

```
index.js
src
app.js
api
cats
index.js
users
index.js
configs
passport.js
controllers
catsController.js
usersController.js
js
db
schemas
schCat.js
schUser.js
index.js
helpers
constants.js
guard.js
repositories
catsRepo.js
usersRepo.js
index.js
services
authService.js
catsService.js
catsService.js
usersService.js
index.js
validation
catsValidation.js
```

```
/db/schemas/schCat.js
const mongoose = require("mongoose");
const { Schema, model } = mongoose;

const catSchema = new Schema(
  {
    name: {
      type: String,
      required: [true, "Set name for cat"],
      unique: true,
    },
    age: {
      type: Number,
      min: 1,
      max: 25,
    },
    isVaccinated: {
      type: Boolean,
      default: false,
    },
    features: {
      type: Array,
      set: (data) => ((data ? [] : data),
    },
    owner: {
      // type: mongoose.Types.ObjectId,
      type: mongoose.SchemaTypes.ObjectId,
      ref: "user",
      // "user" - должно совпадать с "user"
      // а const User = model("user", userSchema);
    },
  },
  {
    versionKey: false,
    timestamps: true, //createdAt and updatedAt
  });

catSchema.virtual("nameAge").get(function () {
  return this.name + " + this.age;
});

const Cat = model("cat", catSchema);
module.exports = Cat;
```

```
/src/repositories/catsRepo.js
const { Cat } = require("../db/schemas/schCat");

class CatsRepo {
  constructor() {
    this.model = Cat;
  }

  async getAll(userId) {
    const results = await this.model.find({ owner: userId }).populate({
      path: "owner",
      select: "name email sex ~_id",
    });
    return results;
  }

  async getById(id, userId) {
    try {
      const result = await this.model
        .findOne({ _id: id, owner: userId })
        .populate({
          path: "owner",
          select: "name email sex ~_id",
        });
      return result;
    } catch (error) {
      error.status = HttpStatusCode.BAD_REQUEST;
      error.data = "Bad request";
      throw error;
    }
  }

  async create(body, userId) {
    const result = await this.model.create({ ...body, owner: userId });
    return result;
  }

  async update(id, body, userId) {
    const result = await this.model.findOneAndUpdate(
      { _id: id, owner: userId },
      { ...body },
      { new: true } // чтоб не возвращал предыдущую запись
    );
    return result;
  }

  async updateStatus(id, body, userId) {
    return this.update(id, body, userId);
  }

  async remove(id, userId) {
    const result = await this.model.findOneAndRemove(
      { _id: id,
        owner: userId,
      });
    return result;
  }
}

module.exports = CatsRepo;
```

```
/src/repositories/usersRepo.js
const User = require("../db/schemas/schUser");

class UsersRepo {
  constructor() {
    this.model = User;
  }

  async findById(id) {
    try {
      const result = await this.model.findOne({ _id: id });
      return result;
    } catch (error) {
      error.status = 400;
      error.data = "Bad request";
      throw error;
    }
  }

  async findByEmail(email) {
    try {
      const result = await this.model.findOne({ email });
      return result;
    } catch (error) {
      error.status = 400;
      error.data = "Bad request";
      throw error;
    }
  }

  async create(body) {
    const user = new this.model(body);
    return user.save();
    //в схеме есть поле на save - userSchema.pre("save", async function (next) {...
