СОФИЙСКИ УНИВЕРСИТЕТ “Св. КЛИМЕНТ ОХРИДСКИ”

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

Курсова работа по Мултимедийни технологии

Тема “Разработка на приложение за процеса на manual QA - Fridment”

Изготвил:

Цветелина Ангелова Борисова 81067

Преподавател:

Траян Илиев

1. Функционалните изисквания

1. Кратко описание на потребителските случаи (Use cases)		
Име на потребителския случай	**Кратко описание (Brief Descriptions)**	**Кратко описание на актьорите (Actor Brief Descriptions)**
1. Регистрация	Всеки потребител може да се регистрира в системата като той ще има само read роля	Нерегистриран потребител
2. Определяне на роля на потребител	Само master администите ще могат да променят ролята на регистрираните потребители, като от read роля да има ‘стандартна’ роля	Администратор - master админ
3. Създаване на milestone, затваряне	При избиране на определени критерии за направата на даден milestone - дата на тикети, label на тикети ще бъдат стартиран скрипт, който да изтегли дадените тикети от Github и да бъде създаден в системата milestone с дадените тикети. По всяко време един milestone може да бъде маркиран като - завършен	Администратори и стандартен потребител
4. Разпределяне на тикетите за тестване между тестерите	Всеки тикет може да има определен брой тестери, които се определят от човека създал milestone.	Администратори и автора на milestone
5. Промяна на тикетите	Изтриване и добавя на тикети	Администратори и автора на milestone

в milestone		
6. Добавяне на коментари в дадения тикет в системата	Към всеки тикет могат да бъдат добавени коментари	Администратори и стандартен потребител
7. Оценка дали даден тикет работи	Всеки тикет ще може да бъде отбелязван като-работещи или не	Администратор, автор на milestone, потребители, които са сложени като тестерти на тикета

2. Нефункционалните изисквания

С цел изследване, оценка и изготвяне на предложения за подобрение на не функционалните атрибути на приложението Fridment ще разгледаме следните характеристики:

- Производителност
- Достъпност
- Потребителски интерфейс - “User Experience”
- Сигурност
- Съвместимост
- Локализация

Метрики

За изследване на производителността, ще проучим времето за отговор от сървъра, времето за показване на съдържането получено от сървъра в браузъра. Възможност за „кеширане“ на статично съдържание.

Тъй като потребителската част е написана на React, това позволява бързо зареждане на елементите на страница, като ако има промяна в един от елементите, то само той се зарежда отново.

Използването на bootstrap значи, че потребителския интерфейс трябва да скалира добре на различни платформи, но тъй като в приложението не е разгърната пълната сила на bootstrap, това е нещо, което може да се промени.

Липсва локализация.

Производителност

Средното време за отговор от сървъра за страница е около 20ms. Най-тежките заявки(взимане на всички milestones или взимане на всички issues е около 40ms) Времето за обработване на елементите на страницата е около 800ms, като времето за цялостно зареждане е около 1 секунда.

Това показва че сървъра има много добра производителност при изготвяне на съдържанието, но факта че цялостното показване на съдържанието изисква няколко пъти повече време. Тъй като няма изображения проблемът идва от много заредени елементи.

Възможно подобрение е страниците с сериозно количество елементи, да бъдат разделени на няколко под-страници или опростени, чрез показване на по-съкратена информация първоначално и добавяне на детайлите едва след зареждане на конкретен ресурс.

Достъпност

В момента потребителския интерфейс има много опростен вид.

За подобряване на достъпността, трябва да се подобрят описанията, да се добавят повече семантични елементи и да се подобри контраста – за целта не е необходимо цялостна подмяна на цветовете, ами подбиране на нюанси предоставящи по-лесна работа за хора с намалена зрителна способност.

Потребителски интерфейс

Потребителския интерфейс има нужда от подобрения - добавяне на breadcrumbs, профил на аутентикирания потребил.

Сигурност

В момента има аутентикация на ниво потребителски интерфейс, но не и сървър частта. Има нужда от добавяне на аутентикация и на back-end частта. Също така в момента всеки може да създава milestones, issues, comments. Трябва да се добави и ауторизация.

Локализация

В момента няма локализация.

3. Използвани технологии и библиотеки

Потребителска част

Използван React - технология, с която лесно може да се дефинират просто компоненти, които след това да се покажат в потребителския интерфейс. React ефективно презарежда само компонентите, които са промени, а не цялото съдържание на страницата.

