Сайт по предмету “Теория автоматов”

Оглавление

Введение

Задачей данного проекта является реализация полноценной электронной среды для упрощения взаимодействия студентов и преподавателей на курсе “Теория автоматов”. Представленная ниже документация описывает реализацию той части функционала, которая была уже реализована.

Для решения поставленной задачи было написано веб-приложение, разделенное на 2 логические части: фронтенд и бэкенд. Для разработки фронтенда была выбрана концепция SPA приложения. В качестве ключевых технологий разработки использовались React, Redux и MaterialUI. Для разработки бэкенда использовалась концепция микросервисной архитектуры в сочетании с такими технологиями как Go, PostgreSql, Redis. Также, для поддержания кроссплатформенности, использовалась система контейнеризации Docker Compose.

Далее будет описаны специфика и детали реализации веб-приложения, а также, идеи для дальнейшего развития и доработки проекта.

1. Общая архитектура фронтенда

Исходный код расположен в папке ta_eos/web/dev/src и разбит на следующие папки:

- components - папка, к которой располагаются React-компоненты приложения
- hooks - папка, где располагаются кастомные хуки для упрощения работы с Redux и jss
- Redux - папка, в которой расположен код для взаимодействия с Redux, т.е. вся работа с данными в приложении
- data - папка, в которой расположены модели приложения, список методов api и сетевые запросы для взаимодействия с api

- utils - папка, в которой расположены вспомогательные модули приложения(например, валидатор для входный ланных)
- также, в папке расположены корневые файлы для сборки приложения. Такие как App.tsx, index.css, index.tsx b setupProxy.js(прокси для обеспечения удобной разработки локально)

Также, в папке ta_eos/web/dev расположены конфигурационные файлы package.json, tsconfig.json, обеспечивающие работу webpack.

Фронтенд приложения построен на следующих технологиях:

- React - библиотека для разработки пользовательских интерфейсов
- MaterialUI — библиотека визуальных компонентов
- Redux - менеджер состояний приложения
- Typescript - язык разработки
- Yarn - менеджер пакетов
- Docker - система контейнеризации

1.1 Proxy

Фронтенд приложение «общается» с серверов при помощи proxy, которая перенаправляет запросы на внутренний адрес сервера - http://127.0.0.1, который находится на порту 8090, данный порт берется из докера, настроить порт можно в файле *docker-compose.yml*, который находится в корне проекта

web/dev/src/setupProxy.js

```
app.use(
  '/api/*',
  createProxyMiddleware({
    target: 'http://127.0.0.1:8090',
    changeOrigin: true,
    secure: false
  })
);
```

docker-compose.yml

```
version: "3.6"
services:
 app:
   build: "./app"
   ports:
     - "8090:8090"
   depends_on:
     - "postgres"
     - "redis"
   restart: "on-failure"
   web:
     build: "./web"
     ports:
       - "80:80"
       - "84:84"
     depends_on:
       - "app"
     restart: "on-failure"
 postgres:
   image: "postgres"
   environment:
     - POSTGRES_USER=postgres
     - POSTGRES_PASSWORD=postgres
   ports:
     - "5432"
   restart: "on-failure"
 redis:
   image: "redis:alpine"
   ports:
```

```
    - "6379"
  restart: "on-failure"
```

1.2 Роутер

Роутер – он же маршрутизатор, выполняет роль «раздатчика компонентов» для всего приложения. Роутер работает с текущим местоположением пользователя и выдает на отрисовку заданные ему компоненты.

```
<Route exact path="/auth" component={Form}/>
<Route path="/home" component={Home}/>
<Route exact path="/admin" component={Search}/>
<Route path='/admin/:userId' component={Admin}/>
<Route exact path="/works" component={Works}/>
<Route exact path="/work/:userId" component={Work}/>
```

При совпадении **path** без пропсы **exact** роутер будет выдавать компоненты при наличии **path** в адресной строке пользователя.

