

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Э. БАУМАНА

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»



Тестирование и отладка ПО

ЛАБОРАТОРНЫЕ РАБОТЫ

Студент: Петухов И.С.

Группа: ИУ7-71

Преподаватель: Rogozin O.V.

Москва, 2016

Содержание

1	Аналитический раздел	3
1.1	Описание тестируемой системы	3
1.2	Рассматриваемые виды тестирования	4
1.3	Интеграционное тестирование	6
1.3.1	Тестирование модуля vuser.js	6
1.3.2	Тестирование модуля referee.js	6
1.4	Регрессионное тестирование	9
1.5	Функциональное тестирование	15
1.5.1	Use Cases	15
1.5.2	Use Case создание пользователем команды	15
1.5.2.1	Основной поток событий	15
1.5.3	Автоматизация функционального тестирования	16
1.5.3.1	Тестирование создания команды	16

1 Аналитический раздел

Цель данной работы - протестировать web-приложение платформа для проведения футбольных чемпионатов.

1.1 Описание тестируемой системы

Web-приложение платформа для проведения футбольных чемпионатов состоит из следующих частей:

- а) сервер
- б) база данных
- в) клиентская часть

Сервер написан на языке JavaScript с использованием программной платформы NodeJS. Основное предназначение - обработка http запросов.

База данных - нереляционная СУБД mongoDB. Обращение к базе данных осуществляется с помощью библиотеки Mongoose. Mongoose - обертка, позволяющая создавать удобные и функциональные схемы документов.

Клиентская часть написана на языке JavaScript с использованием библиотеки ReactJS.

Модели, описывающие объекты веб-приложения/базы данных:

- а) federation
- б) match
- в) stage
- г) team
- д) tournament
- е) user
- ж) vuser

app.js - главный файл тестируемого веб приложения.

В файле **app.js** подключаются обработчики http запросов, описанных в файлах папки **routes**:

- а) federation.js - обработка запросов, связанных с объектом федерация
- б) match.js - обработка запросов, связанных с объектом матч
- в) stage.js - обработка запросов, связанных с объектом этап
- г) team.js - обработка запросов, связанных с объектом команда
- д) tournament.js - обработка запросов, связанных с объектом турнир
- е) vuser.js - обработка запросов, связанных с объектом виртуальный пользователь

ж) referee.js - обработка запросов, связанных с взаимодействием с мобильным приложением

з) main.js - обработка остальных запросов

В мобильном приложении авторизуется пользователь и получает список матчей, на которые он назначен судить. Если в приложении начать судить матч, то на сервер можно отправлять следующие события:

- а) MATCH_STARTED - начало матча
- б) MATCH_FINISHED - конец матча
- в) TIME_STARTED - начало периода
- г) TIME_FINISHED - конец периода
- д) GOAL - гол
- е) OWN_GOAL - автогол
- ж) YELLOW_CARD - желтая карточка
- з) RED_CARD - красная карточка
- и) MIN - просто прислана текущая минута матча
- к) ASSIST - прислано событие с игроком - который отдал голевой пас.

Каждое сообщение, посылаемое на сервер, должно содержать следующие поля:

- а) idAction - число порядковый номер события в матче
- б) idMatch - id матча
- в) minute - число минута матча, на которой произошло событие
- г) idEvent - тип события, описанные выше

1.2 Рассматриваемые виды тестирования

В данной работе будут рассмотрены следующие виды тестирования:

- а) интеграционное
- б) регрессионное
- в) функциональное

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Регрессионное тестирование - это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

1.3 Интеграционное тестирование

В данной части работы будет осуществлено тестирование модулей сервера, обрабатывающие http запросы, взаимодействующие с базой данных, в которых импортированы модели.

1.3.1 Тестирование модуля `vuser.js`

vuser.js - набор методов для обработки запросов, связанных с объектом виртуальный пользователь.

В данном модуле описаны методы-обработчики:

а) **POST /get** - получение информации о виртуальном пользователе по заданном id

Код модуля **vuser.js** небольшой и представлен в листинге 1.1

Листинг 1.1 — Код метода обработчика урла /get

```
1 var express = require('express');
2 var router = express.Router();
3 var Vuser = require('../models/vuser');
4
5 router.post('/get', function(req, res, next) {
6   if (!req.body.id) {
7     return res.status(400).json(null);
8   }
9
10  Vuser.findById(req.body.id, (err, vuser) => {
11    if(err || !vuser) {
12      return res.status(404).json(null);
13    }
14
15    return res.status(200).json(vuser);
16  });
17 });
18
19 module.exports = router;
```

1.3.2 Тестирование модуля `referee.js`

referee.js - набор методов для обработки запросов, связанных с взаимодействием с мобильным приложением.

