САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка
Отчет
Лабораторная работа №2

Выполнил:
Коровин Александр

Группа:
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург
2022 г.

Задача: в рамках данной лабораторной работы Вам предложено выбрать один из нескольких вариантов. Выбранный вариант останется единым на весь курс и будет использоваться в последующих лабораторных работах.

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

ВАРИАНТ №3

Ход работы:

Config для сервера:

```
import dotenv from 'dotenv';

dotenv.config();

const MONGO_USERNAME = process.env.MONGO_USERNAME || '';
const MONGO_PASSWORD = process.env.MONGO_PASSWORD || '';
const MONGO_URL = `mongodb+srv://${MONGO_USERNAME}:${MONGO_PASSWORD}@cluster0.aqfzq.mongodb.net/db`;

const SERVER_PORT = process.env.SERVER_PORT ? Number(process.env.SERVER_PORT) : 5000;

const SECRET_KEY = process.env.SERVER_KEY  || "123";

export const config = {
  mongo: {
    username: MONGO_USERNAME,
    password: MONGO_PASSWORD,
    url: MONGO_URL
  },
  server: {
    port: SERVER_PORT
  },
  jwt:{
    key:SECRET_KEY
  }
};

```

Controllers

```ts
import { NextFunction, Request, Response } from "express";
import mongoose from "mongoose";
import User from "../models/User";
import hashPassword from "../utils/hashPassword";
import checkPassword from "../utils/checkPassword";
import generateAccessToken from "../utils/generateAccessToken";

const createUser = (req: Request, res: Response, next: NextFunction) => {
  const { firstName, lastName, email } = req.body;
  const password = hashPassword(req.body.password);
  const user = new User({
    _id: new mongoose.Types.ObjectId(),
    firstName,
    lastName,
    email,
    password,
  });
  return user
    .save()
    .then((user) => res.status(201).json({ user }))
    .catch((error) => res.status(500).json({ error }));
};
const loginUser = (req: Request, res: Response, next: NextFunction) => {
  const { email, password } = req.body;
  return User.findOne({ email }).then((user) => {
    if (user) {
      const validPassword = checkPassword(user, password);
      if (validPassword) {
        const token = generateAccessToken(email);
        return res.status(201).json({ token });
      } else {
        return res.status(400).json({ message: "Wrong password!" });
      }
    } else {
      return res.status(400).json({ message: "User is not found!" });
    }
  });
};

const updateUser = (req: Request, res: Response, next: NextFunction) => {
  const userId = req.params.userId;

  return User.findById(userId)
    .then((user) => {
      if (user) {
        req.body.password = hashPassword(req.body.password);
        user.set(req.body);
        return user
          .save()
          .then((user) => res.status(201).json({ user }))
          .catch((error) => res.status(500).json({ error }));
      } else {
        return res.status(404).json({ message: "not found" });
      }
    })
    .catch((error) => res.status(500).json({ error }));
};

const deleteUser = (req: Request, res: Response, next: NextFunction) => {
  const userId = req.params.userId;

  return User.findByIdAndDelete(userId)
    .then((user) =>
      user
        ? res.status(201).json({ user, message: "Deleted" })
```

```
        ? res.status(201).json({ user, message: "Deleted" })
        : res.status(404).json({ message: "not found" })
    )
    .catch((error) => res.status(500).json({ error }));
};

const getUser = (req: Request, res: Response, next: NextFunction) => {
  const userId = req.params.userId;

  return User.findById(userId)
    .then((user) =>
      user ? res.status(201).json({user}) : res.status(404).json({ message: "Not found" })
    )
    .catch((error) => res.status(500).json({ error }));
};

const getUserList = (req: Request, res: Response, next: NextFunction) => {
  return User.find()
    .then((users) => res.status(200).json({ users }))
    .catch((error) => res.status(500).json({ error }));
};

export default {
  createUser,
  deleteUser,
  loginUser,
  updateUser,
  getUser,
  getUserList,
};
```

Тут имеются все круд методы на получение пользователя, его создание, на его логин и тд.

