# TALLINNA TEHNIKAÜLIKOOL

# Infotehnoloogia teaduskond

# 

# 

# 

# Meeskonnaprojekt

# Vladislav Suprun (213654IABB), Leonid Demidov (213641IABB)

# Ülikooli Portal

# Infosüsteemide arendamine II:veebirakendused

# 

# 

# Juhataja: Gunnar Piho, Jelena Vendelin

# 

# 

# 

# 

# 

# Tallinn 2022

[Sissejuhatus](#_a1077p28j3la) 3

[Taust](#_h2ndsm5hsaze) 3

[Probleem](#_kfqiln9fkpz8) 4

[Eesmärgid](#_3na8triewh8) 4

[Koostada projekti maksimaalselt nii heal tasemel kui oskame.](#_q95e7f1ht4lk) 4

[Luua ÕIS analoog](#_6r268ua0wvk0) 4

[Ülikooli andme süsteemi värskendamine](#_pak5i21man5g) 4

[Töö struktuur](#_67clgn9tjg77) 4

[Metoodika](#_rq0b2bwoyrlq) 6

[Ülevaade objektist/Esialgsed nõuded](#_gzz1wvvv9ktu) 6

[Kasutatud tööriistad](#_981i0f8t9vik) 6

[Tööprotsess](#_v5100h69i0md) 6

[Peamised tulemused](#_icr72p9ed2ig) 7

[Arhitektuur ja disain](#_fq5p7q2ljonq) 7

[Kasutajaliidese vaated kuvatõmmised ja selgitused](#_ec6qrsogeura) 8

[Olulised järeldused](#_31l7biz6awb1) 10

[Kasutatud mustrid ja nende põhjendus](#_djx8eiw39rm8) 10

[Koodi kvaliteet ja põhjendus](#_1uv9vxsfleos) 13

[Testimine](#_cpbtc3hcewa5) 13

[Edasised tööd](#_52mn8d93fitw) 14

[Kokkuvõtte](#_evbhj8t74c6r) 15

## 

## Sissejuhatus

### Taust

Razer Pages õppimise raames me valisime teha TalTech ÕIS analoog, sest see koosneb elementidest, mis on võimalik realiseerida RazePages ja ASP.net Core tehnoloogia abil. Lisaks see projekt on piisavalt raske ning võimaldab meie programeerimise oskusi kontrollida.

### Probleem

1. ÕISi andme arvestussüsteemi analoogi puudumine ülikoolide jaoks
2. Ülikooli andme süsteemi on vananenud

### Eesmärgid

### Koostada projekti maksimaalselt nii heal tasemel kui oskame.

### Luua ÕIS analoog

### Ülikooli andme süsteemi värskendamine

* Koodi ja veebilehe lihtsustamine

### Töö struktuur

Käesolev dokument on koostatud Tallinna Tehnikaülikooli (TalTech) infotehnoloogia teaduskonna lõputööde juhendi alusel. Dokument koosneb viiest peatükist. Sissejuhatuses antakse lühike ülevaade tööst, kirjeldatakse probleemi ning püstitatakse eesmärk, mida tööga saavutada üritatakse. Metoodikas kirjeldatakse rakenduse tööpõhimõtet, töövahendeid ning töö protsessi. Töö kolmandas peatükis antakse ülevaade töö loogika- ja andmekihist ning kasutajaliidesest. Töö neljandas sisuosas analüüsib meeskond koodi meetrikaid, kasutatud mustreid, teste ning tehtud tööd ja oma tulemust. Antakse ka ülevaade, mida oleks võinud teha paremini nii programmis kui ka meeskonnatöös. Kokkuvõttes on lõplik ülevaade püstitatud eesmärkidest, probleemist, tööprotsessist, projekti tulemustest ning järeldustest.

## 

## Metoodika

### Ülevaade objektist/Esialgsed nõuded

**Use Case diagram:** [https://app.creately.com/diagram/iTrB2FKbVfE/view](https://app.creately.com/diagram/iTrB2FKbVfE)

* Student
  + Võib valida kursused on lehel “Study program”. Sellel lehel ta näeb pealkirjaks oma suunatuse nimi ja selle all näeb nimekiri kursusest, mis kuuluvad selle suunatusele. Ta võib valida need kursused “check box” abil ja kui tema valik on valmis, ta nupub “confirm” et lisada need kursused oma “enrollment” lehele. Ka on võimalus eemaldada valitud kursust mõne aja jooksul.
  + “Enrollment” lehel ta võib vaadata oma kursuse eest hinded ja kogu EAP, mida ta saab selles semestris, kas kurs on läbitud või ei.
* Lecturer
  + Võib lisada ja kutsuda oma loodud kursuse **õppekavast**.
  + Võib liituda teise kursusele nagu “Lecturer”
  + Panna hindeid “Enrollments” lehel kursuse eest igale tudengile
* Admin
  + Saab eemaldada või lisada Lecturers
  + Saab eemaldada või lisada Students
  + Saab eemaldada või lisada Study programs
  + Saab eemaldada või lisada Semesters

### Kasutatud tööriistad

Visual Studio 2022, Razor Page Scaffold, Razor Pages, MS SQL, Asp.Net ja Entity Framework, NuGet Packages source for frameworks, Microsoft Excel, Word, Teams, Google Drive apps, Creately

### **Tööprotsess**

| Vladislav | umbes 170 tundi |
| --- | --- |
| Leonid | umbes 180 tundi |

Kasutasime oma projekti koostamisel paralleel arendusmeetodit. Kuna projekti alguses on programmeerimise kogemus ja kursis kõigil ühesugused, siis otsustati konkreetseid rolle mitte määrata, vaid ülesandeid jagada võrdselt. Seetõttu vastutavad Leonid ja Vladislav võrdselt nii koodi kui ka dokumentatsiooni eest ning oleksid pidanud sellele vastavalt panustama. Töö toimub programmeerimise paaridena, kus üks inimene kirjutab koodi, kontrollib ja testib seda, teine ​​aga töötleb sarnast koodi ning vajadusel abistavad üksteist paranduste tegemisel. Projektis kasutasime järgmised metoodikat:

**Clean code**

Kuna meie projekt me kirjutasime kasutades TDD, siis me püüdsime kirjutada clean tests. Clean code tähendab seda, et testid peavad olema kiired, et neid oleks võimalus lülitada esimesel vajadusel. Need samuti peavad olema sõltumatud teineteisega, st üks test ei tekita tingimust teise testi sooritamiseks. Testid peavad töötama igas ümbruses, täpse vastusega (sooritatud või mitte) ning kirjutatud täpselt production koodi kirjutamise enne (TDD).