  }

  async updateToken(id, token) {
    await this.model.updateOne({ _id: id }, { token });
  }
}

module.exports = UsersRepo;
```

```
/db/schemas/schUser.js
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const SALT_FACTOR = 6;
const { Schema, model } = mongoose;
const { Sex } = require("../helpers/constants");

const userSchema = new Schema(
  {
    name: {
      type: String,
      required: [true, "Email is required"],
      unique: true,
      validate(value) {
        const re = /^[a-zA-Z0-9_\./\s]+$/;
        return re.test(String(value).toLowerCase());
      },
    },
    password: {
      type: String,
      required: [true, "Password is required"],
    },
    token: {
      type: String,
      default: null,
    },
    versionKey: false,
    timestamps: true, //createdAt and updatedAt
  },
  {
    // hook .pre и есть .post (перед и после чего-то)
    // .pre("save") - перед сохранением чего-то должно быть
    // а .post("save") - после сохранения user.save()
    // перед сохранением в базу пароль будет преобразован в хеш
    userSchema.pre("save", async function (next) {
      if (this.isModified("password")) return next();
      this.password = await bcrypt.hash(
        this.password,
        bcrypt.genSaltSync(SALT_FACTOR)
      );
    });
    next();
  });

userSchema.methods.validatePassword = async function (password) {
  return await bcrypt.compare(password, this.password);
};

const User = model("user", userSchema);
// "user" - должно совпадать с ref: "user" в схеме schema
module.exports = User;
```

```
/src/repositories/index.js
const CatsRepo = require("../repositories/catsRepo");
const UsersRepo = require("../repositories/usersRepo");
module.exports = { CatsRepo, UsersRepo };
```

```
/src/services/catsService.js
const { CatsRepo } = require("../repositories");

class CatsService {
  constructor() {
    this.repo = new CatsRepo(client);
  }

  getAll(userId) {
    return this.repo.cats.getAll(userId);
  }

  getById({ id }, userId) {
    return this.repo.cats.getById(id, userId);
  }

  create(body, userId) {
    return this.repo.cats.create(body, userId);
  }

  update({ id }, body, userId) {
    return this.repo.cats.update(id, body, userId);
  }

  updateStatus({ id }, body, userId) {
    return this.repo.cats.updateStatus(id, body, userId);
  }

  remove({ id }, userId) {
    return this.repo.cats.remove(id, userId);
  }
}

module.exports = CatsService;
```

```
/src/services/authService.js
const { UsersRepo } = require("../repositories");
const jwt = require("jsonwebtoken");
const { SECRET_KEY } = process.env.JWT_SECRET_KEY;

class AuthService {
  constructor() {
    this.repo = new UsersRepo();
  }

  async login({ email, password }) {
    const user = await this.repo.users.findByIdByEmail(email);
    if (!user || !user.validatePassword(password)) {
      //userSchema.methods.validatePassword = ....
      return null;
    }
    const id = user.id;
    const payload = { id };
    const token = jwt.sign(payload, SECRET_KEY, { expiresIn: "1h" });
    return token;
  }

  async logout(id) {
    await this.repo.users.updateToken(id, null);
  }
}

module.exports = AuthService;
```

```
/src/services/usersService.js
const { UsersRepo } = require("../repositories");

class UsersService {
  constructor() {
    this.repo = new UsersRepo();
  }

  async create(body) {
    const data = await this.repo.users.create(body);
    return data;
  }

  async findByIdByEmail(email) {
    const data = await this.repo.users.findByIdByEmail(email);
    return data;
  }

  async findById(id) {
    const data = await this.repo.users.findById(id);
    return data;
  }
}

module.exports = UsersService;
```

```
/src/configs/passport.js
const passport = require("passport");
const { Strategy, ExtractJwt } = require("passport-jwt");
const { UsersService } = require("../services");
const { SECRET_KEY } = process.env.JWT_SECRET_KEY;

const params = {
  secretOrKey: SECRET_KEY,
  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
};

passport.use(
  // passport.use не принимает асинхронный токен
  new Strategy(params, async (payload, done) => {
    try {
      const service = new UsersService();
      const user = await service.findById(payload.id);
      if (!user) {
        return done(new Error("User not found"));
      }
      if (!user.token) {
        return done(null, false);
      }
      return done(null, user);
    } catch (error) {
      done(error);
    }
  })
);
```

```
/db/index.js
const mongoose = require("mongoose");
require("dotenv").config();
const uriDb = process.env.URI_DB;

const db = mongoose.connect(uriDb, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
});
// Если в db не будет mongoose

mongoose.connection.on("connected", () => {
  console.log("Mongoose connection to db");
});

mongoose.connection.on("error", (err) => {
  console.log("Mongoose connection error: ${err.message}");
});

mongoose.connection.on("disconnected", () => {
