

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет
Домашняя работа №2

Выполнил: Митурский Богдан Антонович

Группа: K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Продумать свою собственную модель пользователя. Реализовать набор из CRUD-методов для работы с пользователями средствами Express. Написать запрос для получения пользователя по id/email

Ход работы

Т.к. игра будет кроссплатформенной (с поддержкой входа через ВКонтакте, Телеграмм и Одноклассники), нам понадобится три различные коллекции пользователей, с единым доступом и набором параметров. Для этого, создадим основную коллекцию Users со всеми необходимыми полями (имя, аватар, дата регистрации, бонусы, игровые карты, история матчей и так далее):

```
import mongoose, {
  HydratedDocument,
  ObjectId,
  PopulatedDoc,
  Schema,
  Types,
  Document,
} from "mongoose";
import { IEntity } from "../../data/cards";
import { ILobby } from "../../Lobbies";
import { IChestTypes } from "../../data/chests";
import { StoreProducts } from "../../data/shop";
import { MIN_RATING } from "@src/data/rating";
import { Emotions, IEmotion, PopulatedIEmotion } from "../../Emotion";
import { Platforms } from "@src/types/platforms";
import { Lang } from "@src/types/language";
export type Currency = "coins" | "voices" | "ok" | "ton";

export type ICard = {
  entity: IEntity;
  parts?: number;
  position?: number;
};

const CardSchema = new Schema<ICard>({
  entity: {
    type: String,
    required: true,
  },
  parts: Number,
  position: Number,
});
```

```

export type IUserEmotionCard = {
  emotion: IEmotion["_id"];
  position: number | null;
};

export type IUserWithEmotions = Document<any, unknown, IUser> &
  Omit<IUser, "emotions"> & {
    emotions: mongoose.Types.Array<
      IUserEmotionCard & { emotion: PopulatedIEmotion }
    >;
  };

type Bonuses = {
  daily: number;
  subGroup: boolean;
  joinChat: boolean;
  notificationApp: boolean;
  notificationBot: boolean;
  addToFavorites: boolean;
  addToHomeScreen: boolean;
  rewardedGift: {
    lastUpdate: number;
    adWatchedTimes: number;
  };
};

type Settings = {
  isMuted: boolean;
};

export type IChest =
  | { empty: true; _id?: ObjectId }
  | ({
    empty?: false;
    _id?: ObjectId;
    type: IChestTypes;
    adWatchedTimes: number;
  } & (
    | {
      status: "idle";
    }
    | {
      status: "opening";
      willOpenAt: number;
    }
  ));

export type IChestDocs = [
  HydratedDocument<IChest>,

```

```

    HydratedDocument<IChest>,
    HydratedDocument<IChest>,
    HydratedDocument<IChest>
  ];

const ChestSchema = new Schema<IChest>({
  type: String,
  status: String,
  willOpenAt: Number,
  empty: Boolean,
  adWatchedTimes: {
    type: Number,
    default: 0,
    required: true,
  },
});

export interface UserStatus {
  status: mongoose.Types.ObjectId;
  isSelected: boolean;
  isGiftReceived: boolean;
}

export interface IUser {
  markModified(arg0: string): unknown;
  _id: number;
  platform: Platforms;
  name: string;
  photo_100: string;
  photo_200: string;
  registrationDate: Date;
  ban?: boolean;
  cards: ICard[];
  emotions: IUserEmotionCard[];
  balance: number;
  language: Lang;
  history: any[];
  chests: [IChest, IChest, IChest, IChest];
  rating: number;

  stats: {
    draws: number;
    wins: number;
    losses: number;
  };
  lobby: PopulatedDoc<ILobby>;

  bonuses: Bonuses;
  isTutorialPassed: boolean;

```

```

shop: {
  lastUpdate: number;
  boughtProducts: StoreProducts[];
  chest: {
    currency: Currency;
    chestType: IChestTypes | "";
  };

  cardFragment: {
    currency: Currency;
    card: IEntity | "";
  };
};

referrals: {
  refId: string;
  referralsCount: number;
};

statuses: UserStatus[];

tasks: {
  current: string[];
  completed: string[];
  progress: Record<string, number>;
  nextUpdateAt: number;
  lastTasksMessageId?: number;
  madeProgress?: boolean;
};

settings: Settings;
}

const defaultCards: ICard[] = [
  {
    entity: "sheep",
    position: 1,
  },
  {
    entity: "farm",
    position: 2,
  },
  {
    entity: "pasture",
    position: 3,
  },
  {
    entity: "teslaTower",
    position: 4,
  },

