**Имена:**

**Предмет:** MVC Frontend and Backend Frameworks

**GitHub:** https://github.com/fusionart/TimeOffSystemTeamOne

https://github.com/fusionart/TimeOffSystemBackEndTeamOne

Николай Илиев,

Владимир Пеков,

Петра Рашева и

Преслава Кузова

**Дата: 04.11.2017**

Time Off System

**1. Условие**

Целта е да се създаде уеб базирано приложение за подаване, обработка и следене на отпускате на персонала в дадена фирма. Системата трябва да има портал за логин на регистрирани потребителите. След коректно въвеждане на потребителско име и парола се достъпва до модула Create Request, където от календар се избират дните за отпуска и какъв да бъде нейният вид – платенa (“Paid Time Off”/”PTO”), неплатенa (“Unpaid Time Off”/”UPTO”) или болничен (“Sick Leave”/ “SL”). Приложението трябва да разполага с admin panel, от където да се одобряват или отхвърлят подадените заявки и където се регистрира нов потребител. За реализацията трябва да се използват Angular за frond end частта и Spring Boot за back end кода.

**2. Въведение**

Приложението е реализирано с помощта на Angular – TypeScript базирана open-source front-end платформа и Spring Boot – application framework и inversion of control container for the Java platform, който бе използван за back-end частта.

**3. Теория**

Визуалните елементи са реализирани с помощта на текстовия едитор Visual Studio Code, а самият алгоритъм, работещ под Java – благодарение на софтуера Spring Tool Suite. За базата данни се използва MySQL.

**4. Използвани технологии**

За реализирането на задачата са използвани различни езици за частта, която се визуализира на потребителя, и за логиката, отговаряща за съхранението на информацията за потребителите в базата данни. Използвани са и външни библиотеки като Bootstrap и PrimeNG, за улеснение изграждането на графичната част на приложението.

**5. Инсталация и настройки**

За да се стартира приложението е нужно да се отвори command window в папката на front-end частта и да се въведе командата ng serve -o. След зареждането на всички модули, приложението ще се отвори на localhost:4200. За стартиране на back-end частта е нужно да се стартира Spring Tool Suite и да се run-не като Spring Boot Application. След успешно зареждане, back-end частта ще слуша на localhost:8080.

**6. Кратко ръководство на потребителя**

Стартирайте програмите, както е описано в горната точка. Пред потребителя ще се покаже логин екран. Бутоните Request и Create Request не са активни преди въвеждането на коректни данни и пренасочват към Login менюто. След правилно въведени данни, може да се направи заявка за отпуска. Потребителят може да избира от три вида отпуска – платена, неплатена и болничен, като за различните видове са приложими различни правила. Ако в приложението сте логнат като админ, ще имате възможността да приемате подадените от другите потребители отпуски.

**7. Описание на програмния код**

**Front-end:**

За визуалната част на проекта са използвани следните компоненти и техните особености:

1. **Admin Panel** – това е панелът, от който могат да се одобряват подадените от потребителите на уеб приложението заявки или да бъдат регистрирани нови потребители.  
   Панелът използва **AdminPanelService** и **TimeOffRequestInterface.**Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *approve()* – отговаря за одобрението на подадените заявки
* *getTOimage()* – спомага визуализацията на изображение, намиращо се в папка *assets/images,* като намира файла със съответстващо име
* *showDialogToAdd()* – показва диалогов прозорец за добавянето на одобрена от админа заявка
* *onFocusT(type)* - връща типа на зададената отпуска – съответно – платенa (“Paid Time Off”/”PTO”), неплатенa (“Unpaid Time Off”/”UPTO”) или болничен (“Sick Leave”/ “SL”)
* *getTOFimage(typeTO: string): Promise<String>* - връща различно име на файла спрямо вида на зададената отпуска
* *approveRequest(approveObj: any): Observable<any>* - реализацията на този метод в сървиса подпомага одобряването на заявките от админа
* *public getRequests(): Observable<TimeOffRequest[]>* - връща заявките от базата данни под формата на масив от данни