В данном модуле описаны методы-обработчики:

а) **POST /get-my-matches** возвращает список матчей, на которые назначен судить заданный пользователь

Таблица 1.1 — Тестирование модуля vuser.js

1. запрос юзера по существующему id	
Запрос	{id: 12345}
Ожидаемый ответ	200 OK {...}
2. запрос юзера по несуществующему id	
Запрос	{id: 67890}
Ожидаемый ответ	404 NOT FOUND: {null}
3. запрос юзера с отсутствующим полем id в запросе	
Запрос	{}
Ожидаемый ответ	400 BAD REQUEST: {null}

б) **POST** /**set-info** получает событие, произошедшее в заданном матче с описанием события.

Таблица 1.2 — Тестирование модуля referee.js

1. запрос списка матчей по существующему id пользователя	
Запрос	{id: 12345}
Ожидаемый ответ	200 OK {...}
2. запрос списка матчей по несуществующему id	
Запрос	{id: 67890}
Ожидаемый ответ	404 NOT FOUND: {null}
3. запрос списка матчей с отсутствующим полем id в запросе	
Запрос	{}
Ожидаемый ответ	400 BAD REQUEST: {null}
4. отправка события MATCH_STARTED в существующем матче	
Запрос	{idMatch: match.id, idEvent: Match.EVENT.MATCH_STARTED.name, idAction: 0, minute: 0}
Ожидаемый ответ	200 OK: {null}
Ожидаемые действие	Заданный матч изменяет статус на RUNNING {null}
5. отправка события MATCH_FINISHED в существующем матче	
Запрос	{idMatch: match.id, idEvent: Match.EVENT.MATCH_FINISHED.name, idAction: 0, minute: 90}
Ожидаемый ответ	200 OK: {null}
Ожидаемые действие	Заданный матч изменяет статус на FINISHED {null}
6. отправка любого события по несуществующему id	
Запрос	{idMatch: — idEvent: — idAction: 0, minute: 0}
Ожидаемый ответ	404 NOT FOUND: {null}
7. отправка любого события с отсутствующим полем id в запросе	
Запрос	{}
Ожидаемый ответ	400 BAD REQUEST: {null}

1.4 Регрессионное тестирование

Для того чтобы знать, какие тесты перезапускать после того или иного изменения в программе, нужно определить, от каких конкретно частей программы (модулей, методов, и т.п.) зависит результат каждого теста. Для этого часто используется управляющий граф, отображающий поток управления программы, по которому легко отследить зависимости одних блоков/модулей/методов от других.

Web-приложения строятся по принципу микросервисной архитектуры. Поэтому составные модули веб-приложений должны быть независимыми. Более того, методы-обработчики http запросов в этих модулях - также часто не зависят друг от друга. Поэтому на каждый метод-обработчик пишется несколько тестов, которые проверяют качество данного метода. А учитывая специфику веб-приложений, эти тесты никак не будут связаны с другими методами. Из этого я могу сделать вывод, что строить матрицу тестов-методов не имеет смысла (т.к. каждый тест проверяет определенный метод и ни как не связан с другими методами).

Мои одноклассники успешно сдавали регрессионное тестирование как раз на уровне методов. Т.е. строили матрицу зависимости метода от теста. Однако я уже показал, что в моём приложении данный подход не имеет смысла.

На лекции регрессионное тестирование объяснялось на уровне кода. Т.е. я это понимаю так: мы можем взять один метод, взять тесты, которые написаны для этого теста, построить граф потока управления данного метода, построить зависимости узлов графа(строчек кода) от тестов этого метода и из этого сделать вывод, какие тесты необходимо запускать, при изменении определенной строчки кода, а какие нет.

Рассмотрим данный подход на примере метода `/set-info` модуля `referee.js`.
`/set-info` получает событие, произошедшее в заданном матче с описанием события.

