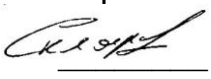
ДОДАТОК А
Текст програми

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник кваліфікаційної роботи

_____ проф. Скляров В.В.
(підпис)

Використання сучасних інструментів тестування та забезпечення якості при створенні веб-застосунку

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

РОЗРОБИВ:
ст. гр. ЗЯм-23-1
Портянніков М.В.

2025 р.

ЗАТВЕРДЖЕНО

Використання сучасних інструментів тестування та забезпечення якості при створенні веб-застосунку

Текст програми

Листів 20

2025 р.

Програмний код застосунку:

```
const Tour = require('../models/tourModel.js');
const { catchAsync } = require('../utils/catchAsync.js');
const AppError = require('../utils/appError.js');
const sharp = require('sharp');
const multer = require('multer');
const {
  deleteOne,
  updateOne,
  createOne,
  getOne,
  getAll,
} = require('./handlerFactory.js');

const multerStorage = multer.memoryStorage();

const multerFilter = (req, file, cb) => {
  if (file.mimetype.startsWith('image')) {
    cb(null, true);
  } else {
    cb(new AppError('Not an image! Please upload only images.', 400),
false);
  }
};

const upload = multer({ storage: multerStorage, fileFilter:
multerFilter });
//Создали middleware, которое будет отвечать за загрузку только одного
изображения
const uploadTourImages = upload.fields(
  //req.files
  { name: 'imageCover', maxCount: 1 },
  { name: 'images', maxCount: 3 },
);

const resizeTourImages = catchAsync(async (req, res, next) => {
  if (!req.files.imageCover || !req.files.images) return next();

  //1) cover image
  const imageCoverFilename = `tour-${req.params.id}-${Date.now()}-
cover.jpeg`;
  await sharp(req.files.imageCover.buffer)
    .resize(2000, 1330)
    .toFormat('jpeg')
    .jpeg({ quality: 100 })
    .toFile(`public/tours/${imageCoverFilename}`);
  req.body.imageCover = imageCoverFilename;
```

```
  //2) images
  req.body.images = [];
  req.files.images.map(async (file, i) => {
    const filename = `tour-${req.params.id}-${Date.now()}-${i +
1}.jpeg`;
    await sharp(file.buffer)
      .resize(2000, 1330)
      .toFormat('jpeg')
      .jpeg({ quality: 100 })
      .toFile(`public/tours/${filename}`);

    req.body.images.push(filename);
  });
  next();
});

const aliasTopTours = function (req, res, next) {
  req.query.sort = '-averageRatings,price';
  req.query.limit = '5';
  // req.query.fields = '';
  next();
};

const getAllTours = getAll(Tour);
const getTour = getOne(Tour, { path: 'reviews' });
const updateTour = updateOne(Tour);
const createTour = createOne(Tour);
const deleteTour = deleteOne(Tour);

const getTourStats = catchAsync(async function (req, res, next) {
  const stats = await Tour.aggregate([
    {
      $match: {
        ratingsAverage: {
          $gte: 4.5,
        },
      },
    },
    {
      $group: {
        _id: { $toUpper: '$difficulty' },
        numTours: { $sum: 1 },
        numRatings: { $sum: '$ratingsQuantity' },
        avgRatings: { $avg: '$ratingsAverage' },
        avgPrice: { $avg: '$price' },
        minPrice: { $min: '$price' },
        maxPrice: { $max: '$price' },
      },
    },
    {
```

```
      $sort: { avgPrice: 1 },
    },
    // {
    //   $match: {
    //     _id: { $ne: 'EASY' },
    //   },
    // },
  ]);

  res.status(200).json({
    status: 'succes',
    data: {
      stats,
    },
  });
});

const getMonthlyPlan = catchAsync(async function (req, res, next) {
  const year = req.params.year * 1;

  const plan = await Tour.aggregate([
    {
      $unwind: '$startDates',
    },
    {
      $match: {
        startDates: {
          $gte: new Date(`${year}-01-01`),
          $lte: new Date(`${year}-12-31`),
        },
      },
    },
    {
      $group: {
        _id: { $month: '$startDates' },
        numTourStarts: { $sum: 1 },
        tours: { $push: '$name' },
      },
    },
    {
      $addFields: { month: '$_id' },
    },
    {
      $project: {
        _id: 0,
      },
    },
    {
      $sort: {
        numTourStarts: -1,
      },
```

```
    },
  ]);

  res.status(200).json({
    status: 'succes',
    data: {
      plan,
    },
  });
});

// /tours-within/:distance/center/:latlng/unit/:unit
const getToursWithin = catchAsync(async function (req, res, next) {