1. **Alert** ­ - при възникване на проблем, този компонент извежда подходящо съобщение, съответстващо на възникналата грешка. Използва **AlertService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* При инициализацията на сървиса (т.е. в конструктора на класа) се изчистват съобщенията при промяна на страницата за визуализация.
* Методите *success()* и *error()* отговарят за коректното подаване на информация при възникване на грешки

1. **Calendar** – потребителят може да избира за какъв период от време да бъде неговият отпуск. При избиране на опцията ”paid time off” се взима предвид оставащият брой дни за този вид отпуска, “unpaid time off” може да се вземе за неопределен период от време, а пък “sick leave” – само за времето, посочено в болничния лист, който трябва да бъде прикачен. При маркирането не се отчитат уикендите, а единствено делничните дни. Използва **CalendarService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *makeGreenDays()* – визуализира в зелено датите на съответната заявка
* *setMyStyles(date)* – визуализира в червено датите на официалните почивни дни
* *onSelect($event)* – след успешно въвеждане на данните, се подават датите и броят на дните, в които ще се отсъства от работа
* *onClear()* – изчиства данните, броят на дните се равнява на нула
* Не би било уместно и да се спомене какво се случва при инициализацията на компонента, а именно в метод *ngOnInit()* – задават се характеристиките на календара – какви са пълните и съкратените имена на дните/месеците, формата на часа и прочие.
* *public getHolidays(): Observable<Holiday[]>* - създава масив от дати, които са почивни дни като използва предварително подадено api със зададени празници

1. **Create Request** – това е компонентът, който отговаря са подаването на заявка за отпуск. По условие трябва да може да се избира от три вида отпуск – платен, неплатен и болничен. Използва **CreateRequstService** и **RequestListService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *getCurrentUser(), getCurentUserAvailablePto(), updatePto()* – това са взаимно обвързани методи, които отговарят за намирането на user-а, който прави заявка. При подаване на запитване за платен отпуск, се взема предвид наличните дни на платен отпуск, с които потребителя разполага, и тяхната промяна след успешна заявка.
* onSubmit() – след като потребителят е подал своята заявка за отпуск, се запазват промените (начало и край на желания отпуск, причините за заявката на такава и бележки). Те се подават към сървиса, където се обработват.
* addRequest(timeOffRequest: TimeOffRequest): Observable<any> - този метод отговаря за запазването на коректно подадената от потребителя заявка
* *getTOFimage(typeTO: string): Promise<String>* - връща различно име на файла спрямо вида на зададената отпуска
* *approveRequest(approveObj: any): Observable<any>* - реализацията на този метод в сървиса подпомага одобряването на заявките от админа
* *public getRequests(): Observable<TimeOffRequest[]>* - връща заявките от базата данни под формата на масив от данни

1. **Dropdown Component** – отваря за избирането на вид отпуск – платен (“Paid Time Off”/”PTO”), неплатен (“Unpaid Time Off”/”UPTO”) или болничен (“Sick Leave”/ “SL”)
2. **Login** – този компонент отговаря за логин функционалността на уеб приложението – след коректно въвеждане на потребителско име и парола на потребителя се дава достъп до полето за правене на заявка за отпуск. Хубаво е да се спомене, че докато влизането в акаунта не е успешно, бутоните за другите функционалности на приложението не се вижда. С помощта на валидатори се осъществяват проверките – дали и двете полета са попълнени, дали потребителското име/паролата са валидни. При грешка, се изписва подходящо съобщение. Използва **AlertService** и **AuthenticationService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *login(),* който използва *login(username: string, password: string): Observable<boolean>* - това е методът, при чието извикване на потребителя се дава правото да има достъп до страницата за заявки. Стойността на token-a се запазва в local storage, така че при refresh-ване на страниците клиентът все още да е в своя акаунт.
* *getCurrentuserDetails(username: string): Observable<any>* - връща данни за потребителя. Те са запазени в localStorage-a.
* *logout(): void* – метод, способстващ logout-а от съответния акаунт. За да се случи това, token-а се изчиства, а от localStorage се премахва настоящия потребител, както и данните за него

