TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Maria Kert 143057IAPB

LAIENDATAVA ARHITEKTUURIGA HINDAMISKESKKOND PROGRAMMEERIMISAINETELE

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Maria Kert

22.05.2017

Annotatsioon

Käesoleva lõputöö “Laiendatava arhitektuuriga hindamiskeskkond programmeerimis-ainetele” eesmärgiks on arendada ainele „Objektorienteeritud programmeerimine keeles Java“ mugav hindamiskeskkond, automatiseerides taustprotsesse ja vähendades sellega drastiliselt hindamisele kuluvat aega. See aitaks oluliselt lihtsustada õppejõu tööd ning tõhustada tudengitele tagasiside andmist.

Töö tulemusena valmis Spring Booti serverirakendus, mille osadeks on andmebaas, skriptid, kontrollerid, teenused ja andmebaasiliidesed. Serveri kihid võimaldavad luua uusi andmebaasiobjekte ning tagavad sujuva suhtluse kavandatava klientrakendusega. Serveriosas on kirjeldatud kõik meetodid, mida klientrakendusel serveriga ühendumiseks vaja läheb.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 5 peatükki, 12 joonist.

Abstract

Grading Tool with an Extensible Architecture for Programming Subjects

The primary goal of the thesis „Grading Tool with an Extensible Architecture for Programming Subjects“ is developing a convenient grading tool for the subject „Object-oriented Programming in Java“ by automating background processes which drastically decreases the time spent on grading students’ assignments. This tool simplifies the professor’s job substantially and makes giving feedback to students more efficient.

The result of this thesis is the server side of the application made using Spring Boot. The server side consists of a database, scripts, controllers, services and database interfaces. Server side components are used to create database objects and ensure smooth interaction with the planned client application. The server side includes every method the client application will need to connect to the server.

The thesis is in Estonian and contains 32 pages of text, 5 chapters, 12 figures.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
Gist	GitHubi koodilõikude salvestamise teenus, salvestatud dokument
GNU	*GNU's Not Unix!* GNU ei ole Unix!
IDE	*Integrated development environment*, integreeritud arenduskeskkond
JPA	*Java Persistence API*
REST	*Representational state transfer*
SQL	*Structured query language*, struktuurpäringukeel
URL	*Uniform Resource Locator*, internetiaadress ehk universaalne ressursilokaator
YAML	*YAML Ain't Markup Language*, YAML pole märgistuskeel

Sisukord

Jooniste loetelu

1 Sissejuhatus

Siiamaani on olnud „Objektorienteeritud programmeerimine keeles Java“ aine hindamissüsteem üsna manuaalne ning ebamugav, muutes sellega hindamise protsessi väga ajakulukaks. Kuna aine esmane idee pole kontrollida atomaarseid programmeerimisoskusi, vaid pigem tarkvara disaini ja arhitektuuri tundmist, siis automaattestimisest antud kursuse raames väga suurt kasu ei ole.

2016. aasta sügissemestril arendati ainele keskkond, kus saab tudengite kodu- ja kontrolltöid lihtsa vaevaga üle vaadata ning neile tagasisidet anda. Küll aga oli selle süsteemi peamiseks puudujäägiks, et koodi ei olnud võimalik manuaalse tööta käivitada ning keskkonnas ei kuvanud süntaktilisi vigu. Samuti ei saanud koodis mugavalt navigeerida nagu IDE-ga töötades.

Käesoleva lõputöö eesmärgiks on arendada ainele mugav hindamiskeskkond, automatiseerides taustprotsesse ja vähendades sellega drastiliselt hindamisele kuluvat aega. See aitaks oluliselt lihtsustada õppejõu tööd ning tõhustada tudengitele tagasiside andmist.

Selle töö tulemusel valmib serverirakendus, milles on olemas kõik funktsionaalsus, mida tulevases hindamissüsteemis vaja läheb. Rakendus koosneb kontrolleritest, teenustest ning andmebaasiliidestest, mis suhtlevad andmebaasiga; skriptidest, mille abil tehakse failisüsteemis muudatusi, ja andmebaasist.

Kontrollerid saavad sisendi REST-i kaudu ja suunavad need edasi teenustele. Teenustes on kogu rakenduse loogika ning neis toimub suhtlus andmebaasiliideste ja skriptidega. Kõik teenused, mille lahendusi oleks kunagi tarvis muuta, realiseeritakse läbi liideste, mis muudab nende välja vahetamise oluliselt lihtsamaks.

2 Ülevaade kasutatud tehnoloogiatest

Antud lõputöö raames kasutati rakenduse loomiseks Spring Booti [1] ja Javat [2], ehitamiseks *Gradle'it* [3] ning andmebaasiks valiti MariaDB [4]. Muudatused failisüsteemis tehti *Bash* skriptidega [5].

Sellised tehnoloogiad said valitud kombinatsioonina autori isiklikest eelistustest tarkvara arendamisel ning lõputöö juhendaja ja aine õppejõuga konsulteerimisel leitud lahendustest.

2.1 Spring Boot

Antud töös on kasutatud Spring Boot versiooni 1.5.3 ning rakenduse ehitamiseks Gradle'it. Spring Boot muudab eraldiseisva Springil [6] põhineva rakenduse loomise lihtsaks [1]. Üle jääb vaid seda jooksutada.

Springi MVC projektide konfigureerimine võib muutuda väga keerukaks ja ajakulukaks, kuna tihti on tarvis otsida vastavale Springi versioonile kokkusobivaid teeke [7]. Samuti muutub sätestamine tüütuks ning üksluiseks, kuna projektide ehitamiseks on tihti vaja taaskasutada samu teeke ja komponente. Selle lahendamiseks lõi Springi tiim Spring Booti [7].

Suurem osa Spring Boot rakendusi vajab väga vähe Springi konfigureerimist. Seda seetõttu, et rakenduse konfiguratsiooni saab määrata Spring Initializr'iga [8], kus saab valida Spring Boot versiooni, milliseid tehnoloogiaid ja teeke arendamiseks soovitakse ning mida kasutada rakenduse ehitamiseks.

Spring Boot on ideaalne veebirakenduse loomiseks, kuna võimaldab lihtsalt tarbida REST [9] teenuseid.

2.2 MariaDB

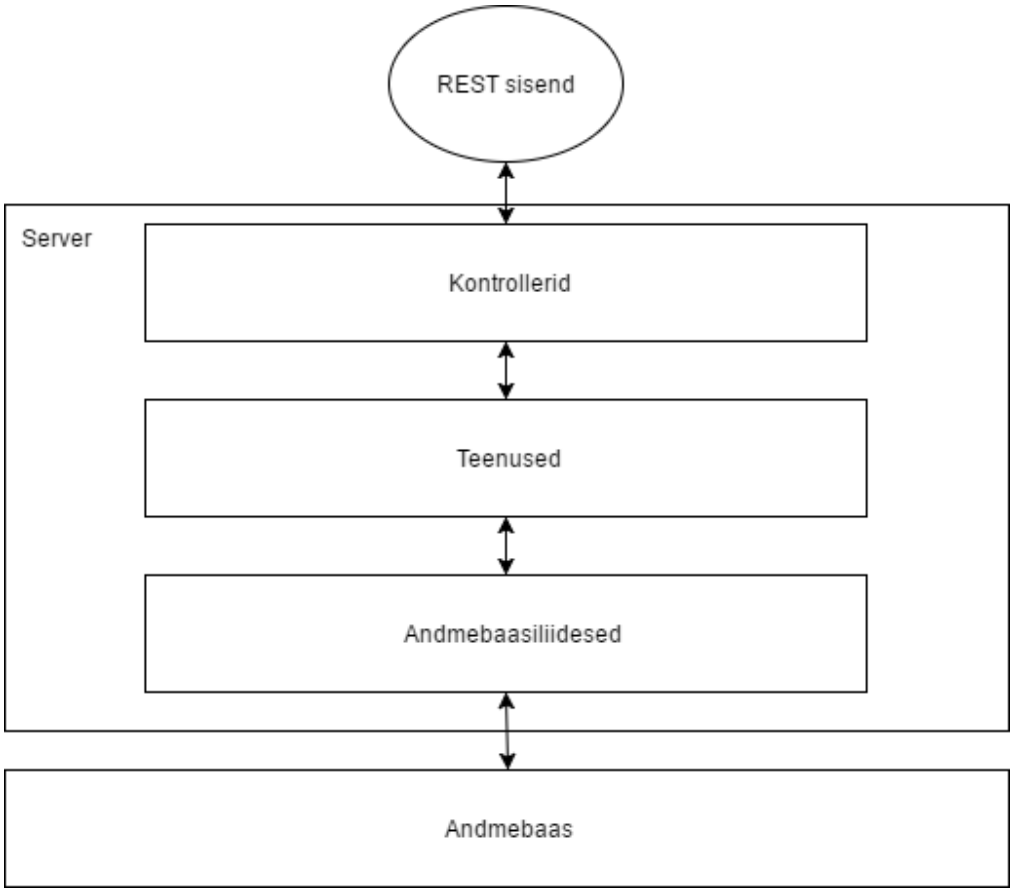
MariaDB [4] Server on üks populaarsemaid avatud lähtekoodiga andmebaasiservereid maailmas, mille lõid MySQL [10] arendajad. MariaDB on kiire, skaleeritav ning robustne, erinevad lisad ja tööriistad muudavad andmebaasi kiireks ning mitmekülgselt kasutatavaks [4]. MariaDB on relatsiooniline andmebaas ning kasutab andmete pärimiseks ja sisestamiseks SQL-i [4].

