои силы в инвестировании без рисков.

Сочетание уроков по инвестированию и финансовый симулятор, позволяющий отточить знания на данных биржи за предыдущие года, является универсальным инструментом для обучения пользователей. Последовательное освоение уроков и применение полученных знаний на практике в игровом формате поможет пользователям повысить уровень знаний и навыков финансовой грамотности.

1 Выбранная тема работы

1. 1 Название системы и краткое описание

Наименование системы: Bulls&Bears

Краткое описание: Веб-сайт для обучения финансовой грамотности. Набор небольших курсов по разным темам: от базовых понятий до сложных финансовых инструментов. Каждый курс состоит нескольких глав, в составе которых обучающий материал, а также практические задачи.

На сайте будет реализована "финансовая песочница", в которой можно попрактиковаться в использовании различных инструментов. Выглядеть это будет как биржевой терминал, но все операции происходят оффлайн.

1.2 Цель создание проекта

Обучение пользователей основам работы с финансами и проверка полученных навыков с помощью внутренней "финансовой песочницы".

1.3 Задачи проекта

1. Создание функционала для регистрации и авторизации пользователей;

2. Разработка личной страницы пользователя (профиль);

3. Составление плана и создание учебных курсов;

4. Разработка функционала по обучению работы с финансами;

5. Создание алгоритма работы внутренней "финансовой песочницы";

6. Разработка функционала "финансовая песочница";

7. Создание обратной связи с разработчиками.

2 Разработка проекта

2.1 Функциональные требования

1. Регистрация пользователя

Для получения доступа к обучающим курсам необходимо пройти процедуру регистрации на сайте. Для этого предусмотрено заполнение данных о себе: имя пользователя, электронная почта и пароль. Повторная регистрация на сайте не требуется, необходима авторизация.

2. Авторизация пользователей

Для работы с сайтом, прохождения обучения и практических упражнений необходимо иметь регистрацию на сайте. Для зарегистрированных пользователей предусмотрена авторизация, необходимо указать имя пользователя и пароль, которые были указаны при регистрации.

3. Просмотр профиля

Для пользователей доступен просмотр основной информации о профиле: имя пользователя, изображение профиля, прогресс в изучении обучающих курсов, результаты работы с “финансовой песочницей”. В графе оценки навыков будут отображены: количество пройденных курсов и лучшие результаты по торгам (самая дорогостоящая сделка, потраченное время, наилучший запас).

4. Создание учебных курсов

Для обучения пользователей необходимо создание структурированного материала в текстовом формате с использованием сопутствующих иллюстраций. Для лучшего восприятия необходимо разделение курса на главы и темы. Материал должен быть составлен из проверенных источников: нормативно-правовые документы, книги, учебные и практические издания. Язык разработанных учебных курсов - английский.

5. Прохождение курса

Для обучения пользователей на сайте доступны обучающие курсы. Веб-страница состоит из текстового материала и иллюстраций. Возможен переход по главам, темам и страницам курса.

6. Работа внутренней «финансовой песочницы»

"Финансовая песочница" – это эмулятор работы биржевого терминала. Вся торговля данного терминала проводится с помощью виртуального счета. Ценовая динамика работы биржевого терминала задается внутренними алгоритмами "финансовой песочницы", основанными на исторических данных.

7. Взаимодействие с «финансовой песочницей»

Для закрепления полученных навыков для пользователей реализована "финансовая песочница". Пользователи могут наблюдать за биржевыми торгами, анализировать ценовую динамику, совершать сделки с ценными бумагами с помощью виртуального счета, отслеживать дальнейшее изменение инвестиционного портфеля и отдельных позиций по бумагам.

8. Обратная связь

Для взаимодействия с администраторами и разработчиками проекта будет предусмотрена обратная связь. Необходимо заполнить поля: тема и текст обращения.

2.2 Нефункциональные требования

1. Требования к пользователям приложения

Пользователь обязан владеть навыками работы с компьютером и сетью Интернет на базовом уровне. Для доступа к системе пользователю понадобится устройство с выходом в сеть Интернет.

2. Требования к надежности

Данная система должна обеспечивать целостность и непротиворечивость хранимых данных при любых действиях пользователей. При соблюдении условий нормального режима функционирования система должна быть доступна пользователям круглосуточно.

При возникновении непредсказуемой ситуации при работе пользователя система обязана выводить сообщение об ошибке без указания технической и внутренней информации о системе.

При аварийных или иных ситуациях, связанных со сбоем в работе аппаратной части системы, система должна автоматически восстанавливать свою работоспособность после устранения сбоя.

3. Требования к производительности

Время ответа системы на действия пользователя не должно превышать 3 секунды. Система должна обеспечивать работоспособность под нагрузкой вплоть до 100 пользователей.

4. Требования к эргономике и технической эстетике

Система должна быть понятна и доступна для пользователя. До разработки необходимо предоставить прототип системы.

5. Требования к лингвистическому обеспечению

* Стек технологий для front-end: JavaScript, React.
* Стек технологий для back-end: Python,Flask.
* DevOps: Docker,Ngrok,Travis.
* База данных: MongoDB

6. Требования к программному обеспечению

ПО должно распространяться свободной лицензией. Язык интерфейса - английский.

7. Требования к аппаратному обеспечению сервера

Минимальные аппаратные требования к серверу:

Процессор 4 ядра (8 логических потоков), частота – 2 ГГц и больше

Оперативная память 8 Гб и больше

Тип накопителя SSD, свободное дисковое пространство 300 Гб и больше

Пропускная способность сетевого интерфейса 1 Гбит/с

Пропускная способность сети между клиентом и сервером приложений должна составлять:

для номинальной работы клиентской части – 1 Мбит/c;

рекомендуемая пропускная способность – 10 Мбит/c или выше.

8. Требования к техническому обеспечению

Система должна быть рассчитана на использование с помощью персональных компьютеров.

Поддерживаемый браузер: Google Chrome.

Поддерживаемый язык интерфейса - английский.

Минимальное разрешение экрана: 1024 х 640.

2.3 Сценарии использования

1. Регистрация пользователя

Название прецедента: регистрация нового пользователя

Действующее лицо: новый пользователь

Цель: добавить нового пользователя в базу пользователей

Предусловие: пользователь заходит на сайт

Основной сценарий:

1. Пользователь нажимает кнопку "Log in/Register" (Вход/Регистрация).
2. Открывается страница входа на сайт.
3. Пользователь нажимает на кнопку "I am a new member" (Я - новый пользователь).
4. Открывается страница регистрации пользователя.
5. Пользователь корректно заполняет поля: "user name" (имя пользователя), "email" (адрес электронной почты), "password" (пароль). Пользователь может использовать буквы латинского алфавита, цифры и символы.
6. Пользователь нажимает кнопку "Join" (присоединиться).
7. Открывается страница профиля зарегистрированного пользователя.

Альтернативный сценарий:

5а Пользователь указывает username или email, который уже существует в системе.

5а.1 Вывод сообщения об ошибке при регистрации, так как для указанной электронной почты и/или имени пользователя уже существует профиль.

5b Пользователь использует символы русского или иного алфавита, символы и знаки, которые не поддерживает система.

5b.1 Вывод сообщения с предупреждением о возможности использовать только буквы латинского алфавита, цифры и символы.

7a. Произошел сбой в работе системы, пользователь не добавлен в базу.

7a.1 Вывод сообщения об ошибке в работе системы, просьба заново заполнить форму для регистрации.

2. Авторизация пользователя

Название прецедента: авторизация пользователя на сайте

Действующее лицо: пользователь

Цель: вход в профиль

Предусловие: пользователь заходит на сайт

Основной сценарий:

1. Пользователь нажимает кнопку "Log in/Register" (Вход/Регистрация).
2. Открывается страница входа на сайт.
3. Пользователь корректно заполняет поля: "user name/email" (имя пользователя/адрес электронной почты) и "password" (пароль). Пользователь может использовать буквы латинского алфавита, цифры и символы.
4. Пользователь нажимает кнопку "Log in" (Войти).
5. Открывается страница профиля пользователя.

Альтернативный сценарий:

3a. Пользователь указывает неверную пару username/email и password.

3a.1. Вывод сообщения об ошибке при заполнении полей.

4a. Пользователь нажимает кнопку "I am a new member" и переходит на страницу регистрации.

4a.1. Открывается страница регистрации, пользователю необходимо нажать на кнопку "I have got my account already" для перехода на страницу авторизации.

3. Просмотр профиля

Название прецедента: просмотр страницы профиля пользователя на сайте

Действующее лицо: пользователь

Цель: просмотр страницы профиля

Предусловие: пользователь успешно проходит процедуру регистрации/авторизации

Основной сценарий:

1. Пользователь нажимает кнопку "Log in/Join" (Вход/Присоединиться) для перехода на страницу пользователя.
2. Открывается страница профиля пользователя с основной информацией:
   1. username (имя пользователя);
   2. profile pic (изображение профиля);
   3. кнопка "change profile pic" (сменить изображение профиля);
   4. My progress (прогресс пользователя):
   5. Editorial (информация о количестве пройденных учебных курсов);
   6. Sandbox (статистика: самая дорогостоящая сделка, потраченное время, наилучший запас)
   7. Log out (выход из профиля).

