

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

**Profesinės darbo praktikos įmonėje EPAM sistemos
ataskaita**

**Report on Professional Work Practice in the Company EPAM
systems**

Profesinės praktikos ataskaita

Atliko:	2 kurso studentas	
	Matas Savickis	(parašas)
Universiteto praktikos vadovas:	Karolis Petrauskas, Doc., Dr.	(parašas)
Įmonės praktikos vadovas:	Nerijus Tenys	(parašas)

TURINYS

1. ĮVADAS	3
1.1. Praktikos vietos pasirinkimo motyvacija.....	3
1.2. Praktikos užduotis.....	3
1.3. Praktikos tikslas	3
1.4. Spręsti praktikos uždaviniai	3
1.5. Praktinės veiklos planas	3
2. ĮMONĖS APIBŪDINIMAS	4
2.1. Įmonės veiklos sritis	4
2.2. Įmonės organizacinė struktūra.....	4
2.3. Įmonės teikiamos paslaugos	4
2.4. Darbo sąlygos	4
3. PRAKTIKOS VEIKLOS APRAŠYMAS	6
3.1. Projektas	6
3.2. Darbo procesas	6
3.3. Naudotos technologijos	6
3.4. Užduotis: Suprojektuoti duomenų bazės schemą.....	7
3.5. Užduotis: Išrinkti Java programavimo karkasą tinkamą debesijos kompiuterijai.	9
3.6. Užduotis: Suprojektuoti duomenų sinchronizavimo architektūrą tarp atskirų duomenų bazių.	11
3.7. Užduotis: Projektui parinkti kokybės užtikrinimo gerąsias praktikas ir jų vykdymo užtikrinimo būdus.	13
3.7.1. JavaDocs validavimas	13
3.7.2. PMD	13
3.7.3. SpotBugs	14
3.7.4. Kodo priklausomybių tikrinimas	14
3.7.5. Checkmarx ir SonarQube	14
3.7.6. Vieneto ir integraciniai testai	14
3.7.7. Kodo formatavimas	14
3.7.8. Integracija su Github Actions	15
3.7.9. Rezultatai	15
4. REZULTATAI, IŠVADOS IR PASIŪLYMAI	16
4.1. Rezultatai	16
4.2. Išvados.....	16
4.3. Privalumai ir trūkumai	16
4.4. Pasiūlymai	16
4.4.1. Pasiūlymai įmonei.....	16
4.4.2. Pasiūlymai universitetui.....	17
LITERATŪRA	18

1. Įvadas

1.1. Praktikos vietos pasirinkimo motyvacija

Pasirinkau atlikti praktiką įmonėje EPAM sistemos, nes norėjau įgauti patirties tarptautinėje įmonėje dirbant su trumpalaikiais klientų užsakymais.

1.2. Praktikos užduotis

- Vykdyti sistemos projektavimo veiklas.
- Vykdyti sistemos kūrimo veiklas.
- Vykdyti projekto apimties įvertinimą ir užduočių analizę.

1.3. Praktikos tikslas

Pritaikyti teorines ir praktines žinias apie programų sistemų kūrimą, įgytas Vilniaus universitete, realiems projektams.

1.4. Spręsti praktikos uždaviniai

- Suprojektuoti duomenų bazės schemą.
- Išrinkti Java programavimo karkasą tinkamą debesijos kompiuterijai.
- Suprojektuoti duomenų sinchronizavimo architektūrą tarp atskirų duomenų bazių.
- Projektui parinkti kokybės užtikrinimo gerąsias praktikas ir jų vykdymo užtikrinimo būdus.

1.5. Praktinės veiklos planas

Praktinė veikla truko nuo 2021-09-01 iki 2021-11-30

1. Įvadas į projektą, naudojamus įrankius ir bendrą įmonės veiklą nuo 2021-09-01 iki 2021-09-10
2. Praktikos užduočių atlikimas nuo 2021-09-11 iki 2019-11-30

2. Įmonės apibūdinimas

2.1. Įmonės veiklos sritis

EPAM sistemos yra Baltarusijos ir JAV informacinių technologijų įmonė, kuri specializuojasi užsakomaisiais projektais ir konsultavimu informacinių technologijų srityje.

2.2. Įmonės organizacinė struktūra

Įmonės padalinys Lietuvoje buvo įsteigtas prieš metus. Šiuo metu įmonėje dirba apie 300 darbuotojų ir šis skaičius vis auga. Didžioji dalis darbuotojų yra atvykusių iš Baltarusijos. Nors įmonė turi nemažai darbuotojų, tačiau tik maža jų dalis būna biure. Biure dažniausiai būna apie 50 žmonių. Darbuotojai norėdami atvykti į biurą turi rezervuoti darbo vietą vidinėje įmonės sistemoje, nes individualių darbo vietų nėra. Mano komandoje dirbo 6 žmonės.