Для данного метода из раздела 1.3.2 описаны тесты:

- а) отправка события **MATCH_STARTED** в существующем матче
- б) отправка события **MATCH_FINISHED** в существующем матче
- в) отправка любого события по несуществующему id
- г) отправка любого события с отсутствующим полем id в запросе

Листинг 1.2 — Код метода обработчика урла `/set-info`

```
1 router.post('/set-info', function(req, res, next) {  
2   let idMatch = req.body.idMatch;  
3   let idEvent = req.body.idEvent;  
4   let idAction = req.body.idAction;  
5   let minute = req.body.minute;  
6  
7   if (!idMatch || !idEvent || isNaN(idAction) || isNaN(minute)) {
```

```

8         return res.status(400).json(null);
9     }
10
11     let now = new Date();
12
13     let event = {
14         idEvent: idEvent,
15         idAction: idAction,
16         minute: minute,
17         realTime: now,
18     };
19
20     Match.findById(idMatch, function (err, match) {
21         if (err || !match) {
22             return res.status(404).json(null);
23         }
24
25         switch (idEvent) {
26             case Match.EVENT.MATCH_STARTED.name:
27                 match.status = Match.STATUS.RUNNING.name;
28                 break;
29             case Match.EVENT.MATCH_FINISHED.name:
30                 match.status = Match.STATUS.FINISHED.name;
31                 break;
32             case Match.EVENT.GOAL.name:
33             case Match.EVENT.OWN_GOAL.name:
34             case Match.EVENT.YELLOW_CARD.name:
35             case Match.EVENT.RED_CARD.name:
36                 event.idTeam = req.body.idTeam;
37                 event.idPlayer = req.body.idPlayer;
38                 event.teamName = req.body.teamName;
39                 event.playerName = req.body.playerName;
40                 break;
41
42         }
43
44         match.events.push(event);
45
46         clients.forEach((ws) => {
47             ws.send(JSON.stringify(event));
48         });
49
50         match.save(function (err) {
51             if (err) {
52                 return res.status(500).json(null);
53             }
54

```

```
55         return res.status(200).json(null);
56     });
57 });
58 });
```

По листингу 1.2 построим граф потока управления. Каждый узел графа (блок) отмечен заглавной латинской буквой. Числа в скобках указывают на строки кода в листинге 1.2.

Для проверки правильности составленной таблицы 1.3 изменим строчку №55, которая соответствует блоку К.

По рисункам 1.2 и 1.3 видно, что после изменения блока К стали падать тесты а, б, что соответствует построенной таблице 1.3