2.3 Käsurida

Käsurida on interaktiivne tööriist, mis võimaldab kasutajatel programme käivitada, failisüsteemi ning süsteemis parasjagu jooksvaid protsesse hallata [11]. Käsureal on sisseehitatud käsud, mis võimaldavad operatsioonisüsteemi toimimist kontrollida [11]. Sisseehitatud käskudest saab moodustada jooksutatavaid programme, mida kutsutakse skriptideks [11]. Antud rakenduses on kasutusel käsurea kasutajaliides *Bash* [12], täpsemalt *Windowsi* operatsioonisüsteemile mugandatud *Bash on Ubuntu on Windows* [13]. Kuna serverirakendus ehitatakse üles Linuxis käivitamiseks, on ülimalt oluline, et skriptid töötaks just Linuxi-põhises süsteemis.

3 Arhitektuur

Rakenduse ülesehitust planeerides oli kõige olulisemaks aspektiks arhitektuuri dünaamilisus. Sooviks oli rakenduses muudatuste sisse viimisel programmikoodi võimalikult vähe muuta. Sellepärast said kõik väliseid API-sid kasutavad teenused realiseeritud liideste abil. See teeb tulevikus selliste teenuste muutmise ja välja vahetamise väga lihtsaks.



Joonis 1. Planeeritud arhitektuuri skeem

3.1 Server

Rakenduse server loodi Spring Booti kasutades. Spring on väga spetsiifiliselt defineerinud kihid, mis aitavad rakenduse arhitektuuri struktureerituna ja arusaadavana hoida.

3.1.1 Kontroller

Kontroller võtab vastu REST päringud ning edastab need teenusele. Kontroller märgitakse *@Controller* annotatsiooniga ning rakenduse käivitudes kaardistab Spring selle vastavalt.

Kõik meetodid, mis töötlevad REST päringuid, tuleb märgistada *@RequestMapping* annotatsiooniga, millele saab anda parameetreid. Kõige olulisem parameeter on *value*, kus määratakse meetodi käivitamise pöördustee. Samuti tuleb defineerida, mis tüüpi päringuga on tegu. Antud lõputöös on kasutusel vaid *GET* ja *POST* tüüpi päringud. Spring ei suuda kõiki objekte veebisaidi jaoks arusaadavale kujule viia. Selle saavutamiseks tähistatakse tagastustüüp annotatsiooniga *@ResponseBody*.

Meetoditele on võimalik anda ka argumente, mis tuleb samuti märkida annotatsiooniga. Käesolevas rakenduses on kasutusel kaht tüüpi argumente: URL-ist tulev argument tähistusega *@PathVariable*, mida kasutatakse põhiliselt *GET*-tüüpi päringutes ning päringu andmetest tulev argument tähistusega *@RequestBody*, mida kasutatakse *POST*-tüüpi päringutes.

```
@Controller
public class CentralGitHookController {

    private CentralGitHookService centralService;

    public CentralGitHookController(CentralGitHookService centralService) {
        this.centralService = centralService;
    }

    @RequestMapping(value="/central/{uniid}/{subjectCode}", method=RequestMethod.GET)
    public @ResponseBody String init(@PathVariable String uniid, @PathVariable String subjectCode) {
        centralService.init(uniid, subjectCode);
        return "done";
    }

}
```

Joonis 2. Kontrolleri koodinäide

3.1.2 Teenus

Teenus võtab kontrollerilt vastu päringud ning edastab need andmebaasiliidesele. Teenuse klassis toimub kogu rakenduse loogika. Teenus tähistatakse *@Service* annotatsiooniga, et Spring saaks seda vastavalt kaardistada.

```java
@Service
public class SubmissionService {

    private SubmissionRepository submissionRepository;

    public SubmissionService(SubmissionRepository repository) {
        submissionRepository = repository;
    }

    public Submission save(Submission submission) {
        return submissionRepository.save(submission);
    }

}
```

Joonis 3. Teenuse koodinäide

3.1.3 Andmebaasiliides

Andmebaasiliides ühendub andmebaasiga ning edastab sellele teenuselt saadud päringud. Andmebaasiliidese kaudu tehakse andmebaasis muudatusi ning küsitakse sealt objekte. Liides ei vaja eraldi annotatsiooni. Käesoleva rakenduse liidesed on realiseeritud *CrudRepository* liidest laiendades. *CrudRepository* pakub sisseehitatud meetodeid andmebaasiga suhtlemiseks, mis võimaldavad andmeobjekte salvestada, pärida ja eemaldada, nende arvu ja olemasolu kontrollida. Liideses saab defineerida ka enda meetodeid, mis tuleb sõnastada täpselt etteantud konventsiooni järgi.

```java
public interface TaskRepository extends CrudRepository<Task, Long> {

    Task findByName(String name);

}
```

Joonis 4. Andmebaasiliidese näide

Joonise meetod *findByName* SQL-kujul näeks välja järgmine:

```
SELECT * FROM Task WHERE Task.name = name;
```

3.1.4 Andmebaasiobjekt

Andmebaasiobjekt on objekt, mille klass on tähistatud *@Entity* annotatsiooniga. Tänu sellele näeb Spring, et antud klass ja selle väljad tuleb teisendada andmebaasile arusaadavale kujule. Käesolevas lõputöös on kasutusel Lombok [14] *plugin*, mis loob automaatselt väljadele *getter*- ja *setter*-meetodid. Selleks tuleb vaid klass märkida annotatsioonidega *@Getter* ja *@Setter*.

Igal andmebaasiobjektil peab olema tähistatud id väli annotatsiooniga *@Id*. Kui tegemist on numbrilise identifikaatoriga, mida soovitakse automaatselt genereerida, märgitakse väli lisaks *@GeneratedValue* annotatsiooniga.

Kui andmebaasiobjekt on relatsioonilises seoses teise andmebaasiobjektiga, tuleb ka see tähistada vastava annotatsiooniga. Antud rakenduses on kasutusel kolm: *@OneToOne*, *@OneToMany* ja *@ManyToOne*.

```
@Entity
@Getter
@Setter
public class Task {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;

    private String name;

    private String subjectCode;

    @OneToMany(cascade=CascadeType.ALL)
    @JsonIgnore
    private List<StudentTask> studentTasks;

    @OneToOne(cascade=CascadeType.ALL)
    private Plagiarism plagiarism;

}
```

Joonis 5. Andmebaasiobjekti näide

Annotatsiooni parameeter *cascade=CascadeType.ALL* tähendab seda, et andmebaasi-objektid salvestatakse enne seda, kui neid mõne teise andmebaasiobjektiga seostada üritatakse. *@JSONIgnore* on vajalik selleks, et REST vastuses ei tagastataks objekti ennast, vältimaks lõpmatu tsükli tekkimist.

3.2 Andmebaas

Andmebaasi tabelite realiseerimiseks on kasutusel JPA (*Java Persistence API*), mistõttu SQL skeeme ja päringuid programmikoodis ise otse ei defineerita. JPA kasutuselevõtt tähendab andmebaasis väga minimaalset seadistamist, mis teostatakse serveri konfiguratsioonifailis. Andmebaas suhtleb serveri andmebaasiliidestega, millelt saab päringuid objektide küsimiseks, lisamiseks või muutmiseks ning millele tagastab päringute tulemused.

3.3 Liidestamine

„Liides on punkt, kus kaks süsteemi kokku saavad ja suhtlevad“ [15]. Liidestes kirjeldatakse ära meetodid, mis vajavad realiseerimist, kui seda kasutada soovitakse. Klass saab liidest kasutada seda implementeerides.

Käesoleva lõputöö kõik teenused, mis kasutavad mingisugust välist API-t, on realiseeritud läbi liideste. Sellega garanteeritakse, et vajadusel saab teenuse API hõlpsasti teise API vastu vahetada, tehes programmikoodis muudatuste sisse viimise lihtsaks ja mugavaks.

4 Realisatsioon

4.1 Spring Boot

4.1.1 Konfiguratsioon

Rakenduse piires on vajalik hoiustada erinevaid staatilisi väärtusi, nagu näiteks failide asukohti failisüsteemis. Nende väärtuste hoiustamiseks kasutatakse konfiguratsioonifaili. Fail on tavaline YAML (*.yml*) fail, kus on ära määratud projekti malli, repositooriumide, *zip*- ja räsifailide ning plagiaadikontrolli kaustade ja skriptide asukohad failisüsteemis, *MailGun* teenuse API võti, GitHubi konto kasutaja, parool ja Gisti lingi mall, plagiaadikontrolli teenuse MOSS [16] kasutaja ning andmebaasiühenduse sätestus. Faili on võimalik vaid käsitsi muuta. Kõik teenused, millel on vaja teada neid väärtusi, saavad need konfiguratsioonifailist, kasutades *@Value* annotatsiooni, mille argumendiks on failis märgitud tee.

```
@Value("${paths.files.repos}")
private String repoPath;

@Value("${github.user}")
private String user;

@Value("${github.pass}")
private String pass;
```

Joonis 6. *@Value* annotatsiooni kasutamise näide

```
paths:
   files:
      hash: "/mnt/d/Users/mammu/workspace/loputoo/hashes/"
      plagiarism: "/mnt/d/Users/mammu/workspace/loputoo/plagiarism/"
      repos: "/mnt/d/Users/mammu/workspace/loputoo/repod/"
      zips: "/mnt/d/Users/mammu/workspace/loputoo/zips/"
      embeddabl: "/mnt/d/Users/mammu/workspace/loputoo/embeddabl/"
```

Joonis 7. Konfiguratsioonifaili andmepuu näide

4.1.2 Abiklassid

FileHandler

FileHandler klass on abiklass, mis tegeleb failist lugemise ja sinna kirjutamisega. Klassil on üks privaatne väli *File startFolder*, mis väärtustatakse *set*-meetodis. *FileHandler*

klassil on kuus avalikku ja neli privaatset meetodit. Objekt on kasutuses viies teenuses: *GitService, EclipseService, MossService, GitHubService* ja *EmbeddablService*.