**TDD**

koodi kirjutamises me kasutasime kolm TDD seadust: Esimene seaduse järgi me ei kirjutanud production code, kui ei kirjutanud selleks module tests, mis ei sooritanud kuna funktsionaal veel ei tehtud. Teise seaduse järgi me kirjutasime selle module koodi maht, millest on piisav, et see test ei sorita (või kui see test ei kompileerib, siis see on sooritamata). Viimase seaduse järgi me kirjutame selle production koodi maht, milles on piisav, et sooritamata test läheb hästi ning uus functional ei tekita teiste module testide ebaõnnestumist.

Production koodi kirjutamise protsess võimalik kirjeldada nagu cycle, kus kõigepealt me kirjutasime module testid ning pärast kirjutasime koodi, mis pidi oma module testi sooritama.

Meie töös me püüdsime vältida nn määrdunud teste kirjutamist. Siin on mõned põhjused: kõigepealt need teste raske muuta, kuna nende muutujad on halvasti nimetatud ja selgitamata, mida see tähendab; testide funktsioonid pikad ja komentaarideta, halvasti projekteeritud ja jagamata. Samuti möödunud testi ümber tegemisele ja parandamisele läheb rohkem aega, kui production koodi kirjutamisele.

Kõige tähtsam põhjus häid teste kirjutada on kindlus midagi uut süsteemisse panna, sest kui testid on määrdunud, siis on olemas kardus midagi muuta süsteemis, sest see võib tekitada ebaõnnestunud teste. Heade testitega on võimalik muuta karduseta muuta süsteemi arhitektuuri ja disaini.

## Peamised tulemused

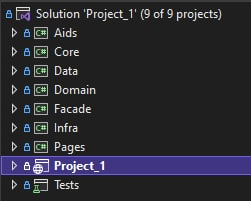
### Arhitektuur ja disain

**Classes hierarchy :**

### 

### Loogikakiht:

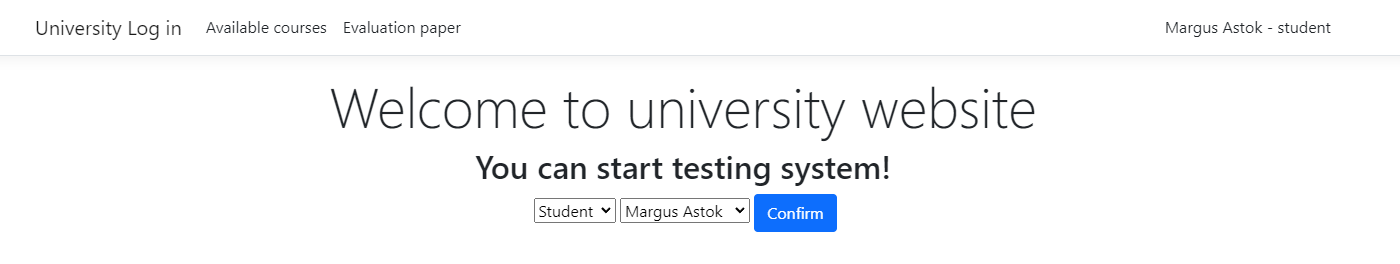
Meie projektis jaotati tarkvara loogikakiht 8 kihiks tulenevalt puhta koodi põhimõtetest.

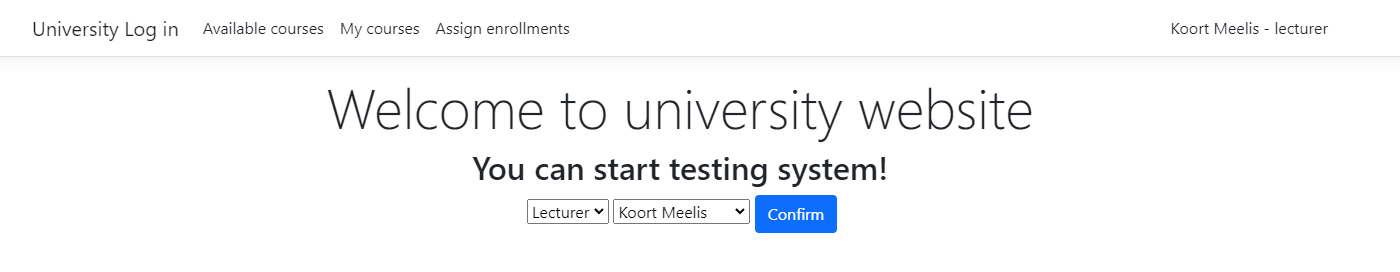


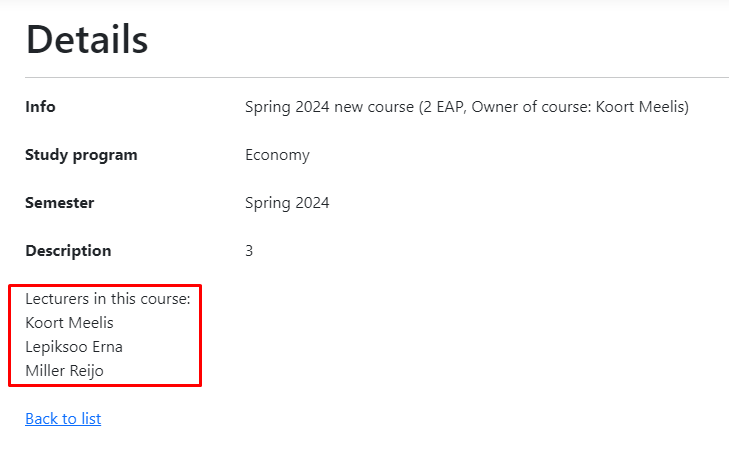
### Kasutajaliidese vaated [kuvatõmmised ja selgitused](#_ec6qrsogeura)