  const { distance, latlng, unit } = req.params;
  const [lat, lng] = latlng.split(',');

  if (!lat || !lng) {
    return next(
      new AppError(
        'Please provide latitude and longitude in the format lat,lng',
        400,
      ),
    );
  }

  const radius = unit === 'mi' ? distance / 3963.2 : distance /
6378.1;

  const tours = await Tour.find({
    startLocation: { $geoWithin: { $centerSphere: [[lng, lat], radius]
} },
  });

  res.status(200).json({
    status: 'success',
    result: tours.length,
    data: {
      data: tours,
    },
  });
});

// /distances/:latlng/unit/:unit
const getDistances = catchAsync(async function (req, res, next) {
  const { latlng, unit } = req.params;
  const [lat, lng] = latlng.split(',');

  if (!lat || !lng) {
    return next(
      new AppError(
        'Please provide latitude and longitude in the format lat,lng',
```

```
        400,
      ),
    );
  }

  const multiplier = unit === 'mi' ? 0.000621371 : 0.001;

  const distances = await Tour.aggregate([
    {
      $geoNear: {
        near: { type: 'Point', coordinates: [lng * 1, lat * 1] },
        distanceField: 'distance',
        distanceMultiplier: multiplier,
      },
    },
    {
      $project: {
        distance: 1,
        name: 1,
      },
    },
  ]);

  res.status(200).json({
    status: 'success',
    data: {
      data: distances,
    },
  });
});

module.exports = {
  getAllTours,
  getTour,
  createTour,
  updateTour,
  deleteTour,
  aliasTopTours,
  getTourStats,
  getMonthlyPlan,
  getToursWithin,
  getDistances,
  uploadTourImages,
  resizeTourImages,
};

const User = require('../models/userModel');
const { catchAsync } = require('../utils/catchAsync');
const AppError = require('../utils/appError');
const jwt = require('jsonwebtoken');
const crypto = require('crypto');
```

```
const { promisify } = require('util');
const { Email } = require('../utils/email');

const createSendToken = function (user, statusCode, res) {
  const token = jwt.sign({ id: user._id }, process.env.SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN,
  });

  const cookieOptions = {
    expires: new Date(
      Date.now() + process.env.JWT_COOKIE_EXPIRES_IN * 24 * 60 * 60 *
1000,
    ),
    httpOnly: true,
  };

  if (process.env.NODE_ENV === 'production') cookieOptions.secure =
true;

  res.cookie('jwt', token, cookieOptions);

  //remove password from output
  user.password = undefined;

  res.status(statusCode).json({
    status: 'success',
    token,
    data: {
      user,
    },
  });
};

const signup = catchAsync(async function (req, res, next) {
  const newUser = await User.create({
    name: req.body.name,
    email: req.body.email,
    password: req.body.password,
    passwordConfirm: req.body.passwordConfirm,
  });

  const url = `${req.protocol}://${req.get('host')}/me`;
  // await new Email(newUser, url).sendWelcome();

  createSendToken(newUser, 201, res);
});

const login = catchAsync(async function (req, res, next) {
  const { email, password } = req.body;

  //1) check if email and password exists\
```

```
  if (!email || !password) {
    return next(new AppError('Provide correct email and password',
400));
  }
  //2) check if user exists && password is correct
  const user = await User.findOne({ email }).select('+password');
  // const passwordCompare = await bcrypt.compare(password,
user.password);

  if (!user || !(await user.correctPassword(password, user.password)))
{
    return next(new AppError('Incorrect email or password', 401));
  }

  //3) if everything ok, send token to client
  createSendToken(user, 200, res);
});

const logout = function (req, res) {
  res.cookie('jwt', 'loggedout', {
    expires: new Date(Date.now() + 10 * 1000),
    httpOnly: true,
  });
  res.status(200).json({ status: 'success' });
};

const protect = catchAsync(async function (req, res, next) {
  //1)getting token and check of it's there
  let token;
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith('Bearer')
  ) {
    token = req.headers.authorization.split(' ')[1];
  } else if (req.cookies.jwt) {
    token = req.cookies.jwt;
  }

  if (!token) {
    return next(
      new AppError('Your not logged in! Please log in to get access.',
401),
    );
  }

  //2)varification token