Альтернативный сценарий:

2a. Произошел сбой в системе и информация загрузилась неверно, частично, отсутствует.

2a.1. Пользователь должен обновить страницу для повторного запроса к данным системы о пользователе. Если информация не появилась, пользователю необходимо связаться с разработчиками.

4. Прохождение курса

Название прецедента: прохождение курса пользователем на сайте

Действующее лицо: пользователь

Цель: изучение предоставленного материала

Предусловие: пользователь переходит на страницу Education

Основной сценарий:

1. Открывается страница с выбором курса или главы.
2. Пользователь нажимает любое изображение главы курса (указано название главы, описание, изображение).
3. Открывается первая страница (слайд) с информацией в данной главе.
4. Пользователь нажимает на кнопку "Next" (следующая страница).
5. Открывается следующая страница (слайд) по данной главе.
6. Пользователь повторяет действие 4 до тех пор, пока данная кнопка активна, далее нажимает кнопку "Finish" (Завершить курс).
7. Открывается страница с выбором курса и глав.

Альтернативный сценарий:

2a. Пользователь нажимает на кнопку "See all"

2a.1. Система должна отобразить список глав в данном курсе.

4а. Пользователь нажимает кнопку "Go back"

4а.1 Открывается страница с выбором курса и глав.

4b. Пользователь нажимает кнопку @Back" (предыдующий слайд)

4b.1 Открывается предыдующая страница (слайд) по данной главе.

5. Взаимодействие с «финансовой песочницей»

Название прецедента: проверка навыков пользователя на сайте

Действующее лицо: пользователь

Цель: проверить свои знания финансирования

Предусловие: пользователь переходит на страницу Sandbox

Основной сценарий:

1. Открывается страница работы с финансовой песочницей
2. Пользователь вводит число
3. Пользователь нажимает на кнопку "Buy"/"Sell" (Купить или продать)
4. Система обрабатывает введенную транзакцию
5. Система обновляет данные пользователя о транзакциях и счете
6. Пользователь повторяет действия с пункта 2, с целью увеличить сумму кошелька

Альтернативный сценарий:

2a. Пользователь вводит неверное число

2a.1. Система выводит сообщение об ошибке с указанием причины (нехватка средств, количества акции)

3а. Пользователь не указал число

3а.1 Система выводит сообщение с просьбой ввести число для транзакции

4а. Ошибка при совершении транзакции

4а.1 Система выводит сообщение об ошибке транзакции, транзакция не совершается

6. Обратная связь

Название прецедента: обратная связь с разработчиками

Действующее лицо: пользователь

Цель: оставить отзыв, жалобу или предложение

Предусловие: пользователь переходит на страницу Contact

Основной сценарий:

1. Открывается страница с возможностью обратной связи с разработчиками.
2. Пользователь заполняет предложенные поля: имя пользователя (для обращения), e-mail для обратной связи, тему обращения и сообщение.
3. Пользователь нажимает на кнопку "Send" (отправить заявку).
4. Система регистрирует заявку.
5. Отображается сообщение об успешной отправке заявки и её номер в системе.

Альтернативный сценарий:

2a. Пользователь заполняет не все предложенные поля

2a.1. Вывод сообщения с просьбой заполнить все поля.

4a. Произошел сбой в системе и заявка не зарегистрирована

4a.1. Вывод сообщения с просьбой отправить обращение позже или использовать другие способы связи с разработчиками с помощью социальной сети VK или через электронную почту.

2.4 Сценарии тестирования

Тест-кейс 1

Номер: 1

Название: Регистрация

Предусловие: открыта страница регистрации

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| Заполнить поле user name латинскими буквами | Отображается введенные данные в строке ввода |
| Заполнить поле user name русскими буквами | Предупреждение об ошибке, имя может быть задано только латинскими буквами, цифрами и знаками |
| Заполнить поле user name данными, которые уже существуют в системе | Сообщение, что данное имя уже зарегистрировано в системе |
| Заполнить поле email корректной записью почты, которая не зарегистрирована в системе | Отображаются введенные данные в строке ввода |
| Заполнить поле email русскими буквами | Предупреждение об ошибке, почта может быть задана только латинскими буквами, цифрами и знаками |
| Заполнить поле email, которая зарегистрирована в системе | Сообщение, что данная почта уже зарегистрирована в системе |
| Заполнить поле password корректно | Отображаются введенные данные в строке ввода, замененные на звездочки |
| Заполнить поле password русскими буквами | Предупреждение об ошибке, пароль может быть задан только латинскими буквами, цифрами и знаками |
| Корректно заполнены все данные, которые не зарегистрированы в системе | Переход на страницу профиля при нажатии на кнопку «Join» |

Таблица 1. Тест-кейс 1

Тест-кейс 2

Номер: 2

Название: Авторизация

Предусловие: открыта страница авторизации

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| Заполнить поле username/email латинскими буквами | Отображается введенные данные в строке ввода |
| Заполнить поле username/email русскими буквами | Предупреждение об ошибке, имя может быть задано только латинскими буквами, цифрами и знаками |
| Заполнить поле password корректно | Отображаются введенные данные в строке ввода, замененные на звездочки |
| Заполнить поле password русскими буквами | Предупреждение об ошибке, пароль может быть задан только латинскими буквами, цифрами и знаками |
| Корректно заполнены все данные, которые зарегистрированы в системе | Переход на страницу профиля при нажатии на кнопку «Log in» |
| Допущена ошибка при заполнении данных | Вывод сообщения, допущена ошибка при вводе username/email или password |

Таблица 2. Тест-кейс 2

Тест-кейс 3

Номер: 3

Название: Прохождение курса

Предусловие: открыта страница курсов

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| Нажимается кнопка «Next» (следующий слайд) | Открывается следующий слайд курса |
| Нажимается кнопка «Back» (предыдущий слайд) | Открывается предыдущий слайд курса |
| Нажимается кнопка "Go back" (вернуться) | Открывается список глав в данном курсе |
| Нажимается кнопка курса | Открывается страница с первым слайдом выбранного курса |
| Нажимается кнопка слайда | Открывается страница выбранного слайда |

Таблица 3. Тест-кейс 3

Тест-кейс 4

Номер: 4

Название: Финансовая песочница

Предусловие: открыта страница песочницы

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| Вводится число и нажимается кнопка «Buy» (покупка) | Совершается транзакция покупки |
| Вводится число и нажимается кнопка «Sell» (продажа) | Совершается транзакция продажи |
| Вводится число, превышающее сумму кошелька и нажимается кнопка «Buy» (покупка) | Вывод ошибки транзакции |
| Вводится число, превышающее сумму акций и нажимается кнопка «Sell» (продажа) | Вывод ошибки транзакции |

Таблица 4. Тест-кейс 4

Тест-кейс 5

Номер: 5

Название: Обратная связь

Предусловие: открыта страница обратной связи

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| Заполнены все поля корректно и нажимается кнопка "Send" (отправить заявку) | Вывод сообщения об успешной отправке |
| Неверно заполнено поле email | Сообщение об ошибки при заполнении полей |
| Заполнены не все поля и нажимается кнопка "Send" (отправить заявку) | Сообщение с просьбой заполнить все поля |

Таблица 5. Тест-кейс 5

2.5 Прототип основных окон проекта

Для создания веб-сайта необходим эскизный проект (прототип) для комплексного понимания в проведении необходимых работ. Создание эскиза помогает понять основные взаимосвязи между частями проекта.

Ссылка на интерактивную версию прототипа: https://app.uizard.io/p/NDCSCtr8b

На рисунках 1-7 представлены эскизы основных окон проекта.



