

СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА



КУРСОВА РАБОТА

ТЕМА:

**„Дигитално гласуване на
корпоративни решения.“**

Изготвили:

Траян Троев – ФН: 81438

Владислав Катранкиев - ФН: 81418

Специалност:

„Компютърни науки“

12.02.2019г София

Функционални изисквания

1. REST API трябва да предоставя следните възможности на следните групи потребители:

1.1. *Админ (Admin) трябва да може да:*

- 1.1.1. Да се логва в акаунта си
- 1.1.2. Да излиза от акаунта си
- 1.1.3. Редактира акунти, както своя, така и на останалите user-и.
- 1.1.4. Активира (удобрява) /деактивира профили, като по този начин няма достъп до профила, до неговата обратна активация
- 1.1.5. Преглежда всички акаунти, както активни, така и неактивни
- 1.1.1. Добавя оферти
- 1.1.2. Преглежда всички оферти
- 1.1.6. Преглежда дадена оферта
- 1.1.7. Редактира своя оферта, докато все още няма направени залагания за нея
- 1.1.8. Да затваря своя оферта
- 1.1.9. Участва в наддаванияУчаства в чат кореспонденция с друг потребител
- 1.1.10. Преглежда чат кореспонденция с друг потребител
- 1.1.11. Публикува постове във форума по дадена тема
- 1.1.12. Добавя теми във форума
- 1.1.13. Преглежда теми във форума
- 1.1.14. Преглежда постове по тема във форума
- 1.1.15. Може да поставя оценка на други потребители
- 1.1.16. Преглежда оценките на всички потребители
- 1.1.3. Преглежда оценка на деден потребител
- 1.1.4. Преглежда своята оценка за даден потребител

1.2. *Дилър (Dealer) трябва да може да:*

- 1.2.1. Да се логва в акаунта си
- 1.2.2. Да излиза от акаунта си
- 1.2.3. Редактира своя акунт
- 1.2.4. Добавя оферти
- 1.2.5. Преглежда всички оферти
- 1.2.6. Редактира своя оферта, докато все още няма направени залагания за нея
- 1.2.7. Да затваря своя оферта, докато няма още направени залагания за нея
- 1.2.8. Да финализира своя оферта, докато няма още направени залагания за нея
- 1.2.9. Преглежда дадена оферта
- 1.2.10. Участва в наддавания
- 1.2.11. Участва в чат кореспонденция с друг потребител
- 1.2.12. Преглежда чат кореспонденция с друг потребител
- 1.2.13. Публикува постове във форума по дадена тема
- 1.2.14. Добавя теми във форума

- 1.2.15. Преглежда теми във форума
- 1.2.16. Преглежда постове по тема във форума
- 1.2.17. Може да поставя оценка на други потребители
- 1.2.18. Преглежда оценките на всички потребители
- 1.2.19. Преглежда оценка на деден потребител
- 1.2.20. Преглежда своята оценка за даден потребител

1.3. *Наддаващ (Bidder) трябва да може да:*

- 1.3.1. Да се логва в акаунта си
- 1.3.2. Да излиза от акаунта си
- 1.3.3. Редактира своя акунт
- 1.3.4. Преглежда всички оферти
- 1.3.5. Преглежда дадена оферта
- 1.3.6. Участва в наддавания
- 1.3.7. Участва в чат кореспонденция с друг потребител
- 1.3.8. Преглежда чат кореспонденция с друг потребител
- 1.3.9. Публикува постове във форума по дадена тема
- 1.3.10. Добавя теми във форума
- 1.3.11. Преглежда теми във форума
- 1.3.12. Преглежда постове по тема във форума
- 1.3.13. Може да поставя оценка на други потребители
- 1.3.14. Преглежда оценките на всички потребители
- 1.3.15. Преглежда оценка на деден потребител
- 1.3.16. Преглежда своята оценка за даден потребител

1.4. *Анонимен (Anonymous) трябва да може да:*

- 1.4.1. *се регистрира, при което се създава деактивиран акаунт, който чака активация от потребител с роля "Админ".*

2. WEB апликацията трябва да предоставя възможност за **лесна връзка** с *REST API* и да позволява на потребителя да достъпва посочените за всяка роля в точка 1) данни и функционалности, в зависимост от неговата роля

