KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

T120B162 Programų sistemų testavimas

IFF-6/11 Nerijus Dulkė

Data: 2019.11.11

KAUNAS
2019

API test cases

Account login tests

√ should login to existing account

√ should fail when email is incorrect

√ should fail when password is incorrect

Account registration tests

√ should create user account

√ should validate user data correctly

√ should not create account with existing email

Account verify tests

√ should verify token

√ should fail with incorrect token

Category get all tests

√ should get categories for user

Category get by id tests

√ should get category by id

√ should fail when category belongs to another user

√ should fail when category does not exist

Category post tests

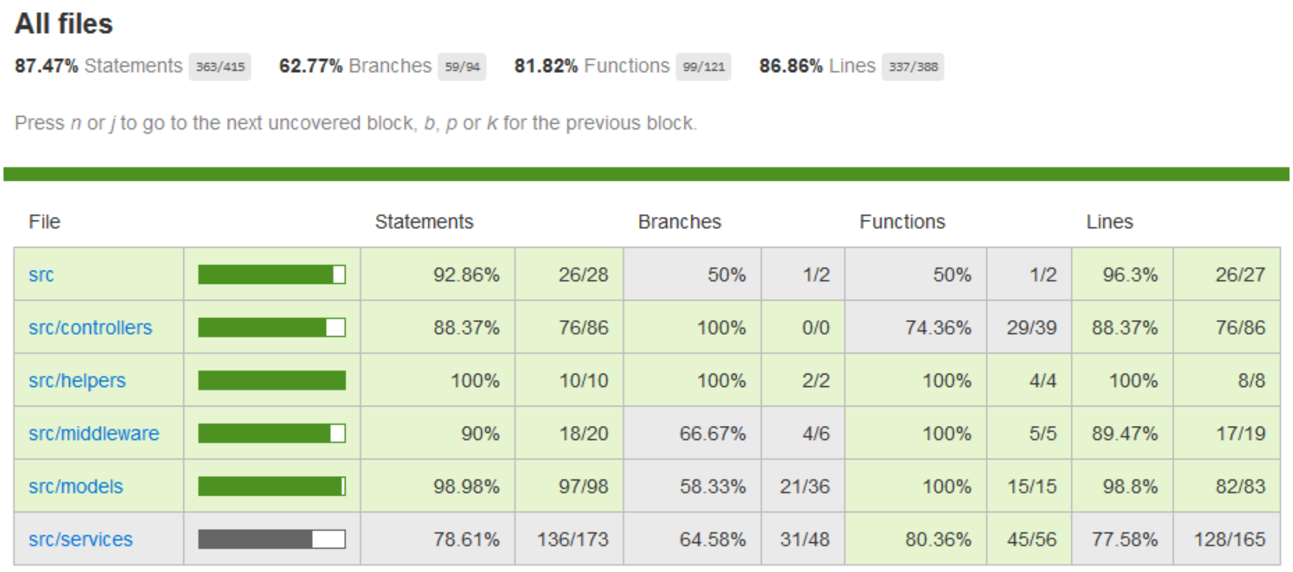
√ should create category

√ should edit category

```
Category delete tests
  √ should delete category
  √ should not delete category of different user

Roadmap get all tests
  √ should get roadmaps for user

Roadmap get by id tests
  √ should get roadmap by id
  √ should get roadmap with categories by id
  √ should get roadmap with tasks by id
  √ should fail when roadmap belongs to another user
  √ should fail when roadmap does not exist

Roadmap post tests
  √ should create roadmap
  √ should edit roadmap

Roadmap delete tests
  √ should delete roadmap
  √ should not delete roadmap of different user

Task get all tests
  √ should get tasks for user
```

```
Task get by id tests
  √ should get task by id
  √ should fail when task belongs to another user
  √ should fail when task does not exist

Task post tests
  √ should create task
  √ should fail when category is incorrect
  √ should fail when roadmap is incorrect
  √ should edit task

Task delete tests
  √ should delete task
  √ should not delete task of different user
```

API Test coverage



All files

87.47% Statements 363/415 **62.77%** Branches 59/94 **81.82%** Functions 99/121 **86.86%** Lines 337/388

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		92.86%	26/28	50%	1/2	50%	1/2	96.3%	26/27
src/controllers		88.37%	76/86	100%	0/0	74.36%	29/39	88.37%	76/86
src/helpers		100%	10/10	100%	2/2	100%	4/4	100%	8/8
src/middleware		90%	18/20	66.67%	4/6	100%	5/5	89.47%	17/19
src/models		98.98%	97/98	58.33%	21/36	100%	15/15	98.8%	82/83
src/services		78.61%	136/173	64.58%	31/48	80.36%	45/56	77.58%	128/165

Used mocks

Local storage mock

```
class LocalStorageMock {
  constructor() {
    this.store = {};
  }

  clear() {
    this.store = {};
  }

  getItem(key) {
    return this.store[key] || null;
  }

  setItem(key, value) {
    this.store[key] = (value || '').toString();
  }

  removeItem(key) {
    delete this.store[key];
  }
}

export default () => {
  global.localStorage = new LocalStorageMock();
};
```

