KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

T120B162 Programų sistemų testavimas

IFF-6/11 Nerijus Dulkė

Data: 2019.12.09

KAUNAS
2019

Task 1. Code review

Code review checklist:

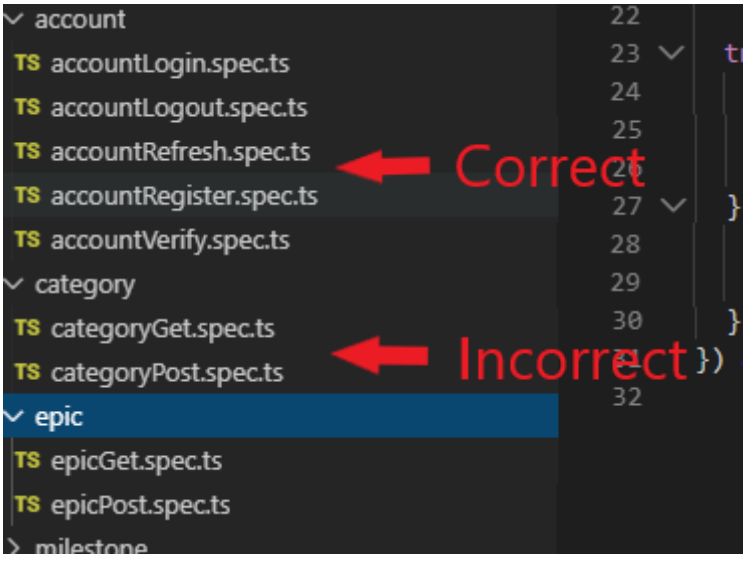
- No unused variables
- Names are consistent
- Consistent code style
- No duplicated logic
- Code is readable and clear upon initial reading
- Code that is unclear has an effective comment
- Unexpected errors are handled
- No commented-out code
- At least 80% coverage on new code
- All tests are passing
- Tech-debt registered to the log

Found issues after code review:

- All controllers implementing CRUD actions have a lot of duplication. Can be split into more generic service or helper.
- Handling promises should be consistent (either *.then()* or *async/await*, not both).
- Authentication middleware should have a more robust way of defining public URLs.

```
const publicUrls = [
  '/api/account/login',
  '/api/account/register',
  '/api/account/refresh',
  '/api/account/logout',
  '/api/health',
  '/swagger'
];      =, a year ago • setu
```

- Test files names are not clear.



- Unresolved *TODO* tag in *app.ts*

```
app.use('/api/roadmaps', RoadmapController); // TODO: move to function in controllers/index.ts
```

Task 2. Static code analysis

For static code analysis TSLint was used (ESLint version for TypeScript). Since this tool was integrated into development environment during the development process, all issues raised are fixed during the development of features.

All recommended rules with a few exceptions are used for analysis (available from https://github.com/palantir/tslint/blob/master/src/configs/recommended.ts).

TSLint config file:

```
{
    "defaultSeverity": "warning",
    "extends": [
        "tslint:recommended"
    ],
    "jsRules": {},
    "rules": {
        "quotemark": [true, "single"],
        "no-console": false,
        "object-literal-sort-keys": false,
        "trailing-comma": false,
        "no-unused-expression": false
    },
    "rulesDirectory": []
}
```

Command to show existing errors:

```
"lint": "tslint -c tslint.json 'src/**/*.ts'"
```

Command output:

```
$ npm run lint

> map-it-api@1.0.0 lint C:\Users\NerijusDulke\Documents\stuff\map-it\api
> tslint -c tslint.json 'src/**/*.ts'
```

Task 3. Custom rule

A custom TSLint rule was added to prevent throwing generic *Error* types. Only extended error types can be thrown to be more understandable, for example custom *HttpError* can be thrown.

Custom rule code:

```
import * as Lint from 'tslint';
import * as ts from 'typescript';

export class Rule extends Lint.Rules.AbstractRule {
  public static FAILURE_STRING = 'no generic errors';

  public apply(sourceFile: ts.SourceFile): Lint.RuleFailure[] {
    return this.applyWithWalker(new NoGemericErrorsWalker(sourceFile, this.getOptio
ns()));
  }
}

// tslint:disable-next-line: max-classes-per-file
class NoGemericErrorsWalker extends Lint.RuleWalker {
  public visitThrowStatement(node: ts.ThrowStatement) {
    if (node.expression.getChildCount() > 1 && node.expression.getChildAt(1).getTex
t() === 'Error') {
      this.addFailure(this.createFailure(node.getStart(), node.getWidth(), Rule.FAI
LURE_STRING));
    }

    super.visitThrowStatement(node);
  }
}
```

Output after running TSLint with custom rule:

```
$ npm run lint

> map-it-api@1.0.0 lint C:\Users\NerijusDulke\Documents\stuff\map-it\api
> tslint -c tslint.json 'src/**/*.ts'


WARNING: src/services/accountService.ts[100, 7]: no generic errors
WARNING: src/tests/helpers/entityFactory.ts[16, 5]: no generic errors
```

Two issues were found:

```
public async register(newUser: User) {
  await validate(newUser);
  const existingUser = await connection()
    .manager
    .findOne(User, { email: newUser.email });

  if (existingUser) {
    throw new HttpError(resources.Registration_EmailExists, 400);
  }
  const pass = await authService.encryptPassword(newUser.password);

  if (!pass) {
    throw new Error('Password hashing failed');
  }
  newUser.password = pass;       You, 17 days ago • use async/await

  const res = await connection()
    .getRepository(User)
    .save(newUser);

  delete res.password;
  return res;
}
```

```
const createAccount = async (modifier?: (u: User) => User) => {
  let user = new User();
  user.name = shortid.generate();
  user.email = shortid.generate() + '@email.com';

  const pass = await authService.encryptPassword(defaultPasswor

  if (!pass) {
    throw new Error('Password hashing failed');
  }
  user.password = pass;

  if (modifier) {
    user = modifier(user);
  }

  return connection().manager.save(user);
};
```

Conclusion

Found some tech-debt issues that cause bad code readability and maintainability. Project is constantly analyzed during development with static code analysis tool. Also learned how to define mt own custom rule.