Компонентите са енкапсулирани и всеки един може да се включва в друг, за да се направи по-сложен потребителки интерфейс. Тъй като компонентите са написани на Javascript, вместо темплейти, лесно може да се предават сложни данни към приложението и състоянието да не се обработва от DOM.

Най-важните използвани библиотеки за front-end частта са: *react-router-dom*, *reactstrap*, *recharts* и *auth0-js*. Първата предоставя декларативно рутиране за React. Втората библиотека предоставя React компоненти, с които да се направи bootstrap интеграция. *recharts* библиотека позволява направата на графика, тя е базирана на d3, като предоставя лесен интерфейс за работа. Последната библиотека предоставя аутентикация чрез auth0, с нея даден потребител може да се аутентикира или чрез Google акаунта си, или с имейл и парола ако има акаунт(ако няма може да се регистрира).

Сървър

За сървъра е използван Express - минималистична и гъвкъв Node.js уеб фреймъурк, който осигурява добър набор от инструменти за уеб и мобилни приложения.

Най-важните библиотеки за сървърната част са: mongojs. body-parser, octonode. Първата библиотека е интеграция на Node.js с MongoDB и позволява удобен интерфейс за работа с избраната база. Втората библиотека улеснява работата с параметрите на заявките. Поседната библиотека е използвана, за да улесни заявките към Github за изтеглянето на тикетите за даден milestone.

4. REST API

Сървъра има 4 логически обекта, които са разделени в отделни файлове: milestones, issues, components, users. По-долу ще бъдат описани всеки един от тях:

milestones:

Всяка от заявките има префикс - milestones:

```
GET  "/"  // взима всички milestones от базата и ги връща
GET "/:id" // взима milestone с подаденото в параметрите id и го връща
GET "/:id/edit" // взима milestone с подаденото в параметрите id и го връща
POST "/" // създава нов milestone от подадените параметри в тялото на заявката
PATCH "/:id" // променя параметрите на milestone с подаденото в параметрите id, като връща променения обект
POST "/finish/:id" // променя статуса на milestone с подаденото в параметрите id на завършен
POST "/open/:id" // променя статуса на milestone с подаденото в параметрите id на отворен
POST "/generate_issues/:id" // създава тикети за milestone с id - подаденото в параметрите
```

Всяка от заявките има префикс - issues:

```
GET "/:milestone_id" // взима всички issues за даденото milestone_id
GET "/show/:id" // взима определен issue спрямо id-то подадено в параметрите
GET "/get_state/:id" // взима процентът на одобрения, които дадения тикет (намерено от id-то в параметрите) има спрямо коментарите към тикета
GET "/burn_down_chart/:milestone_id" // връща масив, в който се съдържат данни за колко процента тикети на дадения milestone са тествани и колко не са.
GET "/get_testers/:id" // взима всички потребители, които трябва да тестват дадения тикет(той се взима спрямо параметъра id, който е подаден)
POST "/add_testers/:milestone_id/:id" // за даден тикет се добавят потребители, които трябва да го изтестват
```

Всяка от заявките има префикс - comments:

```
GET "/:issue_id" // взима всички коментари за даден тикет (който се намира по issue_id) и ги връща
GET "/:id/edit" // взима коментар по id и го връща
POST "/:milestone_id/:issue_id"  създава коментар за даден тикет
PATCH "/:id" // променя атрибутите на даден коментар(само състояние и описание)
DELETE "/:id" // изтрива даден коментар
```

Всяка от заявките има префикс - issues:

```
GET "/" // взима всички потребители
POST "/" // създава потребител по параметрите в тялото на
```

```
з а я в к а т а
```