Рисунок 1. Главная страница веб-сайта

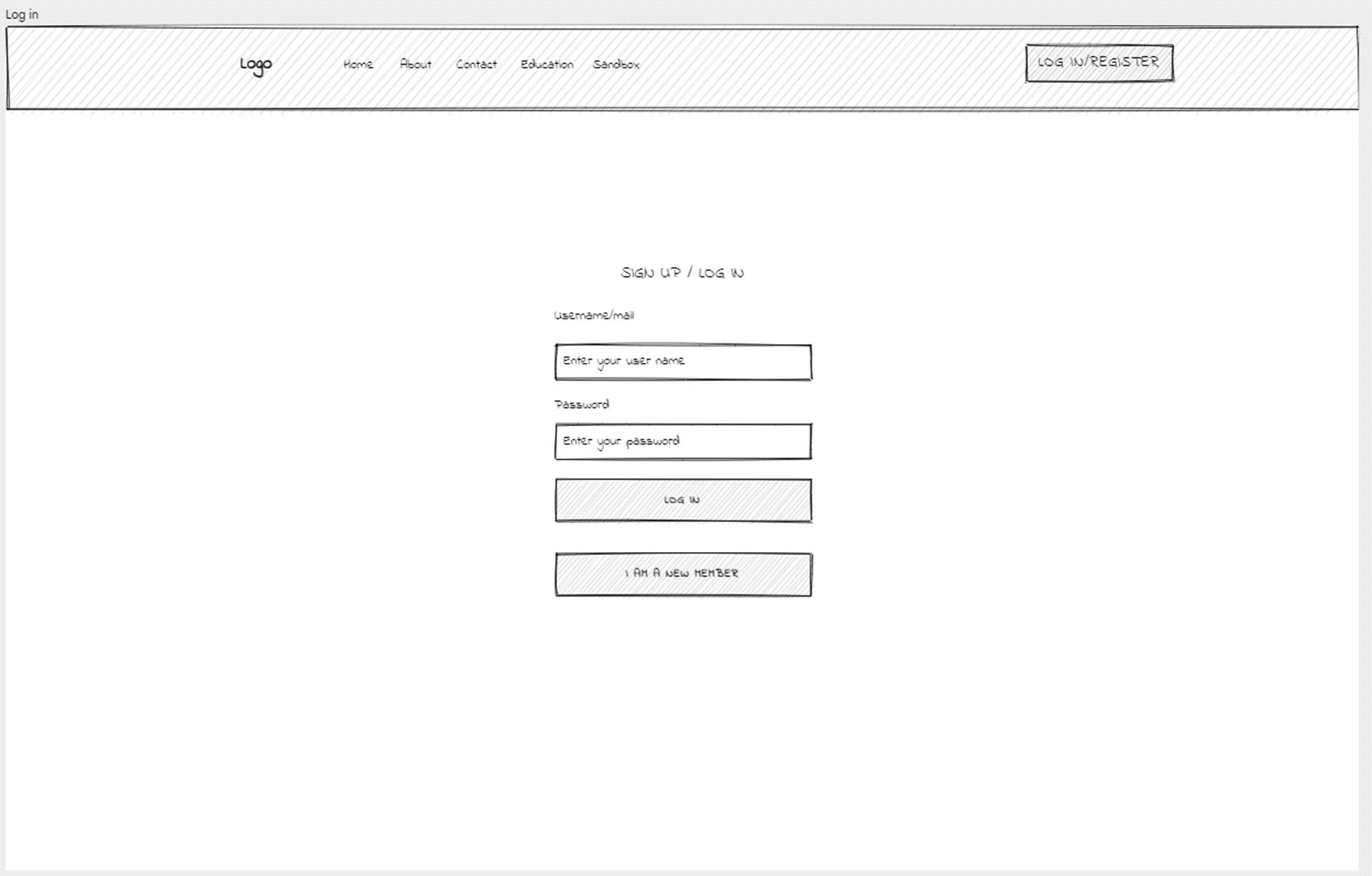


Рисунок 2. Страница авторизации

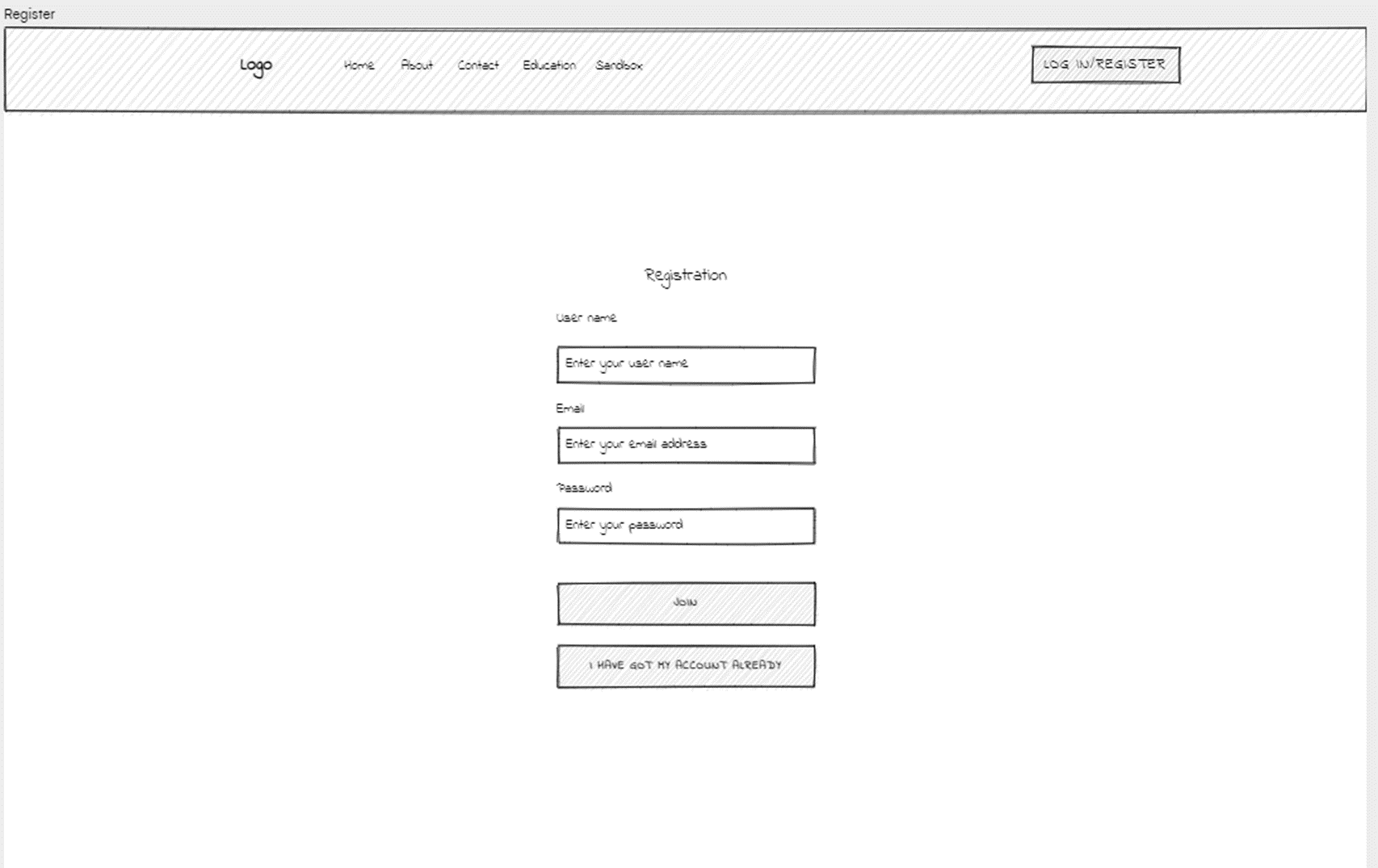


Рисунок 3. Страница регистрации

