САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Лабораторная работа №2

Выполнил: Митурский Богдан Антонович

Группа: K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача
Необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

Заполним связь с основным роутами

```
const router = require("express").Router();

router.use("/user", require("./user"));
router.use("/statuses", require("./statuses"));
router.use("/interface", require("./interface"));
router.use("/lobby", require("./lobby"));
router.use("/waitRoom", require("./waitRoom"));
router.use("/game", require("./game"));
router.use("/shop", require("./shop"));
router.use("/health", require("./health"));

module.exports = router;
```

Добавим все необходимые роуты в путь /user

```
import { checkSignMiddleware } from "../../middlewares/checkSign";

import getUser from "./getUser";
import openChest from "./openChest";
import replaceCard from "./replaceCard";
import unlockChest from "./unlockChest";
import watchAd from "./watchAd";
import rating from "./rating";
import season from "./season";
import friendsRating from "./friendsRating";
import bonuses from "./Bonuses";
import bonusesOk from "./BonusesOk";
import online from "./online";
import setCompleteTutorial from "./completeTutorial";
import getBoughtProducts from "./getBoughtProducts";
import lottery from "./lottery";
import replaceEmotionCard from "./replaceEmotionCard";

const router = require("express").Router();

router.use(checkSignMiddleware);

router.get("/get", getUser);
```

```
router.post("/getBoughtProducts", getBoughtProducts);
router.get("/replaceCard", replaceCard);
router.get("/replaceEmotionCard", replaceEmotionCard);
router.get("/unlockChest", unlockChest);
router.get("/openChest", openChest);
router.get("/watchAd", watchAd);
router.get("/analytics", watchAd);
router.get("/rating", rating);
router.get("/season", season);
router.get("/friendsRating", friendsRating);
router.get("/bonuses", bonuses);
router.get("/bonusesOk", bonusesOk);
router.get("/online", online);
router.get("/completeTutorial", setCompleteTutorial);
router.get("/lottery", lottery);

module.exports = router;
```

Подготовим middleware авторизации, которые будет поддерживать вход из 3х соц сетей (Одноклассники, ВКонтакте, Телеграм)

```
import { IGetUserIDAuthInfoRequest } from "../types/request";
import { checkSignRestApi as checkSignMiddlewareUtils } from
"@src/utils/checkSignRestApi";
import { initData, validate } from "@twa.js/init-data-node";
import { TG_BOT_TOKEN } from "@/src/libs/tg";
import { enviroment } from "@/instances/bot/config/enviroment";
import { Platforms } from "@/src/types/platforms";

export const checkSignMiddleware = (
 req: IGetUserIDAuthInfoRequest,
 res: Response,
 next: NextFunction
) => {
 try {
   if (!req.headers.authorization)
     return res
       .status(400)
       .json({ error: "Authorization header is undefined" });

   // Определяем платформу пользователя
   const platform = req.headers.authorization.includes("vk_user_id")
     ? Platforms.vk
     : req.headers.authorization.includes("is_premium") ||
       req.headers.authorization.includes("auth_date")
```

```
    ? Platforms.tg
    : Platforms.ok;

  req.platform = platform;

  if (
    !checkSignMiddlewareUtils(
      decodeURIComponent(req.headers.authorization),
      [process.env.VK_SECRET_KEY || "", process.env.VK_SECRET_GAME_KEY || ""],
      platform
    )
  ) {
    return res.status(400).json({ error: "Sign is not valid" });
  }

  if (platform === Platforms.ok) {
    req.userId = Number(
      req.headers.authorization?.split("logged_user_id=")[1]?.split("&")[0]
    );
    req.userRefferal = decodeURIComponent(req.headers.authorization)
      ?.split("ref=")[1]
      ?.split("&")[0];
  }

  if (platform === Platforms.vk) {
    req.userId = Number(
      req.headers.authorization?.split("vk_user_id=")[1]?.split("&")[0] ||
        req.headers.authorization?.split("viewer_id=")[1]?.split("&")[0]
    );
    req.userRef = req.headers.authorization
      ?.split("vk_ref=")[1]
      ?.split("&")[0];

    req.userRefferal = String(req?.query?.hash)
      ?.split("ref=")[1]
      ?.split("&")[0];
  }

  if (platform === Platforms.tg) {
    const data = tgAuth(req.headers.authorization);

    if (!data?.user) return console.error("TG auth data is not defined");
    req.userId = data.user.id;
    req.userRefferal = req.headers.authorization
      ?.split("ref_")[1]
```

```
      ?.split("&")[0];
    // TO DO переделать хранение переменных в req
    req.allowsWriteToPmTg = Boolean(
      req.headers.authorization
        ?.split("allows_write_to_pm%22%3A")[1]
        ?.split("%7D&")[0]
    );
  }

  req.userVkPlatform = req.headers.authorization
    ?.split("vk_platform=")[1]
    ?.split("&")[0];

  req.chatId = decodeURIComponent(req.headers.authorization)
    ?.split("vk_chat_id=")[1]
    ?.split("&")[0];

  req.hash = String(req?.query?.hash);

  req.authorization = req.headers.authorization;

  next();
 } catch (err) {
  console.error(err);
  return res.status(400).json({ error: err });
 }
};

const tgAuth = (sign: string) => {
 try {
  if (enviroment.IS_PRODUCTION) {
    validate(sign, TG_BOT_TOKEN);
  }
 } catch (e) {
  throw new Error("Invalid TG signature");
 }

 const parsedInitData = initData.parse(sign);

 if (!parsedInitData.user) {
  throw new Error("User not found");
 }

 if (parsedInitData.user.isBot) {
  throw new Error("Bots is not allowed");
```

```
 }

 return parsedInitData;
};
```