1. **Register** – това е компонентът, отговарящ за регистрацията на потребителите в системата. За да бъде регистрацията успешна и валидна, е нужно да се въведе точно информацията в полетата. За това отговарят валидатори, намиращи се в HTML template-а на този компонент. Всички полета са задължителни. При неточности, се изписва подходящо съобщение, оповестяващо за допуснатата грешка*.* Бутонът не е достъпен освен ако информацията не е въведена коректно. Използва **RegisterUserService** и **UserService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *OnSubmit(form: NgForm)* – след като бутонът “Register” бива натиснат, това е методът, който бива извикван. Той прави проверка дали информацията, въведена в полето *password* и *repeat password* съвпадат, като при несъответствие се изписва подходящо съобщение
* *registerUser(registerUserRequest: RegisterUserRequest): Observable<any>* - същинското регистриране на user-а в базата данни

1. **Request Details** – това е компонентът, който отговаря за визуализацията на подадената от потребителя молба за отпуск. Използва **RequestListService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *getRowData()* – този метод взима информацията от текущия ред стига да има такава
* *showSecondCalendar()* – ако потребителят е избрал дати за своя отпуск, които не са е един и същ месец, за визуализацията на данните е нужно да се ползва втори календар
* *getCurrentUser()* и *public getCurrentUserData(): Observable<User>* – взимат данните за настоящия потребител и ги обработват
* *public getRequests(): Observable<TimeOffRequest[]>* - връща масив от подадените заявки за отпуск на съответния потребител
* *public getAllRequests(): Observable<TimeOffRequest[]>* - връща масив от всички подадени до момента заявки

1. **Requests List** – това е компонентът, който отговаря за визуализацията на детайлите, относно подадените от потребителите на системата отпуски. Използва **RequestListService** и **AuthenticationService.** Някои от по-важните методи, имплементирани за реализацията на този компонент, са:

* *getRequests(),* използващ *public getRequests(): Observable<TimeOffRequest[]>* - връща масив от подадените заявки за отпуск на съответния потребител
* *getCurrentUser(),* използващ *public getCurrentUserData(): Observable<User>* - връща масив от всички подадени до момента заявки
* *getTOimage(typeTO: string): String* - спомага визуализацията на изображение, намиращо се в папка *assets/images,* като намира файла със съответстващо име

**Back-end:**

За логиката е използван Spring Boot framework, като кодът е разработен по MVC моделът. Реализацията е осъществена по следния начин:

1. **Пакет com.tos.timeoffserver**

* *TimeOffSystemTeamOneBackEndApplication.java* – това е класът, който съдържа main метода на сървърната част на приложението

1. **Пакет com.tos.timeoffserver.controllers** – в тази папка се помещават класовете, който играят ролята на контролери – те handle-ват HTTP requests

* *HolidayController.java* – това е контролерът, отговарящ за почивните дни. Използва *public interface HolidayRepository extends CrudRespository<Holiday, Long>* и *@Service public class HolidayService.*
* *TimeOffRequestController.java* – това е контролерът, отговарящ за заявката за отпуск. Използва *public interface TimeOffRequestRepository extends CrudRespository<TimeOffRequest, Long>, public interface UserRepository extends CrudRespository<ApplicationUser, Long>, public interface TimeOffDatesRepository extends CrudRespository<TimeOffDate, Long>* и *@Service public class UserService, @Service public class TimeOffRequestService.* Важни методи, реализирани в този клас, са:
  + *public @ResponseBody String addNewRequest(@RequestBody NewTimeOffRequestBody newTimeOffRequest, HttpServletRequest req) –* отговаря за записването на нова заявка за отпуск в базата данни
  + *public @ResponseBody Iterable<TimeOffRequestResponse> getAllRequest()* – този метод връща лист с всички заявки за отпуск, направени до момента
  + *public ResponseEntity<String> approveRequest(@RequestBody ChangeRequestStatusPost changeStatusPost)* – с помощта на този метод се одобряват заявките, направени от потребителите
  + *public ResponseEntity<TimeOffRequest> deleteTimeOffRequestById(@PathVariable(value = "id") Long id)* – отговаря за изтриването на заявка за отпуск посредством подаденото уникално id на заявката
* *UserController.java* – това е контролерът, отговарящ за потребителите на системата за отпуск. Използва *public interface UserRepository extends CrudRespository<ApplicationUser, Long.* Важни методи, реализирани в този клас, са:
  + *public void signUp(@RequestBody ApplicationUser user) –* криптира паролата на потребителя, за да ограничи прекият достъп до нея от базата данни
  + *UserDetailsResponse getUserInfo(@RequestBody UserRequest userInfoRequest)* – взема наличната информация за потребителя, ако съществува такъв
  + *public @ResponseBody ApplicationUser addNewUser(@RequestParam String firstName, @RequestParam String secondName, @RequestParam String lastName, @RequestParam String address, @RequestParam String email, @RequestParam String personalId, @RequestParam String telephone, @RequestParam String position)* – този метод способства за добавянето на нов потребител в базата данни, изисквайки всички данни
  + *public @ResponseBody ApplicationUser loginUser(@RequestBody LoginData loginData)* – чрез този метод се влиза в потребителски акаунт чрез зададени login данни