2.3. Įmonės teikiamos paslaugos

1. Klientų programavimo projektų vykdymas: pagrindinė įmonės veiklos sritis yra klientų informacinių technologijų projektų vykdymas. Klientas ateina su prašymu atlikti projektą, EPAM įvertina kiek projektas kainuotų klientui ir pateikia pasiūlymą. Jeigu klientui tinka pasiūlymas, tada nurodžius kaštus EPAM savo vidinėje sistemoje suranda reikiamos kompetencijos darbuotojus projekto atlikimui. Jei reikia, paskirti darbuotojai papildomai turi ir darbo pokalbį pas klientą. Po darbo pokalbio EPAM darbuotojas pradeda dirbti pas klientą. Pats EPAM darbuotojas gali nesutikti dirbti jam siūlomame projekte, jei projektas jo nedomina.
2. Klientų konsultavimas informacinių technologijų klausimais: panašiomis sąlygomis, kaip ir buvo minėta kalbant apie projektų vykdymą, įmonė klientams teikia konsultavimo paslaugas, kai EPAM darbuotojas trumpą laiką atlieką kliento konsultavimą informacinių technologijų projektų klausimais. Konsultantas gali padėti įvertinti projekto apimtį, patarti kokių kompetencijų specialistų reikės, įvertinti reikiamą biudžetą ir panašius su projekto pradžia ar tolimesniu vystymu susijusius klausimus.

2.4. Darbo sąlygos

Įmonė yra įsikūrusi Vilniuje, Šeimyniškių g. 19-601. Pastatas yra patogioje miesto vietoje, yra patogus susisiekimas viešuoju transportu. Kompanija suteikia keletą nemokamo stovėjimo vietų, o jei jos užpildytos netoliese yra nebrangios mokamos stovėjimo vietos. Biure nėra asmeninių darbo vietų, todėl norint atvykti ir dirbti reikia rezervuoti darbo vietą vidinėje įmonės sistemoje. Įmonė leidžia dirbti namuose, net ir šiuo metu, kai karantino sąlygos leidžia dirbti biure. Per beveik metus, kai dirbu šioje įmonėje, biure buvau tik keletą kartų ir jokių nusiskundimų dėl to neišgirdau. Darbuotojams dirbant iš namų yra suteikiama beveik visa reikalinga įranga (nešiojamas kompiuteris, monitorius, pelė ir kita). Pradėjęs dirbti įmonėje mėnesį praleidau vidiniame įmonės darbuotojų

apmokymo procese, kuriame galėjau gilinti savo programavimo žinias bei buvau apmokomas kaip sėkmingiau atlikti kliento darbo pokalbio procesus, kokių klausimų dažniausiai klausiama per darbo pokalbius ir kaip į juos atsakyti teisingai. Po mėnesio man buvo paskirtas projektas, po kurio sėkmingo darbo pokalbio pradėjau dirbti su kliento projektu.

3. Praktikos veiklos aprašymas

3.1. Projektas

Aš dirbau su JAV įmonės Vertex vienkartinio prisijungimo projektu. Įmonė Vertex specializuojasi mokesčių skaičiavimo programinės įrangos kūrimu. Vertex parduodami produktai turi daug modulių į kuriuos galima prisijungti skirtingai būdais, su skirtingais prisijungimo duomenimis. Mūsų projekto tikslas buvo sukurti vienkartinio prisijungimo sistemą (angl. Single Sign-On) apjungiančią visus esamus prisijungimo būdus. Komandoje dirbo šeši žmonės: viena projekto vadovė, keturi programuotojai ir vienas testuotojas.

3.2. Darbo procesas

Darbas šiame projekte vyko pagal Agile metodologiją. Kliento produkto savininkas (angl. Product owner) arba sistemos architektas pateikdavo reikalavimus (angl. Epics) į užduočių valdymo platformą Atlasian. Mūsų komanda iš pateiktų reikalavimų sukurdavo istorijas (angl. Stories), kurias mes įvertindavome ir apskaičiuodavome kiek laiko užtruktų atlikti kiekvieną istoriją. Kas dvi savaites mūsų komanda suplanuodavo sprintą (angl. Sprint) ir įvertindavo kiek užduočių galės atlikti per ateinančias dvi savaites. Po dviejų savaitių vienas komandos narys pristatydavo kokius darbus pavyko atlikti per dviejų savaitių sprintą ir pademonstruodavo progresą kliento projektų vadovei. Prieš kito sprinto planavimą komanda atlikdavo savo vidinę retrospektyvą ir padiskutavo kas buvo gerai ir ką reiktų tobulinti, kad ateities sprintai būtų produktyvesni. Sprinto metu komanda atlieka trumpus penkiolikos minučių susitikimus su klientu ir papasakoja ką kiekvienas komandos narys darė praeitą dieną ir ką planuoja nuveikti kitą dieną. Šio susitikimo metu taip pat išsprendžiamos problemos dėl dabartinio sprinto užduočių. Sprinto metu programuotojai atlikinėja suplanuotas užduotis ir atlikinėja kitų komandos narių kodo peržiūras. Testuotojas sprinto metu atlikinėja esamo funkcionalumo testavimą ir suradęs klaidą sukuria klaidos aprašą ir užduotį Atlasian sistemoje. Rasta klaida kode būna pataisoma esamo sprinto metu, įtraukiama į kito sprinto planavimą arba ignoruojama ir neatliekama. Kas atsitiks su rasta klaida nusprendžia pati komanda arba kliento architektas įvertinęs klaidos svarbumą ir kitas prioritetines užduotis.

