

Documentation of Sprint #3: Final Project

12.12.2014

Names:

Boyko Surlev

Nikolaj Desting

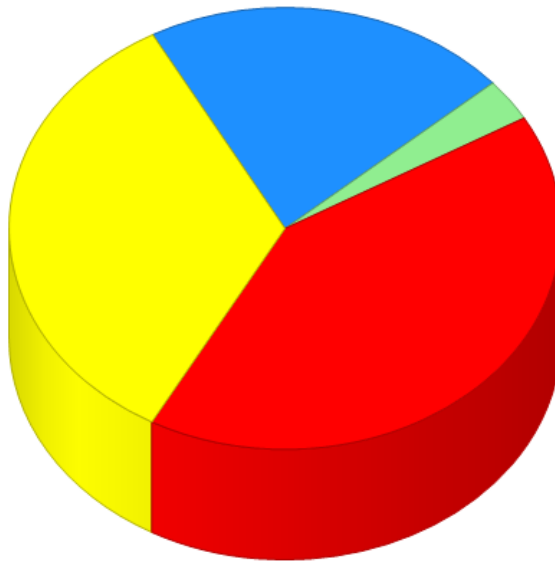
Peter Tomascik



GIT overview

5079 Lines of code

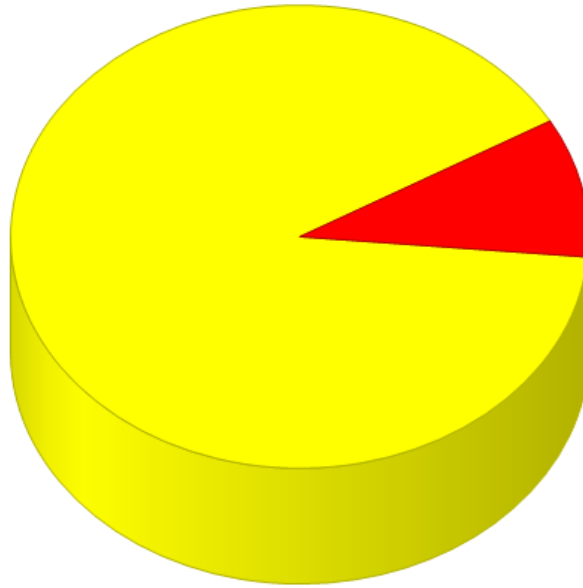
2100 Lines of code in .js files (41.3%)
1728 Lines of code in .html files (34.0%)
1093 Lines of code in .xml files (21.5%)
158 Lines of code in .css files (3.1%)



Excluding merges, **4 authors** have pushed **30 commits** to master and **49 commits** to all branches. On master, **63 files** have changed and there have been **3,752 additions** and **852 deletions**.

494 Lines of test code

494 Lines of testcode (9,7%)
4585 Lines of production code (90,3%)



Sprint #3		
SPRINT BACKLOG	IN PROGRESS	DONE
		<div>As an admin/user, I want to browse a "Stock"</div> <div>As an admin/user, I want to browse a "Product" page, so I can select quantity and add it to a basket</div> <div>As an admin, I want to browse an "Order"</div> <div>As an admin, I want to browse a</div> <div>MongoDB</div> <div>admin functionality (REST_api)</div> <div>customer functionality (REST_api)</div> <div>completed / tested JPA with extra data as planned</div> <div>Login (connection between servers and front-end)</div> <div> <div>Front-end (AngularJS) <ul style="list-style-type: none"> - user can edit old orders - get, getAll, push, put, delete users - user can push, put, delete, get himself - logic for Login + security - tests </div> <div> <div>AngularJS</div> <div>- user/ admin logic</div> <div>USER</div> <div>- user add products to basket</div> <div>- user can create orders from basket</div> <div>- user can edit orders in the basket</div> <div>ADMIN</div> <div>- get , getAll, push, put, delete products and orders</div> </div> <div>filters</div> </div>

Back-end

Rest_API (Node) + Mocha tests

The screenshot shows the Mocha test runner interface. On the left, the 'Test Results' pane lists the following tests:

- Backend testing of Admin REST-API (Order)
 - Testing /order (Get Orders) - Should return all orders.
 - Testing /order/:id (Get Order) - Should return order by id.
 - Testing /order (Create Order) - Should return the added object.
 - Testing /order/:id (Delete Order) - Should return 1, which means deleted.
 - Testing /order/:id (Edit Order) - Should return the new edited object.
- Backend testing of Admin REST-API (Product)
 - Testing /product (Get Products) - Should return all products.
 - Testing /product/:id (Get Product) - Should return product by id.
 - Testing /product (Create Product) - Should return the added object.
 - Testing /product/:id (Delete Product) - Should return 1, which means deleted.
 - Testing /product/:id (Edit Product) - Should return the new edited object.
- Backend testing of Admin REST-API (Payment)
 - Testing /payment (Get Payments) - Should return all payments.
 - Testing /payment/:id (Get Payment) - Should return payment by id.
 - Testing /payment (Create Payment) - Should return the added object.
 - Testing /payment/:id (Delete Payment) - Should return 1, which means deleted.
 - Testing /payment/:id (Edit Payment) - Should return the new edited object.