Нефункционални изисквания

1. **Сигурност (Security)** - Всеки потребител от горе-посочение роли трябва да има достъп само да своите данни и функционалности, без да може да достъпва функционалност и данни, без необходимата оторизация.
2. **Производителност** - Да се постигне възможно най - висока производителност
3. **Наличност** - Да е налична максимално време, с възможно най - кратки прекъсвания на услугата
4. **Модифицируемост** - Сорс кодът на проекта да е организиран така, че лесно да може да се променят, модули, които могат да търпят чести модификации
5. **Тестваемост** - Source кода да е организиран, така че лесно да се тестват отделните модули
6. **Използваемост** - Предоставяне на удобен *WEB* интерфейс за комуникация с REST API частта.

Използвани технологии и модули

1. Реализация на *REST API* със *Spring Boot*:
 - 1.1. **Java 11**
<https://openjdk.java.net/projects/jdk/11/>
<https://docs.oracle.com/en/java/javase/11/docs/api/index.html>
 - 1.2. **Spring Framework (Spring Boot 2)** - Open source, Java базирана технология(framework), използван за създаване на микросървиси
<https://spring.io/projects/spring-boot>
 - 1.3. **Spring Security** - технология(framework), която улеснява прилагането на security ограничния към *Spring Framework*
<https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/>
 - 1.4. **Spring webflux** - асинхронна, реактивна технология, която позволява създаването на напълно асинхронни, неблокиращи приложения, поддържаща *реактивни потоци* (reactive streams)
<https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html>
 - 1.5. **Spring MongoDB** - драйвер за синхронна комуникация с база данни MongoDB
<https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.core>

- 1.6. **Spring Reactive MongoDB** - драйвер за комуникация с база данни MongoDB, който позволява реализирането на асинхронни потоци и така създаване на *Server sent events (SSE)*, където клиента се абонира за информация за настъпването на асинхронни събития от сървъра. В случая на този проект е ползвано при реализацията на чат и наддавания в реално време

<https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.reactive>

- 1.7. **Lombok** - библиотека, която позволява избягването на тривиалния, но задължителен за по - голямата част от приложенията, код по време на реализация, като getter, setter, constructor методи, чрез опция за @<Annotation> базирано дефиниране

<https://projectlombok.org/>

2. Реализация на WEB апликацията с *ReactJs* и *Thymeleaf*:

- 2.1. *ReactJS* – Font-end библиотека(*framework*), позволяваща реализирането *Single page applications SPA*

<https://reactjs.org/>

- 2.2. *Thymeleaf* – JAVA библиотека, реализиране на web клиентни, чрез създаване и презиползване на образци(*templates*) на XML/XHTML/HTML5

<https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>

- 2.3. *ReactJS* базирани модули и библиотеки:

- *react-chat-window*

<https://www.npmjs.com/package/react-chat-window#example>

- *react-star-rating-component*

<https://www.npmjs.com/package/react-star-rating-component>

- *Webpack.js*

<https://webpack.js.org/concepts/loaders/>

- *Babel JS compiler*

<https://babeljs.io/>

- *Create React App*

<https://github.com/facebook/create-react-app>

3. Общо и за двете:

- 3.1. **Gradle** - Отворена(*open source*) система за автоматизиран на процесите по билдване, тестване и деплойване на софтуени проекти

<https://gradle.org/>

Нетривиални аспекти на системата

Реализация на бизнес модела на система, в която си взаимодействат потребители с някоя от три роли, Админ, Дилър и Наддаващ в контекста на проектираната система специфични за конкретната тематика на решавания проблем.

Системата е разделена на малки модули и нива, което работи в полза на не-функционалните изисквания - *Тестваемост* и *Модифицируемост*, но в същото време преминаването през повече нива за осъществяването на процеса на работа на бизнес модела може да доведе до намалява *Производителността* на системата.

Системата е разделена на 6 главни части - “чат” функционалност, “форум” функционалност, “рейтинг” функционалност, функционалност за “наддаване”, функционалност за мениджирание на “оферти” и “потребителски акаунти”. Това води до по - лесното дефиниране и изпълнение на *функционалните изисквания*.