5. Описание на основните модули

Потребителска част

Един от основните нетривиални компоненти на потребителската част е Issues.js.

```
import React, { Component } from 'react';
import { Table } from 'reactstrap';
import Milestone from './Milestone';
import { Link } from 'react-router-dom'
import CommentsView from './CommentsView'

import AddTesters from './AddTesters'
import AddComments from './AddComments'
import TestersView from './TestersView'
import ViewIssue from './ViewIssue'

import { PieChart, Pie, Legend, Cell, Tooltip, ResponsiveContainer, Sector,
  Label, LabelList } from 'recharts';
import { scaleOrdinal, schemeCategory10 } from 'd3-scale';

class Issues extends Component {
  constructor(props) {
      super(props);
      this.state = {
        issues: [],
        milestone: [],
        data01: [
  {
        "name": "tested",
        "value": 22.22222222222222
    },
    {
        "name": "not tested",
        "value": 77.77777777777777
    }
]
      }
      this.findState = this.findState.bind(this);
   };

  findState(issue) {
    fetch(`/issues/get_state/${issue.id}`, {
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      }
    }).then(function(response) {
      return response.json();
    }).then(function(response){
      document.getElementById(`state_of${issue.id}`).innerHTML =
`${JSON.stringify(response).substring(0,4)}%`;
    })
   }
```

```
  componentDidMount() {
    fetch(`/issues/${this.props.match.params.milestone_id}`,
    {
      headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json',
      }
    })
      .then(res => res.json())
      .then(issues => this.setState({ issues }));

    fetch(`/milestones/${this.props.match.params.milestone_id}`, {
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      }
    })
      .then(res => res.json())
      .then(milestone => this.setState({ milestone }));

    fetch(`/issues/burn_down_chart/${this.props.match.params.milestone_id}`, {
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      }
    })
      .then(res => res.json())
      .then(burn_down_chart => this.setState({ burn_down_chart }));

  }

  render(){
    const colors =
["#1f77b4","#ff7f0e","#2ca02c","#d62728","#9467bd","#8c564b","#e377c2","#7f7f7f","#bcbd22
","#17becf"];
    const id = this.props.match.params.mileston_id;

    return(
      <div>
      <h2>Issues for milestone {this.state.milestone.name}</h2>
      <Table>
          <thead>
            <tr>
              <th>ID</th>
              <th>Assignee name</th>
              <th>Issue url</th>
              <th>Name</th>
              <th>Description</th>
              <th>Milestone name</th>
              <th>Created at</th>
              <th>Testers IDs</th>
              <th>Comments</th>
              <th>State</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
          {this.state.issues.map(issue =>
            <tr>
              <td>{issue.id}</td>
              <td>{issue.assignee_name}</td>
              <td>{issue.issue_url}</td>
              <td>{issue.name}</td>
              <td>{issue.description}</td>
              <td><Link to={`/milestones/${this.props.match.params.milestone_id}`}
```

```
component={Milestone}>{this.props.match.params.milestone_id}</Link></td>
              <td>{issue.created_at}</td>
              {issue.testers.length > 0 ? <td><Link to={`/get_testers/${issue.id}`}
component={TestersView}>view testers({issue.testers.length})</Link></td> : <td>No
testers</td>}
              {issue.comments.length > 0 ? <td><Link to={`/comments/${issue.id}`}
component={CommentsView}>view comments({issue.comments.length})</Link></td> : <td>No
comments</td>}
              <td id={`state_of${issue.id}`}>{this.findState(issue)}</td>
              <td><Link to={`/add_testers/${issue.milestone_id}/${issue.id}`}
component={AddTesters}>add testers</Link></td>
              <td><Link to={`/add_comments/${issue.milestone_id}/${issue.id}`}
component={AddComments}>add comments</Link></td>
            </tr>
          )}
          </tbody>
        </Table>
        <PieChart width={800} height={400}>
            <Legend />
            <Pie
              data={this.state.data01}
              dataKey="value"
              cx={200}
              cy={200}
              startAngle={180}
              endAngle={0}
              outerRadius={80}
              label
            >
              {
                this.state.data01.map((entry, index) => (
                  <Cell key={`slice-${index}`} fill={colors[index % 10]}/>
                ))
              }
              <Label value="Tested vs Not tested issues" position="outside" />
              <LabelList position="outside" />
            </Pie>
            </PieChart>
      </div>
    )
  };
};

export default Issues;
```

Основната задача на Issues.js е да покаже всички тикети за даден milestone. Milestone се взима от заявка към в метода ComponentDidMount(). В този метод се правят извиквания и за взимане на тикетите, както и за взимане на данни за направата на burn down chart, която показва какъв процент от тикетите са изтествани и какъв процент не са изтествани. Друга заявка, която се прави за всеки един от тикетите е намирането на неговото състояние и показването му в таблицата. Прави се заявка към сървъра - get_state, която показва процентът на одобрение на даден тикет спрямо коментарите към него, т.е. одобрение - дали функционалността/бъгът са оправени.

Друга интересна част от Issues.js е графиката, която е направена с помощта на recharts и както беше отбелязано показва отношението на изтествани тикети към неизтествани.