3.3. Naudotos technologijos

Projekte buvo naudotas Micronaut 3 karkasas su Java 17 programavimo kalba. Duomenų sluoksniui kurti buvo pasirinkta reliacinė MySQL duomenų bazė ir objektų ryšių suvedimo biblioteka Hibernate. Programa buvo kuriama pagal REST API standartą. Aprašyti REST specifikacijai buvo naudojama OpenAPI specifikacijų kalba. Autorizacijai ir autentikacijai mūsų komanda naudojo Auth0 platformą siekiant užtikrinti saugumą, palengvinti projekto įgyvendinimą ir sumažinti projekto biudžetą. Užtikrinti kodo kokybę mes naudojome įvairius statinio kodo analizės įrankius, tokius kaip: PMD, SpotBug, Checkstyle, Depedency Checker, Pom-lint. Visi šie statiniai kodo analizatoriai padėjo užtikrinti kodo kokybę, tvarką ir saugumą.

3.4. Užduotis: Suprojektuoti duomenų bazės schemą.

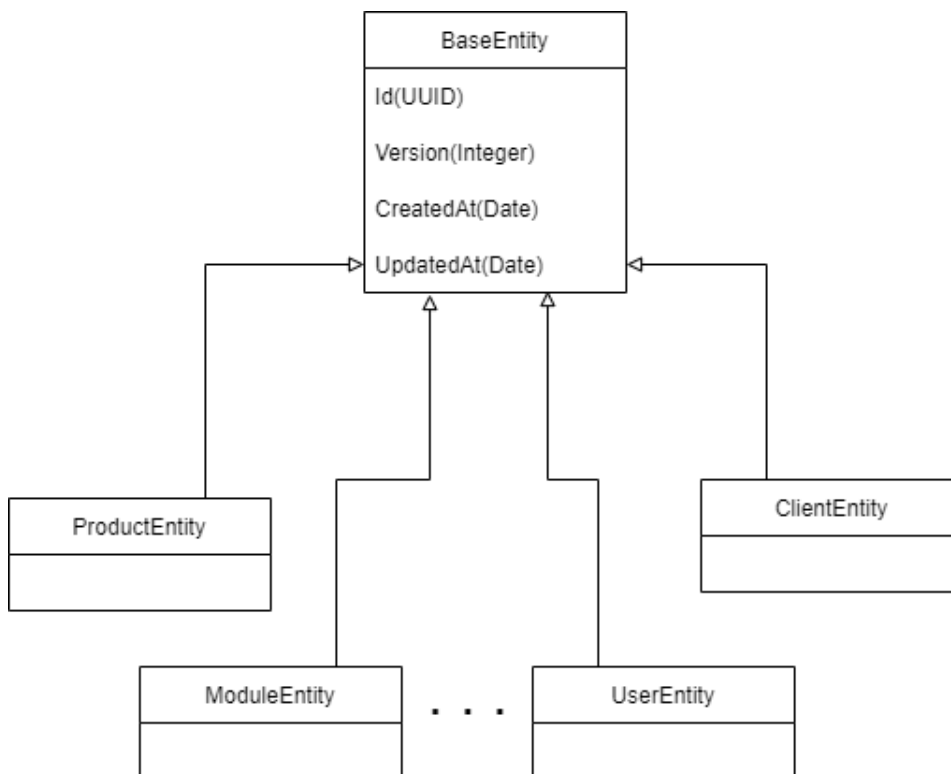
Pradedant projektą klientas pateikė verslo reikalavimus kaip skirtingi mokesčių modeliai sąveikauja tarpusavyje ir kokias reikšmes jie turi. Reikėjo sukurti duomenų bazės schemą, kuri pasižymėtų šiomis savybėmis:

- Esysbė turėtų švelnaus užrakinimo (angl. Soft lock) savybę.
- Kiekviena esybė turėtų unikalų identifikatorių.
- Būtų įmanoma pasakyti kada esybė buvo sukurta.
- Būtų įmanoma pasakyti kada esybė buvo atnaujinta.