Vuex store mock

```
const expectFunc = (commitParam1, commitParam2, funcName, param) => {
  if (funcName) {
    expect(commitParam1).toBe(funcName);
    if (param && typeof param === 'object') {
      expect(commitParam2).toMatchObject(param);
    } else {
      expect(commitParam2).toBe(param);
    }
  }
};

export default (state, funcName, param) => ({
  state,
  commit: (commitParam1, commitParam2) => expectFunc(commitParam1, commitParam2, fu
ncName, param),
  dispatch: (actionParam1, actionParam2) => expectFunc(actionParam1, actionParam2,
funcName, param)
});
```

Test code

accountLogin.spec.ts

```
import chai from 'chai';
import 'mocha';
import supertest from 'supertest';
import app from '../../../app';
import { User } from '../../../models';
import resources from '../../../resources';
import * as database from '../../../services/databaseService';
import entityFactory from '../../helpers/entityFactory';

chai.should();

let server: supertest.SuperTest<supertest.Test>;
let existingUser: User;
before(async () => {
  server = supertest.agent(app);
  await database.initPromise;
  existingUser = await entityFactory.createAccount();
});
after(async () => {
  await database.close();
});
```

```
describe('Account login tests', () => {
  it('should login to existing account', async () => {
    const response = await server
      .post('/account/login')
      .send({
        email: existingUser.email,
        password: entityFactory.defaultPassword
      });

    response.status.should.equal(200);
    response.body.email.should.equal(existingUser.email);
    response.body.token.should.be.a('string');
  });

  it('should fail when email is incorrect', async () => {
    const response = await server
      .post('/account/login')
      .send({
        email: 'a' + existingUser.email,
        password: entityFactory.defaultPassword
      });

    response.status.should.equal(400);
    response.body.message.should.equal(resources.Login_EmailIncorrect);
  });

  it('should fail when password is incorrect', async () => {
    const response = await server
      .post('/account/login')
      .send({
        email: existingUser.email,
        password: 'a' + entityFactory.defaultPassword
      });

    response.status.should.equal(400);
    response.body.message.should.equal(resources.Login_PasswordIncorrect);
  });
});
```

accountRegister.spec.ts

```
import chai from 'chai';
import 'mocha';
import supertest from 'supertest';
import app from '../../../app';
import { User } from '../../../models';
import resources from '../../../resources';
import * as database from '../../../services/databaseService';
import entityFactory from '../../helpers/entityFactory';

chai.should();

let server: supertest.SuperTest<supertest.Test>;
before(async () => {
  server = supertest.agent(app);
  await database.initPromise;
});
after(async () => {
  await database.close();
});

describe('Account registration tests', () =>  {
  it('should create user account', async () => {
    const user = {
      email: 'test@account.com',
      name: 'name',
      password: 'password'
    };

    const response = await server
      .post('/account/register')
      .send(user);

    response.status.should.equal(200);
    const createdUser =  await database
      .connection()
      .manager
      .findOne(User, {
        email: user.email,
        name: user.name
      });

    chai.should().exist(createdUser);
    const passCorrect = await createdUser!.comparePasswords(user.password);
    passCorrect.should.equal(true);
  });

  it('should validate user data correctly', async () => {
    const response = await server.post('/account/register');

    response.status.should.equal(400);
    response.body.message.should.equal(resources.Generic_ValidationError);
    response.body.data.should.be.an('array');

    const emailErrors = response.body.data.find((x: any) => x.property === 'email')
;
```

```
    chai.should().exist(emailErrors);
    emailErrors.errors.should.have.lengthOf(2);

    const nameErrors = response.body.data.find((x: any) => x.property === 'name');
    chai.should().exist(nameErrors);
    nameErrors.errors.should.have.lengthOf(2);

    const passwordErrors = response.body.data.find((x: any) => x.property === 'pass
word');
    chai.should().exist(passwordErrors);
    passwordErrors.errors.should.have.lengthOf(2);
  });

  it('should not create account with existing email', async () => {
    const user = {
      email: '',
      name: 'name',
      password: 'password'
    };

    const existingUser = await entityFactory.createAccount();
    user.email = existingUser.email;

    const response = await server.post('/account/register').send(user);

    response.status.should.equal(400);
    response.body.message.should.equal(resources.Registration_EmailExists);
  });
});
```

accountVerify.spec.ts

```
import chai from 'chai';
import 'mocha';
import supertest from 'supertest';
import app from '../../../app';
import * as database from '../../../services/databaseService';
import entityFactory from '../../helpers/entityFactory';

chai.should();

let server: supertest.SuperTest<supertest.Test>;
before(async () => {
  server = supertest.agent(app);
  await database.initPromise;
});
after(async () => {
  await database.close();
});

describe('Account verify tests', () => {
  it('should verify token', async () => {
    const user = await entityFactory.createAccount();
    const token = entityFactory.loginWithAccount(user).token;

    const response = await server.get('/account/verify').set('Authorization', `Bear
er ${token}`);

    response.status.should.equal(200);
    response.body.email.should.equal(user.email);
    response.body.token.should.equal(token);
  });

  it('should fail with incorrect token', async () => {
    const response = await server.get('/account/verify').set('Authorization', 'Bear
er someRandom.Incorrect.Token');
    response.status.should.equal(401);
  });
});
```

Conclusion

Mocks were used in frontend to replace local storage and vuex store.

Backend code has test coverage of 87%. Some new code hasn't been covered yet and some code can't be covered (or takes too much effort for a small value) like some cases of error handling. For backend testing separate database was used, which is cleared before each test run. Test data is inserted into the database and API methods are called, and tests check if the response is as expected.