Други интересни компоненти са AddComments или AddTesters, които показат как дадена форма се изпраща от потребителската част към сървъра.

```
import React from 'react';
import { Col, Button, Form, FormGroup, Label, Input, FormText, Jumbotron } from
'reactstrap';

export default class AddComments extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
        users: [],
        issue: []
      }
      this.sendForm = this.sendForm.bind(this);
   };

   sendForm(e) {

fetch(`/comments/${this.props.match.params.milestone_id}/${this.props.match.params.issue_
id}`, {
      method: 'POST',
      body: JSON.stringify({
        state: document.getElementById('state').value,
        description: document.getElementById('description').value
      }),
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      }
    }).then(window.location = `/issues/${this.state.issue.milestone_id}`)
   }

  componentDidMount() {
    fetch(`/issues/show/${this.props.match.params.issue_id}`, {
          headers: { 'Accept': 'application/json',
                     'Content-Type': 'application/json',
          }})
      .then(res => res.json())
      .then(issue => this.setState({ issue }));
  }
  render() {
    return (
      <Jumbotron>
      <h2>Add comment for issue {this.state.issue.issue_url}</h2>
      <Form>
        <FormGroup>
          <Label for="state">State</Label>
          <Input type="select" name="state" id="state">
            <option>true</option>
            <option>false</option>
          </Input>
        </FormGroup>
         <FormGroup>
          <Label for="description">Description</Label>
          <Input type="textarea" name="description" id="description" />
        </FormGroup>
        <FormGroup check row>
          <Col sm={{ size: 10, offset: 2 }}>
            <Button onClick={this.sendForm}>Submit</Button>
          </Col>
        </FormGroup>
```

```
      </Form>
      </Jumbotron>
    );
  }
}
```

Функцията sendForm взима въведните параметри с помощта на document.getElementById и изпраща подходяща заявка с правилни хедъри, като при получаване на отговор от сървъра има редирект към страницата с milestones.

Един интересен компонент е Header.js, където се намират дефинициите на всички пътища, които се използват в приложението.

```
import React, { Component } from 'react';
import {
  Collapse,
  Navbar,
  NavbarToggler,
  NavbarBrand,
  Nav,
  NavItem,
  NavLink,
  NavButton
} from 'reactstrap';

import Milestones from './Milestones';
import Issues from './Issues';
import Milestone from './Milestone';
import GenerateIssues from './GenerateIssues'
import FinishMilestone from './FinishMilestone'
import OpenMilestone from './OpenMilestone'
import NewMilestone from './NewMilestone'
import EditMilestone from './EditMilestone'
import Callback from './Callback'
import AuthService from './AuthService'
import {
  BrowserRouter as Router,
  Route,
  Link
} from 'react-router-dom'
import CommentsView from './CommentsView'
import AddTesters from './AddTesters'
import AddComments from './AddComments'
import TestersView from './TestersView'
import ViewIssue from './ViewIssue'
import { login, logout, isLoggedIn } from './AuthService';

  const auth = new AuthService();

const handleAuthentication = (nextState, replace) => {
  if (/access_token|id_token|error/.test(nextState.location.hash)) {
    auth.handleAuthentication();
  }
}
class Header extends Component {
  constructor(props) {
    super(props);
```

```
    this.toggle = this.toggle.bind(this);
    this.state = {
      isOpen: false
    };
  }
  toggle() {
    this.setState({
      isOpen: !this.state.isOpen
    });
  }
  render() {

    return (
      <Router>
        <div>
          <Navbar color="faded" light toggleable>
            <NavbarToggler right onClick={this.toggle} />
            <NavbarBrand><Link to="/milestones">Milestones</Link></NavbarBrand>
            {
             ( auth.isAuthenticated() ) ? <NavbarBrand><Link to="/new_milestone">New
milestone</Link></NavbarBrand> :  ''
            }

            <Collapse isOpen={this.state.isOpen} navbar>
              <Nav className="ml-auto" navbar>
              <NavItem>
                {
                (auth.isAuthenticated()) ? ( <button className="btn btn-danger log"
onClick={() => auth.logout()}>Log out </button> ) : ( <button className="btn btn-info
log" onClick={() => auth.login()}>Log In</button> )
                }
              </NavItem>
              </Nav>
            </Collapse>
          </Navbar>
          <Route exact path="/milestones" component={Milestones}/>
          <Route exact path="/new_milestone" component={NewMilestone}/>
          <Route exact path="/milestones/:milestone_id/edit" component={EditMilestone}/>
          <Route exact path="/issues/:milestone_id" component={Issues}/>
          <Route exact path="/milestones/generate_issues/:milestone_id"
component={GenerateIssues}/>
          <Route exact path="/milestones/:milestone_id" component={Milestone}/>
          <Route exact path="/finish_milestone/:milestone_id"
component={FinishMilestone}/>
          <Route exact path="/open_milestone/:milestone_id" component={OpenMilestone}/>
          <Route exact path="/milestones/:id/open" component={OpenMilestone}/>
          <Route exact path="/comments/:issue_id" component={CommentsView}/>
          <Route exact path="/add_testers/:milestone_id/:issue_id"
component={AddTesters}/>
          <Route exact path="/add_comments/:milestone_id/:issue_id"
component={AddComments}/>
          <Route exact path="/get_testers/:issue_id" component={TestersView}/>
          <Route exact path="/issues/show/:id" component={ViewIssue}/>
          <Route path="/callback" component={Callback} />
          <Route path="/callback" render={(props) => {
            handleAuthentication(props);
            return <Callback {...props} />
          }}/>
        </div>
      </Router>
    );
  }
}

export default Header;
```