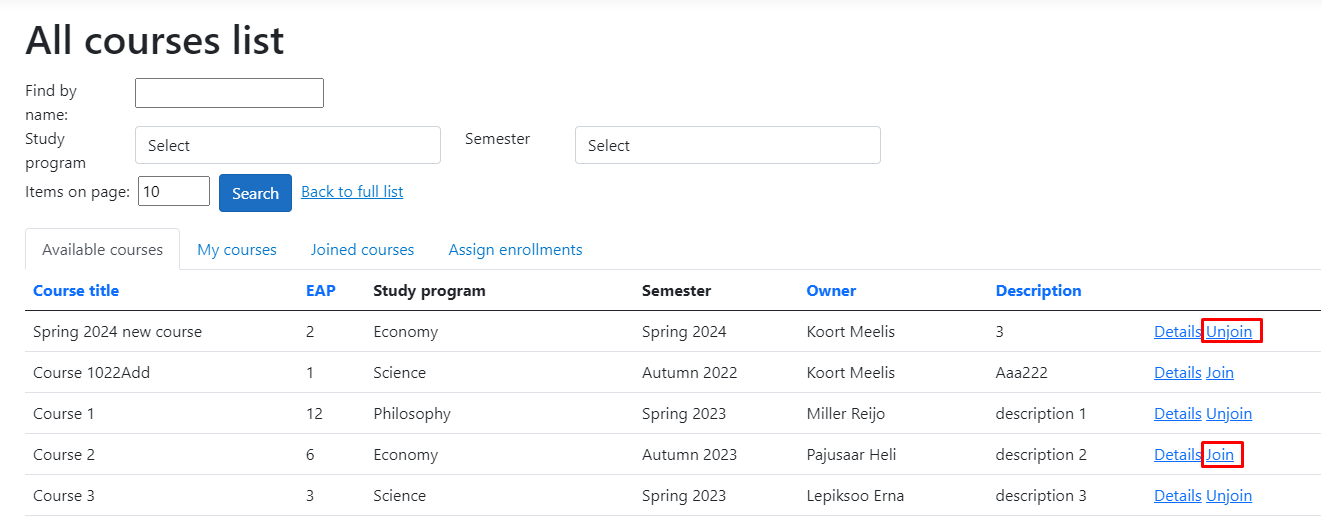
Selle kursuse käigus me võisime luua ÕISi demo versioon, kus on realiseeritud järgmised funktsionaalsus:

1. Rollide (admin, student, lecturer) valimine ja konkreetse kasutaja rolli valimine, et oli võimalus vaadata, kuidas toimub süsteem erinevates kasutajates ühe rollidega (näiteks, kuidas süsteem toimub erinevates tudengites). Samuti Erinevatel rollidel on juurdepääs erinevatele lehtedele, näiteks lecturerile on lubatud hindade pakkumine, aga student ei saa seda teha. Samti lecturer ja student ei saa luua uued lectureri ja studendi kontosid.

