Techninė ataskaita

Komanda „Pyragas“

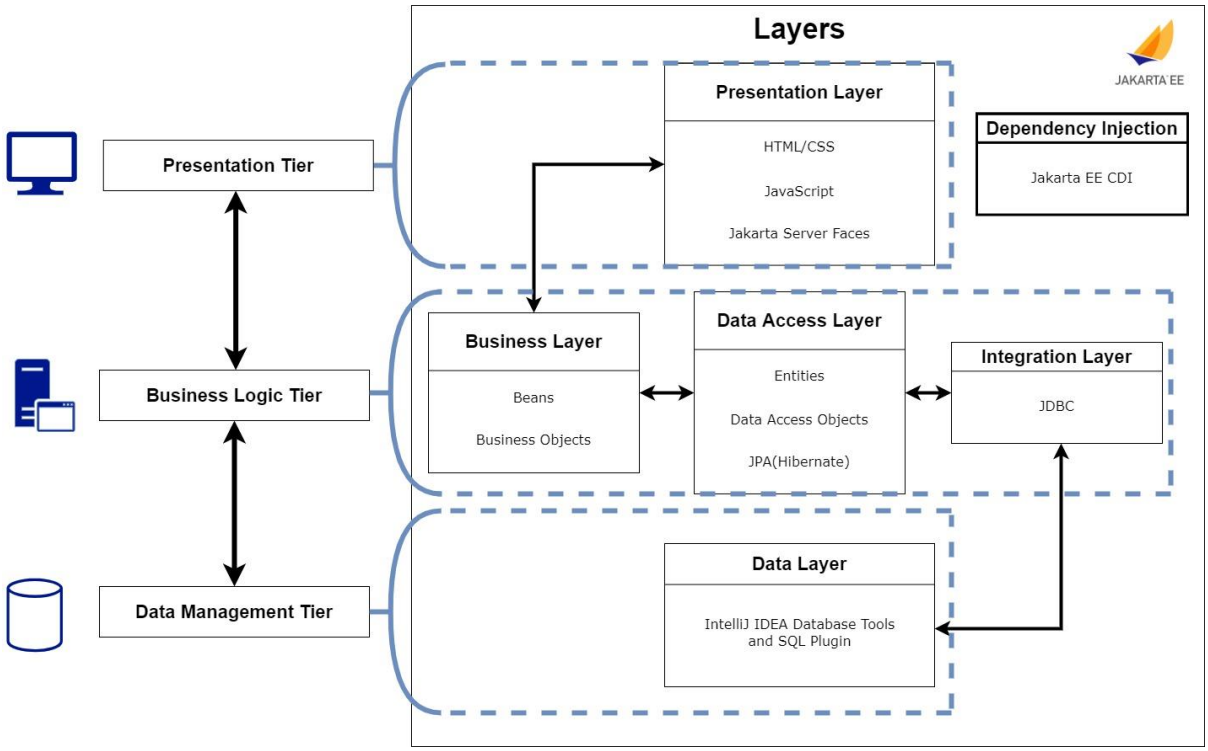
Gintautas Jurgelevičius Marijus Jasinskas Raminta Urbonavičiūtė Mantas Kryževičius Vladislav Kolupayev

Projekto informacija

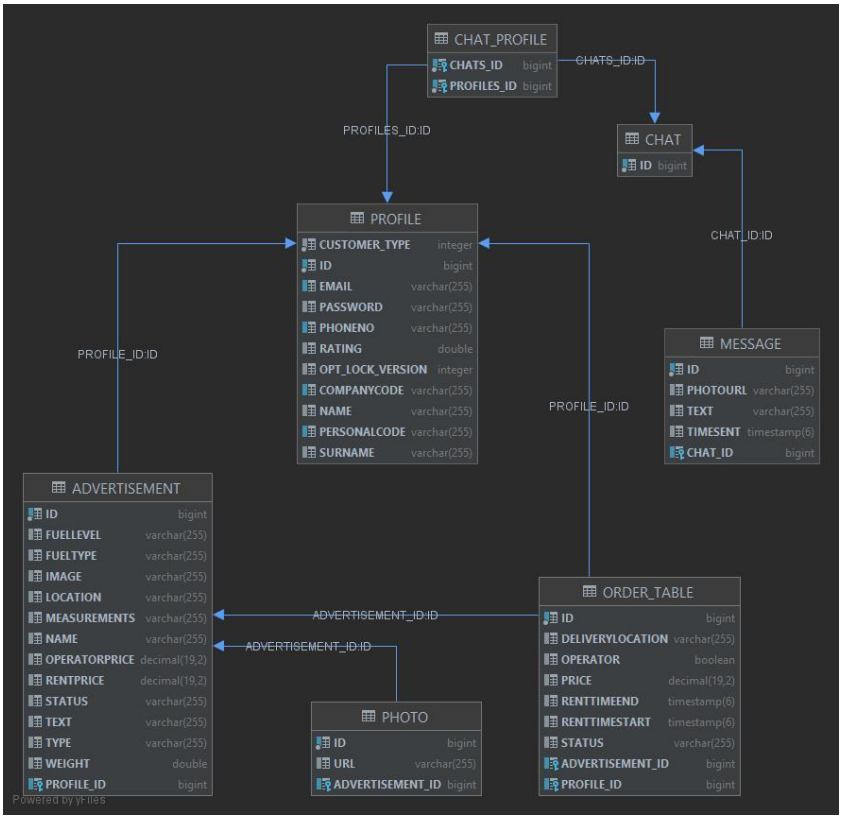
Užduotis: statybinės įrangos dalijimosi platforma