Значими интерфейси

Форум(Forum):

GET */api/forum - Връща лист с всички форум страници.

GET */api/forum/{topic} - При подаден topic връща форум страница отговаряща на тема topic

POST */api/forum - Очаква в body json с подадена тема - topic. Създава нова форум страница

PUT */api/forum/{topic} - Очаква body json с подаден пост. Добавя подадения пост към страницата на темата topic

PUT */api/forum/{topic}/{postId} - Очаква body json с подаден пост. Променя поста с id postId на тема topic с подадения пост.

DELETE */api/forum/{topic}/{postId} - изтрива пост с id postId от тема форум с тема topic.

DELETE */api/forum/{topic} - изтрива форум страница с тема topic и всичките постове в нея

Чат(Chat):

GET */api/chat - Връща лист с всички съобщения.

GET */api/chat/{id} - Връща съобщение, което с идентификатор id.

GET */api/chat/channel/{channelId} - Връща стрийм от съобщения, които са добавени на даден канал channelId. Канал е идентификатор на връзка между двама потребители, които си чатят.

GET */api/chat/{senderId}/{receiverId} - Връща стрийм от съобщения, между дадени потребители с идентификатор senderId праща на друг потребител с идентификатор receiverId

POST */api/chat - Очаква body json с подадено съобщение, получател и изпращател. Подаденото съобщение се запазва в базата.

DELETE */api/chat/{id} - изтрива съобщение с идентификатор id

Рейтинг(Rating):

GET */api/ratings - Връща лист с всички рейтинзи.

GET */api/ratings/evaluatedUser/{userId} - Връща лист с всички оценки за даден потребител.

GET */api/ratings/{evaluatedId}/{userId} - Връща оценката на потребител с ID = userId за потребител с ID = evaluatedId.

GET */api/ratings/average/{userId} - Връща число с плаваща запетая - средната оценка за даден потребител с идентификатор userId

GET */api/ratings/{id} - Връща оценка с идентификатор id

POST */api/ratings - Очаква body json с подадена оценка, идентификатор на оценяващ и идентификатор на оценяван. Запазва оценката в базата.

PUT */api/ratings/{id} - Очаква body json с подадена оценка, идентификатор на оценяващ и идентификатор на оценяван. Променя оценка с идентификатор id като я заменя с подадената.

PUT */api/ratings/{evaluatedUserId}/{graderUserId} - Очаква body json с подадена оценка, идентификатор на оценяващ и идентификатор на оценяван. Променя оценка на оценяващ с идентификатор graderUserId и оценяван с идентификатор evaluatedUserId като я заменя с подадената.

DELETE */api/ratings/{id} - изтрива оценка с идентификатор id.

Потребител(User):

GET */api/users - Връща списък с всички потребителски акаунти

GET */api/users/accounts/inacvtive - Връща списък с всички **неактивни** потребителски акаунти

GET */api/users/accounts/active - Връща списък с всички **активни** потребителски акаунти

GET */api/users/accounts/active/others?id={userId} - Връща списък с всички **активни** потребителски акаунти, освен този с подаденото id = useId

GET */api/users/accounts/excluded?id={userId} - Връща списък на всички акаунти освен този с подаденото userId

GET */api/users?username={username} - Връща акаунт по потребителско име

GET */api/users?id={userId} - Връща акаунт по ID

POST */api/users/register - Очаква request body от тип JSON, който представя полетата на модела на потребител (User) и създава **неактивен** акаунт със съответните данни от request body

PUT */api/users/accounts/deactivate?id={userId} - Деактивира акаунта с ID = userId

PUT */api/users/accounts/activate?id={userId} - Активира акаунта с ID = userId

PUT */api/users?id={userId} - Очаква request body от тип JSON, който представя полетата на модела на потребител (User) и **обновява** акаунта, според подадените данни в request body

Оферта(Offer):

GET */api/offers - Връща списък с всички оферти

GET */api/offers?id={offerId} - Връща офертата с подаденото offerId. Ако тя не съществува връща грешка.