Pagal pateiktus reikalavimus buvo sudaryta duomenų bazės schema.

Kiekviena schemas esybė paveldi iš tėvinės BaseEntity esybės kuri turi šiuos laukus

- Id(UUID) - unikalus esybės identifikatorius UUID formatu užtikrinančiu, kad duplikatinio UUID tikimybė yra arti nulio.
- Version(Integer) - inkrementiškai didėjantis skaičius kartu su Hibernate karkasu užtikrinantis švelnaus užrakto (angl. Soft lock) funkcionalumą.
- CreatedAt(Date) - datos formatas parodantis kada esybė buvo sukurta
- UpdatedAt(Date) - datos formatas parodantis kada esybė buvo pakeista paskutinį kartą.



2 pav. Duomenų bazės esybių paveldėjimo diagrama

Projekte buvo naudojama MySQL duomenų bazė pagal kliento prašymą. Atlikti duomenų bazės pakeitimų migracijai buvo naudojama Liquibase migracijų įrankis.

1 pav. Duomenų bazės esybių diagrama

3.5. Užduotis: Išrinkti Java programavimo karkasą tinkamą debesijos kompiuterijai.

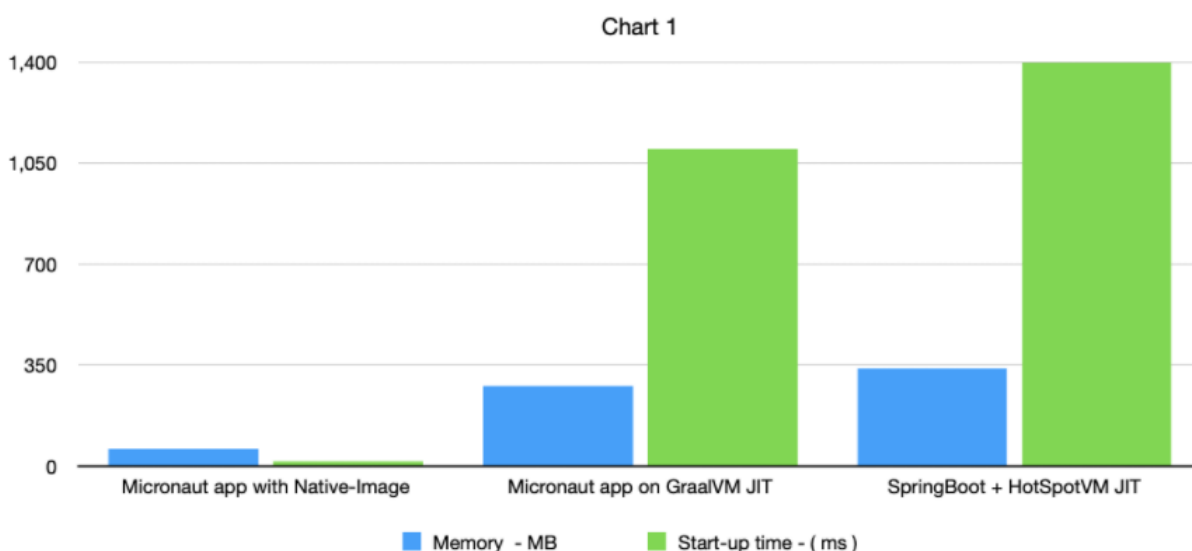
Projektą pradėjome nuo pradžių ir gavome iš kliento prašymą išrinkti Java kalbos karkasą labiau pritaikytą debesijos skaičiavimams. Paprastas sprendimas būtų buvęs pasirinkti šiuo metu populiariausią Spring karkasą ir jį naudoti, tačiau norėjome įsitikinti ar jis yra geriausias pasirinkimas. Atlikus pirminę analizę supratome, kad Spring karkasas turi dvi pagrindines problemas

- Ilgas šalto paleidimo laikas (angl. Cold start).
- Didelis sugeneruotų failų dydis.

Abu šie paminėti aspektai apkrauna debesijos aplinką, dėl jų debesijos paslaugos kainuoja daugiau, ypač jeigu sistemoje yra šimtai tinklo mazgų. Atlikus paiešką internete buvo prieita prie alternatyvų Spring karkasui:

- Micronaut
- Quarkus

Abu šie karkasai pasižymi priešlaikiniu kompiliavimu (angl. Ahead of Time Compilation). Šis kompiliavimo būdas leidžia sukompiliuoti aukšto lygio programinį kodą į vietinį baito kodą (angl. Native byte code). Dėl to vietiniai failai būna mažesnio dydžio nei kompiliuojant tradiciniu pačiu laiku kompiliavimu (angl. Just In Time compilation). Kadangi atliekamas priešlaikinis kompiliavimas, programai nereikia atlikti daug resursų kainuojančių refleksijos operacijų, sulėtinančių programos paleidimą. Vietiniai failai taip pat leidžia naudoti vietinių failų virtualias mašinas tokias kaip GraalVM dėl kurių paleidimo greitis yra sumažinamas.