  const decoded = await promisify(jwt.verify)(token,
process.env.SECRET);
  // console.log(decoded);

  //3)check if user still exists
```

```
  const currentUser = await User.findById(decoded.id);
  if (!currentUser) {
    return next(new AppError('User does not have exist!'));
  }
  //4)check if user changed password after the token was issued
  if (currentUser.changedPasswordAfter(decoded.iat)) {
    return next(
      new AppError('User recently changed password! Log in again!',
401),
    );
  }

  //grant access to protected route
  req.user = currentUser;
  res.locals.user = currentUser;
  next();
});

//Only render pages, no errors!
const isLoggedIn = async function (req, res, next) {
  if (req.cookies.jwt) {
    try {
      //1) verify token
      const decoded = await promisify(jwt.verify)(
        req.cookies.jwt,
        process.env.SECRET,
      );

      //2) check if user still exists
      const currentUser = await User.findById(decoded.id);
      if (!currentUser) {
        return next();
      }
      //3)check if user changed password after the token was issued
      if (currentUser.changedPasswordAfter(decoded.iat)) {
        return next();
      }

      //there is a logged in user
      res.locals.user = currentUser;
      return next();
    } catch (err) {
      return next();
    }
  }
  next();
};

const restrictTo = function (...roles) {
  return function (req, res, next) {
    //roles - array
```

```
    if (!roles.includes(req.user.role)) {
      return next(
        new AppError('You do not have permission to perform this
action', 403),
      );
    }
    next();
  };
};

const forgotPassword = catchAsync(async function (req, res, next) {
  //1) get user based on POSTed email
  const user = await User.findOne({ email: req.body.email });

  if (!user) {
    return next(new AppError('There is no user with email address.',
404));
  }

  //2) generate the random reset token
  const resetToken = user.createPasswordResetToken();
  await user.save({ validateBeforeSave: false });

  //3) send it to user's email
  try {
    const resetURL =
`${req.protocol}://${req.get('host')}/api/v1/users/resetPassword/${res
etToken}`;
    await new Email(user, resetURL).sendPasswordReset();

    res.status(200).json({
      status: 'success',
      message: 'Token send to email!',
    });
  } catch (err) {
    user.passwordResetToken = undefined;
    user.passwordResetExpires = undefined;
    await user.save({ validateBeforeSave: false });

    return next(
      new AppError(
        'There was an error sending the email. Try again later!',
        500,
      ),
    );
  }
});

const resetPassword = catchAsync(async function (req, res, next) {
  //1)get user based on the token
  const hashedToken = crypto
```

```
    .createHash('sha256')
    .update(req.params.token)
    .digest('hex');

  const user = await User.findOne({
    passwordResetToken: hashedToken,
    passwordResetExpires: { $gt: Date.now() },
  });

  //2)if token has not expires,and there is user, set the new password
  if (!user) {
    return next(new AppError('Token is invalid or has expired', 400));
  }
  user.password = req.body.password;
  user.passwordConfirm = req.body.passwordConfirm;
  user.passwordResetExpires = undefined;
  user.passwordResetToken = undefined;
  await user.save();
  //3)update changedPasswordAt property for the user

  //4)log the user in, send jwt

  const token = jwt.sign({ id: user._id }, process.env.SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN,
  });
  res.status(200).json({
    status: 'success',
    token,
  });
});

const updatePassword = catchAsync(async function (req, res, next) {
  //1) get user from collection
  const user = await User.findById(req.user.id).select('+password');

  //2) check if POSTed current password is correct
  if (!(await user.correctPassword(req.body.passwordCurrent,
user.password))) {
    return next(new AppError('Your current password is wrong.', 401));
  }
  //3) if so, update password
  user.password = req.body.password;
  user.passwordConfirm = req.body.passwordConfirm;
  await user.save();

  //4) log user in, send JWT
  createSendToken(user, 200, res);
});

module.exports = {
  signup,
```

```
  login,
  logout,