  console.log("Mongoose disconnected");
});

process.on("SIGINT", async () => {
  await mongoose.connection.close();
  console.log("Connection for DB disconnected and app terminated");
  process.exit(1);
});

module.exports = db;
```

```
/index.js
const app = require("../src/app");
const db = require("../src/db");

const PORT = process.env.PORT || 3000;

db.then(() => {
  app.listen(PORT, () => {
    console.log("Server run on port: ${PORT}");
  });
}).catch((error) => {
  console.log("DB Server isn't running. Error message ${error.message}");
  process.exit(1);
});
```

```
/src/app.js
const express = require("express");
const cors = require("cors");
const logger = require("morgan");
const { HttpStatusCode } = require("../helpers/constants");
const { routerCats, routerUsers } = require("../src/api/users");

const app = express();

const formatLogger = app.get("env") === "development" ? "dev" : "short";

app.use(logger(formatLogger));
app.use(cors());
app.use(express.json());

app.use("/api/cats", routerCats);
app.use("/api/users", routerUsers);

app.use((req, res, next) => {
  res.status(HttpStatusCode.NOT_FOUND).json({
    status: "error",
    code: HttpStatusCode.NOT_FOUND,
    message: "Use api on routes ${req.baseUrl}/api/cats",
    data: "Not found",
  });
});

app.use((err, req, res, next) => {
  err.status = err.status ? err.status : HttpStatusCode.INTERNAL_SERVER_ERROR;
  res.status(err.status).json({
    status: err.status === HttpStatusCode.INTERNAL_SERVER_ERROR ? "fail" : "error",
    code: err.status,
    message: err.message,
    data:
      err.status === HttpStatusCode.INTERNAL_SERVER_ERROR
        ? "INTERNAL_SERVER_ERROR"
        : err.data,
  });
});

module.exports = app;
```

```
/src/helpers/constants.js
const HttpStatusCode = {
  OK: 200,
  CREATED: 201,
  NO_CONTENT: 204,
  BAD_REQUEST: 400,
  UNAUTHORIZED: 401,
  FORBIDDEN: 403,
  NOT_FOUND: 404,
  CONFLICT: 409,
  INTERNAL_SERVER_ERROR: 500,
};

const Sex = {
  MALE: "m",
  FEMALE: "f",
  NONE: "none",
};

module.exports = { HttpStatusCode, Sex };
```

```
/src/api/cats/index.js
const express = require("express");
const controllerCats = require("../controllers/catsController");
const router = express.Router();

const {
  validateCreateCat,
  validateUpdateCat,
  validateUpdateStatusCat,
} = require("../validation/catsValidation");

const { guard } = require("../helpers/guard");

router
  .get("/", guard, controllerCats.getAll)
  .get("/:id", guard, controllerCats.getById)
  .post("/", guard, validateCreateCat, controllerCats.create)
  .put("/:id", guard, validateUpdateCat, controllerCats.update)
  .patch(
    "/:id/vaccinated",
    guard,
    validateUpdateStatusCat,
    controllerCats.updateStatus
  )
  .delete("/:id", guard, controllerCats.remove);

module.exports = router;
```

```
/src/api/users/index.js
const express = require("express");
const controllerUsers = require("../controllers/usersController");
const router = express.Router();

const { guard } = require("../helpers/guard");

router
  .post("/registration", controllerUsers.registration)
  .post("/login", controllerUsers.login)
  .post("/logout", guard, controllerUsers.logout);

module.exports = router;
```

```
/src/helpers/guard.js
const passport = require("passport");
require("../configs/passport");
const { HttpStatusCode } = require("../helpers/constants");

const guard = (req, res, next) => {
  passport.authenticate("jwt", { session: false }, (error, user) => {
    //error callback это done из passport.js
    // console.log(req.get("Authorization"));
    const { token } = req.get("Authorization").split(" ");
    if (error || !user || token !== user.token) {
      return next({
        status: HttpStatusCode.FORBIDDEN,
        message: "FORBIDDEN",
      });
    }
    req.user = user;
    // res.locals.user = user - так хочет express (это будет локальная переменная)
    // req.app.locals.user = user - это будет глобальная переменная
    return next();
  })(req, res, next);
};

module.exports = guard;
```

```
/src/validation/catsValidation.js
const Joi = require("joi");
const { HttpStatusCode } = require("../helpers/constants");

const schemaCreateCat = Joi.object({
  name: Joi.string().alphanum().min(2).max(30).required(),
  age: Joi.number().integer().min(0).max(25).required(),
  isVaccinated: Joi.boolean().optional(),
});

const schemaUpdateCat = Joi.object({
  name: Joi.string().alphanum().min(2).max(30).optional(),
  age: Joi.number().integer().min(0).max(25).optional(),
  isVaccinated: Joi.boolean().optional(),
});

const schemaUpdateStatusCat = Joi.object({
  isVaccinated: Joi.boolean().required(),
});

const validate = (schema, body, next) => {
  const { error } = schema.validate(body);
  if (error) {
    const { message } = error.details[0];
    console.log(error.details);
    return next({
      status: HttpStatusCode.BAD_REQUEST,
      code: HttpStatusCode.BAD_REQUEST,
      message: `Field: ${message.replace(/"/g, "")}`,
      data: "Bad request"