3 pav. Spring ir Micronaut palyginimas naudojant GraalVM

OpenJDK 14 on 2019 iMac Pro Xeon 8 Core. Winner in Red.

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	??? (3)
Request Per Second -Xmx18m	50k req/sec	46k req/sec	??? (3)
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

(1) Verifier Disabled

(2) Measured with: `ab -k -c 20 -n 100000 http://localhost:8080/hello/John`

(3) Spring WebFlux doesn't seem to support keep alive?

(4) Measured with: `ps x -o rss,vsize,command | grep java`

4 pav. Spring, Micronaut ir Quarkus palyginimas

	COLESICO	MICRONAUT	QUARKUS	SPRING
Compile Time (Sec) > mvn clean compile	1.9	1.7	1.6	1.5
Start Time (Sec) > java -jar app.jar	0.23	0.55	0.6	1.6
Requests Per Second > ab -k -c 20 -n 1000000 http://localhost:8080/hello/John Single thread	154k	93k	70k	18k
Requests Per Second with -Xmx16m > ab -k -c 20 -n 1000000 http://localhost:8080/hello/John Single thread	150k	42k	39k	10k
Requests Per Second with -Xmx64m > wrk -t12 -c400 -d30s --latency http://localhost:8080/hello/John 12 threads and 400 connections	16k (21k NB)	5.8k	6.2k	4.2k
Memory Consumption - Heap Usage (Mb)	70	130	160	95
Memory Consumption - Heap usage with -Xmx16m (Mb)	8.5	9	11	10.5
Jar Size With Dependencies (Mb)	5.7	12	11.5	19
* NB – Non-Blocking processing				

5 pav. Spring, Micronaut, Quarkus ir Colesico palyginimas

Palyginus internete rastus testus [Mic20; Pra20; www20] matuojančius Spring, Micronaut ir Quarkus kompiliavimo laiką, atminties sunaudojimą, pasileidimo greitį ir užklausų apdorojimo greitį matome, kad Micronaut ir Quarkus karkasai skiriasi nežymiai, vienuose testuose laimi Micronaut, kituose Quarkus, tačiau matomas skirtumas tarp šiu dviejų karkasų ir Spring karkaso. Spring karkasas laimi tik kompiliavimo greičio kategorijoje, o šis dalykas produkcijos aplinkai nėra svarbu. Nusprendėme pasirinkti Micronaut karkasą, nes jis savo sintakse yra panašus Spring karkasui, o komanda jau turėjo patirties dirbant su Spring. Jeigu ateityje Spring karkaso kūrėjai įgyvendins priešlaikinį kompiliavimą ir vietinių failų kūrimą, bus nesunku perkelti projektą atgal į Spring.

3.6. Užduotis: Suprojektuoti duomenų sinchronizavimo architektūrą tarp atskirų duomenų bazių.

Kurdami integraciją su Auth0 servisu turėjome užtikrinti, kad duomenys tiek Auth0 duomenų bazėje, tiek kliento duomenų bazėje sutaptų. Tiek bandant įrašyti duomenis į Auth0 duomenų bazę tiek į kliento vidinę duomenų bazę (toliau – Vertex duomenų bazė) gali ištikti sutrikimai, kurių metu duomenys neišsisaugotų ir atsirastų nesutapimas tarp šiu bazių. Kadangi duomenys bandomi įrašyti į dvi skirtingas duomenų bases, neįmanoma visko atlikti per vieną transakciją. Reikia sugalvoti mechanizmą užtikrinantį, kad įvykus sutrikimui duomenys būtų atstatomi sutrikimo metu, o atstatymui nepavykus, pranešti palaikymo komandai, kad duomenys būtų pataisyti rankiniu būdu. Kuriant tokį mechanizmą svarbu įgyvendinti šiuos scenarijus:

```
if(duomenys sėkmingai nuskaityti iš Vertex DB){
    if(duomenys sėkmingai nuskaityti iš Auth0 DB) {
        if(duomenys sėkmingai pakeisti Auth0 DB) {
            if(duomenys sėkmingai pakeisti Vertex DB) {
                baigti darbą;
            } else {
                Parodyti klaidos pranešimą;
                Bandyti atstatyti duomenis Vertex DB;
                Bandyti atstatyti duomenis Auth0 DB;
                Nepavykus atstatyti pranešti DB administratoriui;
            }
        } else {
            Parodyti klaidos pranešimą;
            Bandyti atstatyti duomenis Auth0 DB.
            Nepavykus atstatyti pranešti DB administratoriui.
        }
    } else {
        Parodyti klaidos pranešimą;
    }
} else {
    Parodyti klaidos pranešimą;
}
```