Напишем основной эндпоинт для получения пользовательских данных. Внутри себя он также будет актуализировать обновляющиеся поля вроде заданий, магазине и прочего.

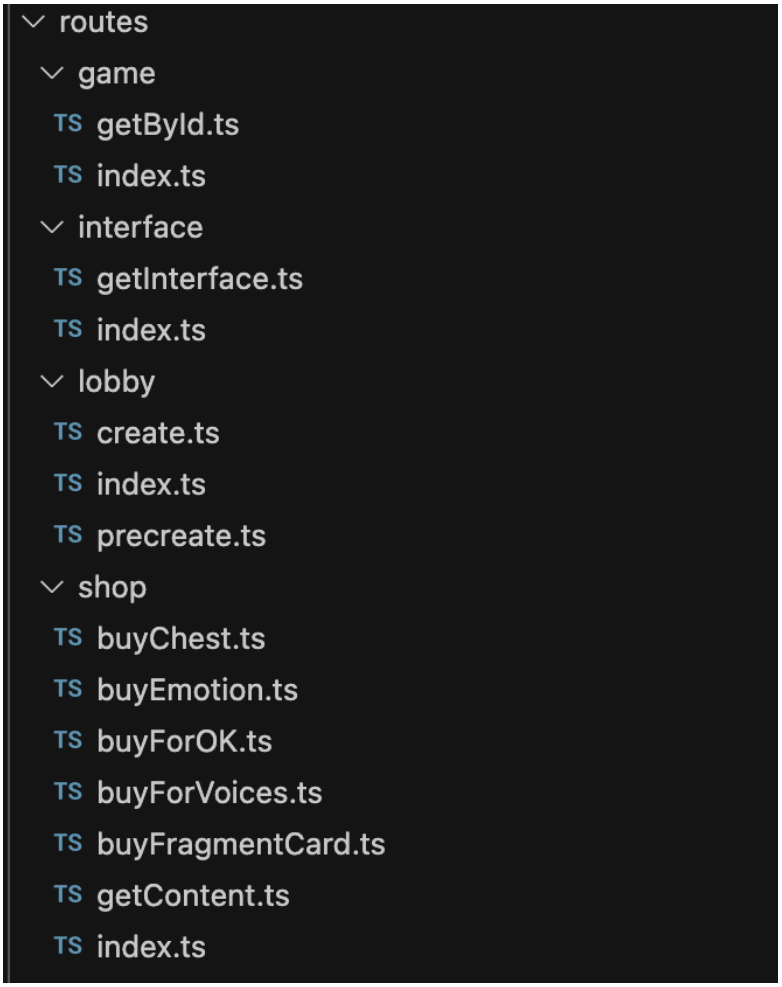
```
import { ICard, IChestDocs } from "@/src/models/Users/Users";
import { findShop } from "@/src/utils/findShop";

import { MIN_RATING } from "@src/data/rating";
import { findUser } from "@src/utils/findUser";
import { Response } from "express";
import { IGetUserIDAuthInfoRequest } from "../../types/request";
import { getCollection } from "../../utils/getCardsInfo";
import { shopContentUpdate } from "../shop/getContent";
import { getUserEmotions } from "../../utils/emotions/getUserEmotions";
import { recordUserAction } from "../../utils/recordUserAction";
import { refreshRewardedGiftTime } from "../../utils/refreshRewardedGiftTime";
import { GAME_HISTORY_LENGTH } from "@/src/configs/users";

import { userInfoPlatform } from "@/src/utils/userInfo";
import { newUserMessage } from "../../utils/tasks/checkTasks";
import { Platforms } from "@/src/types/platforms";

export default async function (req: IGetUserIDAuthInfoRequest, res: Response) {
 const user = await findUser(req);

 if (!user) return;

 const userPlatform = user.platform;

 // Обновляем магазин из-за конфликтов коллекции при входе
 if (user?.shop?.cardFragment?.card === undefined) {
   const globalShop = await findShop();
   if (!globalShop) return;
   user.shop.lastUpdate = 0;
   const shopContent = shopContentUpdate(user, globalShop);
   if (shopContent) {
     await user.update({ shop: shopContent.shop });
   }
 }

 let saveFlag = false;
```

```
// Очищаем историю юзера если изменились конфиги
if (user.history.length >= GAME_HISTORY_LENGTH) {
  user.history.splice(
    GAME_HISTORY_LENGTH,
    user.history.length - GAME_HISTORY_LENGTH
  );

  saveFlag = true;
}

if (user.rating < MIN_RATING) {
  user.rating = MIN_RATING;

  saveFlag = true;
}

if (!req.hash && userPlatform === Platforms.tg) {