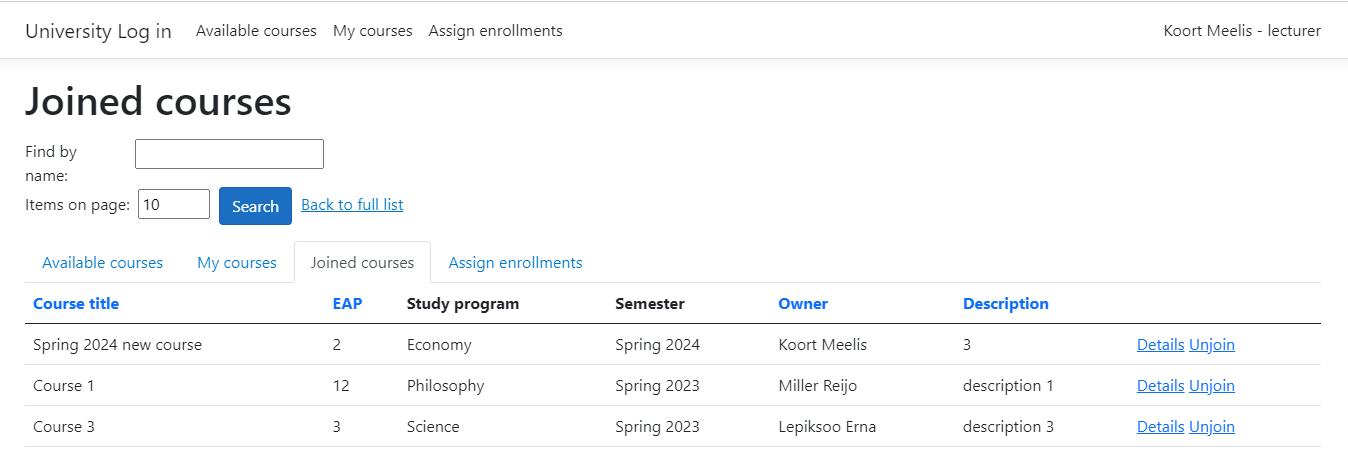




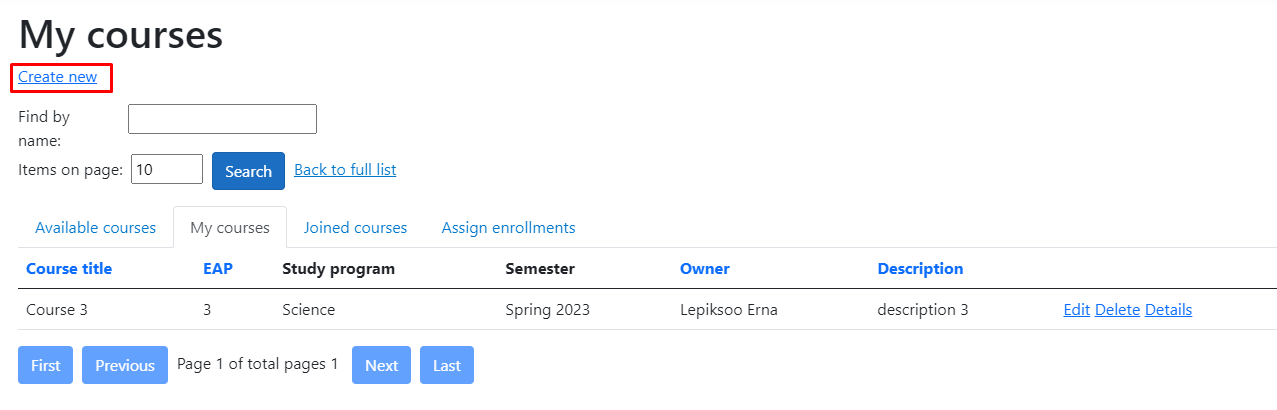
1. Lecturer kontol on juurdepääs järgmistele lehtedele, mis on kujutatud pildil:
   1. Available courses - siis on kujutatud kõik kursused, mis on süsteemis. Igal kursusel on võimalus näha “Details” kõigi selle kursuse andmetega ning kõik Lecturid selles kursuses, ning lisaks on võimalus teha “Join to course”, kasutades nupp “Join”. kui lecturer on osaleb kursuses, siis on näidanud nupp “Unjoin” selleks et lahkuda kursusest.



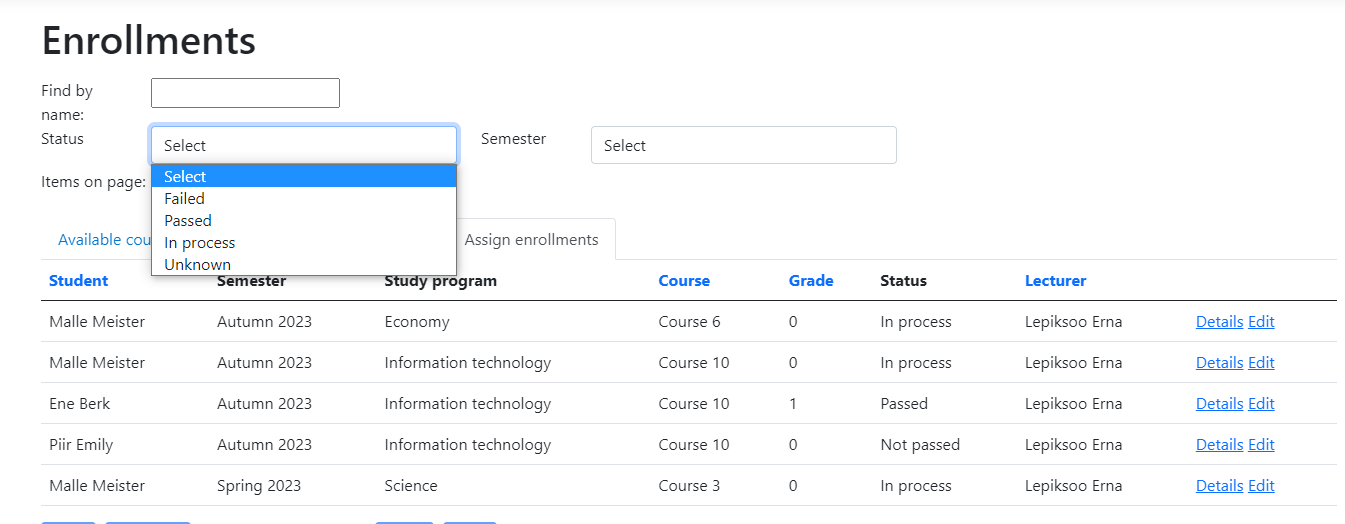
* 1. Lehel “Joined courses”, kus on näidanud need kursused, kus osaleb lecturer



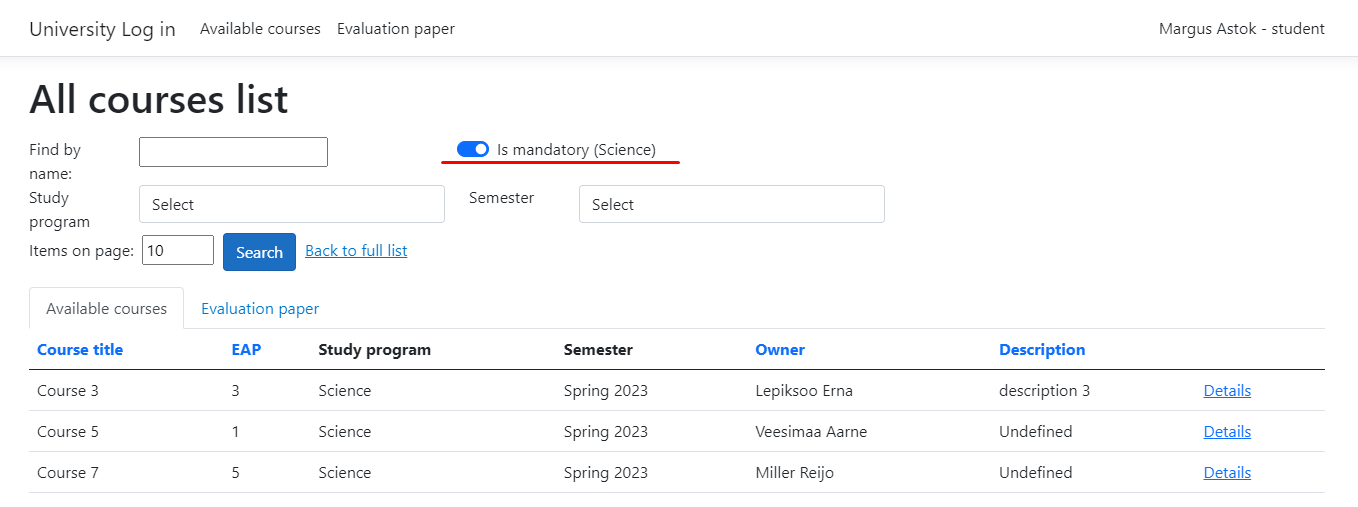
* 1. Lehel “My courses” on näidanud kursused, kus lecturer, mis on süsteemis praegu, on omanik, ehk mis ta lõis ise. Samuti sellel lehel on juurdepääs võimalusele luua uusi kursusi (teistel lehtedel seda pole võimalik teha).