В бъдеще дефинициите на пътищата ще бъдат преместени в отделен компонент.

Сървърна част

Сървъра е разделен на няколко файла, в които се дефинират API endpoints, като те са логически разделени. Някои от по-сложните API endpoints са например issues/get_state, issues/burn_down_chart тъй като в тях има по-сложни MongoDB заявки.

За база беше избрана MongoDB, тъй като направата на графиките трябва да е по-лесна, т.е. взимането на данни и агрегирането им. По-надолу са описани основните полета в таблиците.

milestones

id - използва се. което има стойности числа от 1 до n, подобно на SQL базите. Причината е за по-лесно взимане на записи от други таблици, когато се прави по-сложна заявка
name - име на milestone
description - описание на milestone
author_id - id на потребителя създал milestone
state - opened или closed
created_at - дата на създаване

issues

id - използва се. което има стойности числа от 1 до n, подобно на SQL базите. Причината е за по-лесно взимане на записи от други таблици, когато се прави по-сложна заявка
assignee_name - име на потребител, който е направил съответния тикет
description - описание на тикета
name - име на тикета
milestone_id - id на съответния milestone, към който принадлежи тикета
created_at - дата на създаване

users

id - използва се. което има стойности числа от 1 до n, подобно на SQL базите. Причината е за по-лесно взимане на записи от други таблици, когато се прави по-сложна заявка
name - име на потребителя

email - имейл на потребителя

comments

6. Инсталиране и конфигуриране на системата

Конфигурация и стартиране на сървър частта

Необходимо е да се инсталират NodeJS, MongoDB и да бъде пуснат ***mongod.*** Трябва да се създаде база: fridment-new.

Кода на сървъра може да бъде свален тук: https://github.com/tborisova/fridment-express. След като се клонира, в главната директория на проекта трябва да се изпълнят следните команди:

```
npm install
npm start
```

Първата команда ще инсталира необходимите NodeJS пакети, а втората ще пусне сървъра на порт 3004.

Конфигурация и стартиране на потребителската част

Необходимо е да се инсталира NodeJS.

Кода на сървъра може да бъде свален тук: https://github.com/tborisova/fridment-react. След като се клонира, в главната директория на проекта трябва да се изпълнят следните команди:

```
npm install
npm start
```

Първата команда ще инсталира необходимите NodeJS пакети, а втората ще пусне сървъра на първия свободен порт.

7. Заключение

Не са изпълнени всички функционални изисквания към проекта, но за нещата, които са изпълнени успях да науча малко повече за React, Express и Mongo.

8. Използвани материали

https://www.tutorialspoint.com/reactjs/
https://www.tutorialspoint.com/expressjs/
https://reactstrap.github.io/components/
https://reacttraining.com/react-router/web/
https://github.com/mafintosh/mongojs
https://www.tutorialspoint.com/mongodb
https://hackernoon.com/how-and-why-to-use-d3-with-react-d239eb1ea274
https://medium.com/@Elijah_Meeks/interactive-applications-with-react-d3-f76f7b3ebc71
http://recharts.org/
https://docs.mongodb.com/manual/reference/