  throw new Error("no authorization hash");
}

const usersInfo = await userInfoPlatform({
  userId: user._id,
  platform: user.platform,
  authorization: req.authorization,
});

if (!usersInfo?.length) {
  res.status(500).json({
    message: "failed to get user info",
  });
  return;
}

const userInfo = usersInfo[0];

if (user.photo_200 != userInfo.photo || user.name != userInfo.name) {
  user.photo_200 = userInfo.photo;
  user.name = userInfo.name;

  saveFlag = true;
}

if (user.photo_100 != userInfo.photo_100 || user.name != userInfo.name) {
  user.photo_100 = userInfo.photo_100;
```

```
  user.name = userInfo.name;

  saveFlag = true;
}

if (!user.cards.find((card: ICard) => card.entity === "bee")) {
  user.cards.push({ entity: "bee" });
  saveFlag = true;
}
if (!user.cards.find((card: ICard) => card.entity === "teslaTower")) {
  user.cards.push({ entity: "teslaTower" });
  saveFlag = true;
}

if (saveFlag) {
  await user.save();
}

// Отправление ежедневных заданий
if (user.bonuses.notificationBot && user.tasks.nextUpdateAt <= Date.now()) {
  const oldUpdateTime = user.tasks.nextUpdateAt;
  newUserMessage(user._id, oldUpdateTime, userPlatform);
}
await refreshRewardedGiftTime(user);

const { cards, ...userObject } = user.toObject();

const response = {
  ...userObject,
  units: getCollection(cards),
  emotions: await getUserEmotions(user),
  chests: (user.chests as IChestDocs).map((doc) => {
    const { _id, willOpenAt, ...chest } = doc.toObject() as any;
    return chest.empty
      ? null
      : {
          ...chest,
          willOpenAt:
            willOpenAt === undefined ? undefined : willOpenAt - Date.now(),
        };
  }),
};

// [ANALYTICS]
recordUserAction(userObject._id, "login");
```

```
 res.status(200).json(response);
}
```

Подготовим все остальные роуты, исходя из необходимых для проекта эндпоинтов и пропишем для них логику (Магазин, предсоздание лобби, получение данных для изменений в интерфейсе и др.)