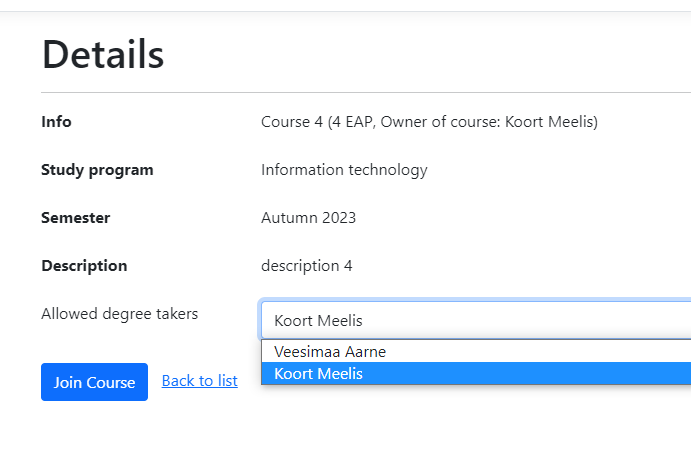
* 1. Lehel “Assign enrollments” lecturer võib näha kõik oma studendid, on võimalik filtering hinde statuse järgi ning panna hindeid tudengitele. Loomulikult Student samuti näed oma hind ja hinne status oma kontos.



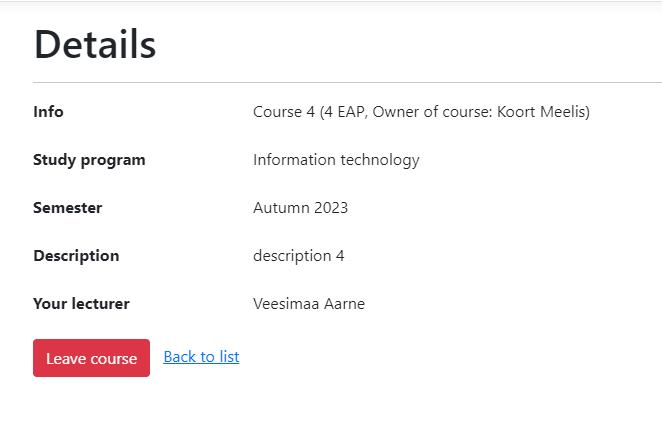
1. Student kontos on juurdepääs järgmistele lehtedele, mis on kujutatud pildil:
   1. Lehel “Available courses” tudeng näed kõik kursused süsteemis. Lisaks ta võib filtreerida neid Studieprogram ja Semestri järgi ning lisaks näidata neid kursused, mis on kohustuslikud konkreetse tudengi, mis praegu on süsteemis (Is mandatory course). Kui vajutada nuppu “Details” siis tudengil on võimalus lisada konkreetsetele lecturer’ile või vastupidi lahkuda kursust.



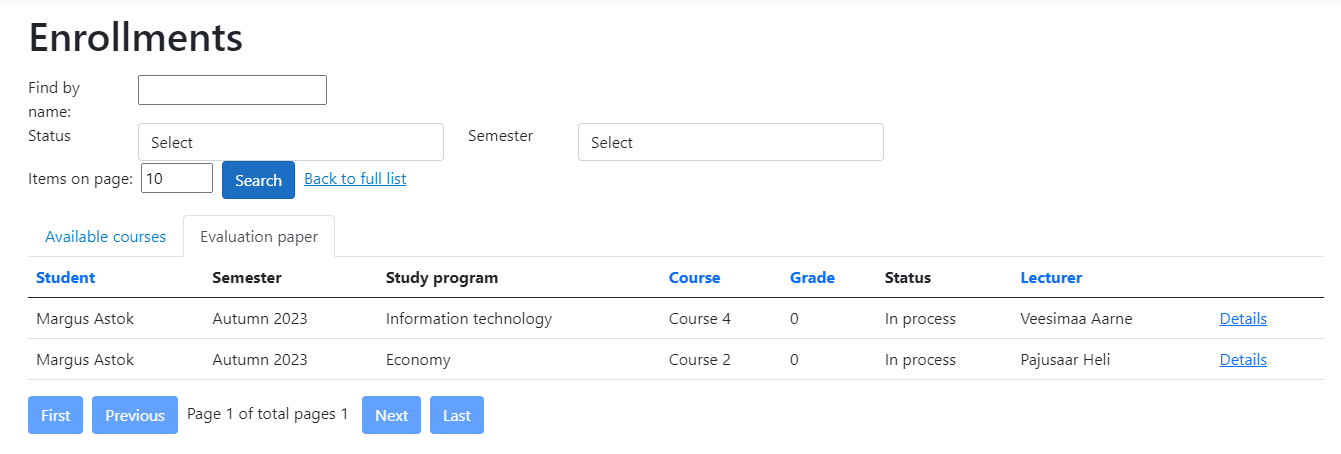
* 1. Kui tudeng pole kursusel, siis võib valida õppejõudu, kes juhib see kursus.