  protect,
  isLoggedIn,
  restrictTo,
  forgotPassword,
  resetPassword,
  updatePassword,
};

const mongoose = require('mongoose');
const validator = require('validator');
const bcrypt = require('bcryptjs');
const crypto = require('crypto');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please tell us your name!'],
  },
  email: {
    type: String,
    required: [true, 'A user mast have a email'],
    unique: true,
    lowercase: true,
    validate: {
      validator: validator.isEmail,
      message: 'Please provide a valid email',
    },
  },
  photo: {
    type: String,
    default: 'default.jpg',
  },
  role: {
    type: String,
    enum: ['admin', 'user', 'guide', 'lead-guide'],
    default: 'user',
  },
  active: {
    type: Boolean,
    default: true,
    select: false,
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 8,
    select: false,
  },
  passwordCurrent: String,
```

```
  passwordConfirm: {
    type: String,
    required: [true, 'Please confirm your password'],
    validate: {
      validator: function (val) {
        return val === this.password;
      },
      message: 'Passwords are not the same! ',
    },
  },
  passwordChangedAt: Date,
  passwordResetToken: String,
  passwordResetExpires: Date,
});

userSchema.pre('save', async function (next) {
  //only run this func if pass was actually modified
  if (!this.isModified('password')) return next();

  //hash the pass with cost of 12
  this.password = await bcrypt.hash(this.password, 14);

  //delete passwordConfirm field
  this.passwordConfirm = undefined;
  next();
});

userSchema.pre('save', async function (next) {
  if (!this.isModified('password') || this.isNew) return next();

  this.changedPasswordAt = Date.now();
  next();
});

userSchema.pre(/^find/, function (next) {
  this.find({ active: { $ne: false } });

  next();
});

userSchema.methods.correctPassword = async function (
  candidatePassword,
  userPassword,
) {
  return await bcrypt.compare(candidatePassword, userPassword);
};

userSchema.methods.changedPasswordAfter = function (JWTTimestamp) {
  if (this.passwordChangedAt) {
    const changedTimestamp = parseInt(
      this.passwordChangedAt.getTime() / 1000,
```

```
      10,
    );

    return JWTTimestamp < changedTimestamp; //100 < 200
  }

  //false means not changed
  return false;
};

userSchema.methods.createPasswordResetToken = function () {
  const resetToken = crypto.randomBytes(32).toString('hex');
  this.passwordResetToken = crypto
    .createHash('sha256')
    .update(resetToken)
    .digest('hex');
  this.passwordResetExpires = Date.now() + 10 * 60 * 1000;

  return resetToken;
};

module.exports = mongoose.model('User', userSchema);

const express = require('express');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const mongoSanitize = require('express-mongo-sanitize');
const xss = require('xss-clean');
const hpp = require('hpp');
const fs = require('fs');
const path = require('path');
const cookieParser = require('cookie-parser');
const cors = require('cors');
const compression = require('compression');

const AppError = require('./utils/appError.js');
const { globalErrorHandler } =
require('./controllers/errorController.js');
const tourRouter = require('./routes/tourRoutes.js');
const userRouter = require('./routes/userRoutes.js');
const reviewRouter = require('./routes/reviewRoutes.js');
const viewRouter = require('./routes/viewRoutes.js');
const bookingRouter = require('./routes/bookingRutes.js');

//swagger
const swaggerUi = require('swagger-ui-express');
const { login } = require('./controllers/authController.js');
const swaggerDocument = JSON.parse(
  fs.readFileSync('./swagger-output.json', 'utf-8'),
);
```

```
const app = express();

app.set('view engine', 'pug');
app.set('views', path.join(__dirname, 'views'));

//global middleware

//serving static files
// app.use(express.static(`${__dirname}/public`));
app.use(express.static(path.join(__dirname, 'public')));

app.use(cors());

//set security http headers
app.use(helmet());

//development logging
if (process.env.NODE_ENV === 'development') {
  app.use(morgan('dev'));
}

//limit requests from same API
const limiter = rateLimit({
  max: 200,
  windowMs: 60 * 60 * 1000,