Также в роутер в приложении умеет определять какие страницы отдать пользователю в зависимости от того авторизован он или нет, если в хранилище приложения не имеется информация об авторизованном пользователе, то он перенаправляется на страницу авторизации и регистрации, иначе он может взаимодействовать с приложением на всех доступных ему страницах.

1.3 Конфигурационный файл Nginx

Конфигурационный файл расположен по пути ts_eos/web/.nginx и представляет собой стандартный конфигурационный файл для веб-сервера nginx. Сервер запускается на 80 порту(стандартный порт для протокола http) и имеет несколько блоков location:

- блок /, по которому раздается статические файлы(фронтенд в виде spa)
- блок /swagger/, по которому раздается документация по бэкенд части проекта

- блок /api, в котором настроено проксирование на бэкенд приложения(расположено на порту 8090)

Все статические файлы расположены в папке /usr/share/nginx/html/app (папка является виртуальной папкой внутри docker контейнера, на этапе сборки в нее копируется все статические файлы с сервера).

1.4 Dockerfile

К качестве системы контейнеризации в проекте используется Docker. Dockerfile для сборки фронтенда расположен по пути ts_eos/web.

Dockerfile логически разделен на 2 части: сборка и запуск веб-сервера(nginx).

В первой части происходит сборка и посредством утилиты yarn и копирование собранных статических файлов в контейнер докер. Во второй части происходит копирование конфигурационного файла nginx и собранной папки со статическими файлами в рабочие директории веб-сервера(nginx) и его запуск в режиме демона.

1.5 Авторизация

Чтобы пользоваться приложением пользователю необходимо авторизоваться. Есть два типа пользователей – студент и админ, чтобы получить роль студента необходимо зарегистрироваться в форме авторизации во вкладке «регистрация», а для получения роли админа необходимо сходить в БД и выдать ее «вручную».

Для авторизации в приложении необходимо ввести свои учетные данные во вкладке «логин», если бэкенд вернет ответ на данный запрос с кодом 200 (OK), то в хранилище приложения (Redux) сохранится информация о пользователе, а именно его имя, фамилия, группа, email, кол-во выполненных контрольных и информация о них.

```
enum authAPI {
 AUTHORIZE = 'AUTHORIZE',
 LOG_OUT = 'LOG_OUT',
 GET_CURRENT_USER = 'GET_CURRENT_USER',
```

```
 NOT_INSIDE = 'NOT_INSIDE',
 REGISTER = 'REGISTER',
 SET_GRADE = 'SET_GRADE'
}
```

1.6 Админка

Взаимодействовать со страницами администрирования может ТОЛЬКО пользователь с ролью «админ». Страницы доступные админу для взаимодействия:

- Поиск по пользователям
- Просмотр статистики пользователя (студента или же другого админа)

На странице поиска администратор может найти студента или другого админа по его уникальной информации, например имени и попасть на страницу с его статистикой по работе в модуле «тренажер»

1.7 Тренажер

В тренажере пользователь может «набивать руку» на введенных лично примерах, которые он может решать и получать обратную связь в виде «правильно» или «неправильно», также пользователь может пользоваться подсказками для упрощения процесса. Решение происходит пошагово, т.е. пользователь должен вводить решение примера шаг за шагом.

В тренажере реализована следующая API

```
math: {
  directCode: {
    highLeftShift: apiHost + '/calculations/direct_code/high_digits/left_shift',
    highRightShift: apiHost + '/calculations/direct_code/high_digits/right_shift',
    lowLeftShift: apiHost + '/calculations/direct_code/low_digits/left_shift',
    lowRightShift: apiHost + '/calculations/direct_code/low_digits/right_shift',
  },
  additionalCode: {
    correctiveStep: apiHost + '/calculations/additional_code/corrective_step',
```

```
  },
},
```

Из данной api можно увидеть что реализованы следующие методы умножения

1. Прямой код со старших разрядов сдвигом влево
2. Прямой код со старших разрядов сдвигом вправо
3. Прямой код с младших разрядов сдвигом влево
4. Прямой код с младших разрядов сдвигом вправо
5. Дополнительный код с корректирующим шагом

1.8 Реализация контрольных

Раздел контрольных позволяет студенту написать одну из нескольких подготовленных контрольных работ в реальном времени с сохранением результата.