```

```

    },
    {
      entity: "bigSheep",
    },
    {
      entity: "bee",
    },
    {
      entity: "doubleFarm",
      parts: 1,
    },
  ],
];

const defaultEmotions: IUserEmotionCard[] = [
  { emotion: "sheepyHello", position: 1 },
  { emotion: "sheepSorry", position: 2 },
  { emotion: "sheepAngry", position: 3 },
];

export const UserSchema = new Schema<IUser>({
  _id: { index: true, required: true, type: Number },
  platform: {
    type: String,
    required: true,
    enum: Platforms,
    default: Platforms.vk,
  },
  ban: {
    type: Boolean,
    required: false,
  },
  name: {
    type: "string",
    required: true,
  },
  photo_100: {
    type: "string",
    required: true,
  },
  photo_200: {
    type: "string",
    required: true,
  },
  registrationDate: {
    type: Date,
  },
  cards: {
    type: [CardSchema],
    required: true,
  },
});

```

```
    default: defaultCards,
  },
  emotions: {
    type: [
      {
        emotion: { type: String, ref: Emotions },
        position: { type: Number, default: null },
      },
    ],
    required: true,
    default: defaultEmotions,
    _id: false,
  },
  balance: {
    type: Number,
    required: true,
    default: 0,
  },
  language: {
    type: String,
    required: true,
    default: "ru",
  },
  history: {
    type: [],
    required: true,
    default: [],
  },
  stats: {
    wins: {
      type: Number,
      default: 0,
      required: true,
    },
    losses: {
      type: Number,
      default: 0,
      required: true,
    },
    draws: {
      type: Number,
      default: 0,
      required: true,
    },
  },
},

  chests: {
    type: [ChestSchema],
    default: [
```

```
    { empty: true },
    { empty: true },
    { empty: true },
    { empty: true },
  ],
  required: true,
},
rating: {
  type: Number,
  default: MIN_RATING,
  required: true,
  index: -1,
},
lobby: { type: Schema.Types.ObjectId, ref: "lobbies" },

bonuses: {
  daily: {
    type: Number,
    default: 0,
    required: true,
  },
  subGroup: {
    type: Boolean,
    default: false,
    required: true,
  },
  joinChat: {
    type: Boolean,
    default: false,
    required: true,
  },
  notificationApp: {
    type: Boolean,
    default: false,
    required: true,
  },
  notificationBot: {
    type: Boolean,
    default: false,
    required: true,
  },
  addToFavorites: {
    type: Boolean,
    default: false,
    required: true,
  },
  addToHomeScreen: {
    type: Boolean,
```



```
    default: false,
    required: true,
  },
  rewardedGift: {
    lastUpdate: {
      type: Number,
      default: Date.now(),
      required: true,
    },
    adWatchedTimes: {
      type: Number,
      default: 0,
      required: true,
    },
  },
},
isTutorialPassed: {
  type: Boolean,
  default: false,
  required: true,
},
shop: {
  lastUpdate: {
    type: Number,
    default: 0,
    required: true,
  },
  boughtProducts: {
    type: Object,
    default: [],
    required: true,
  },
  chest: {
    chestType: {
      type: String,
      default: "",
    },
    currency: {
      type: String,
      default: "coins",
    },
  },
},
cardFragment: {
  card: {
    type: String,
    default: "",
  },
  currency: {
    type: String,
```

```

        default: "coins",
    },
},
},
tasks: {
    completed: {
        type: [String],
        required: true,
        default: [],
    },
    current: {
        type: [String],
        required: true,
        default: [],
    },
    progress: {
        type: {},
        default: {},
        required: true,
    },
    nextUpdateAt: {
        type: Number,
        required: true,
        default: 0,
    },
    lastTasksMessageId: {
        type: Number,
    },
    madeProgress: { type: Boolean },
},

referrals: {
    refId: {
        type: String,
        default: "",
    },
    referralsCount: {
        type: Number,
        default: 0,
    },
},
},

statuses: [
    {
        status: {
            type: Types.ObjectId,
            required: true,
            unique: false,
            ref: "emoji_status",

```

```

    },
    isGiftReceived: { type: Boolean, default: false },
    isSelected: { type: Boolean, default: false },
  },
],

settings: { isMuted: { default: false, type: Boolean, required: true } },
});

export const Users = mongoose.model("users", UserSchema);

```

Далее, создадим коллекции для не основных соц сетей (ТГ и ОК):

```

import mongoose from "mongoose";

import { UserSchema } from "../Users";

export const UsersOk = mongoose.model("usersok", UserSchema);

