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

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 техническому обеспечению

Система должна быть рассчитана на использование с помощью персональных компьютеров.

Поддерживаемый браузер: Google Chrome.

Поддерживаемый язык интерфейса - английский.

Минимальное разрешение экрана: 1024 x 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:

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

5a.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:

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

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

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

4а.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:
    a. username (имя пользователя);
    b. profile pic (изображение профиля);
    c. кнопка "change profile pic" (сменить изображение профиля);
    d. My progress (прогресс пользователя):
    e. Editorial (информация о количестве пройденных учебных курсов);
    f. Sandbox (статистика: самая дорогостоящая сделка, потраченное время, наилучший запас)
    g. Log out (выход из профиля).

Альтернативный сценарий:

2а. 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.

4a. Пользователь нажимает кнопку "Go back"

4a.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:

2а. Пользователь вводит неверное число

2а.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:

2а. Пользователь заполняет не все предложенные поля

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

4а. Произошел сбой в системе и заявка не зарегистрирована

4а.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.

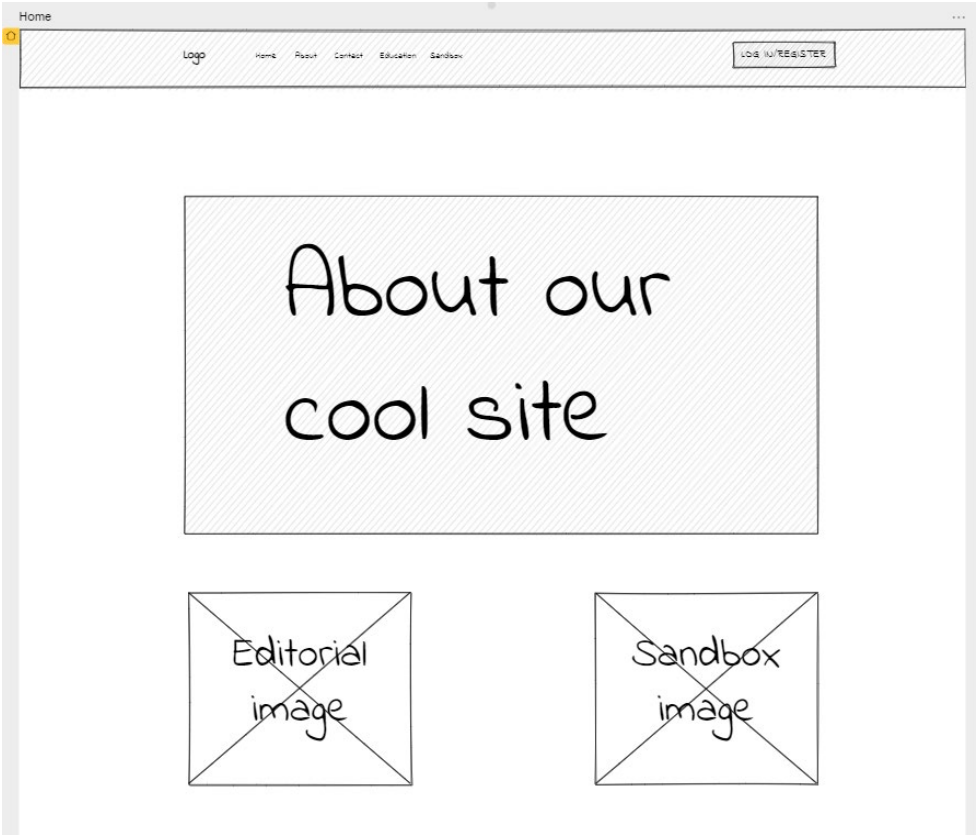