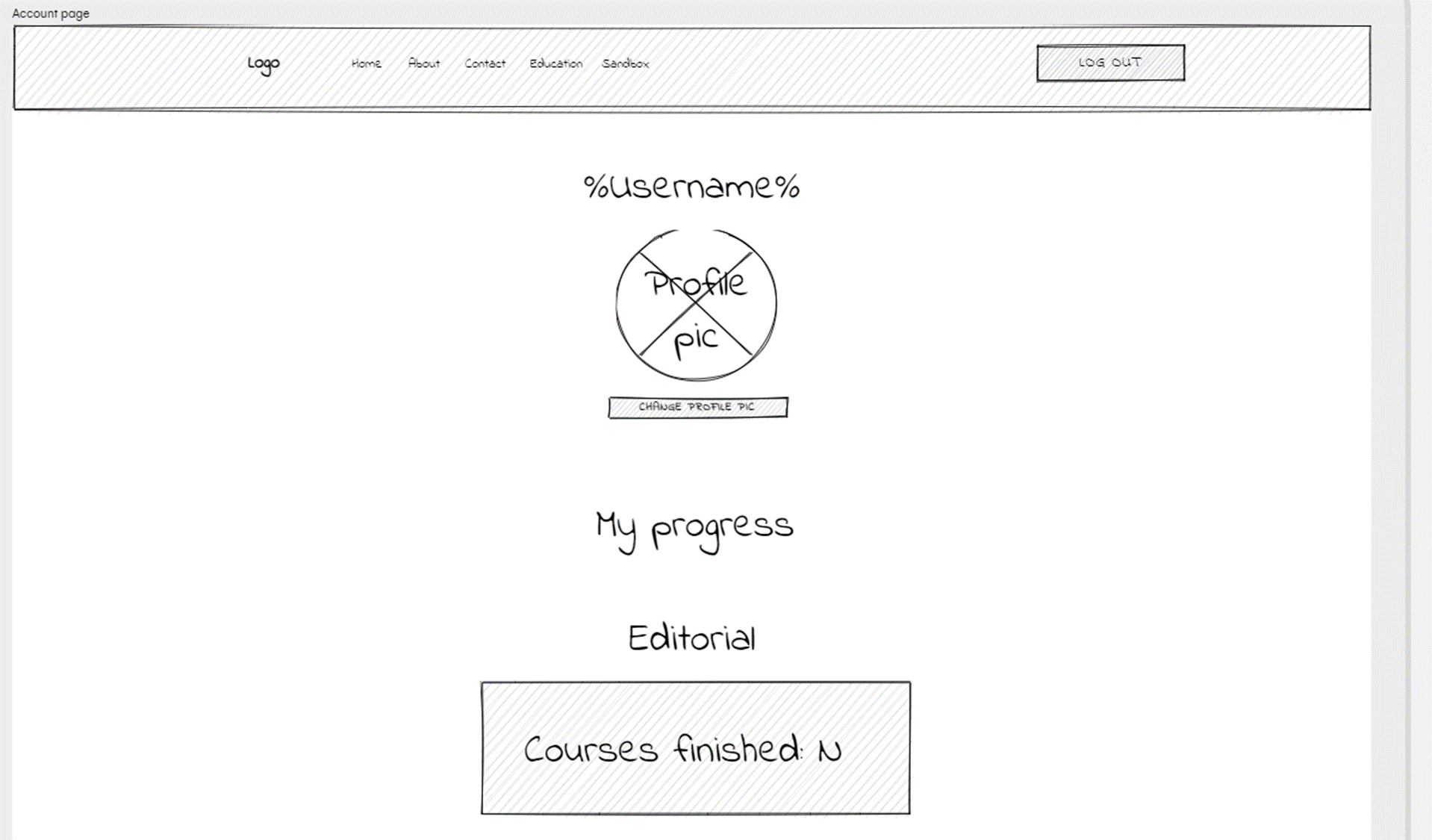


Рисунок 4. Страница профиля

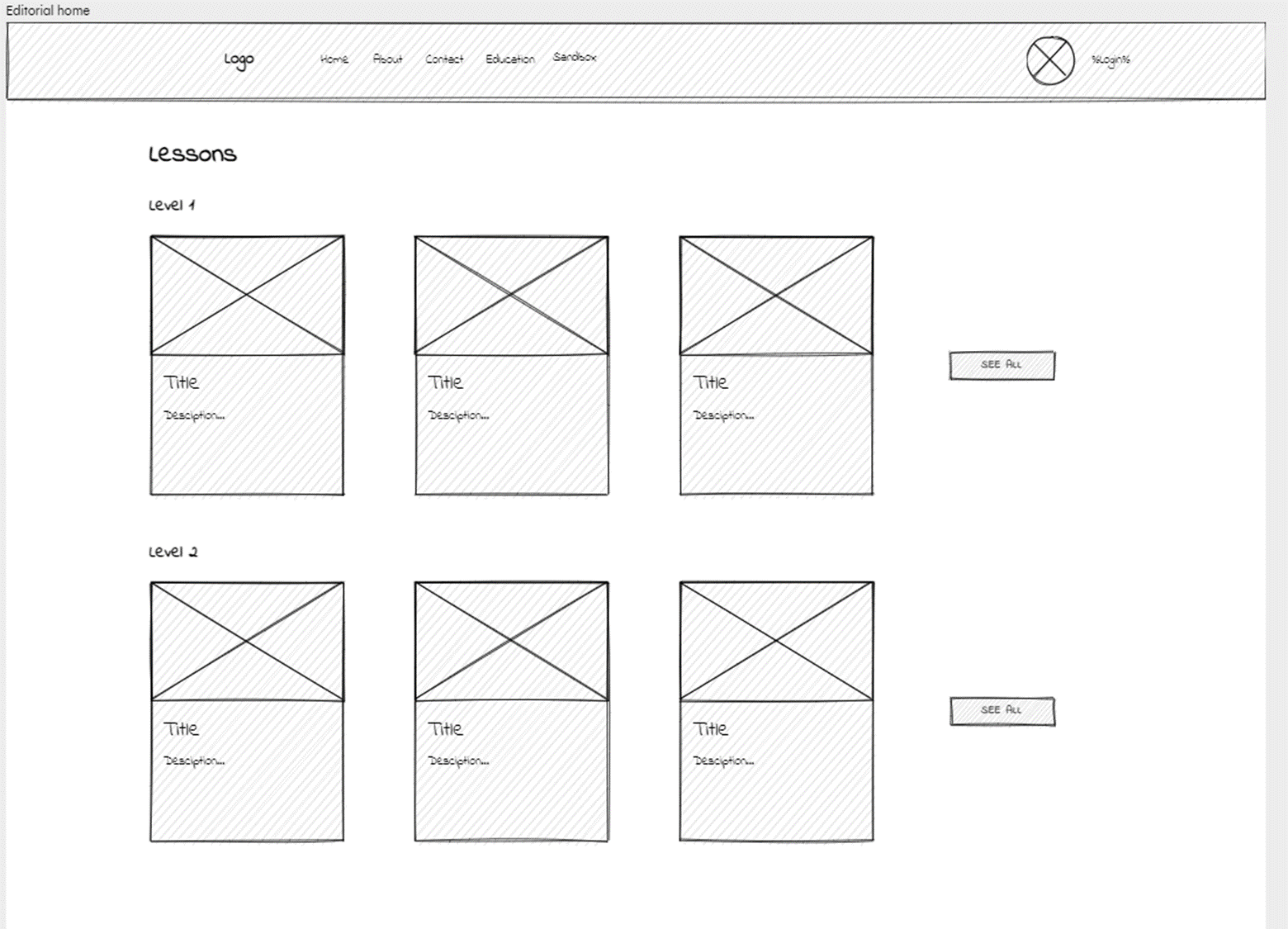


Рисунок 5. Страница выбора курсов

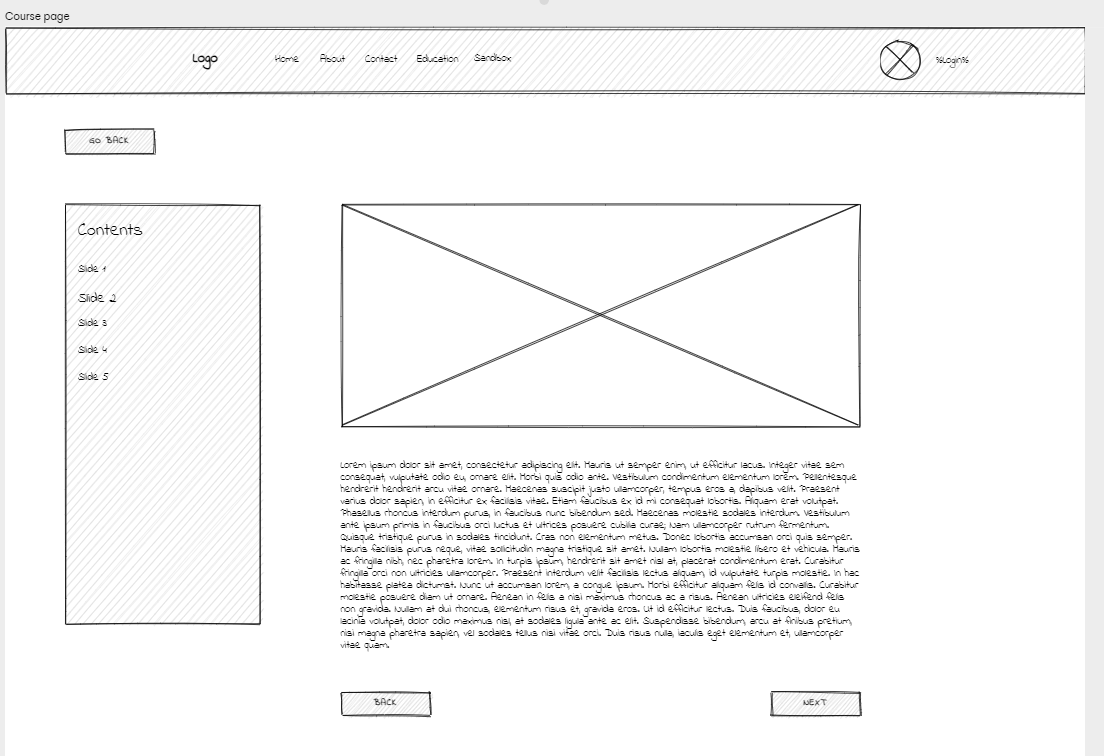


Рисунок 6. Страница курса (слайд 2)