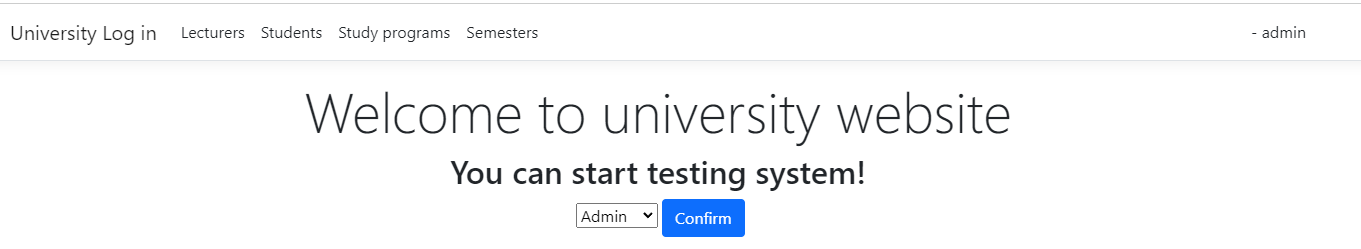
* 1. Kui tudeng pole kursusel, siis võib lahkuda kursust.



* 1. Lehel “Evaluation paper” on näidatud tudengi hinned, on võimalus filtreerida oma “evaluations” statusi ja semestri järgi. Vajutades nuppu “Details” võib lahkuda kursust.



1. Admin võib lisada süsteemi uued õppejõudu, tudengeid, studyprogramms ja semesters.



## Olulised järeldused

### Kasutatud mustrid ja nende põhjendus

Koodi põhistruktuuriks ja kujunduseks võeti loengutest saadud kood, kuna see vastab "Clean code" nõuetele. Allpool on kokkuvõte projekti põhikujundusest, millisteks osadeks see on jagatud ja kuidas need on omavahel seotud.

Projekt on jagatud 8 osaks, mis mängisid suurt rolli projektis:

**Aids**

Kaust *Aids* sisaldab abifunktsioone, mis on salvestatud nende staatilisse klassi, kuna hiljem pole projektis vaja selle klassi eksemplare luua, neid on vaja ainult teatud funktsioonide sisaldamiseks. Näiteks klassi *Lists* loomisel mõistetakse, et see sisaldab ainult funktsioone, mis suhtlevad otseselt mis tahes tüüpi objekte sisaldava lehega. Projektis on 10 sellist abiklassi: Chars, Enums, GetAssembly, GetNamespace, GetRandom, Lists, Methods, Safe, Strings, Types.

**Data**

Kaust salvestab need klassid, mis loovad andmebaasis mallid tulevaste tabelite jaoks. Puhta koodi reegleid järgides loodi ka abiklassid, mis salvestavad kõikidele tabelitele ühiseid välju. *UniqueData* sisaldab absoluutselt kõiki klasside ainsat unikaalset ID-välja. ID automaatseks loomiseks kasutatakse Guid Object ja selle meetodit *NewGuid*(), mis genereerib sarnased võtmed: *0f8fad5b-d9cb-469f-a165-70867728950e*. Seega lahendatakse hilisemad vead sama ID-ga kirjete lisamisega andmebaasi. *NamedData* on suurem abstraktsioon, kuna see on mõeldud tabelite jaoks, mis salvestavad kasutaja jaoks andmeid. Neid iseloomustavad järgmised väljad: Eesnimi, Perekonnanimi, Unikaalne ID, Sugu. Selle klassi pärivad *LecturerData* ja *StudentData*. Samuti jagati klassid õigel tasemel struktuuri hoidmiseks 2 kausta: Party, mis salvestab konkreetseid andmeid rakenduse osa kohta (näiteks õppejõud (*LecturerData*), programmid (*StudyProgramData*) või kursused (*CourseData*)) ja Ühendus, mis sisaldab Partei linkide tabeleid ID järgi (näiteks *CourseLecturerData* sisaldab õppejõudude ID-sid ja kursusi, milles nad osalevad). Samuti otsustati luua kaust *IsoEnums*, mis sisaldab enum objekte, mis võimaldavad kuvada andmeid kasutajasõbralikul kujul (näiteks *IsoSemester* sisaldab semestrite nimetusi Sügis, Talv, kuid salvestatakse ainult nende semestrite nimedele vastavad numbrid andmebaasis)

**Domain**

*Domain* on koostatud 2 osast. Esimene osa on seotud *Data* classidega, ehk räägib, mis andmed saatakse *Data*’st ja mida on vaja nendega teha, et õigesti pakutada neid *View* jaoks. Samuti Domain classid sisaldavad teist *Doamin*’i classid, et mingi andmed võtta teistest tabelitest. Näiteks *Course* sisaldab property OwnerID (see on Lecturer) ja selleks, et saada selle omaniku andmed on kirjutatud *Lecturer* property, mis võtab *LecturerData*’st need andmed (näiteks omaniku (lecturer) nimi või sugu), kasutades static meetod GetRepo. *Domain* lisaks sisaldab IRepo, interface, mis ütleb kohustuslikud meetmed iga Repo jaoks. IRepo sisaldab üldised kohustuslikud meetmed (CRUD), ja samuti iga spetsiifiline Repo sisaldab oma interface Repo (näiteks Couse sisaldab ICoursesRepo).

**Facade**

Facade’s põhimõtteliselt on need classid, mis ütlevad mis andmed oleks näidatud kliendi lehel. Need andmed on võimalik olla rohkem, kui Domain classis, aga igal juhul seal on sarnased property. Samuti nimelt Faсade’s tehakse validation iga property jaoks, mis teade näidatakse, kui valesti täidetakse formi (näiteks Create või Edit lehel). Lisaks igal klassil on oma *Factory* class, mis sisaldab meetod tegemiseks Domain into View või vastupidi.