Meetodite kirjeldused:

- **public String getMainPath(String path)** – tagastab antud kaustast tee selle klassini, kus asub *main* meetod. Selle puudumisel tagastab *nulli*. Kasutab tagastamiseks privaatset meetodit *getMainPath*.

- **public List<File> getAllFiles(String path)** – tagastab kõik selles kaustas olevad failid listina. Kui kaustas pole ühtki faili, tagastab tühja listi. Kasutab tagastamiseks privaatset meetodit *getAllFiles*.

- **public String read(String filePath)** – loeb faili kõik read ning tagastab faili sisu sõnena. Kui faili ei leitud või fail on tühi, tagastatakse *null*.

- **public List<String> readAllLines(String filePath)** – loeb faili kõik read listi ning tagastab selle. Vigade korral tagastatakse *null*.

- **public String getPackagePath(String path)** – tagastab argumendist etteantud projekti selle paketi, kus asub *main* meetod. Kui *main* meetodit ei leita, tagastatakse *null*. Kui failid asuvad *src* paketis, tagastatakse tühi *String*.

- **public boolean writeLines(List<String> lines, String path)** – kirjutab listis olevad read argumendina saadud faili. Kui kirjutamine õnnestub, tagastatakse *true*, muul juhul tagastatakse *false*.

- **private void setPath(String path)** – moodustab argumendist *File* tüüpi objekti ja salvestab selle klassi väljale.

- **private List<File> getAllFiles(File folder, List<File> files)** – tagastab kõik selles kaustas olevad failid listina. Kui kaustas pole ühtki faili, tagastab tühja listi.

- **private String getMainPath(File folder)** – leiab antud kaustast tee selle klassini, kus asub *main* meetod. Selle puudumisel tagastab *nulli*.

- **private boolean isMain(String path)** – kontrollib, kas antud fail sisaldab *main* meetodit. Selleks loeb meetod kõik faili read ning filtreerib need *“public static*

void main(String[] args)” olemasolu alusel. Kui *main* meetod eksisterib, tagastatakse *true*, vastasel juhul *false*.

ScriptRunner

ScriptRunner on abiklass skriptide jooksutamiseks ja nende väljundi lugemiseks. Klassil on üks avalik meetod skriptide käivitamiseks. *ScriptRunner* klassi kasutab 4 teenust: *GitService, EclipseService, MossService* ja *EmbeddablService.*

Meetodite kirjeldus:

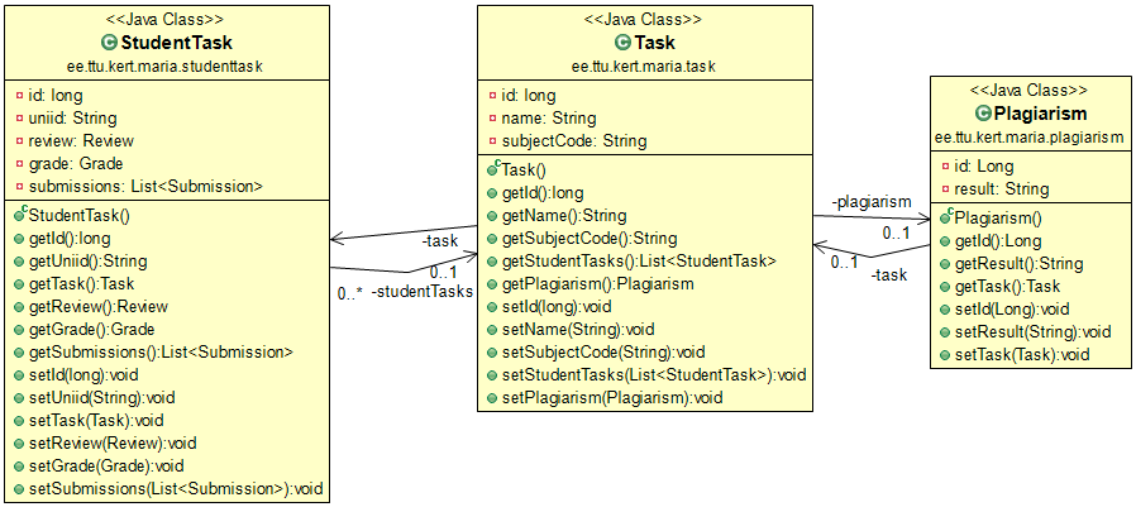
- **public String run(String[] command)** – käivitab argumendina sisestatud käsu ning tagastab selle väljundi. Vigade ilmnemisel tagastatakse *null*. Kui skripti jooksutamisel tekib viga, prindib meetod lisaks konsooli ka veateate.

4.1.3 Ülesanded ja nende jaotus

Selleks, et paremini mõista, kuidas rakendus toimib, tuleb lahti seletada andmebaasi põhiobjektid ning nende jaotuse. Ülesannete tüübid on rakenduses jaotatud kolmeks: *Task* (edaspidi ülesanne), *StudentTask* (harjutus) ja *Submission* (esitus).

Task

Task on ülesande ülemtüüp, mis on otseselt seotud ainega, kus ülesanne on deklareeritud. Ülesandel on väljad *Long id*, *String name*, *String subjectCode*, *Plagiarism plagiarism* ning list sellega seotud *StudentTaskidest*. Iga ülesandega on seotud täpselt üks plagiaadikontroll, mis käib läbi kõik objektiga seotud harjutused ning võrdleb nende sarnasust.



Joonis 8. Ülesande klassidiagramm

TaskController

Kontroller, mis võtab vastu REST päringuid ülesande objektide kohta. Kontrolleril on üks privaatne väli *TaskService taskService*, üks avalik konstruktor ning kaks avalikku meetodit.

Meetodite kirjeldus:

- **public TaskController(TaskService taskService)** – klassi konstruktor, kus Spring initsialiseerib automaatselt *taskService* välja.
- **public List<StudentTask> getStudentTasks(String taskName)** – meetod, mis tagastab kõik antud ülesandega seotud harjutused listina. Meetod kutsutakse välja REST teenuse kaudu ning päringu tüüp on *GET*. Argument *taskName* initsialiseeritakse päringu URL-i parameetrina (*/task/taskName*).
- **public Iterable<Task> getAllTasks()** – meetod, mis tagastab kõik ülesanded itereeritava objektina. Meetod kutsutakse välja REST teenuse kaudu ning päringu tüüp on *GET*.

Koodinäide:

```
@RequestMapping(value="/task/{taskName}", method=RequestMethod.GET)
public @ResponseBody List<StudentTask> getStudentTasks(@PathVariable String
taskName) {
	Task task = taskService.getByName(taskName);
	if (task == null) return new ArrayList<>();
	return task.getStudentTasks();
}
```

TaskService

Teenus, mis tegeleb kontrollerilt saadud päringute edastamisega andmebaasi. Klassil on üks privaatne väli *TaskRepository taskRepository*, üks avalik konstruktor ning neli avalikku meetodit. Andmete edastamine toimub *taskRepository* välja kaudu.

Meetodite kirjeldus:

- **public Task(TaskRepository taskRepository)** – klassi konstruktor, mille abil Spring initsialiseerib automaatselt *taskRepository* välja.
- **List<StudentTask> getStudentTasks(String taskName)** – meetod, mis leiab andmebaasist vastava nimega ülesande ja tagastab kõik sellega seotud harjutused. Kui sellise nimega ülesannet ei eksisteeri, tagastatakse tühi list.
- **public Task getByName(String name)** – meetod, mis võimaldab nime järgi andmebaasist *Task* tüüpi objekte otsida. Kui objekti andmebaasis ei leidu, tagastatakse *null*.
- **public Iterable<Task> getAll()** – tagastab itereeritava objekti kõikidest andmebaasis olevatest *Task* objektidest.
- **public Task save(Task task)** – meetod, mis salvestab argumendina antud *Task* objekti andmebaasi.