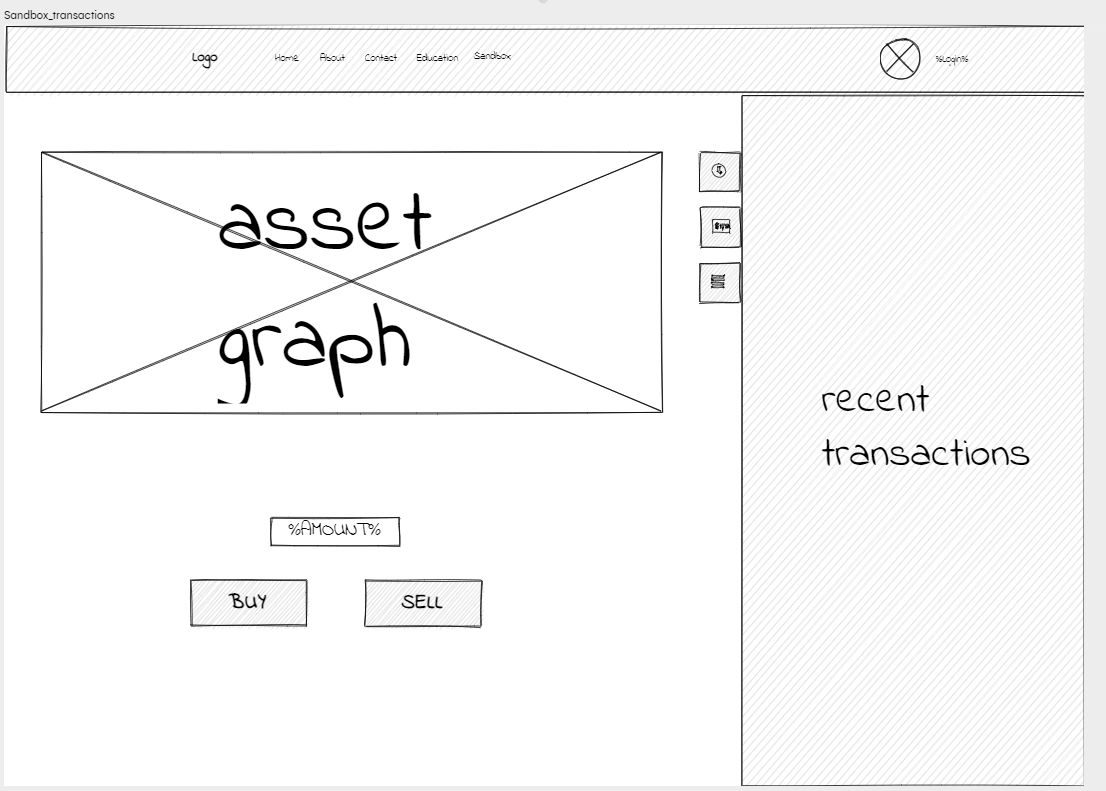


Рисунок 7. Страница «внутренней песочницы»

Заключение

В рамках данной работы были сформулированы цель и задачи проекта. Составлены функциональные и нефункциональные требования, созданы сценарии использования (use-cases) и сценарии тестирования (test-cases). Создан прототип всех основных окон программы.

Проект был реализован в тестовом режиме, с поддержкой всех указанных в работе требований. Код проекта находится в открытом доступе на GitHub и в Приложении 1.

В корне проекта распложены подробные инструкции по локальной сборке проекта (серверная и интерфейсная часть) через Docker.

Список литературы

1. "ГОСТ 34.602-89. Межгосударственный стандарт. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы"// М.: ИПК Издательство стандартов, 2004;

2. "ГОСТ 19.101-77\* (СТ СЭВ 1626-79). Государственный стандарт Союза ССР. Единая система программной документации. Виды программ и программных документов"// СПС КонсультантПлюс;

3. "IEEE Recommended Practice for Software Requirements Specifications," in IEEE Std 830-1998 , vol., no., pp.1-40, 20 Oct. 1998, doi: 10.1109/IEEESTD.1998.88286

4. Б.И. Заманский, Ф.Г. Кирдяшов «Основы системной инженерии»: учебник. – М.:Изд. Дом НИТУ «МИСиС», 2019.

Приложение 1. Листинг кода

Создание базы данных:

database\_dummy\_impl.py

from typing import List, Optional

from .idatabase import IDatabase

from ..model.user import User

from ..model.lesson import Lesson

from ..model.lesson\_data import LessonData

class DatabaseDummyImpl(IDatabase):

# auth

def get\_user(self, uid: str) -> Optional[User]:

print("get\_user")

return User.dummy()

def get\_user\_with\_username(self, username: str) -> Optional[User]:

print("get\_user\_with\_username")

return User.dummy()

def authorize\_user(self, username: str, email: str, password: str) -> Optional[User]:

print("authorize\_user")

return User.dummy()

# education

def get\_lesson(self, uid: str) -> Optional[Lesson]:

print("get\_lesson")

return Lesson.dummy()

def get\_all\_lessons(self) -> List[Lesson]:

print("get\_all\_lessons")

return [Lesson.dummy()]

def get\_lessons\_for\_level(self, level\_name: str) -> List[Lesson]:

print("get\_lessons\_for\_level")

return [Lesson.dummy()]

def get\_lesson\_data(self, uid: str) -> Optional[LessonData]:

print ("get\_lesson\_data")

return LessonData.dummy()

# sandbox

def sandbox\_init(self, user\_id: str, virtual\_start: str, balance: float) -> Optional[User]:

print("sandbox\_init")

return self.get\_user(user\_id)

def sandbox\_step(self, user\_id: str, virtual\_current: str) -> Optional[User]:

print("sandbox\_step")

return self.get\_user(user\_id)

def sandbox\_transaction(self, user\_id: str, ticker: str, price: float, amount: int,

operation\_type: str) -> Optional[User]:

print("sandbox\_transaction")

return self.get\_user(user\_id)

**database\_factory.py**

from enum import Enum

from .idatabase import IDatabase

from .database\_mongo\_impl import DatabaseMongoImpl

from .database\_dummy\_impl import DatabaseDummyImpl

class DatabaseType(Enum):

DUMMY = 0

MONGO = 1

class DatabaseFactory:

@staticmethod

def get(database\_type: DatabaseType) -> IDatabase:

if database\_type == DatabaseType.DUMMY:

return DatabaseDummyImpl()

elif database\_type == DatabaseType.MONGO:

return DatabaseMongoImpl()

**database\_mongo\_impl.py**

from typing import List, Optional

from pymongo import MongoClient

from bson.objectid import ObjectId

from .idatabase import IDatabase

from ..model.user import User

from ..model.lesson import Lesson

from ..model.lesson\_data import LessonData

from ..model.transaction import Transaction, OperationType

class DatabaseMongoImpl(IDatabase):

def \_\_init\_\_(self):

cluster = MongoClient(

"mongodb+srv://dbAdmin:StonkApps2021@cluster0.ww5af.mongodb.net/bulls-bears?retryWrites=true&w"

"=majority")

db = cluster["bulls-bears"]

self.\_lessons\_collection = db["lessons"]

self.\_users\_collection = db["users"]

self.\_data\_collection = db["data"]

# auth

def get\_user(self, uid: str) -> Optional[User]:

result = self.\_users\_collection.find\_one(ObjectId(uid))

if result is None:

return None

return User.from\_json(result)

def get\_user\_with\_username(self, username: str) -> Optional[User]:

result = self.\_users\_collection.find\_one({"username": username})

if result is None:

return None

return User.from\_json(result)

def authorize\_user(self, username: str, email: str, password: str) -> Optional[User]:

user = User(username, email, password, "", "", None, None)

inserted\_user = self.\_users\_collection.insert\_one(user.to\_json())

user.set\_user\_id(str(inserted\_user.inserted\_id))

return user

# education

def get\_lesson(self, uid: str) -> Optional[Lesson]:

result = self.\_lessons\_collection.find\_one(ObjectId(uid))

if result is None:

return None

return Lesson.from\_json(result)

def get\_all\_lessons(self) -> List[Lesson]:

results = self.\_lessons\_collection.find({})

return [Lesson.from\_json(result) for result in results]

def get\_lessons\_for\_level(self, level\_name: str) -> List[Lesson]:

results = self.\_lessons\_collection.find({"level\_name": level\_name})

return [Lesson.from\_json(result) for result in results]

def get\_lesson\_data(self, uid: str) -> Optional[LessonData]:

result = self.\_data\_collection.find\_one(ObjectId(uid))

if result is None:

return None

return LessonData.from\_json(result)

# sandbox

def sandbox\_init(self, user\_id: str, virtual\_start: str, balance: float) -> Optional[User]:

self.\_users\_collection.update\_one({'\_id': ObjectId(user\_id)},

{'$set': {

'sandbox\_data.virtual\_start': virtual\_start,

'sandbox\_data.virtual\_current': virtual\_start,

'sandbox\_data.balance': balance,

'sandbox\_data.assets': []