```

```

import mongoose from "mongoose";

import { UserSchema } from "../Users";

export const UsersTg = mongoose.model("userstg", UserSchema);

```

Далее, реализуем функцию которая будет возвращать нам нужную коллекцию пользователей, в зависимости от платформы:

```

import { Users } from "../Users";
import { Platforms } from "@src/types/platforms";
import { UsersOk } from "../UsersOk";
import { UsersTg } from "../UsersTg";

function UsersModel(platform: Platforms) {
  switch (platform) {
    case Platforms.vk:
      return Users;
    case Platforms.tg:
      return UsersTg;
    case Platforms.ok:
      return UsersOk;
  }
}

export default UsersModel;

```

Теперь, создадим базовые метод для получения записи о пользователе:

- findUser (Позволяет найти пользователя в бд. Создаёт его при отсутствии, также, обрабатывает базовую логику обновления необходимых данных пользователя при входе)

```
import { IUserWithEmotions, Users } from "../models/Users/Users";
import "../models/Lobbies";
import { Platforms } from "../types/platforms";
import UsersModel from "../models/Users/Utils/getUsersModel";
import { newUserMessage } from "@/instances/hub/src/Utils/tasks/checkTasks";
import { ONLINE_SECONDS_TIMEOUT } from "../configs/online";

export type findUserParams = IGetUserAuthInfo & {
  userRegistrationData?: {
    firstName: string;
    lastName?: string;
    languageCode?: string;
  };
};

export const findUser = async (req: findUserParams) => {
  try {
    let user;
    req.userId = Number(req.userId);

    user = await UsersModel(req.platform)
      .findOne({ _id: req.userId })
      .populate("lobby")
      .populate<Pick<IUserWithEmotions, "emotions">>("emotions.emotion");

    if (!user) {
      let usersInfo;

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

      if (!usersInfo) {
        throw new Error("failed to get user info for create");
      }

      const userInfo = usersInfo[0];

      const date = new Date();

      let userData = {
        _id: req.userId,
```

```

    platform: userInfo.platform,
    name: userInfo.name,
    photo_100: userInfo.photo_100,
    photo_200: userInfo.photo,
    registrationDate: date.toISOString(),
    referrals: {},
    bonuses: { notificationBot: false },
    language: "ru"
  };

  // Отмечаем для телеграмма, если пользователь разрешил ему писать
  if (userInfo.platform === Platforms.tg && req?.allowsWriteToPmTg) {
    userData.bonuses.notificationBot = true;
  }
  if (userInfo?.languageCode) {
    userData.language = userInfo?.languageCode;
  }

  const referrerID = Number(req?.userRefferal);

  // Если передан ref, проверяем валидность всех данных и обновляем
  модели
  if (referrerID) {
    console.info("ref", referrerID, req.platform);
    const referrer = await UsersModel(req.platform).findOne({
      _id: referrerID,
    });
    /**
     * В каких случаях выбрасываем ошибку:
     * 1. Если referrer не найден (тот, кто пригласил в игру)
     * 2. Если айди приглашающего совпадает с приглашенным (пригласил
сам себя)
     */

    if (!referrer || req.userId === referrerID) {
    } else {
      // Сохраняем айди, чтобы чуть позже выдать бонус
      userData = { ...userData, referrals: { refId: String(referrerID) }
    };
  }
}
let user;

user = await UsersModel(req.platform).create(userData);

if (!user) throw new Error("failed to create user");

user = await user.populate<Pick<IUserWithEmotions, "emotions">>(
  "emotions.emotion"

```

```

    );

    req.platform === Platforms.vk &&
        recordUserAction(user._id, "registration", {
            vkRef: req?.userRef,
            vkPlatform: req?.userVkPlatform,
            vkInitialParams: req?.authorization,
            hash: req?.hash,
        });

    return user;
}

redis.setex(
    `online_list:${user._id}`,
    ONLINE_SECONDS_TIMEOUT,
    user._id + ""
);

return user;
} catch (e) {
    console.log("Find User Error", e);
}
};

```

И, на его основе, учитывая все игровые особенности, напомним метод для получения пользователя. Который одновременно с этим будет проверять актуальность пользовательских данных, обновлять его аватар, таймеры бонусов и так далее.

```

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

let saveFlag = false;

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

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(),
        };
    })),
};
```

Вывод

В ходе работы была продумана основная модель пользователя, реализован метод для получения и создания пользователя, а также основной эндпоинт для получения пользователя исходя из строки авторизации