TaskRepository

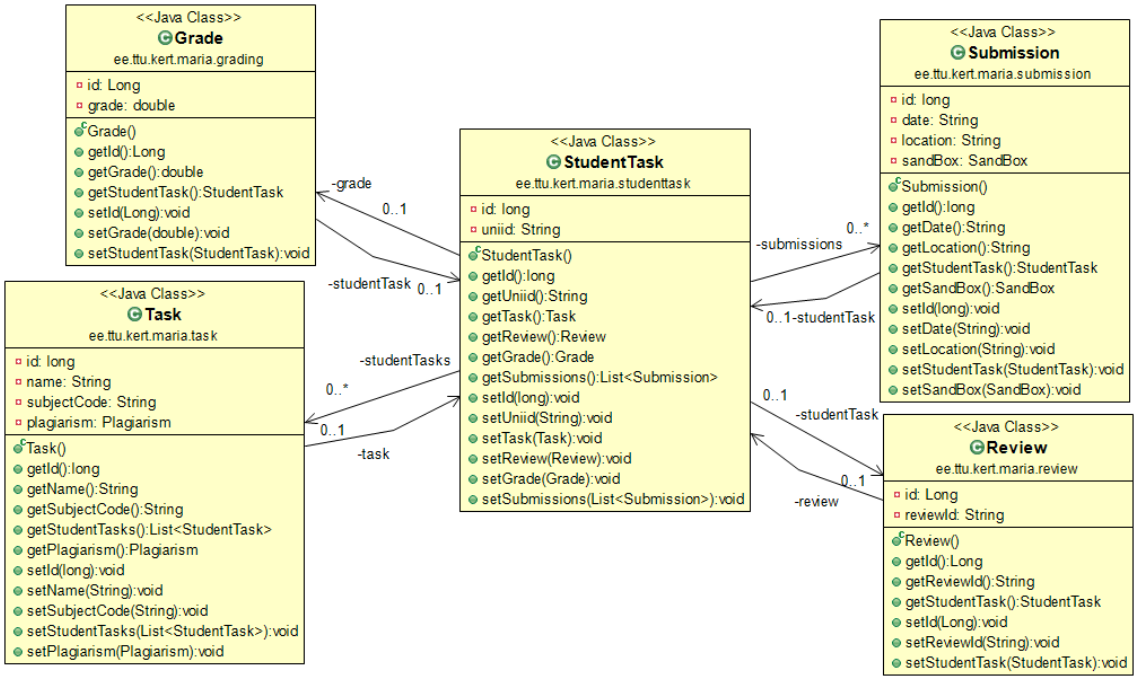
Liides, mis suhtleb andmebaasiga ning mille abil on võimalik salvestada ning pärida *Task* tüüpi objekte. *TaskRepository* laiendab *CrudRepository* liidest.

Programmikood:

```
public interface TaskRepository extends CrudRepository<Task, Long> {

	Task findByName(String name);

}
```

StudentTask

Harjutus on ülesande vahetüüp, mis on seotud tudengiga, kes harjutuse üles laadis. Harjutusel on väljad *Long id, String uniid, Task task, Review review, Grade grade* ning list sellega seotud *Submissionidest*. Iga harjutus on seotud täpselt ühe ülesandega. Iga harjutusega on seotud täpselt üks hinne ning ülevaatus.



Joonis 9. Harjutuse klassidiagramm

StudentTaskController

Kontroller, mis võtab vastu REST päringuid harjutuse objektide kohta. Kontrolleril on üks privaatne väli *StudentTaskService studentTaskService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus:

- **public StudentTaskController(StudentTaskService studentTaskService)** – konstruktor, mille abil Spring initsialiseerib automaatselt *studentTaskService* välja.
- **public List<Submission> getSubmissions(Long id)** - meetod, mis tagastab kõik antud harjutusega seotud esitused listina. Meetod kutsutakse välja REST teenuse kaudu ning päringu tüüp on *GET*. Argument id initsialiseeritakse päringu URL-i parameetrina (*/task/id)*.

StudentTaskService

Teenus, mis tegeleb kontrollerilt saadud päringute edastamisega andmebaasi. Klassil on üks privaatne väli *StudentTaskRepository repository*, üks avalik konstruktor ning kolm avalikku meetodit. Andmete edastamine toimub *repository* välja kaudu.

Meetodite kirjeldus:

- **public StudentTask(StudentTaskRepository repository)** – klassi konstruktor, mille abil Spring initsialiseerib automaatselt *repository* välja.
- **public List<Submission> getSubmissions(Long id)** – meetod, mis leiab id järgi harjutuse ning tagastab kõik sellega seotud esitused. Kui harjutust ei eksisteeri, tagastatakse tühi list.
- **public Task getByTaskAndUniid(Task task, String uniid)** – meetod, mis võimaldab ülesande ja uniid järgi andmebaasist *StudentTask* objekte otsida. Kui objekti andmebaasis ei leidu, tagastatakse *null*.
- **public StudentTask save(StudentTask task)** – meetod, mis salvestab argumendina antud *StudentTask* objekti andmebaasi.

StudentTaskRepository

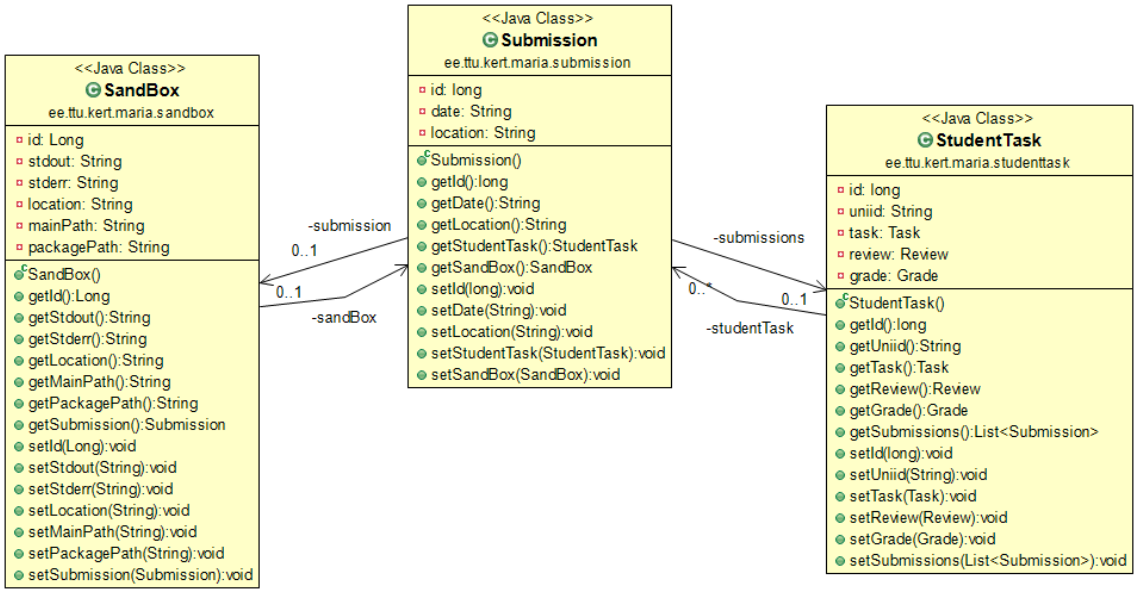
Liides, mis suhtleb andmebaasiga ning mille abil on võimalik salvestada ning pärida *StudentTask* tüüpi objekte. *StudentTaskRepository* laiendab *CrudRepository* liidest.

Programmikood:

```
public interface StudentTaskRepository extends CrudRepository<StudentTask,
Long> {

	StudentTask findByTaskAndUniid(Task task, String uniid);

}
```

Submission

Esitus on ülesande alamtüüp ning vajalik selleks, et eristada koodi üleslaadimisi kuupäeva ja kellaaja järgi. Esitusel on väljad *Long id*, *StudentTask studentTask*, *String* kuupäev, *String location* ja *Sandbox sandbox*. Iga esitus on seotud ainult ühe harjutusega. Iga esitusega on seotud täpselt üks *sandbox*.



Joonis 10. Esituse klassidiagramm

SubmissionService

Teenus, mis tegeleb päringute edastamisega andmebaasi. Klassil on üks privaatne väli *SubmissionRepository submissionRepository*, üks avalik konstruktor ning üks avalik meetod. Andmete edastamine toimub *submissionRepository* välja kaudu.

Meetodite kirjeldus:

- **public SubmissionService(SubmissionRepository repository)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt *repository* välja.
- **public Submission save(Submission submission)** – meetod, mis salvestab argumendina antud *Submission* objekti andmebaasi.

SubmissionRepository

Liides, mis suhtleb andmebaasiga ning mille abil on võimalik salvestada ning pärida *Submission* tüüpi objekte. *SubmissionRepository* laiendab *CrudRepository* liidest.

4.1.4 Tsentraalne Git *Hook*

Kontroller

Tsentraalne kontroller võtab vastu päringud TTÜ Gitist. Kui tudeng laeb oma koodi Giti üles, käivitatakse Giti serveris *hook* (skript), mis saadab vajalikud andmed (ainekood ja uniid) antud rakendusele. Seejärel tõmmatakse kohalikku serverisse antud repositooriumi