Рисунок 1. Главная страница веб-сайта

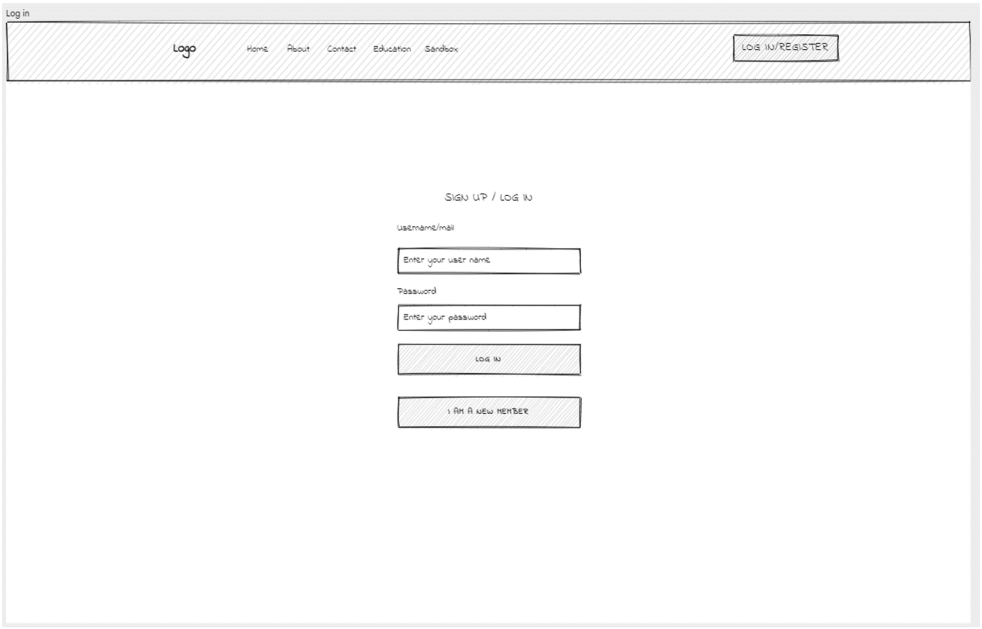

Рисунок 2. Страница авторизации

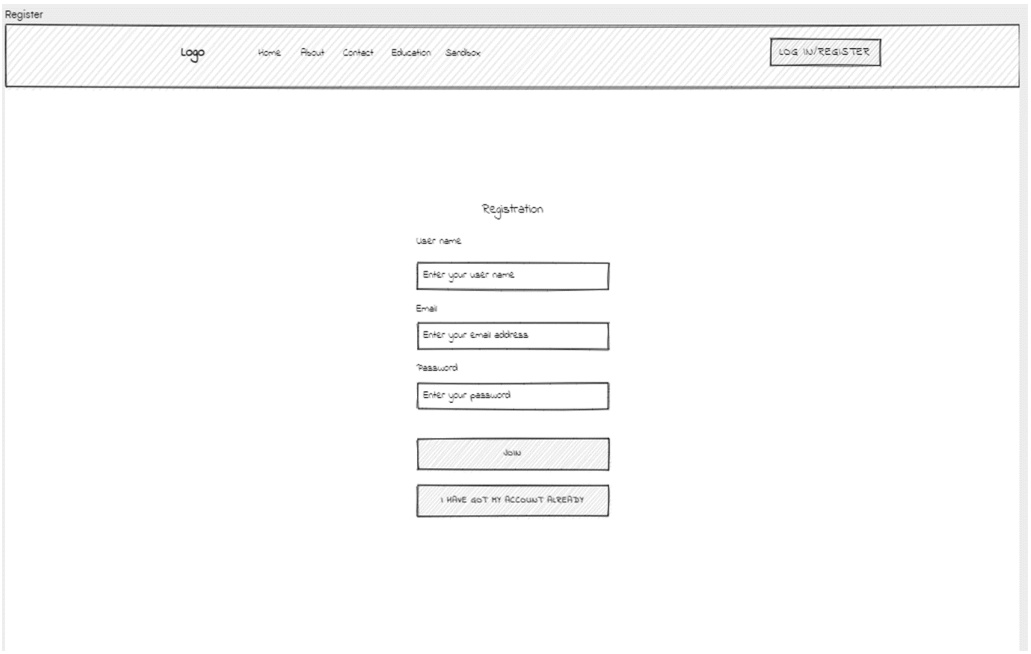

Рисунок 3. Страница регистрации

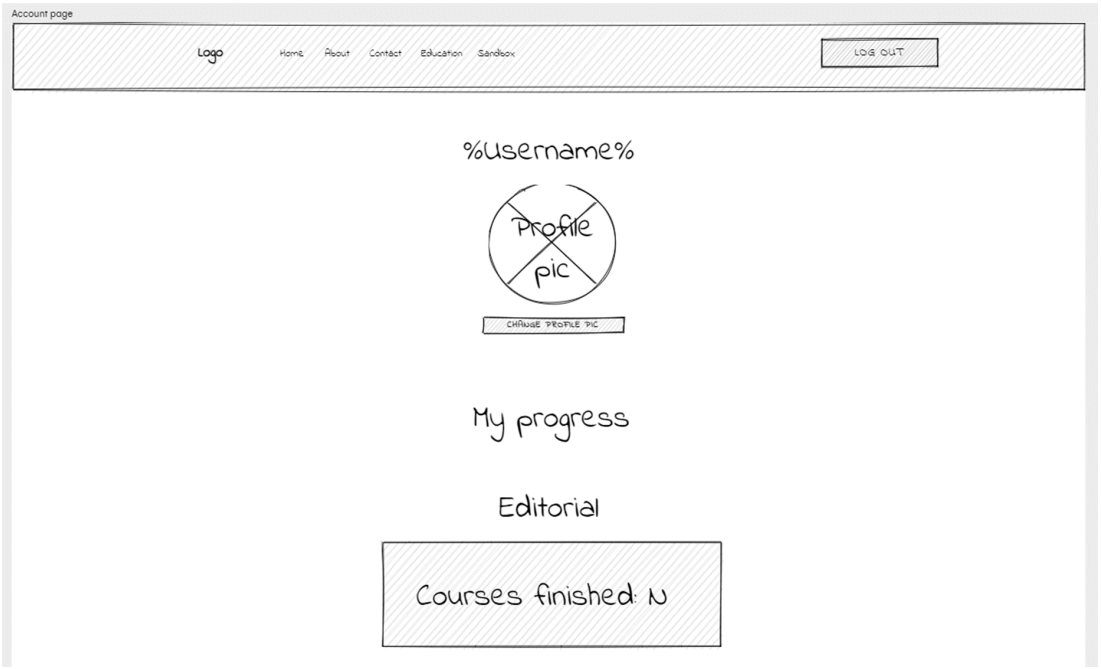

Рисунок 4. Страница профиля

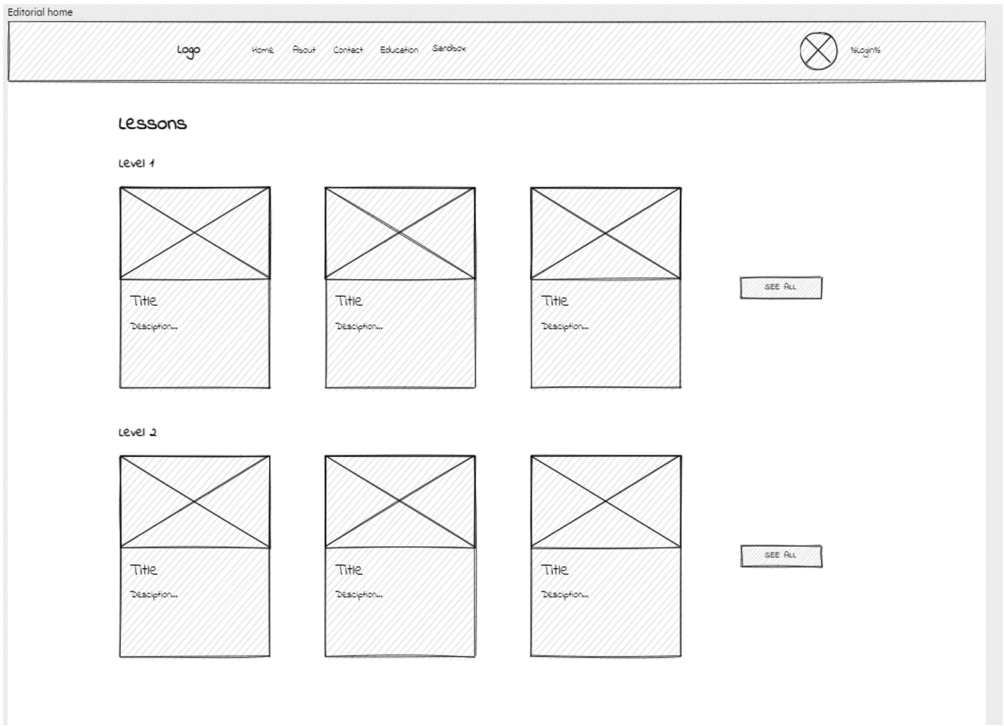

Рисунок 5. Страница выбора курсов

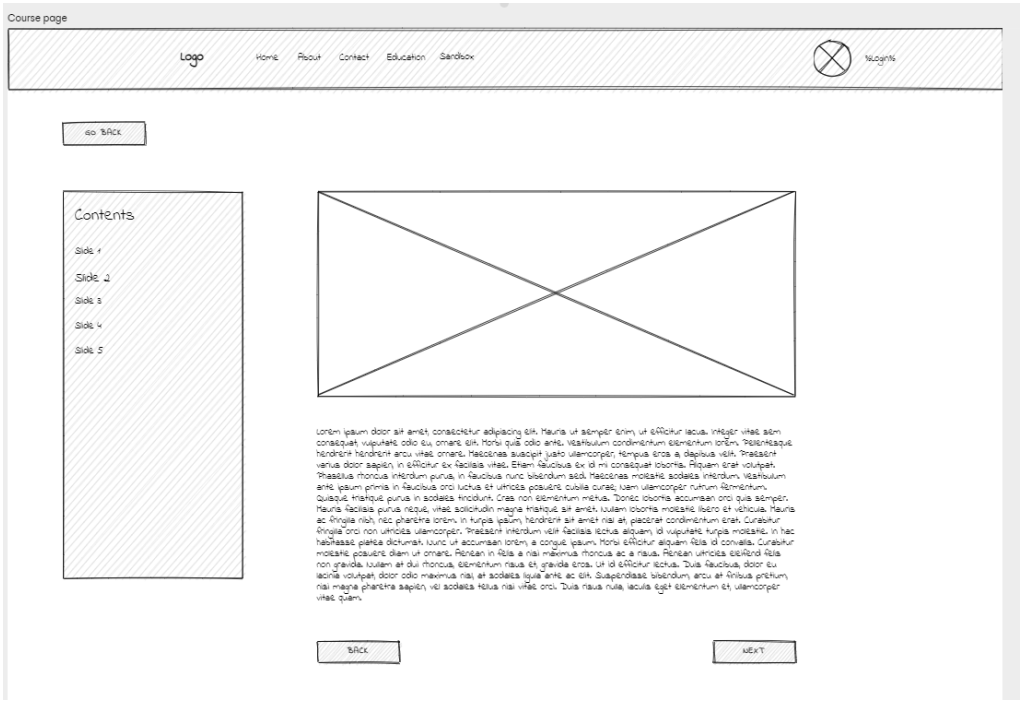

Рисунок 6. Страница курса (слайд 2)

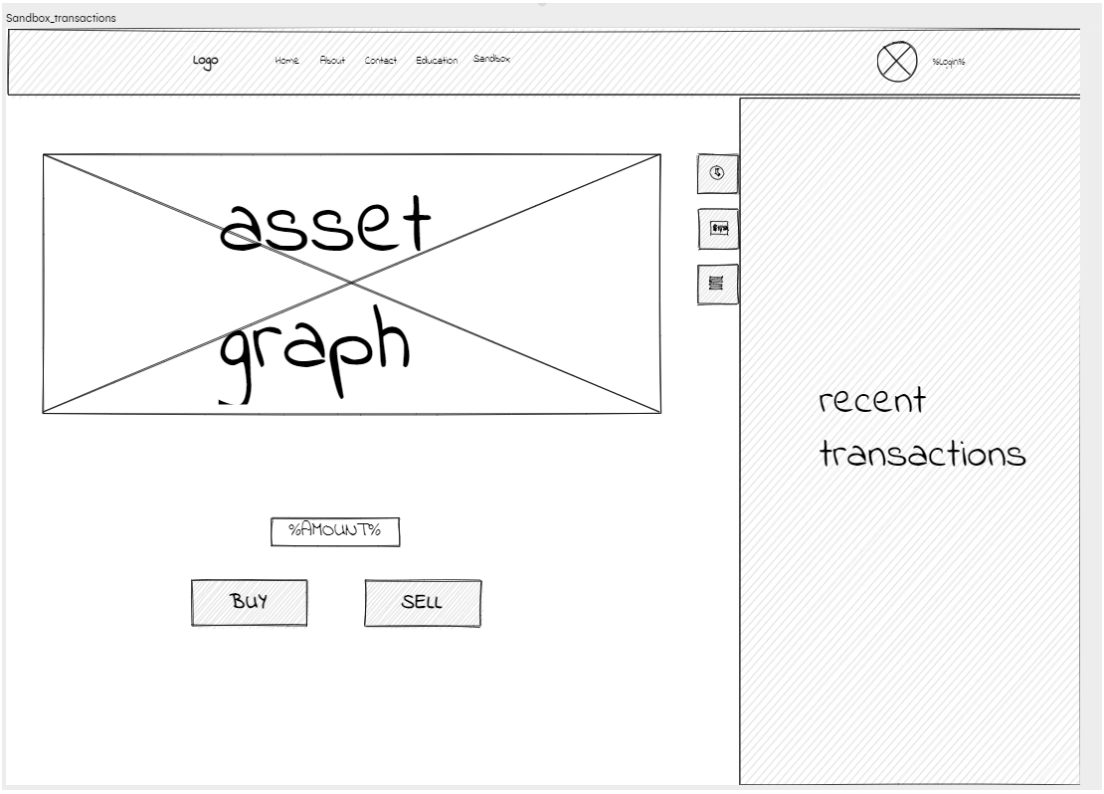

Рисунок 7. Страница «внутренней песочницы»

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"]
```