6 pav. Sinchronizavimo mechanizmo pseudo kodas

Kaip matome, kuriant funkcionalumą kodas dažnai tampa labai kompliktuotas, sunkiai skaitomas ir palaikomas. Taip pat atliekant operacijas su skirtingomis klasėmis atsiranda kodo duplikacijos. Šį pseudokodą galima perdaryti į labiau skaitomą ir bendresnį kodą:

```

public <E, A> E execute(SagaLikeExecution<E, A> execution) {
    ExecutionContext<E, A> context = new ExecutionContext<>();

    try {
        return transactionExecutor.call(() -> {
            E existingDatabaseEntity = execution.getDatabasePrepare().prepare();
            context.setExistingDatabaseEntity(existingDatabaseEntity);

            A existingAuth0Entity = auth0RetryHelper.retryManagementApiCall(() ->
                execution.getAuth0SaveState().save(existingDatabaseEntity));
            context.setExistingAuth0Entity(existingAuth0Entity);

            A updatedAuth0Entity = auth0RetryHelper.retryManagementApiCall(() ->
                execution.getAuth0Write().write(existingDatabaseEntity, existingAuth0Entity));
            context.setAuth0WriteOccurred(true);
            context.setUpdatedAuth0Entity(updatedAuth0Entity);

            return execution.getDatabaseWrite().write(existingDatabaseEntity, existingAuth0Entity, updatedAuth0Entity);
        });
    } catch (RuntimeException exception) {
        rollbackAuth0ChangesIfEligible(execution, context, exception);
        throw exception;
    }
}

```

7 pav. Sinchronizavimo mechanizmo įgyvendinimas

Naudojant šį kodą duomenų sinchronizavimas atrodytų taip:

```

@Override
public Module updateModule(Module moduleToUpdate, UUID moduleId) {
    return Objects.requireNonNull(executor.execute(SagaLikeExecution.<>newUpdateExecution()
        .readEntityFromDatabase(() -> {
            // Skaityti duomenis iš Vertex DB
        })
        .saveEntityFromAuth0(existingModule -> {
            // Skaityti duomenis iš Auth0 DB
        })
        .updateEntityInAuth0(existingModule -> {
            // Pakeisti duomenis Auth0 DB
        })
        .updateEntityInDatabase(() -> {
            // Pakeisti duomenis Vertex DB
        })
        .rollbackChangesFromAuth0(existingResourceServer -> {
            // Duomenų atstatymas iš Auth0 DB
        })
        .rollbackChangerFromVertex(
            // Duomenų atstatymas iš Vertex DB
        )
        .getMessageOnOutOfSync(existingModule -> {
            //Pranešimas DB administratoriui
        })
        .build());
}

```

8 pav. Sinchronizavimo mechanizmo panaudojimas

Šiuo metu pademonstruotas kodas yra sėkmingai naudojamas projekte. Sinchronizavimo mechanizmo pasekoje pagerėjo kodo skaitomumas, klaidų valdymas ir duomenų bazių atstatymas iš-tikus klaidai.

3.7. Užduotis: Projektui parinkti kokybės užtikrinimo gerąsias praktikas ir jų vykdymo užtikrinimo būdus.

Komanda nutarė, jog reikia išlaikyti aukštą kodo kokybę, sumažinti kodo klaidas ir saugumo spragas. Norint įgyvendinti šias užduotis reikia surasti reikiamus įrankius padėsiančius užtikrinti užsibrėžtus tikslus bei integruoti juos į programos kūrimo procesą taip, kad būtų kuo sunkiau ap-eiti šiuos įrankius, nes gera programuotojų valia ne visuomet galima pasitikėti. Užduotį pradėjau vykdyti atlikdamas įvairių statinės analizės įrankių paiešką ir įvertinimą. Internetu pavyko rasti nemažai mokamų ir nemokamų priemonių vykdančių kodo analizę.