1. **Пакет com.tos.timeoffserver domain.entities** – в тази папка се помещават класовете, които ще играят ролята на @Entity-та (модели за базата данни, които Hibernate автоматично ще превърне в таблици)

* *ApplicationUser.java* – съдържа модела на потребителя в базата данни и акомпаниращите го getters и setters
* *Document.java –* създава таблица с име *documents,* с колони съответно *document\_id* и *file\_name*
* *Holiday.java -* съдържа модела на почивните дни (например национални празници) в базата данни и акомпаниращите го getters и setters
* *TimeOffDate.java -* съдържа модела на датите, в рамките на които ще се състои отпуската, в базата данни и акомпаниращите го getters и setters
* *TimeOffRequest.java -* съдържа модела на заявката за отпуск, която потребителя прави, в базата данни (съответно датата, когато заявката е подадена, тип (платенa (“Paid Time Off”/”PTO”), неплатенa (“Unpaid Time Off”/”UPTO”) или болничен (“Sick Leave”/ “SL”)), брой на дните, начало и старт, причина, статус и бележки) и акомпаниращите го getters и setters
* *UserDetailsServicempl.java* - това е сървисът, който връща потребителското име и паролата на потребителя от базата данни, и отговарящ за handle-ването на грешки, когато такива не са намерени в базата данни. Използва *UserRepository extends CrudRespository<ApplicationUser, Long>.* Важни методи, реализирани в този клас, са:
  + *public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException* *–* връща потребителското име и паролата на потребителя, ако такива са налични

1. **Пакет com.tos.timeoffserver.domain.model** – в тази папка се помещават класовете, които ще играят ролята на модели в back end частта на приложението

* *CurrentUser.java*
* *LoginData.java*
* *NewTimeOffRequestBody.java*
* *TimeOffRequestProxy.java*
* *TimeOffRequestResponse.java*
* *UserDetailsResponse.java*
* *UserRequest.java*
* *UserResponse.java*

1. **Пакет com.tos.timeoffserver.domain.respositories ­**– в тази папка се помещават класовете, които ще играят ролята на respository-та в back end частта на приложението

* HolidayRepository.java
* TimeOffDatesRepository.java
* TimeOffRequestRepository.java
* UserRepository.java

1. **Пакет com.tos.timeoffserver.security** – в тази папка се помещават класовете, които отговарят за сигурността и за авторизацията на потребителя

* *AllowLocalhost.java*
* *JWTAuthenticationFilter.java*
* *JWTAuthorizationFilter.java*
* *SecurityConstants.java*
* *WebSecurity.java*