**Infra**

Infra juba kirjeldab, mida täpsemalt tähendavad need Repo, mille interface on loodud Domain’is. BaseRepo’s tehakse ühendust DataBase’le ning kirjeldatakse abstract meetmed (CRUD), mis juba kirjutatakse üle CrudRepo`s. Kõikidel nendel meetmetel on sarnased osad: kõik need võtavad Domain objekti, et näiteks seda kirjutada Databaasisse,või vastupidi, pakutavad andmed selles formaadis. Lisaks on vaja märkida, et kõik osad Repost on seotud ühe vajaliku meetmega - createSql. Iga Crud meetod (need, mis võtavad DataBaasist andmed) kasutab createSql et pakutada andmed kliendile. Seda meetod koostatakse iga Repo klassi osadest. Alustatakse Sql request standaartselt “from s in set select s”, mis räägib, et võtta kõiki andmed data baasist (set on konkreetne databaasi tabel). Pärast, iga osa Repost lisab oma tingimused selle request’ile, näiteks PageRepo ütleb, mitu on vaja maha võtta (Skip(num)) ja mitu elementi võtta (Take(num)), FilterRepo lisab tingimused, kui klient sooviks filtreerida andmed näiteks mingi tingimuse järgi (meie programmis on realiseeritud filtreering näiteks kohustuse kursuse järgi, ning kui seda on vaja, siis lisatakse tingimus kasutades Where(tingimus Expression)). Samuti oli lisatud SessionRepo, mis ütleb Repo jaoks, mis User praegu on süsteemis ja mis on selle User ID ja Roll. Need andmed on vaja selleks, et teha automaatiline andmete filtreerimine, näiteks, kui Roll on “student”, siis lehel “enrollments” on vaja näidata need enrollments, mis kuulutatud sellele UserID jaoks. Samuti Infras on kirjutatud Initializers classid, mis täiendavad Databaasisse standardsete andmetega, kui konkreetne tabel on tühi.

**Pages**

Pages koosneb põhimõtteliselt sarnasest klassidest, mis oli Infras. Aga erinevuses Infrast, Pages juba on otseselt seotud kliendiga (kliendi tegevusega lehtedel). Pages tegeleb klientide GET-i päringutega kasutades handlereid (Näiteks OnGetIndexAsync), ning iga meetod sisaldab parameetrid, mis on võimalik saada urlist nagu GET parameteriks. Pärast nende saamiseks need on vaja edasi andma Reposse ning selleks oli loodud setAttributes meetod, mis seotab GET parameetrid Repo property’ga, et näiteks teha filtreering või sorting. Samuti nimelt siin tegeletakse sellega, et kasutada loodud Infras CrudRepos meetmed andmete saamiseks, ning pärast nende saamiseks nimelt Pages CrudPage’s need saadud andmed formaadis Domain konverteeritakse View formaadisse, et juba need näidata lehel. Lihtsalt öeldes, Pages on vaja selleks, et teha ühendus kliendi ja databaasi vahel.

Nagu on juba öeldud, Pages sisaldab kõiki, mis on seotud kliendi osaga, ning sellele ka kuulutatakse html elemente. Selleks, et need genereerida automaatselt ja vältida code duplication in cshtml main Project folders (Project\_1 folder), oli loodud staatilised meetmed, mis genereerivad näiteks html struktuur tabelite jaoks, Drop Down listi jaoks, jne. Kõik need meetmed on juba kasutatakse Project\_1.

**Project\_1**

Selles folderis on loodud kõik need, mida näeb klient lehtedel. Iga kaust on selle page root ning leht on juba fail, mis selles kaustas on. Samuti, et vältida code duplication oli

loodud shared osad, mis sisaldava sarnased osad nag html klasside nimetused, mõned

html tags jne. Igal selle shared failis on olemas meetod RenderBody(), kuhu

kompilaator täiendab selle html code, mida kirjutatakse juba spetsiifilis failides.

Selleks, et kuskil kasutada seda shared faili struktuuri, on vaja spetsiifilisse property

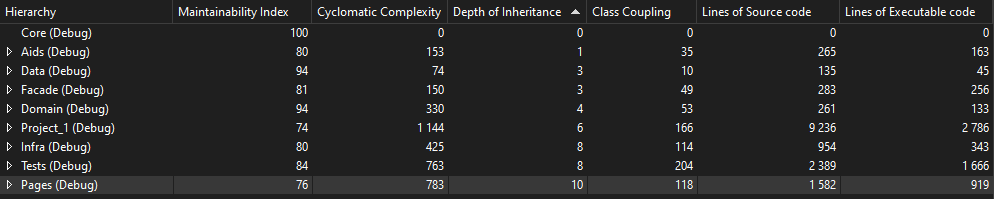
Layout (mis võetakse PageModel classist) täiendada selle shared failinimi. Samuti

projektis tihti kasutatakse partial tags, et vältida code duplication ning teha koodi

rohkem loetavamaks.