Контрольная работа представлена в виде таблицы со следующими компонентами:

- кастомный инпут(находится в файле TableInput.tsx), и представляет собой структуру из заданного числа инпутов под каждый разряд числа + 1 скрытый инпут для хранения полного значения
- таблица для отображения введенных студентом значений. Код располагается в файле CustomTable.tsx. Таблица принимает при своем создании пустой шаблон контрольной, пришедший с бэкенда и сохраняет его в виде состояния(хук useState). В процессе заполнения студентом таблицы, его ответы записываются в шаблон и отправляются на бэкенд либо при истечении отведенного времени, либо при нажатии на кнопку “Отправить”
- псевдотаблица CollapseTable(файл CollapseTable.tsx), отображающаяся только для операции сложения. Содержит в себе отображения всех полей данных, необходимых для проверки правильности операции сложения(таких полей 7)
- компонент таймера, отсчитывающего остаток времени для написания контрольной работы

2. Общая архитектура бекенда

2.1 Аутентификация

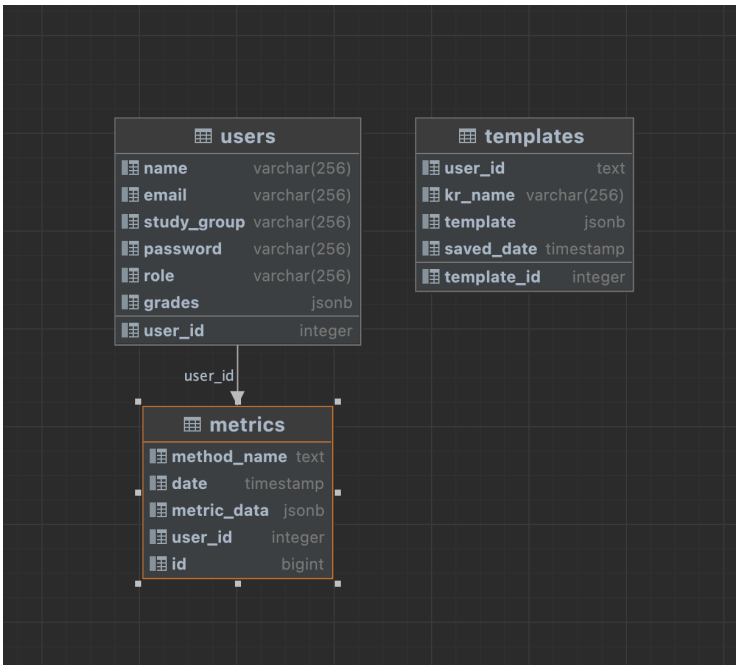
Регистрация и авторизация реализованы на беке с использованием общих архитектурных подходов, те разделения на слои и использование в слое репозитория PostgreSQL, в случае с регистрацией и авторизацией мы реализуем стандартный CRUD для модели User, а также используется AuthMiddleware из класса SessionStore, которая занимается пробрасыванием Cookie в метадату GRPC запроса, так как на бекенде используется GRPC как основной протокол. Также, эта миддлвара занимается вспомогательной ролью, логирует текущий запрос в таблицу метрик. А сама кука выставляется при регистрации и авторизации используя метод SetCookieGRPC

2.2 Метрики

Каждый запрос логируется, используя AuthMiddleware, методы для работы с БД содержатся в metricsRepo, в перспективе можно было бы хранить метрики в ClickHouse.

2.3 Описание БД

Предлагается расширять текущее использование PostgreSQL, в проекте подключены автомиграции, использующие Goose, соответственно миграции описываются в папке migrations, схема текущих таблиц описана ниже



Redis используется исключительно для хранения сессий, в перспективе его можно использовать и для кеширования, но нагрузка на сервис думаю не подразумевает такого использования.