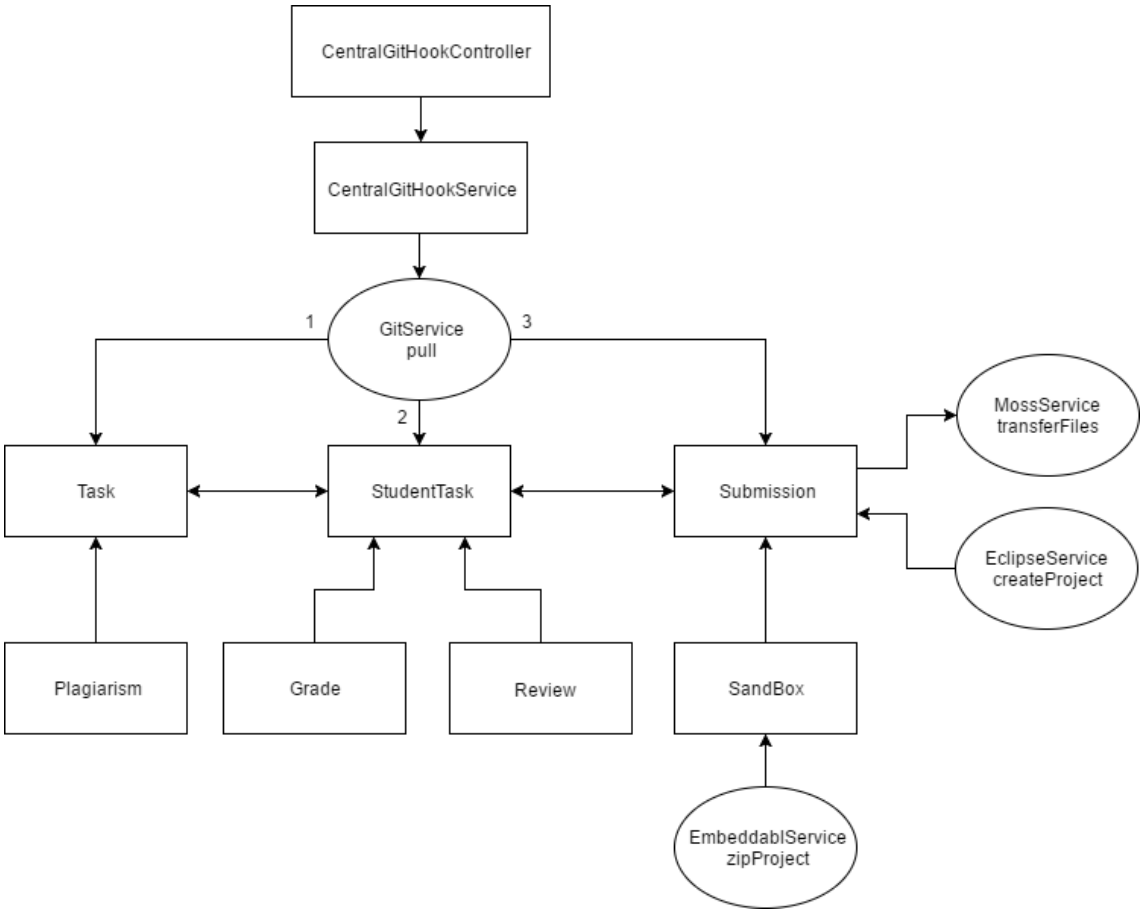
muudatused. *Hookiga* käivitatud keskne kontroller on ka kogu programmi põhiline sisendpunkt, kust saab alguse andmebaasiobjektide loomine.

Teenus

Pärast muudatuste allalaadimist algab muudatuste kontroll keskses teenuses. Selleks käiakse tudengi repositooriumis läbi kõik kaustad ning kontrollitakse, kas sellise nimega ülesanne juba eksisteerib. Vastasel juhul eksemplar luuakse. Igale ülesandele luuakse ka plagiaadikontrolli objekt.

Seejärel vaadatakse, kas ülesandega on seotud harjutus, mille autoriks on repositooriumi omanik. Kui on vajalik eksemplar luua, lisatakse sellele *Review* ja *Grade* objekt. Eksemplar seotakse ülesande objektiga.

Viimaseks tehakse kindlaks, et harjutuse kaustas on toimunud koodi uuendus ning selle põhjal luuakse uus esitus. Igale esitusele lisatakse *SandBox* objekt, seejärel leitakse *main* klassi asukoht ning pakitakse esitus *zip*-failiks. Siis tehakse esitusest uus *Eclipse'i* projekt, mis pakitakse samuti *zip*-failiks. Faili asukoht salvestatakse esituse objekti väljale. Esituse failid kopeeritakse plagiaadikontrolli kausta ning objekt seostatakse harjutusega.



Joonis 11. Taustal tehtava töö skeem

4.1.5 Versioonihaldus

Versioonihaldusena kasutame TTÜ Giti. Gitist tulevad päringud võtab vastu tsentraalne kontroller, mis suunab need edasi kesksele teenusele, mis suhtleb omakorda versioonihaldusteenusega. Seetõttu pole versioonihaldusel eraldi kontrollerit tarvis.

VersionControlService

Liides, mis kirjeldab ära vajalikud meetodid versioonihalduse teostamiseks.

Programmikood:

```
public interface VersionControlService {

	public String pull(String uniid, String subjectCode);

	public String getHash(String uniid, String taskName);

	public String createHash(String uniid, String taskName);

}
```

GitService

GitService implementeerib *VersionControlService* klassi. *GitService* klassil on privaatsed väljad *String pullScriptPath, String hashScriptPath, String repoPath* ja *String hashPath*, mille väärtused saadakse konfiguratsioonifailist, ning privaatne väli *ScriptRunner scriptRunner*.

Meetodite kirjeldus:

- **public String pull(String uniid, String subjectCode)** – loob sisenditest ja *pullScriptPath* väljast käsu, edastab selle *scriptRunner* objektile ning tagastab käsu käivitamise tulemuse.

- **public String getHash(String uniid, String taskName)** – loob sisenditest ja *hashPath* väljast tee failisüsteemis olevale failile, loeb *FileHandler* objekti abil selle sisu ning tagastab tulemuse.

- **public String createHash(String uniid, String taskName)** – loob *repoPath* väljast ja sisenditest tee projekti kausta. Seejärel ehitab *hashScriptPath* ja *hashScript* väljadest ning projekti kausta teest käsu, mille käivitab *scriptRunner* objekti abil ning tagastab skripti tulemuse.

4.1.6 Koodi ülevaatus

Koodi ülevaatusel on kaks põhikomponenti – IDE ja ülevaatus. IDE on vajalik selleks, et hindajal oleks võimalik mugavalt koodi läbi vaadata ning kommenteerida. IDE-s tehtud muudatustest uuendatakse *Review* objekti.

IDE

IDE-na eelistab aine õppejõud kasutada *Eclipse'i*, seega on IDE osa realiseeritud *Eclipse'ist* lähtudes. Kuna IDE-ga seotud päringud tulevad kesksest kontrollerist ja teenusest, pole IDE osale eraldi kontrollerit vaja. IDE osa on vajalik selleks, et õppejõul oleks vajadusel võimalik tudengi üles laetud programmikoodi viimane versioon lihtsa vaevaga kätte saada ja käivitada.

IDEService

Liides, mis kirjeldab ära vajalikud meetodid IDE osa realiseerimiseks.

Programmikood:

```
public interface IDEService {

	public String createProject(String uniid, String taskName);

}
```

EclipseService

Teenus, mis implementeerib *IDEService* liidest. Klassil on privaatne väli *ScriptRunner*, mis initsialiseeritakse konstruktoris ja 5 privaatset välja, mille väärtused saadakse konfiguratsioonifailist: *String repoPath, String zipPath, String zipScriptPath, String projectCreatorPath* ja *String projectTemplatePath*. Teenusel on üks avalik konstruktor, üks avalik ja üks privaatne meetod.

Meetodite kirjeldus:

- **public EclipseService()** – konstruktor, kus initsialiseeritakse *scriptRunner* väli.
- **public String createProject(String uniid, String taskName)** – meetod, mille argumentide abil luuakse mallist uus projekt, pakitakse see kokku ning kopeeritakse vastavalt kausta, kust sellele pärast ligi saada. Meetod tagastab projekti asukoha kettal. Vigade puhul tagastatakse *null*.
- **private List<String> replaceLine(List<String> lines, String search, String replace)** – meetod, mis argumendina antud listis vahetab ära esimese otsitava sõnega rea argumendist tuleva sõne vastu. Tagastatakse vahetatud reaga list.

Review

Review on andmebaasiobjekt, millel on kolm privaatset välja: *Long id*, *String reviewId* ja *StudentTask studentTask*. Objektis talletatud *reviewId* järgi saab kätte ülevaatuse lingi, kus on õppejõu kommentaarid koodile. Iga *Review* tüüpi objekt on üks-ühele seoses harjutusega.

ReviewController

Kontroller, mis võtab vastu REST päringuid *Review* objektide kohta. Kontrolleril on üks privaatne väli *GitHubService gitHubService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus:

- **public ReviewController(GitHubService gitHubService)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt välja *gitHubService*.
- **public Review update(Review review, String uniid, String taskName)** – meetod, mille argumentide abil tagastatakse vastava harjutusega seotud uuendatud *Review* tüüpi objekt. Meetod kutsutakse välja REST teenuse kaudu ning päringu tüüp on *POST*. Argument *studentTask* saadakse *POST* andmetest. Ülejäänud argumendid tulevad päringu URL-ist.

ReviewService

Liides, milles on kirjeldatud koodi ülevaatuseks vajalikud meetodid.

Programmikood:

```
public interface ReviewService {

	public Review updateReview(String uniid, String taskName, Review
review);

}
```

GithubService

GitHubService on teenus, mis kasutab GitHub API-t [17] Gistide loomiseks [18] ja uuendamiseks ning implementeerib *ReviewService* klassi. Klassil on kaheksa privaatset välja: *String repoPath, String user, String pass, String gistTemplateLink*, mis tulevad

konfiguratsioonifailist ja *ReviewRepository reviewRepository, GitHubClient client, FileHandler reader* ning *SecureRandom secureRandom*. Teenus sisaldab avalikku konstruktorit, kaht avalikku ning viit privaatset meetodit.

Meetodite kirjeldus:

- **public GitHubService(ReviewRepository reviewRepository)** – Spring loob argumendist automaatselt *ReviewRepository* objekti ning salvestab selle klassi väljale. Samuti initsialiseerib konstruktor privaatsed muutujad *reader*, *gitHubClient* ja *secureRandom*.