Модели для базы данных:

```typescript
import mongoose, { Document, Schema } from "mongoose";

export interface IHotel {
  title: string;
  description: string;
  address: string;
  guestLimit: number;
  price: number;
  bookings: string
}

export interface IHotelModel extends IHotel, Document { }

const HotelSchema: Schema = new Schema(
  {
    title: { type: String, required: true },
    description: { type: String, required: true },
    address: { type: String, required: true },
    guestLimit: { type: Number, required: true },
    price: { type: Number, required: true },
    bookings: { type: Schema.Types.ObjectId, required: false, ref: "Booking" },
  },
  {
    versionKey: false,
  }
);

export default mongoose.model<IHotelModel>("Hotel", HotelSchema);

```

Модель отеля MongoDB

Routes:

```typescript
import express from "express";
import controller from "../controllers/BookingController";
import { Schemas, ValidateJoi } from "../middleware/ValidateSchema";

const router = express.Router();

router.post("/create", ValidateJoi(Schemas.booking.create),controller.createBooking);
router.get("/get/:bookingId", controller.getBooking);
router.get("/get", controller.getBookingList);
router.patch("/update/:bookingId", ValidateJoi(Schemas.booking.create),controller.updateBooking);
router.delete("/delete/:bookingId", controller.deleteBooking);

export = router;

```

Routes с валидацией данных

Сервер:

```
    });

    next();
  });

  router.use(express.urlencoded({ extended: true }));
  router.use(express.json());

  /** Rules of our API */
  router.use((req, res, next) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header(
      "Access-Control-Allow-Headers",
      "Origin, X-Requested-With, Content-Type, Accept, Authorization"
    );

    if (req.method == "OPTIONS") {
      res.header(
        "Access-Control-Allow-Methods",
        "PUT, POST, PATCH, DELETE, GET"
      );
      return res.status(200).json({});
    }

    next();
  });

  /** Routes */
  router.use("/users", userRoutes);
  router.use("/bookings", bookingRoutes);
  router.use("/hotels", hotelRoutes);
```

```
import express from "express";
import http from "http";
import mongoose from "mongoose";
import { config } from "./config/config";
import Logging from "./library/Logging";
import userRoutes from "./routes/UserRoutes";
import bookingRoutes from "./routes/BookingRoutes";
import hotelRoutes from "./routes/HotelRoutes";

const router = express();

/** Connect to Mongo */
mongoose
  .connect(config.mongo.url, { retryWrites: true, w: "majority" })
  .then(() => {
    Logging.info("Mongo connected successfully.");
    StartServer();
  })
  .catch((error) => Logging.error(error));

/** Only Start Server if Mongoose Connects */
const StartServer = () => {
  /** Log the request */
  router.use((req, res, next) => {
    /** Log the req */
    Logging.info(
      `Incomming - METHOD: [${req.method}] - URL: [${req.url}] - IP: [${req.socket.remoteAddress}]`
    );

    res.on("finish", () => {
      /** Log the res */
      Logging.info(
        `Result - METHOD: [${req.method}] - URL: [${req.url}] - IP: [${req.socket.remoteAddress}] - STATUS: [${res.statusCode}]`
```

```
67    /** Healthcheck */
68
69    router.get("/ping", (req, res, next) =>
70      res.status(200).json({ hello: "world" })
71    );
72
73    /** Error handling */
74    router.use((req, res, next) => {
75      const error = new Error("Not found");
76
77      Logging.error(error);
78
79      res.status(404).json({
80        message: error.message,
81      });
82    });
83
84    http
85      .createServer(router)
86      .listen(config.server.port, () =>
87        Logging.info(`Server is running on port ${config.server.port}`)
88      );
89  };
90
```

Папка .env:

```
1   MONGO_USERNAME="admin"
2   MONGO_PASSWORD="admin"
3   SERVER_PORT="5000"
4   SECRET_KEY="SECRET_KEY_RANDOM"
5
```

Данные для логина в MongoDB

Вывод: Я Реализовал REST API сайта для бронирования отелей. С использованием данных в задании фреймворков и библиотек.