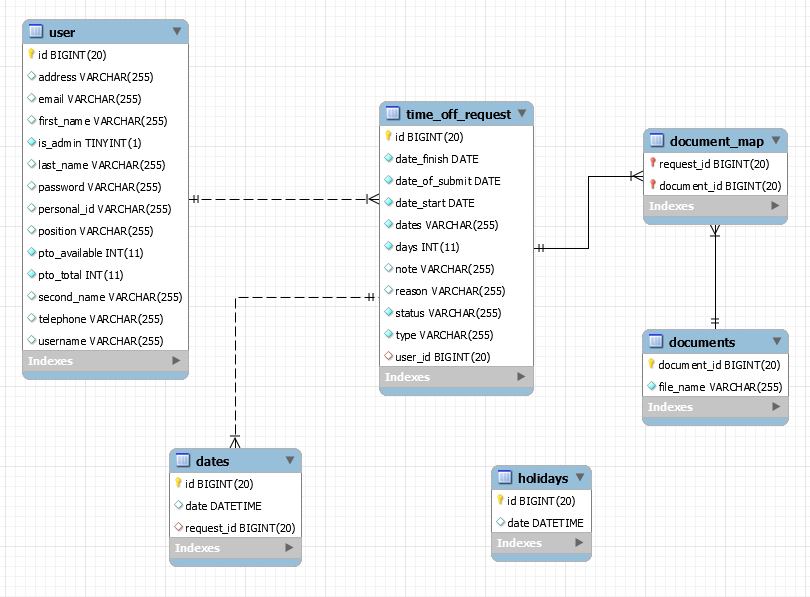
1. **Пакет com.tos.timeoffserver.services**

* *HolidayService.java* – това е сървисът, отговарящ за задаването на дните, които се считат за почивни. Използва *public interface HolidayRepository extends CrudRespository<Holiday, Long>.* Важни методи, реализирани в този клас, са:
  + *public void addHoliday(String dateString) throws ParseException* – отговаря за маркирането на точно определен ден като почивен такъв
  + *@PostConstruct public void initDb() throws ParseException* – инициализира дните, които са почивни
* *TimeOffDatesService.java* – това е сървисът, отговарящ за дните, който потребителят изискал за почивни. Използва *public interface HolidayRepository extends CrudRespository<Holiday, Long>.*
* *TimeOffRequestService.java* – това е сървисът, отговарящ за вземането на отпуск. Използва *public interface HolidayRepository extends CrudRespository<Holiday, Long>* и *public interface TimeOffRequestRepository extends CrudRespository<TimeOffRequest, Long>.* Важни методи, реализирани в този клас, са:
  + *public int getTimeOffDays(Date startDate, Date finishDate, int workingDays)* – отговаря за задаването на периода на отпуската (начална дата и крайна такава), както и за броя на работните дни, в които потребителя иска да отсъства от работното си място
  + *private Date[] orderDates(Date startDate, Date finishDate)* – подрежда датите във възходящ ред, за да бъдат подавани коректно
  + *private List<java.util.Date> getWorkdays(Date dateStart, Date dateFinish)* – връща лист от датите, които за работни дни, за които потребителя си взема отпуск
  + *private boolean isSaturday(java.util.Date nextDay)* и *private boolean isSunday(java.util.Date nextDay)* – правят проверка дали дните не са почивни

Важно е да се отчете, че избраните дати могат да бъдат както последователни, така е не последователни. Преобразува се формата (от числов в текстов) на календарни единици съставляващи една заявка за отпуск.

* *UserService.java* – това е сървисът, отговарящ за потребителите на приложението. Използва *public interface UserRepository extends CrudRespository<ApplicationUser, Long>.* Важни методи, реализирани в този клас, са:
  + *public void addUser(String firstName, String secondName, String lastName, String username, String password, String email, String address, String telephone, String position, boolean isAdmin, int PtoAvailable, int PtoTotal)* – отговаря за добавянето на нов потребител в базата данни с user-и
  + *public void changeUserProAvailable(String type, int days, ApplicationUser currentUser)* – при вземане на платен отпуск (“Paid Time Off”/”PTO”), броят на оставащите дни за този вид отпуск се променя
  + *public boolean isUserAdmin(ApplicationUser currentUser)* – прави проверка дали настоящият потребител има правомощията на admin

1. **Структура на база данни**



1. **Използвани източници**

Основни източници, помогнали при създаването на кода, са:

1. <https://www.primefaces.org/primeng/#/>
2. <https://angular.io>
3. <https://projects.spring.io/spring-boot/>
4. <https://stackoverflow.com>