Как итог, получаем рабочее api содержащее в себе весь необходимый для проекта функционал. Ниже представлены примеры запросов и ответов от разработанного api.

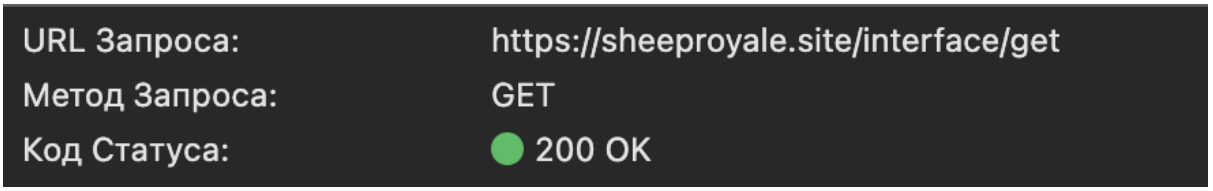


Рисунок 1 - запрос interface/get

Рисунок 2 - ответ interface/get

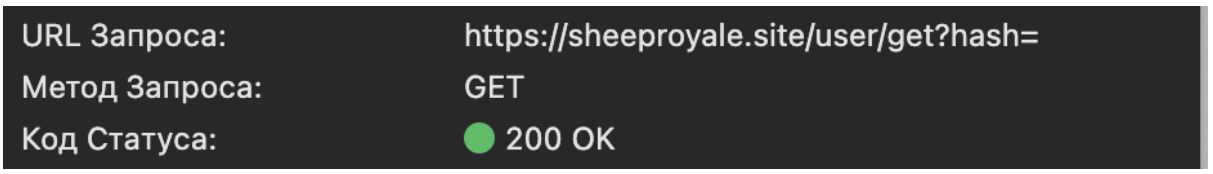


Рисунок 3 - запрос user/get

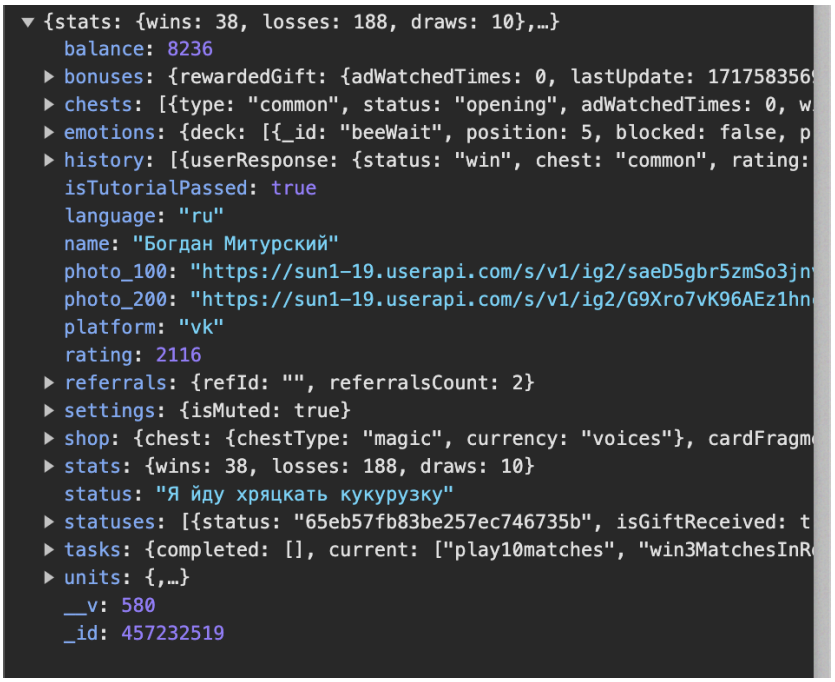


Рисунок 4 - ответ user/get

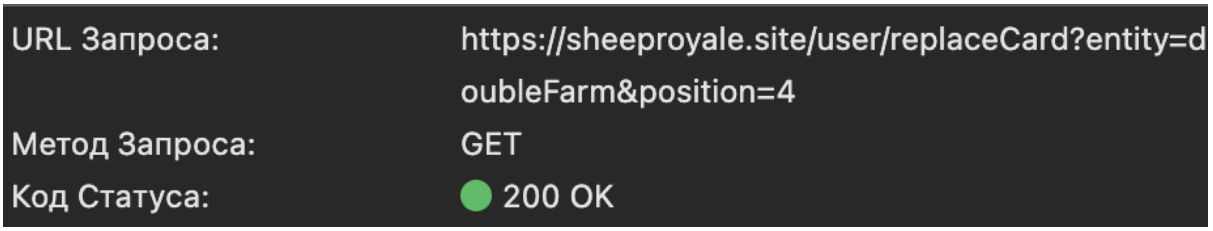