Projekto repozitorijos nuoroda

Sistemos architektūra



Sistemos duomenų bazės schema



Kokybiniai sistemos reikalavimai

Concurrency

Vartotojo prisijungimas išlieka keliuose languose. Prisijungimo įsiminimui naudojame ExternalContext klasės getSessionMap() metodą. Gautame Map laikome vartotojo prisijungimo būseną (src/main/java/com/psk/pyragas/ToolRent/usecases/Registration.java:47 eilutė):

```
@Transactional
public String createProfile() {
    profilesDAO.persist(profileToCreate);
    externalContext.getSessionMap().put("user", profileToCreate);
    return "index.xhtml?faces-redirect=true";
}
```

Kai vartotojas atsijungia, atitinkamai pakeičiame ir ExternalContext esančią prisijungimo būseną (src/main/java/com/psk/pyragas/ToolRent/usecases/Logout.java:8 eilutė):

```
@Model
public class Logout {
    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();

    public String logoutUser(){
        externalContext.invalidateSession();
        return "index.xhtml?faces-redirect=true";
    }
}
```

Security

Klasėje ProfilesDAO naudojame query parametrus, kad sukurtume užklausą, kuri yra atspari SQL atakoms (src/main/java/com/psk/pyragas/ToolRent/dao/ProfilesDAO.java: 29 eilutė):

```
public Profile findOneByEmailAndPassword(String email, String password) {
    return this.em.createNamedQuery("Profile.findOneByEmailAndPassword",
Profile.class)
            .setParameter("email", email)
            .setParameter("pass", password)
            .getSingleResult();
}
```

Užklausos sakinys su užvardintais parametrais (src/main/java/com/psk/pyragas/-ToolRent/entities/Profile.java: 14 eilutė):

```
@NamedQueries({
        @NamedQuery(name = "Profile.findAll", query = "select p from
Profile as p"),
        @NamedQuery(name="Profile.findOneByEmailAndPassword", query =
"select p from Profile as p where p.email = :email and p.password = :pass")
})
```

Data Access

Nesu įsitikinęs, ką čia tiksliai parašyti. Tarsi pas mus yra @Model klasės, o tos, kurios yra @ViewScoped turi Transactional metodus. Tad tokią informaciją ir pateikti? Kad dėl šito yra įgyvendintas reikalavimas?