- **public Review updateReview(String uniid, String taskName, Review review)** – tagastab uuendatud *Review* tüüpi objekti. Kui objektil on juba väärtustatud väli *reviewId*, uuendatakse selle identifikaatoriga Gisti, vastasel juhul luuakse uus Gist ja initsialiseeritakse objekti tühi väli. Tagastatav väärtus saadakse *reviewRepository* välja abil.

- **public Review saveReview(Review review)** – meetod, mis salvestab *Review* objekti andmebaasi.

- **private String createGist(String uniid, String taskName)** – loob ühenduse GitHub API *GistService*'iga, loob Gist objekti ning lisab *addAllFiles* meetodi abil kõik kaustas olevad failid loodud objekti. Seejärel laseb meetod API-l Gisti realiseerida ning tagastab loodud Gisti id. Kui Gisti loomisel läheb midagi valesti, tagastatakse *null*.

- **private String updateGist(String id)** – loob ühenduse *GistService*'iga ning pärib sealt id järgi Gisti. Õnnestumise korral lisatakse *addAllFiles()* meetodiga objekti uued failid ning API kirjutab olemasoleva Gisti üle, vastasel juhul tagastatakse *null*.

- **private Map<String, GistFile> addAllFiles(String taskPath)** – leiab parameetri ja *reader* objekti abil kõik kaustas olevad failid, teeb igast failist *GistFile* objekti ning määrab parameetrid *File* objektist. Seejärel lisab meetod kõik objektid *HashMapi*, mille võtmeks on faili nimi, ning tagastab selle. Kui kaustas faile ei ole, tagastatakse tühi *HashMap*.

- **private String getRandomString()** – tagastab juhusliku sõne, kasutades *secureRandom* välja.

- **private Authorization getGistAuthorization()** – autoriseerib GitHub API kaudu Gisti teenuse kasutamise. Autoriseerimiseks on vajalik unikaalse kirjelduse olemasolu. Selle genereerimiseks kasutatakse *getRandomString* meetodit. Õnnestumise korral tagastatakse autoriseering, vastasel juhul *null*.

ReviewRepository

Liides, mis suhtleb andmebaasiga. Liidese kaudu on võimalik salvestada *Review* tüüpi andmeid ning neid lugeda. Klass laiendab *CrudRepository* liidest.

4.1.7 Sandbox

Tudengite tööde hindamise teeb tunduvalt lihtsamaks, kui on võimalik näha programmi väljundit seda manuaalselt käivitamata. Selleks on *SandBox* klass, kuhu salvestatakse tudengi üles laetud koodi käivitamise tulemused *Embeddabl* API-t kasutades. API loojaks on Sven Kadak, kes arendas spetsiaalselt antud projekti jaoks Java *sandbox* teenuse. Praegune serveri versioon näeb ette, et API on kasutusel klientrakenduses, saab päringu sisendi teenuselt ning salvestab API vastuse andmebaasi. Lahendus on vajalik, kuna kõiki programmikoode ei ole tingimata vaja alla laadida ning käsitsi käivitada. *SandBoxil* on seitse privaatset välja *Long id, String stdout, String stderr, String location, String mainPath, String packagePath* ning *Submission submission*. Iga *SandBox* tüüpi objekt on üks-ühele seoses esituse objektiga.

SandBoxController

Kontroller, mis võtab vastu REST päringuid *SandBox* objektide kohta. Kontrolleril on üks privaatne väli *EmbeddablService embeddablService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus

- **public SandBoxController(EmbeddablService embeddablService)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt *embeddablService* välja.

- **public String update(String uniid, String taskName, SandBox sandBox)** – meetod, mis tagastab uuendatud *SandBox* tüüpi objekti. Meetod kutsutakse välja REST teenuse kaudu ning päringu tüüp on *POST*. Argument *sandBox* saadakse *POST* andmetest. Ülejäänud argumendid tulevad päringu URL-ist.

SandBoxService

Liides, kus on kirjeldatud *SandBox* teenuse realiseerimiseks vajalikud meetodid.

Programmikood:

```
public interface SandBoxService {

	public SandBox updateSandBox(String uniid, String taskName, SandBox
sandBox);

	public String zipProject(String uniid, String taskName);

}
```

EmbeddablService

Teenus, mis implementeerib *SandBoxService* liidest ning mis saab sisendid kontrollerist. Klassil on viis privaatset välja *SandBoxRepository sandBoxRepository, FileHandler reader* ja *ScriptRunner scriptRunner*. Ülejäänud kahe välja *String repoPath* ja *String zipPath* väärtused tulevad konfiguratsioonifailist. Teenusel on avalik konstruktor, kolm avalikku ja kaks privaatset meetodit.

Meetodite kirjeldus:

- **public EmbeddablService(SandBoxRepository sandBoxRepository)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt välja *sandBoxRepository*. Konstruktoris väärtustatakse lisaks ka väljad *reader* ja *scriptRunner*.

- **public SandBox updateSandBox(String uniid, String taskName, SandBox sandBox)** – meetod, mis uuendab argumendiks olevat SandBox objekti ja tagastab selle.

- **public String zipProject(String uniid, String taskName)** – meetod, mis leiab repositooriumi asukoha, argumentide uniid ja *taskName* järgi projekti ning pakib selle *zip*-failiks. Kui pakkimine ei õnnestu, tagastatakse *null*.

- **public SandBox save(SandBox sandBox)** – meetod, mis salvestab *SandBox* tüüpi objekte andmebaasi ning tagastab salvestatud objekti.

- **private String getMainPath(String taskPath)** – meetod, mis leiab repositooriumide ja ülesande asukoha järgi tee selle failini, milles asub *main*-meetod. Kui sellist faili ei eksisteeri, tagastatakse *null*.

- **private String getPackagePath(String uniid, String taskName)** – meetod, mis tagastab etteantud argumentide põhjal selle paketi, mille sees on *main* meetod. Kui *main* meetodit ei eksisteeri, tagastatakse *null*.

SandBoxRepository

Liides, mis suhtleb andmebaasiga. Liidese kaudu on võimalik salvestada *SandBox* tüüpi andmeid ning neid lugeda. Klass laiendab *CrudRepository* liidest.

4.1.8 Plagiaadikontroll

Plagiaadikontroll (*Plagiarism*) on klass, millega salvestatakse plagiaadikontrolli tulemusi andmebaasi. Plagiaadikontrollil on kolm privaatset välja: *Long id*, *String result* ja *Task task*. Plagiaadikontroll on üks-ühele seoses ülesande objektiga.

PlagiarismController

Kontroller, mis võtab vastu REST päringuid *Plagiarism* objektide kohta. Kontrolleril on üks privaatne väli *MossService mossService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus:

- **public PlagiarismController(MossService mossService)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt *mossService* välja.

- **public Plagiarism run(String taskName, Plagiarism plagiarism)** – meetod, mis tagastab plagiaadikontrolli uuendatud objekti. Meetod kutsutakse välja REST

teenuse kaudu ning päringu tüüp on *POST*. Argument *plagiarism* saadakse *POST* andmetest.

PlagiarismService

Liides, mis kirjeldab ära plagiaadikontrolli teenuse käivitamiseks vajalikud meetodid.

Programmikood:

```
public interface PlagiarismService {

	public Plagiarism run(String taskName, Plagiarism plagiarism);

	public void transferFiles(String uniid, String taskName);

}
```

MossService

Teenus, mis kasutab MOJI API-t [19] *Moss* plagiaadikontrolliteenuse [16] kasutamiseks. Klassil on 5 privaatset välja, millest neli – *String plagiarismPath, String repoPath, String mossUserid* ja *String copyScriptPath* – väärtustatakse konfiguratsioonifaili abil. Viimane klassi väli on *PlagiarismRepository plagiarismRepository*. Teenusel on avalik konstruktor ja kaks avalikku meetodit.

Meetodite kirjeldus:

- **public MossService(PlagiarismRepository plagiarismRepository)** – konstruktor, mille argumendist initsialiseerib Spring automaatselt klassi välja *plagiarismRepository*.

- **public Plagiarism run(String taskName, Plagiarism plagiarism) –** meetod, mille argumendist leitakse tee projektini, millele plagiaadikontrolli teostada soovitakse. Seejärel laetakse failid MOJI API abil *Moss* serverisse, mis loob tulemuste üle vaatamiseks lingi. Link seostatakse *plagiarism* objektiga, tagastatakse. Vigade ilmnemisel tagastatakse *null*.

- **public void transferFiles(String uniid, String taskName)** – meetod, mis kopeerib argumentide ja repositooriumide asukoha järgi leitud projekti failid vastavasse plagiaadikontrolli kausta.

PlagiarismRepository

Liides, mis suhtleb andmebaasiga. Liidese kaudu on võimalik salvestada *Plagiarism* tüüpi andmeid ning neid lugeda. Klass laiendab *CrudRepository* liidest.

4.1.9 Meiliteenus

Meiliteenus on vajalik tudengitele tagasiside saatmiseks. Tagasisideks on link koodi ülevaatusele. Praegu kasutab rakendus meiliteenusena *MailGuni,* mis võimaldab lihtsa vaevaga meile saata.