2.4 Библиотека для математических вычислений

Библиотека для упрощения реализации логических операций сложений умножений и др находится в директории *ta_eos/app/internal/pkg/value_lib/.* Основной структурой данной библиотеки является структура *Value,* которая обладает следующими полями:

- value uint64 — само число в бинарном представлении со знаковым битом
- grid uint8 — размер сетки в котором производятся вычисления
- overflow bool — бло ли переполнение
- valueType ValueType — тип числа (прямой код, обратный, дополнительный)

Обладает следующими методами для работы:

- InitValue — инициализирует объект Value из бинарного представления числа в формате uint64
- InitValueFromInt64 - инициализирует объект Value из десятичного представления числа в int64
- InitValueFromString - инициализирует объект Value из строкового представления бинарного числа в string
- Sign — позволяет получить знак числа (0 — положительное, иначе отрицательное)
- Value — позволяет получить бинарное представление числа без знака
- ToInt — позволяет получить десятичное значение числа в формате string
- Overflow — позволяет получить происходило ли переполнение или нет.
- LeftShift — позволяет сделать сдвиг влево числа с учетом знаковой части и формата числа
- RightShift - позволяет сделать сдвиг вправо числа с учетом знаковой части и формата числа
- Invert — позволяет инвертировать число

- Add — позволяет добавить к числу требуемое значение
- ChangeGreed — позволяет изменить разрядность сетки числа
- Inc — позволяет увеличить число на 1
- ConvertType — позволяет изменить тип числа

2.5 Микросервис templates

Микросервис templates отвечает за генерацию, сохранение и валидацию контрольных для пользователей. В общем виде шаблон контрольной представляет из себя json объект следующего формата:


```
{
  "krName":"first",
  "data":{
    "UI":[
      {
        "data":[
          {
            "data":[
              {
                "name":"A",
                "value":"A"
              },
              {
                "name":"B",
                "value":"B"
              },
              {
                "name":"-A",
                "value":"-A"
              },
              {
                "name":"-B",
                "value":"-B"
              }
            ],
            "name":"Переменные"
          },
          {
            "data":[
              {
                "name":"A",
                "value":"11"
```

```
      },
      {
        "name":"B",
        "value":"28"
      },
      {
        "name":"-A",
        "value":null
      },
      {
        "name":"-B",
        "value":null
      }
    ],
    "name":"Значения"
  },
  {
    "data":[
      {
        "name":"A",
        "value":null
      },
      {
        "name":"B",
        "value":null
      },
      {
        "name":"-A",
        "value":null
      },
      {
        "name":"-B",
        "value":null
      }
    ],
    "name":"Прямой код"
  },
  {
    "data":[
      {
        "name":"A",
        "value":null
      },
      {
        "name":"B",
```

```
                "value":null
              },
              {
                "name":"-A",
                "value":null
              },
              {
                "name":"-B",
                "value":null
              }
            ],
            "name":"Обратный код"
          },
          {
            "data":[
              {
                "name":"A",
                "value":null
              },
              {
                "name":"B",
                "value":null
              },
              {
                "name":"-A",
                "value":null
              },
              {
                "name":"-B",
                "value":null
              }
            ],
            "name":"Дополнительный код"
          }
        ],
        "name":"table"
      }
    ],
    "template_name":"first",
    "what_to_do":"Первая контрольная работа"
  }
}
```

Описание полей:

- *krName — название контрольной работы*
- *template_name — название темплейта*
- *what_to_do — описание контрольной работы*
- *data — набор UI элементов, которые есть в контрольной*
- *UI — некоторый структурный элемент контрольной*
- *name — тип UI элемента*

В данном сервисе есть метод для сериализации данных абстрактных json структур в структуры в go GetTemplate.