    });
  }
  next();
};

module.exports.validateCreateCat = (req, res, next) => {
  return validate(schemaCreateCat, req.body, next);
};

module.exports.validateUpdateCat = (req, res, next) => {
  return validate(schemaUpdateCat, req.body, next);
};

module.exports.validateUpdateStatusCat = (req, res, next) => {
  return validate(schemaUpdateStatusCat, req.body, next);
};
```

```
/src/controllers/usersController.js
const { HttpStatusCode } = require("../helpers/constants");
const { AuthService, UsersService } = require("../services");

const authService = new AuthService();
const usersService = new UsersService();

const { registration } = async (req, res, next) => {
  const { name, email, password, sex } = req.body;
  const user = await usersService.findByIdByEmail(email);
  if (user) {
    // если пользователь уже существует или ранее пер под этим email
    return next({
      status: HttpStatusCode.CONFLICT,
      data: "Conflict",
      message: "This email is already use",
    });
  }
  try {
    const newUser = await usersService.create({ name, email, password, sex });
    return res.status(HttpStatusCode.CREATED).json({
      status: "success",
      code: HttpStatusCode.CREATED,
      data: {
        id: newUser.id,
        email: newUser.email,
        sex: newUser.sex,
      },
    });
  } catch (e) {
    next(e);
  }
};

const login = async (req, res, next) => {
  const { email, password } = req.body;
  try {
    const token = await authService.login({ email, password });
    if (token) {
      return res.status(HttpStatusCode.OK).json({
        status: "success",
        code: HttpStatusCode.OK,
        data: { token },
      });
    }
  } catch (e) {
    next(e);
  }
};

const logout = async (req, res, next) => {
  const { id } = req.user;
  await authService.logout(id);
  return res
    .status(HttpStatusCode.NO_CONTENT)
    .json({ status: "success", code: HttpStatusCode.NO_CONTENT, message: "Nothing" });
};

module.exports = {
  registration,
  login,
  logout,
};
```

```
/src/controllers/catsController.js
const { HttpStatusCode } = require("../helpers/constants");
const { CatsService } = require("../services");

const catsService = new CatsService();

// а guard на каждом req добавим пользователя req.user = user;
const { getAll } = async (req, res, next) => {
  // const { id } = req.user;
  try {
    const { id } = req.user;
    const cats = await catsService.getAll(id);
    return res.status(HttpStatusCode.OK).json({
      status: "success",
      code: HttpStatusCode.OK,
      data: { cats },
    });
  } catch (e) {
    next(e);
  }
};

const { getById } = async (req, res, next) => {
  try {
    const { id } = req.user;
    const cat = await catsService.getById(req.params, id);
    return res.status(HttpStatusCode.OK).json({
      status: "success",
      code: HttpStatusCode.OK,
      data: { cat },
    });
  } else {
    // по умолчанию в app -> app.use(err, req, res, next) => {...
    return next({
      status: HttpStatusCode.NOT_FOUND,
      code: HttpStatusCode.NOT_FOUND,
      message: "Cat not found",
      data: "Not found",
    });
  }
  } catch (e) {
    next(e);
  }
};

const { create } = async (req, res, next) => {
  try {
    const { id } = req.user;
    const cat = await catsService.create(req.body, id);
    return res.status(HttpStatusCode.CREATED).json({
      status: "success",
      code: HttpStatusCode.CREATED,
      data: { cat },
    });
  } else {
    // по умолчанию в app -> app.use(err, req, res, next) => {...
    return next({
      status: HttpStatusCode.NOT_FOUND,
      code: HttpStatusCode.NOT_FOUND,
      message: "Cat not found",
      data: "Not found",
    });
  }
  } catch (e) {
    next(e);
  }
};

const { update } = async (req, res, next) => {
  try {
    const { id } = req.user;
    const cat = await catsService.update(req.params, req.body, id);
    return res.status(HttpStatusCode.OK).json({
      status: "success",
      code: HttpStatusCode.OK,
      data: { cat },
    });
  } else {
    // по умолчанию в app -> app.use(err, req, res, next) => {...
    return next({
      status: HttpStatusCode.NOT_FOUND,
      code: HttpStatusCode.NOT_FOUND,
      message: "Cat not found",
      data: "Not found",
    });
  }
  } catch (e) {
    next(e);
  }
};

const { remove } = async (req, res, next) => {
  try {
    const { id } = req.user;
    const cat = await catsService.remove(req.params, id);
    if (cat) {
      return res.status(HttpStatusCode.OK).json({
        status: "success",
        code: HttpStatusCode.OK,
        data: { cat },
      });
    }
  } else {
    // по умолчанию в app -> app.use(err, req, res, next) => {...
    return next({
      status: HttpStatusCode.NOT_FOUND,
      code: HttpStatusCode.NOT_FOUND,
      message: "Cat not found",
      data: "Not found",
    });
  }
  } catch (e) {
    next(e);
  }
};

module.exports = {
  getAll,
  getById,
  create,
  update,
  updateStatus,
  remove,
};
```