  message: 'To many requests from this IP, please try again in an
hour!',
});
app.use('/api', limiter);

//body parser, reading data from body into req.body
app.use(express.json({ limit: '10kb' }));
app.use(cookieParser());

//data sanitization against NoSQL query injection
app.use(mongoSanitize());

//data sanitization against XSS
app.use(xss());

//prevent HTTP parameter pollution
app.use(
  hpp({
    whitelist: [
      'duration',
      'price',
      'difficulty',
      'ratingsAverage',
      'ratingsQuantity',
      'maxGroupSize',
```

```
    ],
  }),
);

app.use(compression());

//middleware будет срабатывать при поступлении нового запроса
//Middleware который добавляет в объект запроса время, когда был
выполнен запрос, просто добавим это свойство в ответы и все

app.use((req, res, next) => {
  req.requestTime = new Date().toISOString();
  // console.log(req.cookies);
  next();
});

app.use((req, res, next) => {
  res.setHeader(
    'Content-Security-Policy',
    "img-src 'self' data: https://a.tile.openstreetmap.org
https://b.tile.openstreetmap.org https://c.tile.openstreetmap.org",
  );
  next();
});

//mounting
//Как только достигнем middleware tourRouter, userRouter то мы
отправимся и будем выполнять код который содержит этот middleware, в
данном случае мы используем middleware param и выполним код для одного
из маршрутов, что будет получен из запроса

app.use('/api/v1/tours', tourRouter);
app.use('/api/v1/users', userRouter);
app.use('/api/v1/reviews', reviewRouter);
app.use('/api/v1/booking', bookingRouter);
app.use('/', viewRouter);

app.use('/api-docs', swaggerUi.serve,
swaggerUi.setup(swaggerDocument));
// console.log('Swagger UI is being initialized at /api-docs');

app.all('*', function (req, res, next) {
  // const err = new Error(`Can't find ${req.originalUrl} on this
server!`);
  // err.status = 'fail';
  // err.statusCode = 404;

  next(new AppError(`Can't find ${req.originalUrl} on this server!`,
404));
});
```

```
app.use(globalErrorHandler);

module.exports = app;

const request = require('supertest');
require('dotenv').config({ path: './.env.test' });
const mongoose = require('mongoose');
const Tour = require('../models/tourModel'); // Модель
const app = require('../app'); // Экспортированное приложение
const { expect } = require('@jest/globals');

// console.log('DB URI:', process.env.MONGO_URL);
const t = {
  name: 'Secret Tour',
  duration: 9,
  maxGroupSize: 20,
  difficulty: 'easy',
  ratingsAverage: 4.6,
  ratingsQuantity: 54,
  price: 1197,
  summary: "Living the life of Wanderlust in the US' most beatiful
cities",
  description:
    'Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat lorem ipsum dolor sit amet.\nConsectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur, nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat!',
  imageCover: 'tour-4-cover.jpg',
  images: ['tour-4-1.jpg', 'tour-4-2.jpg', 'tour-4-3.jpg'],
  startDates: ['2021-03-11,10:00', '2021-05-02,10:00', '2021-06-
09,10:00'],
  secretTour: true,
  startLocation: {
    type: 'Point',
    coordinates: [-80.185942, 25.774772],
    address: '301 Biscayne Blvd, Miami, FL 33132, USA',
    description: 'Miami, USA',
  },
};

beforeAll(async () => {
  await mongoose.connect(process.env.MONGO_URL);

  console.log('MongoDB Connected:', mongoose.connection.readyState);
// Должно быть "1"
});
```

```
afterAll(async () => {
  await mongoose.disconnect();
});

afterEach(async () => {
  await Tour.deleteMany({ name: 'Secret Tour' });
});

describe('GET /api/v1/tours', () => {
  it('must return all documents with the correct status and result',
async () => {
    const response = await
request(app).get('/api/v1/tours').expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(9);
    expect(response.body.data.data.length).toBe(9);
    expect(response.body.data.data[0]).toHaveProperty(
      'name',
      'The Sea Explorer',
    );
    expect(response.body.data.data[1]).toHaveProperty(
      'name',
      'The Park Camper',
    );
  });

  it('should correctly filter data by query parameters', async () => {
    const response = await request(app)
      .get('/api/v1/tours?price[gte]=500')
      .expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(7);
    expect(response.body.data.data[0]).toHaveProperty(