Рисунок 5 - запрос user/replaceCard

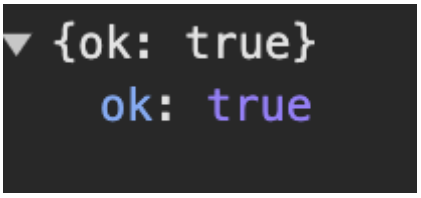


Рисунок 6 - ответ user/replaceCard

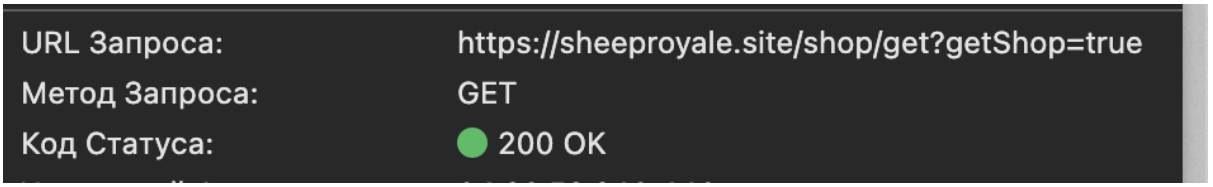


Рисунок 7 - запрос shop/get

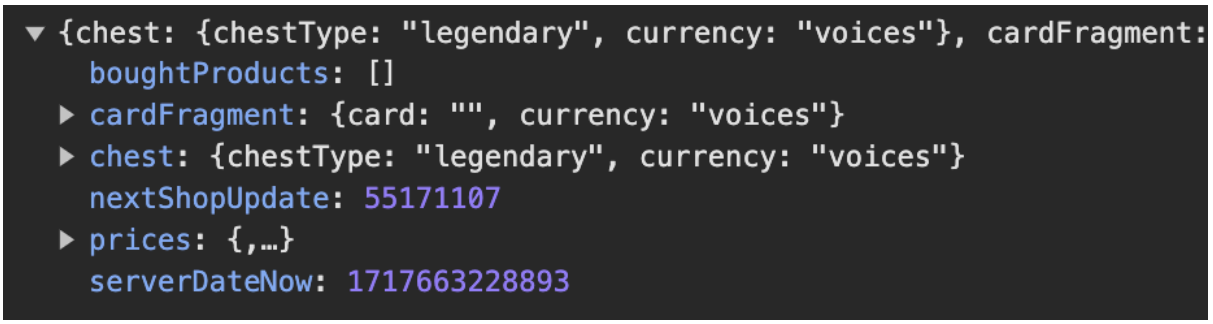


Рисунок 8 - ответ shop/get

Вывод

В рамках выполнения работы удалось реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate). Были разработаны все необходимые для функционирования бекенда эндпоинты.