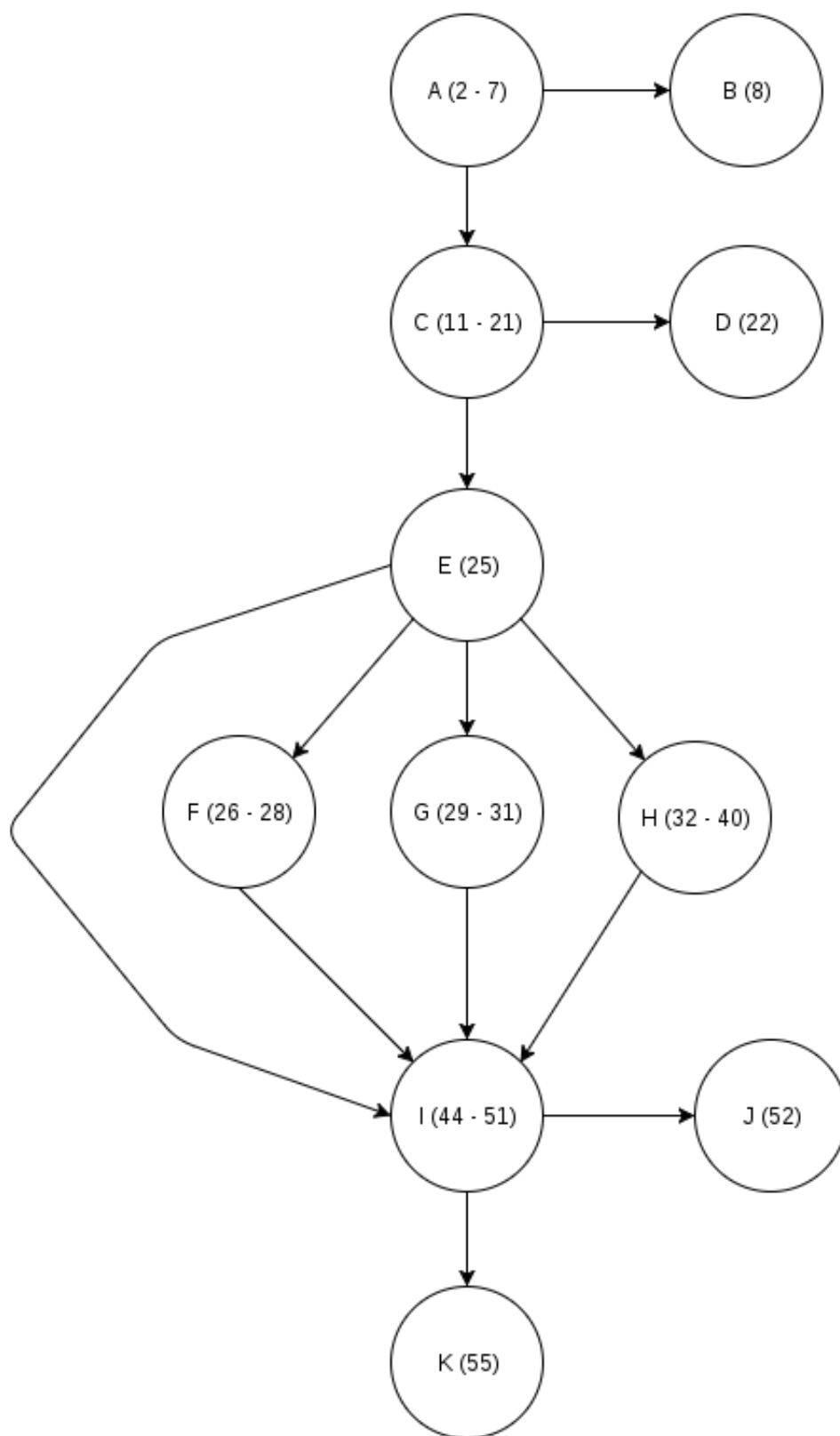


Рисунок 1.1 — Граф потока управления метода обработчика /set-info

Таблица 1.3 — Матрица зависимостей блоков графа от тестов

	тесты			
блок	а	б	в	г
A	-	-	-	-
B	-	-	-	+
C	-	-	-	-
D	-	-	+	-
E	-	-	-	-
F	+	-	-	-
G	-	+	-	-
H	-	-	-	-
I	-	-	-	-
J	-	-	-	-
K	+	+	-	-

```

xpress запущен на http://localhost:8080; нажмите Ctrl+C для завершения.
API Referee
  /get-my-matches
    ✓ it should get matches by the given user id (310ms)
    ✓ it should get empty matches by the given not exist user id
    ✓ it should get bad request
  /set-info
    ✓ it should match.status == Match.STATUS.RUNNING.name by set event MATCH_STARTED
    ✓ it should match.status == Match.STATUS.FINISHED.name by set event MATCH_FINISH
    ✓ it should get NOT FOUND by the given not exist match id
    ✓ it should get bad request

API Vuser
  /get
    ✓ it should get a vuser by the given id
    ✓ it should get 404 (not found) by not exist id
    ✓ it should get 400 (bad query)

10 passing (2s)

```

Рисунок 1.2 — Прохождение тестов до изменения блока К

```
API Referee
  /get-my-matches
    ✓ it should get matches by the given user id (59ms)
    ✓ it should get empty matches by the given not exist user id
    ✓ it should get bad request
  /set-info
    1) it should match.status == Match.STATUS.RUNNING.name by set event MATCH_START
    2) it should match.status == Match.STATUS.FINISHED.name by set event MATCH_FINISH
    ✓ it should get NOT FOUND by the given not exist match id
    ✓ it should get bad request

API Vuser
  /get
    ✓ it should get a vuser by the given id
    ✓ it should get 404 (not found) by not exist id
    ✓ it should get 400 (bad query)

8 passing (323ms)
2 failing
```

Рисунок 1.3 — Прохождение тестов после изменения блока К

1.5 Функциональное тестирование

Функциональные тесты основываются на функциях, выполняемых системой. Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

1.5.1 Use Cases

Use Case (вариант использования, ВИ, Прецедент, юскейс) — это сценарная техника описания взаимодействия. С помощью Use Case может быть описано и пользовательское требование, и требование к взаимодействию систем, и описание взаимодействия людей и компаний в реальной жизни. В общем случае, с помощью Use Case может описываться взаимодействие двух или большего количества участников, имеющее конкретную цель. В разработке ПО эту технику часто применяют для проектирования и описания взаимодействия пользователя и системы, поэтому название Use Case часто воспринимает как синоним требования человека-пользователя к решению определенной задачи в системе.