}}, upsert=False)

return self.get\_user(user\_id)

def sandbox\_step(self, user\_id: str, virtual\_current: str) -> Optional[User]:

self.\_users\_collection.update\_one({'\_id': ObjectId(user\_id)},

{'$set': {

'sandbox\_data.virtual\_current': virtual\_current,

}}, upsert=False)

return self.get\_user(user\_id)

def sandbox\_transaction(self, user\_id: str, ticker: str, price: float, amount: int,

operation\_type: str) -> Optional[User]:

user = self.get\_user(user\_id)

if user is None:

return None

sandbox\_data = user.sandbox\_data

commission = 0.0 # WARNING! HARDCODE

transaction = Transaction(amount, price, commission, sandbox\_data.virtual\_current, OperationType.from\_string(operation\_type))

did\_apply = sandbox\_data.apply\_transaction(ticker, transaction)

if did\_apply:

self.\_users\_collection.update\_one({'\_id': ObjectId(user\_id)},

{'$set': {

'sandbox\_data': sandbox\_data.to\_json(),

}}, upsert=False)

return self.get\_user(user\_id)

**idatabase.py**

from typing import List, Optional

from ..model.user import User

from ..model.lesson import Lesson

from ..model.lesson\_data import LessonData

class IDatabase:

# auth

def get\_user(self, uid: str) -> Optional[User]:

raise NotImplementedError()

def get\_user\_with\_username(self, username: str) -> Optional[User]:

raise NotImplementedError()

def authorize\_user(self, username: str,

email: str, password: str) -> Optional[User]:

raise NotImplementedError()

# education

def get\_lesson(self, uid: str) -> Optional[Lesson]:

raise NotImplementedError()

def get\_all\_lessons(self) -> List[Lesson]:

raise NotImplementedError()

def get\_lessons\_for\_level(self, level\_name: str) -> List[Lesson]:

raise NotImplementedError()

def get\_lesson\_data(self, uid: str) -> Optional[LessonData]:

raise NotImplementedError()

# sandbox

def sandbox\_init(self, user\_id: str, virtual\_start: str, balance: float) -> Optional[User]:

raise NotImplementedError()

def sandbox\_step(self, user\_id: str, virtual\_current: str) -> Optional[User]:

raise NotImplementedError()

def sandbox\_transaction(self, user\_id: str, ticker: str, price: float, amount: int,

operation\_type: str) -> Optional[User]:

raise NotImplementedError()

Работа с Flask

**\_\_init\_\_.py**

import os

from flask import Flask

from flask\_cors import CORS

from . import db, fin, auth

def create\_app():

app = Flask(\_\_name\_\_)

CORS(app)

SECRET\_KEY = os.environ.get("SECRET\_KEY")

if SECRET\_KEY is None:

raise RuntimeError("No SECRET\_KEY")

app.secret\_key = SECRET\_KEY

@app.route('/')

def hello\_world():

return 'Flask server for Bulls&Bears'

app.register\_blueprint(db.bp)

app.register\_blueprint(fin.bp)

app.register\_blueprint(auth.bp)

return app

**auth.py**

import os

from typing import Optional

from flask import Blueprint, g, request, session

from werkzeug.security import check\_password\_hash, generate\_password\_hash

from functools import wraps

import jwt

import datetime

from . import util

from . import db as database

from ..model.user import User

bp = Blueprint('auth', \_\_name\_\_, url\_prefix='/auth')

def token\_required(f):

@wraps(f)

def decorator(\*args, \*\*kwargs):

token = None

if 'x-access-tokens' in request.headers:

token = request.headers['x-access-tokens']

if not token:

return util.message\_to\_json("No valid token")

try:

SECRET\_KEY = os.environ.get('SECRET\_KEY')

if SECRET\_KEY is None:

return util.message\_to\_json("No SECRET\_KEY")

data = jwt.decode(token, SECRET\_KEY, algorithms="HS256")

user = database.get\_db().get\_user(data['user\_id'])

except:

return util.message\_to\_json("Token is invalid")

return f(user, \*args, \*\*kwargs)

return decorator

@bp.route('/register', methods=['POST'])

def register():

username = request.form['username']

email = request.form['email']

password = request.form['password']

db = database.get\_db()

error: Optional[str] = None

if not username:

error = 'Username is required.'

elif not password:

error = 'Password is required.'

elif not email:

error = 'Email is required.'

user = db.get\_user\_with\_username(username)

if user is not None:

error = f'User with username={username} already exists'

if error is None:

user = db.authorize\_user(username, email,

generate\_password\_hash(password))

if user is None:

error = 'Could not authorize user'

if error is None:

return util.message\_to\_json("Success"), 201

else:

return util.message\_to\_json(error), 418

@bp.route('/login', methods=['POST'])

def login():

username = request.form['username']

password = request.form['password']

db = database.get\_db()

error: Optional[str] = None

if not username:

error = 'Username is required.'

elif not password:

error = 'Password is required.'

user: Optional[User] = None

if error is None:

user = db.get\_user\_with\_username(username)

if user is None:

error = 'Incorrect username.'

elif not check\_password\_hash(user.password\_hash, password):

error = 'Incorrect password.'

if error is None:

session.clear()

session['user\_id'] = user.user\_id

SECRET\_KEY = os.environ.get('SECRET\_KEY')

if SECRET\_KEY is None:

return util.message\_to\_json("No SECRET\_KEY")

token = jwt.encode({'user\_id': user.user\_id,

'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)},

SECRET\_KEY, algorithm="HS256")

token\_str: str = ""

if isinstance(token, bytes):

token\_str = token.decode('UTF-8')

elif isinstance(token, str):

token\_str = token

else:

return util.message\_to\_json("Couldn't generate token"), 418

return {'token': token\_str}

else:

return util.message\_to\_json(error), 418

@bp.before\_app\_request

def load\_logged\_in\_user():

user\_id = session.get('user\_id')

if user\_id is None:

g.user = None

else:

g.user = database.get\_db().get\_user(user\_id)

@bp.route('/logout')

def logout():

session.clear()

return util.message\_to\_json("Success")

**db.py**

from flask import Blueprint, request, g

from . import util, auth

from ..database.database\_factory import \*

bp = Blueprint('db', \_\_name\_\_, url\_prefix='/db')

def get\_db():

if 'db' not in g:

g.db = DatabaseFactory.get(DatabaseType.MONGO)

return g.db

@bp.route('/get\_user')

@auth.token\_required

def get\_user(user):

return user.to\_json(), 200

@bp.route('/get\_all\_lessons')

def get\_all\_lessons():

db = get\_db()

lessons = db.get\_all\_lessons()

json\_data: dict = {"lessons": [l.to\_json() for l in lessons]}

return json\_data

@bp.route('/get\_lesson')

def get\_lesson():

db = get\_db()

uid: str = str(request.args.get("uid"))

lesson = db.get\_lesson(uid)

if lesson is None:

return util.message\_to\_json("Lesson not found"), 404

json\_data: dict = {"lesson": lesson.to\_json()}

return json\_data

@bp.route('/get\_lesson\_data')

def get\_lesson\_data():

db = get\_db()

uid: str = str(request.args.get("uid"))

lesson\_data = db.get\_lesson\_data(uid)

if lesson\_data is None:

return util.message\_to\_json("Lesson data not found"), 404

json\_data: dict = {"lesson\_data": lesson\_data.to\_json()}

return json\_data

@bp.route('/sandbox\_init', methods=['POST'])

@auth.token\_required

def sandbox\_init(user):

db = get\_db()

virtual\_start: str = str(request.form["virtual\_start"])

balance: float = float(request.form["balance"])

user = db.sandbox\_init(user.user\_id, virtual\_start, balance)

if user is None:

return util.message\_to\_json("Couldn't initialize sandbox"), 401

return user.to\_json(), 201

@bp.route('/sandbox\_step', methods=['POST'])

@auth.token\_required

def sandbox\_step(user):

db = get\_db()

virtual\_current: str = str(request.form["virtual\_current"])

user = db.sandbox\_step(user.user\_id, virtual\_current)

if user is None:

return util.message\_to\_json("Couldn't perform step in sandbox"), 401

return user.to\_json(), 201

@bp.route('/sandbox\_transaction', methods=['POST'])

@auth.token\_required

def sandbox\_transaction(user):

db = get\_db()

ticker: str = str(request.form["ticker"])

price: float = float(request.form["price"])

amount: int = int(request.form["amount"])

operation\_type: str = str(request.form["operation\_type"])

user = db.sandbox\_transaction(user.user\_id, ticker, price, amount,

operation\_type)

if user is None:

return util.message\_to\_json("Couldn't perform transaction in sandbox"), 401

return user.to\_json(), 201

**fin.py**

from flask import Blueprint, request

from . import util

from ..ticker\_warehouse.ticker\_warehouse import TickerWarehouse

bp = Blueprint('fin', \_\_name\_\_, url\_prefix='/fin')

tw = TickerWarehouse()