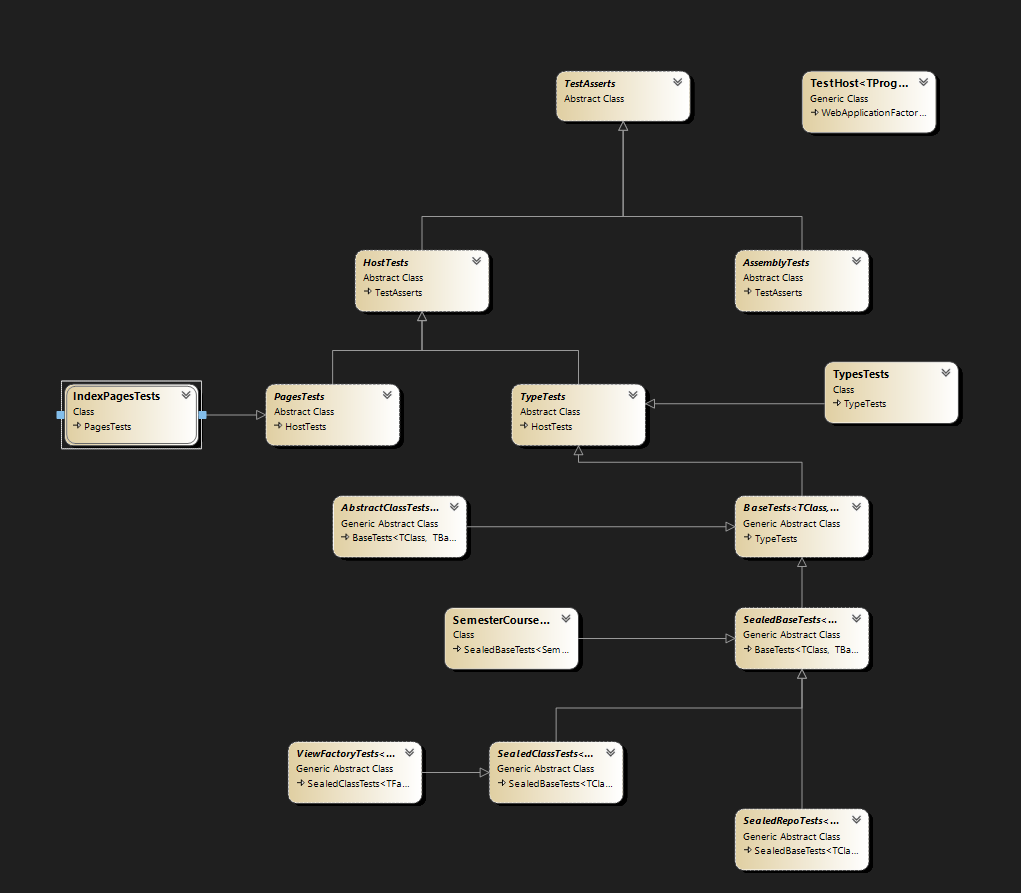
### Koodi kvaliteet ja põhjendus

Koodi meetrika järgi on projektis 5554 koodirida. Kõige mahukamad koodi ridade arvu poolest on Project\_1 2023 rida ja Pages 925 rida. Kokku on projektis 10 tabelit University andmebaasis. Vigu ei esine, hoiatused on viidud miinimumini.



### 

### Testimine

**Tests Hierarchy:**

Koodi testimiseks kasutati ühikteste, mis testisid üksikuid koodi osasid. Kokku tehti 600 ühiktesti, koodi katvus ühiktestidega on umbes 70% (v.a. Pages and Soft). Kuna meeskonna kogemus testimisega oli enne projektiga alustamist väga väike, siis selle tõttu ei jõudnud meeskond rohkem teste teha. Kuna projekti käigus oli rõhk rollide ja täiustatud filtrite loomisel, siis aja piiratuse tõttu pühendati testidele vähem aega, puudus ka võimalus lehekülgi kontrollida, kuna testidele (Vastuvõtu testid) ei antud ligipääsu rollide tegemise tõttu.

### Edasised tööd

Kuna projekti loomiseks oli piiratud aeg, siis kõigepealt edasine töö koosneb meie projekti parandamiseks. Alla on pakutud need parandused:

* Realiseerida paralleelne andmete kutsumine tabelitest, millega on seosed (näiteks, kui kutsutatakse student, siis kustutada kõik enrollments temaga seotud)
* Lehtede testid (praegu ei ole juurdepääs lehtedele)
* Süsteemi optimeerimine (et kõik töötas kiiresti)
* Lisada teised filtrid lehtedele (näiteks filtreerimine sugu või vanuse järgi)
* lõpuni parandada need olukorrad, kui süsteemis on palju kasutajaid (et kui ühele databaasi elemendile kontakteeruvad mitme kasutajat)
* Normaalne authoriseerimine rollide valimisega ning email kinnitamisega (praegu on tehtud demo versioon, kus avalehel on võimalus valida rolli ja userID)

Kui kõik ületoodud parandused on tehtud, siis on kavas teha meie projekti Core, et teha edasist laiendamist lihtsamaks ja tõhusamaks.

## 

## Kokkuvõtte

Antud projekti eesmärgiks on praktiliselt rakendada baaskursust õpituid teadmisi ning omandada uusi programmeerimise alaseid kogemusi ja oskuseid veebirakenduse tegemisel ning harjutada meeskonnatööd. Probleem, mida antud töö käsitleb, on Eesti ülikooli süsteemi analoogi moodustamine ja rakenduse lihtsustamine. Meie projektis on realiseeritud põhimõtteliselt kõik funktsioonid mis peavad olema, et tagada hea vastastikmõju ülikooli eluga. On võimalus tudengitele registreerida semestrile, õppekaval, kui õppejõududele luua uusi kursusi ja panna hinded nende eest.

Töö tegemisel lähtub meeskond puhta koodi põhimõtetest, tarkvara arendatakse Visual Studio’s ning koodi jagamiseks kasutatakse GitLab’i. Diagrammide koostamiseks kasutatakse Visual Studio ja Creately app. Suhtlus meeskonnaliikmete vahel toimub valdavalt Microsoft Teams’is ning dokumentatsioon koostatakse Google Docs’i abiga. Tööülesanded olid jaotatud võrdselt ning kõikide meeskonnaliikmete panus nii tarkvara arendamisesse kui ka dokumentatsiooni on võrdne, ning koodi kirjutamine toimis paaris.

Tehtud projekti demo versioon edukalt täidab püstitatud eesmärgid, sest projekti funktsionaalsus võimaldab minimaalselt teha ülikooli raamatupidamist erinevate kasutaja tüübiks (admini, õppejõu ja tudengi poolest) ja vastab minimaalsetele ohutusnõuetele (näiteks tudeng ei lisada uusi tudengeid, seda võib teha ainult admin). Loomulikult, süsteem on projekteeritud nii, et oleks lihtsam laiendada süsteemi nii funktsionaalsuse kui ka testimise poolest.

Üldiselt on meeskond rahul oma tööga, sest vaatamata sellele, et enne meil puudus kogemus töötamises meeskonnas, arvame, et suutsime vastutus ratsionaalselt jaotada ning tõhusalt projekti kallal töötada. Eriti hästi meil oli uusi lahendusi otsida või uusi ideid arutleda ja nende integreerimise kava koostada. Meeskonnaliikmed hindavad teineteist “väga heaks” ning töö meeskonnas läks piisavalt nii, nagu kavatsesime. Muidugi need saadud selles semestris teadmised on kasulikud (eriti praktikal) ja loomulikult on kasutatud meie tuleviku projektides.