Tema:

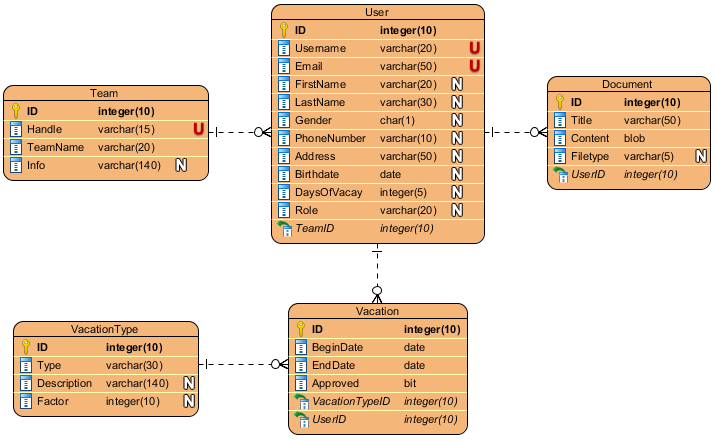
FirmInternalApp

## Kratak opis teme

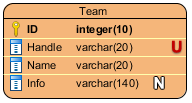
Svaka ozbiljnija IT firma posjeduje aplikaciju koja se koristi interno za operacije poput upravljanja korisnicima, dokumentima i zahtjevima. FirmInternalApp je upravo primjer takve aplikacije, koja olakšava svakodnevni rad u firmi, omogućava pregled relevantnih dešavanja i novosti.

Aplikacija će raspolagati sa 4 modula, te će omogućiti sljedeće funkcionalne zahtjeve: upravljanje obavijestima, upravljanje godišnjim odmorima, upravljanje dokumentima, kao i dvije vrste korisnika i privilegije shodno dodijeljenom tipu. Ukoliko se radi o administratoru, dodatne privilegije bi omogućile brisanje i dodavanje dokumenata. Uposlenik HR tima će se razlikovati po mogućnostima odobravanja ili odbijanja zahtjeva za godišnji odmor. Svi ostali korisnici će imati pristup predaji zahtjeva, pregledu uposlenika po timovima, dodavanju novih timova i sl.

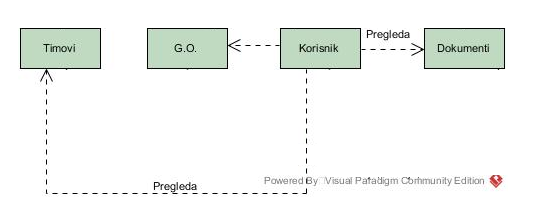
## ERD sistema



## ERD modula za Timove



## Prikaz komunikacije između modula



## Enkripcija konfiguracijskog servera

Za enkripciju konfiguracijskog servera korištena je simetrična kriptografija, tj. isti se ključ (privatni i javni) koristi i za enkripciju i dekripciju. Da bismo omogućili konfiguracijskom serveru da enkriptuje *plain* stringove kao i da dekriptuje enkriptovane stringove, u njegov application.properties dodamo sljedeće polje:

encrypt.key = skafiskafnjak

čija vrijednost predstavlja vrijednost ključa. Konfiguracijski server će, pri usluživanju konfiguracijskih datoteka klijentskim aplikacijama, koristiti ovaj ključ da dekriptuje lozinke za pristup njihovim bazama podataka.

Prije svega toga, a nakon dodjeljivanja ključa serveru, te lozinke smo ručno enkriptovali u Postmanu POST pristupom na localhost:8888/encrypt i prosljeđivanjem lozinke za bazu u x-www-form-urlencoded formatu. Vrijednost koju dobijemo kao tijelo odgovora zalijepimo u konfiguracijsku datoteku umjesto lozinke nakon ključne riječi {cipher}.

Da bismo provjerili ispravnost enkripcije, možemo izvršiti POST zahtjev na localhost:8888/decrypt i proslijediti kriptovanu vrijednost lozinke, a kao odgovor ćemo dobiti našu stvarnu lozinku za bazu podataka.

## Implementacija komunikacije između modula

Komunikacija između pojedinačnih modula (mikroservisa) je ostvarena gdje god u sistemu postoji situacija da u tabeli jednog modula postoji instanca stranog ključa na tabelu koja se nalazi u nekom od preostalih modula. U tom modulu se zbog toga nalazi skraćena verzija te tabele u koju se onda vrše promjene svaki put kada dođe do promjene u odgovarajućoj tabeli iz drugog modula.

Uzmimo za primjer module UsersModule i DocumentsModule. Ovaj prvi sadrži tabelu korisnika, dok tabela dokumenata u drugom modulu sadrži strani ključ na korisnika koji je ujedno i autor dokumenta. Zbog toga u modulu DocumentsModule postoji tabela Authors koja je skraćena verzija tabele korisnika. Tako, svaki put kada se izvrši POST, PUT ili DELETE zahtjev prema tabeli korisnika, ova promjena će se propagirati i u tabelu autora dokumenata u sasvim drugom modulu.

Za implementaciju ove komunikacije korištena je RestTemplate klasa. Unutar klase UsersService dodamo privatnu metodu restInit() koja vraća novu instancu RestTemplate klase:

@Service

**public** **class** UsersService {

...

@Bean

@LoadBalanced

**private** RestTemplate restInit() {

**return** **new** RestTemplate();

}

...

Da bismo propagirali POST zahtjeve prema tabeli korisnika u tabelu autora u modulu DocumentsModule, potrebno je da u metodi addUser() UsersService servisa pozovemo metodu postForObject() RestTemplate klase:

**public** **void** addUser(User user) {

usersRepository.save(user);

String docClient = sirc.getService("documents-client");

String vacClient = sirc.getService("vacations-client");

String url = docClient + "/users";

String url2 = vacClient + "/users";

Logger.*getLogger*("").info(docClient);

RestTemplate rt = restInit();

rt.postForObject(url, user, User.**class**);

rt.postForObject(url2, user, User.**class**);

}

Isto ovo uradimo za PUT i DELETE zahtjeve (ovaj put se radi o put() i delete() metodama RestTemplate klase):

**public** **void** updateUser(User user) {

usersRepository.save(user);

String docClient = sirc.getService("documents-client");

String vacClient = sirc.getService("vacations-client");

String url = docClient + "/users/{id}";

String url2 = vacClient + "/users/{id}";

RestTemplate restTemplate = restInit();

restTemplate.put(url, user, user.getId());

restTemplate.put(url2, user, user.getId());

}

**public** **void** deleteUser(Long id) {

usersRepository.delete(id);

String docClient = sirc.getService("documents-client");

String vacClient = sirc.getService("vacations-client");

String url = docClient + "/users/{id}";

String url2 = vacClient + "/users/{id}";

Map<String, String> params = **new** HashMap<String, String>();

params.put("id", String.*valueOf*(id));

RestTemplate restTemplate = **new** RestTemplate();

restTemplate.delete(url, params);

restTemplate.delete(url2, params);

}