MailController

Kontroller, mis võtab vastu REST päringuid, et käivitada tagasiside saatmine tudengile. Kontrolleril on üks privaatne väli *MailGunService mailGunService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus:

- **public MailController(MailGunService mailGunService)** – konstruktor, mille argumendist väärtustab Spring automaatselt *mailGunService* välja.
- **public String send(String uniid, String reviewId, String subject)** – meetod, mis käivitab meili saatmise.

MailService

Liides, milles on kirjeldatud meiliteenuse kasutamiseks vajalikud meetodid.

Programmikood:

```
public interface MailService {

	public String sendFeedback(String uniid, String reviewLink, String
subject);

}
```

MailGunService

Teenus *MailGun* serveriga ühendamiseks. Klassil on kaks privaatset välja *String apiKey* ja *String gistLink* mis saadakse konfiguratsioonifailist. *MailGunService* klassil on üks avalik ja üks privaatne meetod.

Meetodite kirjeldus:

- **public void sendFeedback(String uniid, String reviewId, String subject)** – kasutab *MailGun* serverit ja *sendEmail* meetodit, et saata email koodi ülevaatuse lingiga õigele adressaadile.

- **private ClientResponse sendEmail(String uniid, String reviewId, String subject)** – loob ühenduse *MailGun* serveriga ning edastab sellele meili parameetrid. Õnnestumise korral tagastab *ClientResponse* objekti, vastasel juhul veateate.

4.1.10 Hindamine

Hinne (*Grade*) on andmebaasiobjekt, kus hoiustatakse harjutusega seotud hinnet. Klassil on privaatsed väljad *Long id, double grade* ja *StudentTask studentTask*. Hinne on üks-ühele seoses harjutusega.

GradeController

Kontroller, mis võtab vastu REST päringuid *Grade* tüüpi objektide kohta. Kontrolleril on üks privaatne väli *GradeService gradeService*, üks avalik konstruktor ning üks avalik meetod.

Meetodite kirjeldus:

- **public GradeController(GradeService gradeService)** – konstruktor, mille argumendist väärtustab Spring automaatselt *gradeService* välja.
- **public Grade update(Grade grade)** – meetod, mis uuendab andmebaasis olevat Grade objekti. Meetodi päringu tüüp on *POST*.

GradeService

Teenus, mis suhtleb hinnete andmebaasiliidesega. Klassil on privaatne väli *GradeRepository gradeRepository*, üks avalik konstruktor ja üks avalik meetod.

Meetodite kirjeldus:

- **public GradeService(GradeRepository gradeRepository)** – konstruktor, mille argumendist väärtustab Spring automaatselt *gradeRepository* välja.

- **public Grade saveGrade(Grade grade)** – meetod, mis salvestab *Grade* tüüpi objekti andmebaasi.

GradeRepository

Liides, mis suhtleb andmebaasiga. Liidese kaudu on võimalik salvestada *Grade* tüüpi andmeid ning neid lugeda. Klass laiendab *CrudRepository* liidest.

4.2 Skriptid

Kuna käesolev lõputöö on kirjutatud *Windowsi* operatsioonisüsteemis, tuli leida lahendus skriptide käivitamisele. Selleks valiti *Bash on Ubuntu on Windows* [13], mis on Linuxi terminal otse *Windowsi* operatsioonisüsteemis ning mille kasutamiseks ei ole vaja virtuaalmasinat. Skriptid on rakenduses kasutusel selleks, et teha muudatusi failisüsteemis. Kõik olemasolevad skriptid on *Bash* skriptid [5]. Aktiivselt on kasutusel kuus skripti.

4.2.1 copyfiles.sh

copyfiles.sh (vt. Lisa 2) toimimiseks on vaja kaht argumenti. Esimeseks argumendiks on failide lähteasukoht ning teiseks argumendiks failide sihtasukoht. Kui mõlemad sisestatud asukohad on korrektsed, kopeerib skript edukalt failid sihtkohta.

4.2.2 embeddabl.sh

embeddabl.sh (vt. Lisa 3) vajab toimimiseks nelja argumenti. Esimeseks argumendiks on ülesande nimi, teiseks argumendiks uniid, kolmandaks repositooriumide asukoht ning neljandaks *zip*-failide asukoht failisüsteemis. Skript loob vajalikud kaustad ning kopeerib projekti failid sellisele kujule, et need oleks *Embeddabl* API-le loetaval kujul.

4.2.3 hashcreator.sh

hashcreator.sh (vt. Lisa 4) toimimiseks on vaja kaht argumenti. Esimeseks argumendiks on projekti asukoht failisüsteemis, teiseks räsifailide asukoht failisüsteemis. Kõigepealt leiab skript kõikide projekti failide räsi. Seejärel tehakse kindlaks ülesande kausta nimi ning uniid. Lõpuks luuakse sihtkaust, kuhu tekitatakse fail, mille sisuks on eelnevalt leitud räsi.

4.2.4 projectcreator.sh

projectcreator.sh (vt. Lisa 5) toimimiseks on vaja viit argumenti. Esimeseks argumendiks on ülesande nimi, teiseks uniid, kolmandaks repositooriumide asukoht, neljandaks *zip*-failide ja viiendaks projekti malli asukoht failisüsteemis. Skript loob uue *Eclipse'i* projekti, mille sisuks on argumentidest leitud projekti failid.

4.2.5 pull.sh

pull.sh toimimiseks on vaja kaht argumenti: uniid ja repositooriumide asukoht failisüsteemis. Skripti kasutamiseks peab olema installeeritud Git. Nende argumentide abil navigeeritakse õigesse repositooriumisse ja laetakse Gitist uuendused.

```
#!bin/bash

uniid=$(echo $1 | awk '{print tolower($0)}')
echo $uniid
repopath=$2
path=$repopath$uniid

cd $path

git pull
```

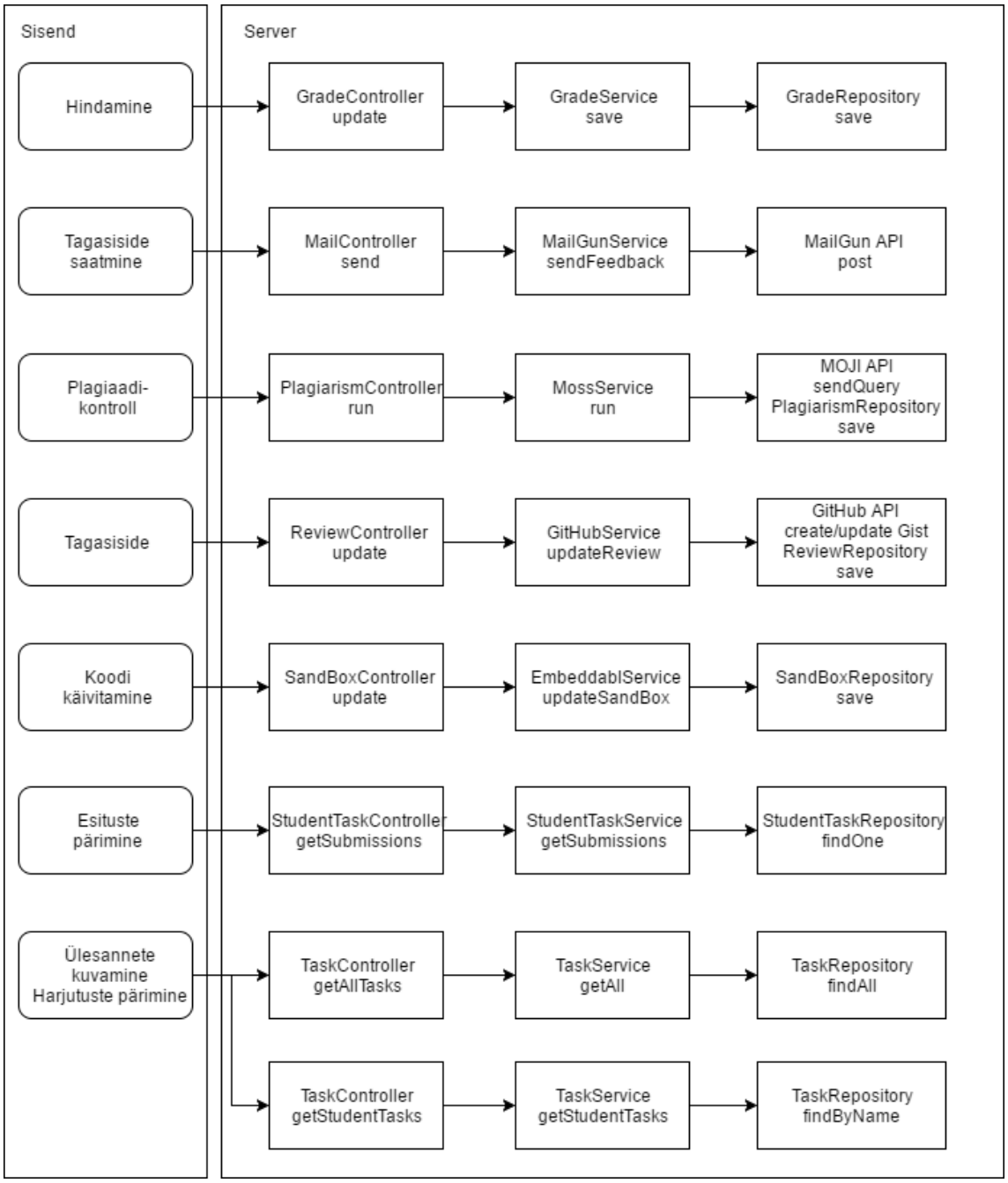
4.2.6 zipper.sh

zipper.sh toimimiseks on vaja nelja argumenti. Esimeseks argumendiks on uniid, teiseks kaust, mida *zipiks* pakitakse, kolmandaks kausta asukoht ja neljandaks *zip*-faili sihtkoht. Skripti kasutamiseks peab olema installeeritud Java. Argumentide ja *jar*-käsu abil luuakse *zip*-fail ning ajutine kaust, kus on programmi failid, kustutatakse.

```
#!/bin/bash

uniid=$1
folder=$2

source=$3 #"D:/Users/mammu/workspace/loputoo/repod/maria.kert/EX05/src"
dest=$4 #"D:/Users/mammu/workspace/loputoo/zips/EX05"

cd $source
jar -cMf $dest/$uniid.zip $folder
chmod +x $dest/$uniid.zip
rm -r $folder
echo "$dest/$uniid.zip"
```

4.3 Kasutaja interaktsioon süsteemiga

Mõned muudatused süsteemis käivituvad vaid kasutajapoolse sisendi tagajärjel. Järgneval joonisel on kirjeldatud viisid, kuidas saab toimuda kasutaja interaktsioon käesoleva rakendusega.