@bp.route('/get\_available\_tickers')

def get\_available\_tickers():

return {"tickers": list(tw.get\_tickers\_set())}

@bp.route('/get\_all\_ticker\_history')

def get\_all\_ticker\_history():

ticker: str = str(request.args.get("ticker"))

json\_data = tw.get\_ticker\_history\_as\_json(ticker)

if json\_data is None:

return util.message\_to\_json("Ticker history not Found"), 404

return json\_data

@bp.route('/get\_ticker\_history')

def get\_ticker\_history():

ticker: str = str(request.args.get("ticker"))

begin: int = int(str(request.args.get("begin")))

end: int = int(str(request.args.get("end")))

json\_data = tw.get\_ticker\_history\_in\_range\_json(ticker, begin, end)

if json\_data is None:

return util.message\_to\_json("Ticker history not Found"), 404

return json\_data

**util.py**

def message\_to\_json(message: str) -> dict:

return {"message": message}

****Модель****

**asset.py**

from typing import List

from .transaction import Transaction

class Asset:

\_ticker: str = ""

\_amount: int = 0

\_transactions: List[Transaction] = []

def \_\_init\_\_(self, ticker: str, amount: int, transactions: List[Transaction]):

self.\_ticker = ticker

self.\_amount = amount

self.\_transactions = transactions

@property

def ticker(self) -> str:

return self.\_ticker

@property

def amount(self) -> int:

return self.\_amount

@amount.setter

def amount(self, value: int):

self.\_amount = value

@property

def transactions(self) -> List[Transaction]:

return self.\_transactions

def add\_transaction(self, transaction: Transaction):

self.\_transactions.append(transaction)

@classmethod

def from\_json(cls, json\_data):

ticker = json\_data["ticker"]

amount = json\_data["amount"]

transactions = [Transaction.from\_json(t) for t in json\_data["transactions"]]

return cls(ticker, amount, transactions)

@classmethod

def dummy(cls):

return cls("", 0, [])

def to\_json(self):

json\_data = {

"ticker": self.\_ticker,

"amount": self.\_amount,

"transactions": [t.to\_json() for t in self.\_transactions]

}

return json\_data

**edudata.py**

from typing import List

class EduData:

\_score: int = 0

\_in\_progress: List[str] = [] # maybe LessonProgress class?

\_done: List[str] = []

\_id: str = ""

def \_\_init\_\_(self, score: int, in\_progress: List[str], done: List[str], uid: str):

self.\_score = score

self.\_in\_progress = in\_progress

self.\_done = done

self.\_id = uid

@classmethod

def from\_json(cls, json\_data):

if json\_data is None:

return None

score = json\_data["score"]

in\_progress = json\_data["in\_progress"]

done = json\_data["done"]

uid = str(json\_data["\_id"])

return cls(score, in\_progress, done, uid)

@classmethod

def dummy(cls):

return cls(0, [], [], "")

def to\_json(self):

json\_data = {

"score": self.\_score,

"in\_progress": self.\_in\_progress,

"done": self.\_done,

"\_id": self.\_id

}

return json\_data

**lesson.py**

class Lesson:

\_level\_name: str = ""

\_index: int = -1 # lesson index in a level

\_title: str = ""

\_description: str = ""

\_photo\_url: str = ""

\_data: str = "" # id of lesson\_data

\_id: str = ""

def \_\_init\_\_(self, level\_name: str, index: int, title: str,

description: str, photo\_url: str, data: str, uid: str):

self.\_level\_name = level\_name

self.\_index = index

self.\_title = title

self.\_description = description

self.\_photo\_url = photo\_url

self.\_data = data

self.\_id = uid

@classmethod

def from\_json(cls, json\_data):

level\_name = json\_data["level\_name"]

index = json\_data["index"]

title = json\_data["title"]

description = json\_data["description"]

photo\_url = json\_data["photo\_url"]

data = str(json\_data["data"])

uid = str(json\_data["\_id"])

return cls(level\_name, index, title, description, photo\_url, data, uid)

@classmethod

def dummy(cls):

return cls("", -1, "", "", "", "", "")

def to\_json(self):

json\_data = {

"level\_name": self.\_level\_name,

"index": self.\_index,

"title": self.\_title,

"description": self.\_description,

"photo\_url": self.\_photo\_url,

"data": self.\_data,

"\_id": self.\_id

}

return json\_data

lesson\_data.py

from typing import List

from .lesson\_slide import LessonSlide

class LessonData:

\_slides: List[LessonSlide] = []

\_id: str = ""

def \_\_init\_\_(self, slides: List[LessonSlide], uid: str):

self.\_slides = slides

self.\_id = uid

@classmethod

def from\_json(cls, json\_data):

slides = [LessonSlide.from\_json(s) for s in json\_data["slides"]]

uid = str(json\_data["\_id"])

return cls(slides, uid)

@classmethod

def dummy(cls):

return cls([], "")

def to\_json(self):

json\_data = {

"slides": [s.to\_json() for s in self.\_slides],

"\_id": self.\_id

}

return json\_data

**lesson\_slide.py**

from typing import List

class LessonSlide:

\_text: str = ""

\_media: List[str] = []

\_slide\_number: int = 0

def \_\_init\_\_(self, text: str, media: List[str],

slide\_number: int):

self.\_text = text

self.\_media = media

self.\_slide\_number = slide\_number

@classmethod

def from\_json(cls, json\_data):

text = json\_data["text"]

media = json\_data["media"]

slide\_number = json\_data["slide\_number"]

return cls(text, media, slide\_number)

@classmethod

def dummy(cls):

return cls("", [], -1)

def to\_json(self):

json\_data = {

"text": self.\_text,

"media": self.\_media,

"slide\_number": self.\_slide\_number

} return json\_data

sandboxdata.py

from typing import List, Optional

from .asset import Asset

from .transaction import Transaction, OperationType

class SandboxData:

\_virtual\_start: str = ""

\_virtual\_current: str = ""

\_balance: float = 0.0

\_assets: List[Asset] = []

def \_\_init\_\_(self, virtual\_start: str, virtual\_current: str, balance: float, assets: List[Asset]):

self.\_virtual\_start = virtual\_start

self.\_virtual\_current = virtual\_current

self.\_balance = balance

self.\_assets = assets

@property

def virtual\_current(self) -> str:

return self.\_virtual\_current

@virtual\_current.setter

def virtual\_current(self, value: str):

self.\_virtual\_current = value

@property

def balance(self) -> float:

return self.\_balance

@balance.setter

def balance(self, value: float):

self.\_balance = value

@property

def assets(self) -> List[Asset]:

return self.\_assets

@classmethod

def from\_json(cls, json\_data):

if json\_data is None:

return None

virtual\_start = json\_data["virtual\_start"]

virtual\_current = json\_data["virtual\_current"]

balance = json\_data["balance"]

assets = [Asset.from\_json(a) for a in json\_data["assets"]]

return cls(virtual\_start, virtual\_current, balance, assets)

@classmethod

def dummy(cls):

return cls("", "", 0.0, [])

def to\_json(self):

json\_data = {

"virtual\_start": self.\_virtual\_start,

"virtual\_current": self.\_virtual\_current,

"balance": self.\_balance,

"assets": [a.to\_json() for a in self.\_assets]

}

return json\_data

def apply\_transaction(self, ticker: str, transaction: Transaction) -> bool:

if int(transaction.timestamp) > int(self.\_virtual\_current):

return False

current\_asset: Optional[Asset] = None

for asset in self.\_assets:

if asset.ticker == ticker:

current\_asset = asset

break

if transaction.operation\_type == OperationType.SELL:

if current\_asset is None or current\_asset.amount < transaction.amount:

return False

current\_asset.amount -= transaction.amount

if transaction.operation\_type == OperationType.BUY:

if self.\_balance + transaction.sum() < 0:

return False

if current\_asset is None:

current\_asset = Asset(ticker, 0, [])

self.\_assets.append(current\_asset)

current\_asset.amount += transaction.amount

current\_asset.add\_transaction(transaction)

for asset in self.\_assets:

if asset.ticker == ticker:

asset = current\_asset

self.\_balance += transaction.sum()

return True

**transaction.py**

from enum import Enum

class OperationType(Enum):

BUY = 0

SELL = 1

@classmethod

def from\_string(cls, value: str):

if value == "BUY":

return cls.BUY

elif value == "SELL":

return cls.SELL

else:

return cls.BUY

def operation\_type\_to\_string(operation\_type: OperationType) -> str:

if operation\_type == OperationType.BUY:

return "BUY"

elif operation\_type == OperationType.SELL:

return "SELL"

else:

return "BUY"

class Transaction:

\_amount: int = 0

\_price: float = 0.0

\_commission: float = 0.0

\_timestamp: str = ""