GET */api/offers?userId={userId} - Връща списък с всички оферти на потребителя с подаденото userId

GET */api/offers - Връща списък с всички оферти

PUT */api/offers - Очаква request body от тип JSON, който представя полетата на модела на оферта (Offer) и **обновява** офертата, според подадените данни в request body

POST */api/offers - Очаква request body от тип JSON, който представя полетата на модела на оферта (Offer) и добавя нова оферта, според подадените данни в request body

PUT */api/offers/finalize - Очаква request body от тип JSON, който представя полетата на модела на оферта (Offer) и финализира офертата, ако има наддавания за нея, или я затваря(отменя), ако няма такива. В случая с финализиране на сделката, то се прехвърля сумата на печелившия залог от акаунта на наддаващия в акаунта на автора на офертата.

Наддаване(Bid):

GET */api/bids/save/stream - Връща stream от всички залози за офертата с ID = offerId

GET */api/bids/save/stream - Връща stream от всички залози направени от потребителя с ID = userId

POST */api/bids/save - Очаква request body от тип JSON, който представя полетата на модела на залог(Bid) Запазва залог(bid) подаден по данните от Request body.

Инсталация и конфигуриране

Системата е проектирана и тествана на основа *Java версия 11*.

Препоръчително е ползването на *Java 11*, за *build* и стартиране на сървърната част на прокета. Също така е нужно изтеглянето на всички dependency-та описани в секция “Използвани технологии и модули” - т.1. Това може да стане най - лесно, използвайки *Gradle* и файла “build.gradle” от основната директория на проекта.

За да се build-не react front-end апликацията е нужно:

1. Да се изпълни командата “npm install” в поддиректорията на проекта “webapp”. Тя ще инсталира всички нужни *dependency*-та дефинирани в “package.json” файла от същата директория в директорията “node_modules”.
2. След това с командата “npm run build” в същата директория се *build*-ва пакета на react апликацията в поддиректорията “dist/js” под име “index_bundle.js”.
3. Следващата стъпка е да се копира “index_bundle.js” в поддиректорията на проекта “resources/static/js”.
4. Остана само да се стартира сървъра на проекта, който ще сервира *web* апликацията на “/”.

За всяка една от горните стъпки е дефиниран скрипт в “build.gradle”.

Потребителска документация

На потребителя е предоставен удобен интерфейс за работа със системата под формата на *WEB* апликация. Може също да бъде използван всяка една web клиент апликация.

Заклучение

Реализацията на този проект доведе до запознанството ни с много нови за екипа технологии и техните специфики, като например: **ReactJS**, **Spring Framework**, използване на **Http протокола** и други.

Трудности, срещнати по време на реализацията бяха основно свързани с конфигуриране на някои *dependency*-та, на Security конфигурацията, на реализацията на *WEB* апликацията, използвайки *ReactJS*, както и други. Процеса по преодоляването им, обаче, ни донесе много нови знания в областта на разработка на REST API-та, *microservices*, сигурност на данните и много други.

Плановите за бъдещо развитие са свързани с подобряване на GUI на *WEB* апликацията, намиране и отстраняване на допуснати грешки и дупки в сигурността, както и разширяване на системата с още функционалности, от които има нужда конкретния бизнес модел.

Източници

<https://github.com/iproduct/course-spring5/wiki> - FMI Spring5 course, by Trayan Iliev, 2019/2020 educational year edition

<https://www.baeldung.com/servlet-json-response> - Returning a JSON Response from a Servlet, by baeldung, October 26, 2018

<https://www.baeldung.com/spring-data-mongodb-reactive> - Spring Data Reactive Repositories with MongoDB, by baeldung, December 22, 2019

<https://howtodoinjava.com/spring5/webmvc/spring-mvc-cors-configuration/> - CORS with Spring, By Lokesh Gupta | Filed Under: Spring5 WebMVC

<https://spring.io/guides/tutorials/react-and-spring-data-rest/> - React.js and Spring Data REST, by Spring community

<https://shekhargulati.com/2019/01/13/running-tests-and-building-react-applications-with-gradle-build-tool/> - Issue #40: 10 Reads, A Handcrafted Weekly Newsletter For Software Developers, by Shekhar Gulati, January 13, 2019