Примеры Use Case для тестируемого приложения:

- а) создание пользователем команды
- б) создание пользователем федерации
- в) создание пользователем турнира по футболу
- г) создание пользователем матчей в турнире его федерации
- д) подача пользователем заявки на участие его команды в турнире

1.5.2 Use Case создание пользователем команды

Рассмотрим подробнее use case создание пользователем футбольной команды.

Система - сайт web-приложения

Основное действующее лицо - пользователь-капитан команды

Цель - создать команду

Триггер - пользователь решает создать команду и заходит на главную страницу сайта

Результат - информация о команде сохранена в базе данных, пользователь получил ответ от сервера.

1.5.2.1 Основной поток событий

- а) пользователь заходит на главную страницу сайта
- б) пользователь авторизуется на сайте через сайт-вконтакте
- в) пользователь нажимает на кнопку «создать команду»

- г) пользователь заполняет поля «название команды» и «город»
- д) пользователь нажимает отправить
- е) система получает запрос
- ж) система проверяет запрос
- з) при верном запросе система сохраняет данные в базе
- и) система посылает ответ пользователю

1.5.3 Автоматизация функционального тестирования

В связи с большим количеством usecase и большим количеством монотонных действий в каждом usecase достаточно распространенной является автоматизация функционального тестирования.

Автоматизация функционального тестирования выполнена с помощью Selenium WebDriver.

Selenium WebDriver - автоматизирует действия с браузером. Это именно то, что необходимо для тестирования сайта web-приложения. Частым является использование паттерна Page Object при написании тестов с использованием Selenium.

1.5.3.1 Тестирование создания команды

Код теста написан с учетом рассмотренного потока событий данного usecase в листинге 1.3

Листинг 1.3 — Тестирование use case создание команды

```
1  it('create team', function(done) {
2      client.manage().timeouts().implicitlyWait(5000);
3
4      let teamName = 'NAME TEAM1';
5      let teamCity = 'CITY TEAM1';
6
7      vkPage.open()
8          .then(() => {
9          vkPage.auth(secret.emailVk, secret.passVk);
10         mainPage.open(URL);
11         mainPage.auth();
12         mainPage.createTeam(teamName, teamCity);
13         client.quit();
14     })
15     .then(() => {
16         return Team.find({name: teamName}, function (err, team) {
17             assert.isNull(err);
18             assert.isNotNull(team);
19             done();
20         })
21     })
22 })
```


21 }))

Листинг 1.4 — Объект mainPage

```
1 function mainPage(driver) {
2     let self = this;
3
4     self.driver = driver;
5
6     self.open = function(url) {
7         return self.driver.get(url);
8     };
9
10    self.auth = function() {
11        self.driver.findElement(wd.By.id('vk-auth-btn')).click();
12
13        self.driver.findElements(wd.By.className("button_indent"))
14            .then(arr => {
15                if (arr.length > 0) {
16                    arr[0].click();
17                }
18            });
19    };
20
21    self.createTeam = function(teamName, teamCity) {
22        // var createTeamBtn =
23        self.driver.wait(wd.until.elementLocated(wd.By.id('create-team-btn')),
24            10000);
25        // createTeamBtn.click();
26        self.driver.findElement(wd.By.id('create-team-btn')).click();
27
28        self.driver.findElement(wd.By.id('team-name')).sendKeys(teamName);
29        self.driver.findElement(wd.By.id('team-city')).sendKeys(teamCity);
30
31        self.driver.findElement(wd.By.id('send-team-btn')).click();
32    }
33 }
```

Листинг 1.5 — Результаты тестирования

```
1 ilyaps@debian-ilyaps:~/projects/MyFootballFederation$ npm test
2
3 > mff@1.0.0 test /home/ilyaps/projects/MyFootballFederation
4 > mocha test/gui/ --no-timeoutsзапущенная
5
6
7 база: mongodb://localhost:27017/football_test
```

```
8
9
10 Express запущен на http://localhost:8080; нажмите Ctrl+C для завершения.
11   GUI Create team
12     OK normal create team (21944ms)
13
14
15   1 passing (22s)
```