3.7.1. JavaDocs validavimas

Vienas iš būdų užtikrinti sklandų kodo palaikymą ateityje, kai prie jo dirbs kiti programuotojai, yra kodo dokumentacija. Dažnai projektuose nutinka taip, kad dokumentacija būna daroma atmetinai arba jos išvis nebūna ir kitai komandai perėmus projektą prireikia daug laiko ir pastangų toliau ją vystyti. Dėl šių priežasčių komanda nutarė, kad dokumentacijos rašymas turi būti integruotas į programų kūrimo procesą. Vienas iš būdų Java kode rašyti dokumentaciją yra naudoti Javadocs dokumentų generatorių, kurio pagalba atitinkami Java kodo komentarai būtų sugeneruoti į dokumentaciją. Užtikrinti, kad JavaDocs komentarai būtų rašomi kiekvienam viešam klasės metodu ir kontraktui naudotame maven-javadocs-plugin biblioteką, kuri užtikrina, kad visi komentarai būtų rašomi tinkamu formatu, visų reikiamų metodų ir kontraktų parametrai būtų nurodyti ir būtų aprašyta ką daro kiekvienas iš metodų. Jeigu vienas iš nurodytų kriterijų nėra įgyvendintas maven-javadocs-plugin parodo klaidą ir pasako ko trūksta, kad dokumentacija būtų teisinga.

```
/**
 * Gets the geocoordinates of roadrunners based on your city and state.
 *
 * @param city the city you want to browse for roadrunners
 * @param state the state you want to browse for roadrunners
 * @return the coordinates of the roadrunner in your area
 * @throws IOException if you put integers instead of strings
 */
public String findRoadRunner(String city, String state) throws IOException {
    System.out.println("location: " + city + ", " + state);
    System.out.println("getting geocoordinates of roadrunner.... ");
    System.out.println("roadrunner located at " + LongLat);
    return LongLat;
}
```

9 pav. JavaDocs pavyzdys

3.7.2. PMD

Gero kodo požymis yra kai jame laikomasi tvarkos: nėra nepanaudotų kintamųjų ar nenaudojamų bibliotekų, kintamųjų pavadinimai turi prasmę ir laikomasi jų pavadinimo standartų, kode nėra magiškų skaičių (angl. Magic number). Dažnai tokius dalykus galima pastebėti kodo peržiūros stadijoje, tačiau visi žmonės klysta ir kartais praleidžia šiuos dalykus pro pirštus. Taip pat, jeigu

būtų įrankis galintis patikrinti šiuos dalykus kodo peržiūros užimtų mažiau laiko. Vienas iš atviro kodo įrankių plačiai naudojamas tokio tipo statinei kodo analizei yra PMD. Jo pagalba galima patikrinti didelę aibę kodo stiliaus klaidų, gerųjų praktikų trūkumų bei daug kitų panašių dalykų. PMD turi savų trūkumų, kartais yra rodomos klaidos kurių negalima ištaisyti, pavyzdžiui, rodomos klaidos naudojant Lombok generuojamą kodą. Išspręsti tokius trūkumus padeda lanksti PMD konfigūracija, kurios pagalba galima išjungti norimas kodo analizės taisykles visam projektui arba tik vienam failui.

3.7.3. SpotBugs

SpotBugs yra statinės kodo analizės įrankis padedantis surasti kodo klaidas, tokias kaip neinicializuoti laukai, laukai turintys null reikšmę, kai kode anotacijos null reikšmės neleidžia, sąlygos sakiniai visuomet grąžinantys tą pačią reikšmę bei kitas klaidas, kurios sintaksiškai yra teisingos, bet gali sukelti programos sutrikimus ateityje.

3.7.4. Kodo priklausomybių tikrinimas

Siekiant užtikrinti, kad projekto kodas būtų saugus svarbu užtikrinti, kad kode naudojamos priklausomybės būtų saugios. Šiam tikslui naudojome dependency-check-maven ir maven-enforcer-plugin bibliotekas. Jos užtikrino, kad naudojamos kodo priklausomybių versijos neturi žinomų saugumo spragų ir priklausomybių versijos nesiduplikuoja arba nėra nurodytos dvi skirtingos versijos tai pačiai priklausomybei.

3.7.5. Checkmarx ir SonarQube

Kliento prašymu turėjome naudoti Checkmarx saugumo statinį analizatorių ir SonarQube statinį kodo analizatorių. Kai kurie dalykai analizuojami šių įrankių jau buvo daromi aukščiau minėtuose įrankiuose, bet buvo nutarta palikti naudojamus įrankius, nes jų vykdymo laikas yra trumpesnis ir perteklinė analizė nėra blogai.

3.7.6. Vieneto ir integraciniai testai

Dar vienas svarbus kodo kokybės aspektas yra testai. Nutarėme, kad projekto išeities kodas turi būti 80 procentų padengtas testais. Tą užtikrinti naudojome Jacoco biblioteką išreiškiančią procentais projekto kodo padengimą testais. Jacoco biblioteka skaičiuoja kiek Java baitų kodo eilučių pasiekia testai.

3.7.7. Kodo formatavimas

Paskutinis aspektas kurį norėjome užtikrinti buvo kodo stiliaus pastovumas. Jeigu kodas visame projekte atrodo panašiai, jį tampa lengviau skaityti ir suprasti. Šiam tikslui pasirinkome plačiai naudojama Google Java style konvenciją, kurią tikrinome pagal Google suteiktą stiliaus tikrinimo įrankį checkStyle.