      'name',
      'The Snow Adventurer',
    );
  });

  it('testing the limit and page parameters', async () => {
    const response = await
request(app).get('/api/v1/tours?limit=1&page=2');

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.length).toBe(1);
  });
});

describe('POST /api/v1/tours', () => {
  it('create new tour', async () => {
```

```
    const response = await request(app).post('/api/v1/tours').send(t);

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.name).toBe(t.name);
  });
});

const request = require('supertest');
require('dotenv').config({ path: './.env.test' });
const mongoose = require('mongoose');
const User = require('../models/userModel'); // Модель
const app = require('../app'); // Экспортированное приложение
const { expect } = require('@jest/globals');

// console.log('DB URI:', process.env.MONGO_URL);

beforeAll(async () => {
  await mongoose.connect(process.env.MONGO_URL);

  console.log('MongoDB Connected:', mongoose.connection.readyState);
// Должно быть "1"
});

afterAll(async () => {
  await mongoose.disconnect();
});

afterEach(async () => {
  await User.deleteMany({ email: user.email });
});

const user = {
  name: 'new user',
  email: 'newemail@gmail.com',
  password: 'pass1234',
  passwordConfirm: 'pass1234',
};

describe('GET /api/v1/users', () => {
  it('should get all users with the admin role', async () => {
    const response = await
request(app).get('/api/v1/users?role=admin');

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(1);
    expect(response.body.data.data[0]).toHaveProperty('role',
'admin');
  });
});

describe('POST /api/v1/users', () => {
```

```
  it('Create new user', async () => {
    const response = await request(app)
      .post('/api/v1/users')
      .send(user)
      .expect(201);

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.name).toBe(user.name);
  });

  it('signup', async () => {
    const response = await request(app)
      .post('/api/v1/users/signup')
      .send(user)
      .expect(201);

    expect(response.body.status).toBe('success');
    expect(response.body).toHaveProperty('token');
    expect(response.body.data.user.email).toBe(user.email);
  });

  it('login', async () => {
    const response = await request(app)
      .post('/api/v1/users/login')
      .send({ email: 'm.portyannikov@gmail.com', password: 'pass1234'
})
      .expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body).toHaveProperty('token');

expect(response.body.data.user.email).toBe('m.portyannikov@gmail.com')
;
  });
});
const nodemailer = require('nodemailer');
const pug = require('pug');
const htmlToText = require('html-to-text');

class Email {
  constructor(user, url) {
    this.to = user.email;
    this.firstName = user.name.split(' ')[0];
    this.url = url;
    this.from = `Maxim Portyannikov <${process.env.EMAIL_FROM}>`;
  }

  //с помощью метода мы создаем траспортер, который отвечает за
отправку писем и вызовем его в методе send, который будет
использоваться в других методах как sendWelcome и другие
  newTransport() {
```

```
    if (process.env.NODE_ENV === 'production') {
      //sendgrid
      return 1;
    }

    return nodemailer.createTransport({
      host: process.env.EMAIL_HOST,
      port: process.env.EMAIL_PORT,
      secure: true,
      auth: {
        user: process.env.EMAIL_FROM,
        pass: process.env.EMAIL_PASSWORD,
      },
    });
  }

  async send(template, subject) {
    //1) render html based on a pug template
    const html = pug.renderFile(
      `${__dirname}/../views/emails/${template}.pug`,
      {
        firstName: this.firstName,
        url: this.url,
        subject,
      },
    );

    //2) define email options
    const mailOptions = {
      from: this.from,
      to: 'm.portyannikov@gmail.com', // this.to
      subject: subject,
      html: html,
      text: htmlToText.convert(html),
    };

    //3) create a transport and send email
    await this.newTransport().sendMail(mailOptions);
  }

  async sendWelcome() {
    await this.send('welcome', 'Welcome to the Natours Family');
  }

  async sendPasswordReset() {
    await this.send(
      'passwordReset',
      'Your password reset token (valid for only 10 minutes)',
    );
  }
}
```