Joonis 12. Kasutaja interaktsioon süsteemiga

5 Kokkuvõte

Lõputöö eesmärgiks oli arendada ainele „Objektorienteeritud programmeerimine keeles Java“ mugav hindamiskeskkond, automatiseerides taustprotsesse ja vähendades sellega drastiliselt hindamisele kuluvat aega. Rakenduse lisakriteeriumiks oli arhitektuuri dünaamilisus, mis saavutati liidestamise teel.

Bakalaureusetöö raames valmis andmebaasiga serverirakendus, mis saab sisendid REST-teenustelt. Rakendus võimaldab tuvastada, millistes failides on toimunud muudatused ning loob selle põhjal automaatselt erinevad ülesande tüübid ja nendega seotud andmebaasiobjektid. Tudengite üles laetud koodile saab teostada plagiaadikontrolli, anda tagasisidet ning seda hinnata. Igale tudengi üles laetud esitusele on olemas meetodid, mille abil salvestatakse koodi välises süsteemis käivitamiseks vajalikud parameetrid. Rakendus võimaldab saata tagasiside lingiga emaile.

Failisüsteemi muudatusi tehakse skriptidega. Käesoleva töö raames tegelevad skriptid kaustade ja failide loomise, kopeerimise, *zip*-failiks pakkimisega ning versioonihalduse muudatuste alla laadimisega.

Lõputöö põhiliseks puudujäägiks oli autori kogenematus sellise mastaabiga projekti arendamisel, mistõttu osutusid mitmed esialgselt välja mõeldud lahendused üsnagi utoopiliseks. Soov on projektiga jätkata ning realiseerida ka töötav klientrakendus, et projekt oleks võimalik järgmisel aastal hindamiseks kasutusele võtta. Võimalusel ühendada hindamine ka *ained.ttu.ee* keskkonnaga, automatiseerides protsessi veelgi enam.

Bakalaureusetöö eesmärk sai täidetud. Valmis toimiv serverirakendus, mis ühendamisel klientrakendusega muudab aine „Objektorienteeritud programmeerimine keeles Java“ hindamisprotsessi kindlasti lihtsamaks ja vähem ajakulukaks. Lisaks on rakenduse kood lihtsasti modifitseeritav, kuna väliseid ressursse tarbivad teenused realiseeriti läbi liideste.

Kasutatud kirjandus

[1] Pivotal Software, Inc., „Spring Boot,“ [Võrgumaterjal]. Available: http://projects.spring.io/spring-boot/. [Kasutatud 10 Mai 2017].

[2] Oracle Corporation, „Java,“ [Võrgumaterjal]. Available: https://java.com/en/. [Kasutatud 10 Mai 2017].

[3] Gradle Inc. , „Gradle,“ [Võrgumaterjal]. Available: https://gradle.org/. [Kasutatud 10 Mai 2017].

[4] MariaDB Foundation, „MariaDB,“ [Võrgumaterjal]. Available: https://mariadb.org/. [Kasutatud 10 Mai 2017].

[5] „Bash (UNIX shell),“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Bash_(Unix_shell). [Kasutatud 20 Mai 2017].

[6] Pivotal Software, Inc., „Spring Framework,“ [Võrgumaterjal]. Available: https://spring.io/. [Kasutatud 10 Mai 2017].

[7] S. P. R. Katamreddy, „Why Spring Boot?,“ [Võrgumaterjal]. Available: https://dzone.com/articles/why-springboot. [Kasutatud 10 Mai 2017].

[8] „Spring Initializr,“ [Võrgumaterjal]. Available: http://start.spring.io/. [Kasutatud 10 Mai 2017].

[9] „Representational state transfer,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Kasutatud 10 Mai 2017].

[10] Oracle Corporation, „MySQL,“ [Võrgumaterjal]. Available: https://www.mysql.com/. [Kasutatud 10 Mai 2017].

[11] C. B. Richard Blum, Bible: Linux Command Line and Shell Scripting Bible (3rd Edition), Indianapolis, Indiana: John Wiley and Sons, Inc., 2015.

[12] Free Software Foundation, Inc., „GNU Bash,“ [Võrgumaterjal]. Available: https://www.gnu.org/software/bash/. [Kasutatud 21 Mai 2017].

[13] Microsoft, „Bash on Ubuntu on Windows,“ [Võrgumaterjal]. Available: https://blogs.msdn.microsoft.com/commandline/learn-about-bash-on-windows-subsystem-for-linux/. [Kasutatud 21 Mai 2017].

[14] „Project Lombok,“ [Võrgumaterjal]. Available: https://projectlombok.org/. [Kasutatud 20 Mai 2017].

[15] J. Friesen, „Java 101: Interfaces in Java,“ [Võrgumaterjal]. Available: http://www.javaworld.com/article/3171300/java-language/java-101-interfaces-in-java.html. [Kasutatud 21 Mai 2017].

[16] „Plagiarism Detection,“ [Võrgumaterjal]. Available: https://theory.stanford.edu/~aiken/moss/. [Kasutatud 20 Mai 2017].

[17] eclipse, „GitHub Java API,“ [Võrgumaterjal]. Available: https://github.com/eclipse/egit-github/tree/master/org.eclipse.egit.github.core. [Kasutatud 09 Mai 2017].

[18] kevinsawicki, „GistViaOAuthToken.java,“ 11 November 2011. [Võrgumaterjal]. Available: https://github.com/kevinsawicki/github-api-examples/blob/master/examples/src/main/java/com/github/kevinsawicki/api/GistViaOAuthToken.java. [Kasutatud 09 Mai 2017].

[19] nordicway, „MOJI,“ [Võrgumaterjal]. Available: https://github.com/nordicway/moji. [Kasutatud 13 Mai 2017].

Lisa 1 – projekti Git link

https://bitbucket.org/mariakert/loputoo

Lisa 2 – copyfiles.sh

```
#!/bin/bash

sourcepath=$1
destpath=$2

echo $sourcepath
echo $destpath

mkdir -p $destpath
cd $sourcepath

for file in "$sourcepath*" ; do
    echo $file
    cp $file $destpath
done
```

Lisa 3 – embeddabl.sh

```
#!/bin/bash

task=$1
uniid=$2

repopath=$3 #"D:/Users/mammu/workspace/loputoo/repod/"
zippath=$4 #"D:/Users/mammu/workspace/loputoo/zips/"

cd $repopath$uniid/$task/src/
mkdir -p $zippath$task/ #../../zips/$task/'

mkdir -p $task

for d in "$repopath$uniid/$task/src"/* ; do
    if [[ "$d" != "$repopath$uniid/$task/src/$task" ]]; then
        cp -r $d $task
    fi
done

echo "$repopath$uniid/$task/src/$task"
```

Lisa 4 – hashcreator.sh

```
#!/bin/bash

projectpath=$1
hashpath=$2 #"D:/Users/mammu/workspace/loputoo/hashes/"

cd $projectpath

folders=$(echo $projectpath | tr "/" "\n")
value="$(find $projectpath -type f -print0 | xargs -0 sha1sum)"
taskfolder=""
uniid=""

for folder in $folders
do
    if [ "$folder" != "src" ]
    then
        taskfolder="$folder"
    fi
done

for folder in $folders
do
    if [ "$folder" == "$taskfolder" ]
    then
        break
    else
        uniid="$folder"
    fi
done

target="$hashpath$uniid"

mkdir -p $target
touch $target/$taskfolder.txt
echo $value > $target/$taskfolder.txt
echo $value
```

Lisa 5 – projectcreator.sh

```
#!/bin/bash

task=$1
uniid=$2

repopath=$3 #"D:/Users/mammu/workspace/loputoo/repod/"
zippath=$4 #"D:/Users/mammu/workspace/loputoo/projects/"
templatepath=$5 #"D:/Users/mammu/workspace/loputoo/templates/Template"

cd $repopath$uniid/$task/src/
mkdir -p $zippath$task/ #../../zips/$task/

mkdir -p $task

cp -r "$templatepath/.settings" $task
cp -r "$templatepath/.classpath" $task
cp -r "$templatepath/.project" $task
cp -r $templatepath/* $task

for f in */ ; do
    if [[ "$f" != "$task/" ]]
    then
        cp -r $f "$task/src/"
    fi
done

echo "$repopath$uniid/$task/src/$task"
```