The right pane shows the terminal output of the test execution, including the command used to run the tests and the results of each test, such as 'GET /adminApi/order 200 8.665 ms - -' and 'DELETE /adminApi/order/5477542c0132d346688c296a 200 1.618 ms - -'.

The screenshot shows a code editor with the file 'REST_Admin_API.js' open. The code defines a REST API for orders, products, and payments. It includes routes for GET, POST, PUT, and DELETE operations. The code is as follows:

```

var router = express.Router();

var OrderDataLayerModel = require('../model/OrderDataLayer');
var ProductDataLayerModel = require('../model/ProductDataLayer');
var PaymentDataLayerModel = require('../model/PaymentDataLayer');

//see all users. see all user details. create new admin user etc. customer <<
//***additional : change see user details (also admin should see all users details + change see delete

router.get('/order', function (req, res) {...});
router.get('/order/:id', function (req, res) {...});
router.post('/order', function (req, res) {...});
router.delete('/order/:id', function (req, res) {...});
router.put('/order/:id', function (req, res) {...});

router.get('/product', function (req, res) {...});
router.get('/product/:id', function (req, res) {...});
router.post('/product', function (req, res) {...});
router.delete('/product/:id', function (req, res) {...});
router.put('/product/:id', function (req, res) {...});

router.get('/payment', function (req, res) {...});
router.get('/payment/:id', function (req, res) {...});
router.post('/payment', function (req, res) {...});
router.delete('/payment/:id', function (req, res) {...});
router.put('/payment/:id', function (req, res) {...});

module.exports = router;

```

JAVA + J-Unit

```
//Singleton
private static FacadeLogic instance = null;

protected FacadeLogic() {...3 lines }

public static FacadeLogic getInstance() {...6 lines }

private void initializeTransactions() {...3 lines }

@Override
public String getUsersAsJSON() {...18 lines }

@Override
public String authenticateUserViaJSON(String username, String password) {...19 lines }

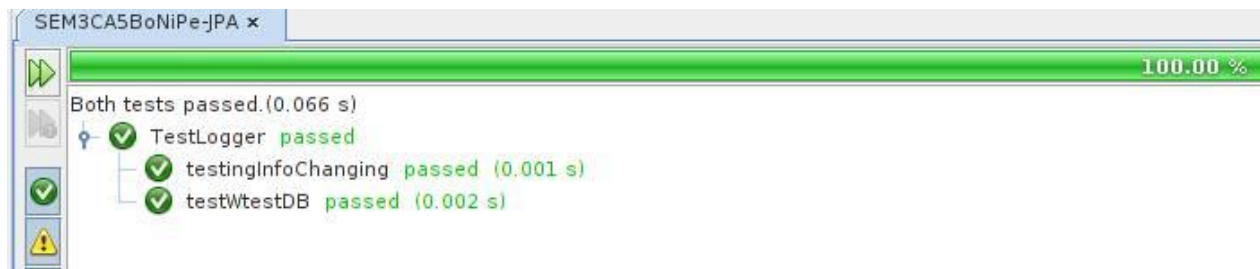
@Override
public String addUserViaJSON(String json) {...20 lines }

@Override
public String editUserViaJson(String json) {...45 lines }

@Override
public Integer changePasswordViaJson(String json) {...22 lines }

@Override
public Integer deleteUserViaJSON(String username) {...16 lines }

@Override
public String hashMe(String hashed) throws NoSuchAlgorithmException {...11 lines }
```



Who did what?

What?	Who?
Java server + JPA	Peter / Boyko
Node-Express + MongoDB	Boyko
AngularJS	Boyko
server connection	Boyko / Peter
majority of tests (karma / JUnit)	Boyko
sprint documentations	Peter
final documentation	Peter / Boyko
product Backlog	Nick
sprint backlogs	Peter
architecture diagram	Peter
sequence diagram	Boyko
deploying (heroku/ azur)	Nick / Peter
design	Boyko
UI mock-ups	Nick / Peter

NOTE: Nik and Peter created 2 functions of Node and AngularJS, particularly Payment and Your Orders. Rest of the work was done by Boyko.

How far we have got

If we look at the backlog, we discovered that we had to be realistic about what we were able to implement and what we were not able to.

We have all the requirements fulfilled except for the front-end tests, which gave us problems. If we look at the product backlog there are only a few of the stories that we never implemented. We never divided the orders into different tables depending on their status, but the main functionality that we thought was most important is implemented and works.

The same thing goes for the company profiles that we wanted to connect the users with. Instead of spending time on trying to make the complete application, we developed the functions for an admin and a user, which was the core logic of the whole system.

In the end, we are pleased with our application and we think this could be worth developing further in the future.