\_operation\_type: OperationType = OperationType.BUY

def \_\_init\_\_(self, amount: int, price: float, commission: float, timestamp: str, operation\_type: OperationType):

self.\_amount = amount

self.\_price = price

self.\_commission = commission

self.\_timestamp = timestamp

self.\_operation\_type = operation\_type

@property

def amount(self) -> int:

return self.\_amount

@property

def timestamp(self) -> str:

return self.\_timestamp

@property

def operation\_type(self) -> OperationType:

return self.\_operation\_type

@classmethod

def from\_json(cls, json\_data):

amount = json\_data["amount"]

price = json\_data["price"]

commission = json\_data["commission"]

timestamp = json\_data["timestamp"]

operation\_type = OperationType.from\_string(json\_data["operation\_type"])

return cls(amount, price, commission, timestamp, operation\_type)

@classmethod

def dummy(cls):

return cls(0, 0.0, 0.0, "", OperationType.BUY)

def to\_json(self):

json\_data = {

"amount": self.\_amount,

"price": self.\_price,

"commission": self.\_commission,

"timestamp": self.\_timestamp,

"operation\_type": operation\_type\_to\_string(self.\_operation\_type)

}

return json\_data

def sum(self) -> float:

\_sum: float = self.\_price \* self.\_amount

if self.\_operation\_type == OperationType.BUY:

\_sum = -\_sum - self.\_commission

else:

\_sum = \_sum - self.\_commission

return \_sum

**user.py**

from typing import Optional

from ..model.sandboxdata import SandboxData

from ..model.edudata import EduData

class User:

\_username: str = ""

\_email: str = ""

\_password: str = ""

\_id: str = ""

\_token: str = ""

\_edu\_data: EduData = EduData.dummy()

\_sandbox\_data: SandboxData = SandboxData.dummy()

def \_\_init\_\_(self, username: str, email: str, password: str, uid: str, token: str,

edu\_data: Optional[EduData], sandbox\_data: Optional[SandboxData]):

self.\_username = username

self.\_email = email

self.\_password = password

self.\_id = uid

self.\_token = token

if edu\_data is not None:

self.\_edu\_data = edu\_data

if sandbox\_data is not None:

self.\_sandbox\_data = sandbox\_data

@property

def password\_hash(self) -> str:

return self.\_password

@property

def user\_id(self) -> str:

return self.\_id

@property

def sandbox\_data(self) -> SandboxData:

return self.\_sandbox\_data

def set\_user\_id(self, uid: str):

self.\_id = uid

@classmethod

def from\_json(cls, json\_data):

username = json\_data["username"]

email = json\_data["email"]

password = json\_data["password"]

uid = str(json\_data["\_id"])

token = json\_data["token"]

edu\_data = EduData.from\_json(json\_data["edu\_data"])

sandbox\_data = SandboxData.from\_json(json\_data["sandbox\_data"])

return cls(username, email, password, uid, token,

edu\_data, sandbox\_data)

@classmethod

def dummy(cls):

return cls("", "", "", "", "", None, None)

def to\_json(self):

json\_data = {

"username": self.\_username,

"email": self.\_email,

"password": self.\_password,

"token": self.\_token,

"edu\_data": self.\_edu\_data.to\_json(),

"sandbox\_data": self.\_sandbox\_data.to\_json()

}

return json\_data

Работа «финансовой песочницы»

**mongo\_demo.py**

from pymongo import MongoClient

# init

cluster = MongoClient("mongodb+srv://dbAdmin:StonkApps2021@cluster0.ww5af.mongodb.net/bulls-bears?retryWrites=true&w"

"=majority")

db = cluster["bulls-bears"]

collection = db["data"]

# generate some data

post = {

"app\_name": "Bulls&Bears",

"company\_name": "StonkApps",

"year": 2021

}

# insert in db

insert\_result = collection.insert\_one(post)

print(insert\_result.inserted\_id)

# search

search\_results = collection.find({"year": 2021})

for result in search\_results:

print(result)

# remove

delete\_result = collection.delete\_one({"\_id": insert\_result.inserted\_id})

print("Deleted", delete\_result.deleted\_count, "documents")

# lesson insertion example

lesson = {

"level\_name": "Level 1",

"index": 0,

"title": "Introduction",

"description": "Basic introduction to a course",

"data": "Lesson content should be here..."

}

lessons\_collection = db["lessons"]

inserted\_lesson = lessons\_collection.insert\_one(lesson)

search\_result = lessons\_collection.find\_one({"\_id": inserted\_lesson.inserted\_id})

print(search\_result)

lessons\_collection.delete\_one({"\_id": inserted\_lesson.inserted\_id}) # delete to keep db clean

**yfinance\_demo.py**

import yfinance as yf

msft = yf.Ticker("MSFT")

info = msft.info

print(info.keys()) # json

hist = msft.history(start="2020-12-01", end="2021-01-01", interval="1h")

print(hist) # pandas dataframe

Warehouse

**ticker\_list.json**

{

"Tickers":

[

"MSFT",

"MMM",

"GOOGL",

"AMZN",

"AXP",

"BAC",

"KO",

"COST",

"DIS",

"GS",

"HD",

"IBM",

"JNJ",

"MCD",

"AAPL",

"NKE",

"SBUX",

"VZ",

"V",

"WMT"

]

}

**ticker\_warehouse.py**

from datetime import datetime

import json

import os

import yfinance as yf

import pandas as pd

from typing import Dict, List, Optional

from pandas import Series

class TickerWarehouse:

\_ticker\_info = {}

\_ticker\_history: Dict[str, pd.DataFrame] = {}

\_ticker\_set = set()

\_load\_file = "ticker\_list.json"

def \_\_init\_\_(self):

self.\_import\_start()

def \_import\_ticker(self, ticker\_string: str):

if ticker\_string not in self.\_ticker\_set:

ticker = yf.Ticker(ticker\_string)

# todo: can be deleted for speed

self.\_ticker\_info[ticker\_string] = ticker

self.\_ticker\_history[ticker\_string] = ticker.history(period="max", interval="1d")

self.\_ticker\_set.add(ticker\_string)

# for initialization

def \_import\_start(self):

cur\_dir = os.path.dirname(\_\_file\_\_)

file\_open = os.path.join(cur\_dir, self.\_load\_file)

with open(file\_open) as json\_file:

json\_data = json.load(json\_file)

for ticker\_string in json\_data["Tickers"]:

self.\_import\_ticker(ticker\_string)

# tickers set

def get\_tickers\_set(self) -> set:

return self.\_ticker\_set

# get dictionary of all tickers str->DataFrame

def get\_all\_history(self) -> Dict[str, pd.DataFrame]:

return self.\_ticker\_history

def get\_ticker\_history\_in\_range\_df(self, ticker\_name: str, start, finish, is\_timestamp: bool = True) -> Optional[pd.DataFrame]:

if(ticker\_name not in self.\_ticker\_set):

#todo: log

return None

datetime\_start = start

datetime\_finish = finish

if is\_timestamp:

datetime\_start = datetime.fromtimestamp(start).date()

datetime\_finish = datetime.fromtimestamp(finish).date()

retval\_df = self.\_ticker\_history[ticker\_name].loc[datetime\_start:datetime\_finish]

return retval\_df

def get\_ticker\_history\_in\_range\_json(self, ticker\_name: str, start, finish, is\_timestamp: bool = True) -> Optional[str]:

df\_ans = self.get\_ticker\_history\_in\_range\_df(ticker\_name, start, finish, is\_timestamp)

if(df\_ans is None):

return None

return df\_ans.to\_json(orient="index", date\_unit="s")

def get\_ticker\_history(self, ticker\_name: str) -> Optional[pd.DataFrame]:

return self.get\_ticker\_history\_in\_range\_df(ticker\_name, "1700-01-01", "2050-01-01", is\_timestamp=False)

def get\_ticker\_history\_as\_json(self, ticker\_name: str) -> Optional[str]:

return self.get\_ticker\_history\_in\_range\_json(ticker\_name, "1700-01-01", "2050-01-01", is\_timestamp=False)

def get\_data\_for\_ticker\_at\_time\_as\_json(self, ticker\_name: str, time, is\_timestamp: bool = True) -> Optional[str]:

single\_frame = self.get\_ticker\_history\_in\_range\_df(ticker\_name, time, time, is\_timestamp=is\_timestamp)

if (single\_frame.size != 7):

return None

return single\_frame.iloc[0].to\_json(orient="index")

Основная часть

**main.py**

import os

from .flask import create\_app

if \_\_name\_\_ == '\_\_main\_\_':

port = os.getenv("PORT", 5000)

app = create\_app()

app.run(debug=False, host='0.0.0.0', port=port)

Dockerfile

FROM ubuntu:latest

RUN apt-get update -y

RUN apt-get install -y python3-pip python3-dev build-essential

COPY ./requirements.txt /app/

WORKDIR /app

RUN pip3 install -r requirements.txt

COPY . /app

ARG secret\_key

ENV SECRET\_KEY=$secret\_key

CMD ["python3", "-m", "src.main"]