В основном использование данного сервиса выглядит следующим образом: пользователь запрашивает шаблон с контрольной работой. Он генерируется в микросервисе и сохраняется в базу данных, с учетом времени начала работы. Далее фронтенд отображает пользователю вариант контрольной. После её заполнения вызывается метод ApproveKr в микросервисе для валидацию заполненной контрольной. После чего в ответе пользователю присылается оценка за контрольную вида решено правильно / общее число задач. Для пользователя на стороне фронтенд данное значение нормализуется и приводится в нужный вид.

2.6 Микросервис калькулятора

Микросервис калькулятора представляет из себя несколько ручек для расчёта умножений в различных кодах для тренировки. Включает в себя следующие ручки :

- AdditionalCodeWithCorrectiveStepCalculation — Умножение в дополнительном коде с коррекционным шагом.
- DirectCodeHighDigitsLeftShiftCalculation - Умножение в прямом коде со старших разрядов и сдвигом влево.
- DirectCodeHighDigitsRightShiftCalculation - Умножение в прямом коде со старших разрядов и сдвигом вправо.
- DirectCodeLowDigitsLeftShiftCalculation - Умножение в прямом коде со младших разрядов и сдвигом влево.

- DirectCodeLowDigitsRightShiftCalculation - Умножение в прямом коде со младших разрядов и сдвигом вправо.

Далее планируется использовать часть методов этого сервиса для проверки контрольных работ пользователей

2.7 Миграции

Миграции располагаются в папочке migrations и позволяют автоматически при запуске backend приложения выполнять sql код в базе данных с учётом уже примененных миграций. Для такого управления миграция и используется goose

2.8 Сваггер

Сваггер - некоторая удобная оболочка для локального запуска ручек backend приложения, доступна по 127.0.0.1/swagger/

2.9 Docker-compose

Docker-compose так как структура проекта в целом из себя представляет Нахождение и backend и frontend части приложения в одном месте то и приложение в целом собирается в одном месте. В данном случае стоит выделить 2 основные директории... И.... В них располагается код backend и frontend частей соответственно. Для каждой из частей указаны соответствующие docker файлы, в которых определенные шаги, необходимые для запуска каждой из частей по отдельности. Отдельно стоит выделить настройку nginx, который в данном случае настроен так, что проксирует все вызовы 127.0.0.1:80/ на frontend часть приложения и 127.0.0.1:80/api/ на backend часть приложения

3. О будущем развитии

Мы реализовали веб приложение (SPA) с использованием фреймворка “React” и языка “Typescript” в фронтенд части сервиса, в бекенд части приложения была использована микросервисная архитектура, язык

программирования “Golang”, базы данных “PostgresSQL” и “Redis”, средство для развертывание “Docker”.

Сейчас сервис располагается на ip адресе http://188.35.161.40/, исходный код доступен на https://github.com/Elderly-AI/ta_eos.

Уже сделанные части сервиса - авторизация, журнал оценок и просмотра активности для преподавателя, шаблон для создания контрольных работ и пример для этого шаблона “Контрольная работа №1”, тренажер для операция умножения, представленных в различных двоичных кодах.

3.1 Какие важные части нужно добавить или доработать?

1. По шаблону создания контрольных работ (смотреть в микросервисе контрольных работ) можно добавлять новые контрольные работы. Само наполнение контрольных работ можно брать у преподавателя;
2. Тренажер - нужно реализовать на бекенде в том же формате тренажеры для других операций в различных кодах. Лучше сначала добавить все недостающие варианты, которые даются на лекциях;
3. Система просмотра контента. Сейчас в сервисе не реализовано. Однако в этом сервисе желательно реализовать функционал сохранения презентаций и показа следующего контента: презентации, видеозаписи, тренажеры по лекциям с всплывающими подсказками.

3.2 Как стоит дополнять функционал?

Если идейно новый функционал укладывается в существующие микросервисы, то добавлять новые методы в api, или модифицировать их. На фронтенде нужно будет поддержать изменения.

Если идейно лучше выделить новый функционал в отдельный микросервис, то нужно настроить взаимодействие с уже существующими, добавить новые методы api для этого функционала, на стороне фронтенда создать новые страницы и добавить их в роутер, поддержать новые методы api согласно задумке.