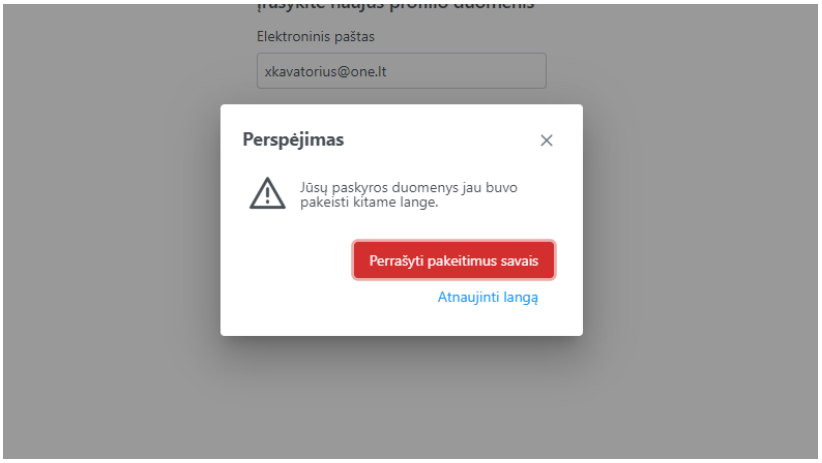
Data consistency and Optimistic locking

Klasėje EditProfile implementavome metodą updateProfile(), kuris leidžia tam pačiam arba keliems vartotojoms redaguojant tos pačios paskyros duomenis pasirinkti, ar perrašyti kito vartotojo pakeitimus, ar nieko nedaryti ir atnaujinti langą (src/main/java/com/psk/pyragas-/ToolRent/usecases/EditProfile.java: ?? eilutė):

```
public void updateProfile() {
    try {
        profilesDAO.update(profile);
        System.out.println("putting user");
        externalContext.getSessionMap().put("user", profile);

FacesContext.getCurrentInstance().getExternalContext().redirect("index.xhtm
l?faces-redirect=true");
    }
    catch (OptimisticLockException | IOException e) {
        System.out.println("exception");

PrimeFaces.current().executeScript("PF('optimisticButton').jq.click()");
    }
}
```



Memory management

SessionScoped komponentų nenaudojame nenaudojame use-case įgyvendinimui. Verslo klasės projekte yra arba @Model (RequestScoped) arba @ViewScoped. Pavyzdžiui, klasė CreateAdvertisement yra @Model (src/main/java/com/psk/pyragas/ToolRent/usecases/-CreateAdvertisement.java):

```
@WillBeLogged
@Model
public class CreateAdvertisement {

    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();

    @Inject
    private AdvertisementsDAO advertisementsDAO;

    @Getter @Setter
    private Advertisement advertisementToCreate = new Advertisement();
```

Klasėje EditProfile teko naudoti @ViewScoped. EditProfile gyvavimo metu nenaudoja daug resursų (src/main/java/com/psk/pyragas/ToolRent/usecases/EditProfile.java):

```
@WillBeLogged
@ViewScoped
@Named
@Getter
@Setter
@Transactional
public class EditProfile implements Serializable {
    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();

    private Profile profile;

    @Inject
    private ProfilesDAO profilesDAO;
```

Reactive programming and Asynchronous/non-blocking communication

Šitą dar reikėtų įgyvendinti.

Cross-cutting functionality and Interceptors

Dalykinio funkcionalumo veiksmus žurnalizuojame klasėje MethodLogger. Tai klasė su @Interceptor anotacija ir metodu logMethodInfo(). Šiame metode į žurnalizavimo failą įrašoma tam tikra informacija prieš ir po metodo įvykdymo. Žurnalizuojame, kada metodas buvo pradėtas vykdyti, kokia klasė kvietė metodą ir koks buvo metodo rezultatas po vykdymo (src/main/java/com/psk/pyragas/ToolRent/interceptors/MethodLogger.java: Įdėti nuorodą ir eilutės numerį):

```
@AroundInvoke
public Object logMethodInfo(InvocationContext context) throws Exception {
    String logMessage = "";
    logMessage += "--METHOD LOGGER START--" + "\n";
    logMessage += "Time before method call: " + new
Timestamp(System.currentTimeMillis()) + "\n";
    fileWriter.writeToTextFile(logFileLocation, logMessage);
    Object methodResult = context.proceed();
    logMessage = "";
    logMessage += "Time after method call: " + new
Timestamp(System.currentTimeMillis()) + "\n";
    logMessage += "Called method: " + context.getMethod().getName() + "\n";
    logMessage += "Caller class: " + context.getTarget().getClass() + "\n";
    logMessage += "Method result: " + (methodResult == null ? "No Result
(void)" : methodResult.toString()) + "\n";
    logMessage += "--METHOD LOGGER END--" + "\n";
    fileWriter.writeToTextFile(logFileLocation, logMessage);
    return methodResult;
}
```

Extensibility/Glass-box extensibility

Naudojame dekoratorių SummerSaleAdvertisementsDAO, kuris implementuoja IAdvertisementsDAO interfeisą ir užtikrina, kad visi skelbimai turėtų žemesnę nuomos kainą vasarinio išpardavimo metu (src/main/java/com/psk/pyragas/ToolRent/decorators/Summer-SaleAdvertisementsDAO.java: Įdėti nuorodą):

```
@Decorator
public abstract class SummerSaleAdvertisementsDAO implements
IAdvertisementsDAO {

    @Inject
    @Delegate @Any
    private IAdvertisementsDAO advertisementsDAO;
```