3.7.8. Integracija su Github Actions

Supratome, kad reikia priversti programuotojus naudotis statinio kodo analizės įrankiais, nes kitu atveju niekas jų nenaudos. Buvo nutarta šiuos įrankius sukonfigūruoti Github Action aplinkoje, kuri užtikrina, kad joks kodas negalėtų patekti į kodo bazę prieš tai nepraėjęs visų minėtų įrankių. Visi šie įrankiai kartu su Github Actions integracija yra toliau sėkmingai naudojami komandoje.

3.7.9. Rezultatai

Pradėjus naudoti visus aukščiau minėtus įrankius kilo keblumų, įrankiai rodė kodo klaidas ten kur jų nebuvo, kai kurių kodo klaidų nebuvo įmanoma pataisyti, nes jos kilo iš generuojamo kodo (pavyzdžiui naudojant Lombok biblioteką). Tačiau skiriant tam laiko, konfigūracija buvo pritaikyta komandos poreikiams ir kodas atrodo labai švariai ir tvarkingai. Sėkmingai surandamos kodo klaidos ir saugumo spragos.

4. Rezultatai, išvados ir pasiūlymai

4.1. Rezultatai

- Suprojektuota ir įgyvendinta duomenų bazės schema.
- Įvertinti Java kalbos karkasai ir išrinktas karkasas leidžiantis greitesnį šaltą paleidimą (angl. Cold start) ir mažesnius failo dydžius.
- Sėkmingai suprojektuotas ir įgyvendintas duomenų sinchronizavimas tarp dviejų duomenų bazių.
- Išrinktos ir integruotos kodo analizės priemonės pagerinančios kodo kokybę ir projekto saugumą.
- Pagilinti darbo komandoje įgūdžiai.
- Susipažinta su projektinių įmonių darbo praktika.

4.2. Išvados

Praktika įvyko sėkmingai. Jos metu išmokau naujų technologijų bei pagilinau žinias tose technologijose kurias jau mokėjau. Praktikos užduotys buvo įgyvendintos sėkmingai. Sėkmingai pritaikiau teorines ir praktines žinias įgytas universitete. Darbe aprašytos užduotys buvo tik dalis per praktika nuveikto darbo ir žinių, kurias įgavau. Įgytas praktikos žinias ir toliau taikysiu savo darbe ir kituose ateities projektuose.

4.3. Privalumai ir trūkumai

Privalumai: praplėčiau savo žinias apie REST tipo projektus, modernius programavimo karkasus, statinius kodo analizatorius ir gerasias programavimo praktikas. Turėjau galimybę programuoti naujausia Java kalbos versija, o tai – retas malonumas.

Trūkumai: projekto metu vykdavo labai daug susitikimų su klientu, kurie atitraukdavo dėmesį nuo darbo. Informacija iš visų šių susitikimų galėdavo būti perduodama išsiuntus elektroninį laišką vietoje pačio susitikimo.

4.4. Pasiūlymai

4.4.1. Pasiūlymai įmonei

Siūlyčiau sumažinti susitikimų trukmę ir dažnumą. Būna tokių dienų kai visas dienos darbas būna dalyvauti susitikimuose, kurie net nėra susiję su mūsų projekto darbu. Net jeigu susitikimai būna trumpi, jie atitraukia dėmesį nuo užduočių ir reikia laiko vėl susikaupti po susitikimo, o tai mažina dienos produktyvumą.

4.4.2. Pasiūlymai universitetui

Manau profesinė praktika magistro studentams yra nenaudingas dalykas kuris neturi daug prasmės, per praktikos laikotarpį esi priverstas likti toje pačioje įmonėje net jeigu ir gauni geresnių pasiūlymų dirbti kitur, apriboja karjeros galimybes. Manau visi studijuojantys magistro studijuose jau turi darbus ir savo užduotis pritaiko prie to, ką jau darė darbe ko pasekoje praktika tampa tiesiog aprašyti ką veikia įmonėje ir nemanau kad kažkam tai laibai įdomu. Mano nuomone darbo praktika turėtų būti pasirenkamas dalykas.

Literatūra

- [Mic20] Micronaut.io. Micronaut vs Quarkus vs Spring Boot performance on JDK 14. (<https://micronaut.io/2020/04/07/micronaut-vs-quarkus-vs-spring-boot-performance-on-jdk-14/>), 2020.
- [Pra20] Pratik Prakash. Microservice and Serverless With Micronaut + GraalVM. (<https://dzone.com/articles/micronaut-build-for-next-wave-of-microservices-amp>), 2020.
- [www20] [www.github.com/colesico](https://github.com/colesico). Colesico-Framework 3.7 vs Micronaut 2.0 vs Quarkus 1.5 vs Spring Boot 2.3 Benchmark - 2020. (<https://github.com/colesico/java-frameworks-comparison>), 2020.