CHALMERS

SchoolIT - En lärplattform i Java EE

Examensarbete inom Data- och Informationsteknik

Albin Becevic

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige, 2022

EXAMENSARBETE

En webbaserad lärplattform utvecklad med olika mjukvaruspecifikationer inom Java EE

Albin Becevic

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, 2022

En webbaserad lärplattform utvecklad med Java EE

ALBIN BECEVIC

Examinator: Jonas Duregård

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik
Göteborg 2022

Sammanfattning

För att underlätta kommunikation mellan elev och lärare utnyttjar de flesta akademiska institutioner läroplattformer som är en webbaserad kursmiljö för både elever och lärare. Målet med projektet var att utveckla en lärplattform på gymnasienivå utifrån de olika mjukvaruspecifikationerna som ingår i Java EE (Java Enterprise Edition) som är Oracles egna samling av mjukvaror för att distrubera företagsövergripande webbapplikationer. Den utvecklade webbapplikationen använder sig av Apache Derby som databasehanteringssystem tillsammans med Java Persistence API för att interagera med den. För användargränsnitt används Java Server Faces tillsammans med primefaces komponent bibliotek. Java EEs Context och Dependency Injektion teknologi används för instansiering av objekt och Enterprise Java bön teknologi för att hantera affärslogik inom webbapplikationen. Resultatet blev en funktionell webbaserad lärplattform där lärare kan skapa en kurs med en kursplan som en elev kan gå med i och se kursplanen.

Nyckelord: Java EE, Läroplattform, Företagsövergripande webbapplikationer

Abstract

Inorder to facilitate communication between student and teacher most of the academical institutions utilize learning management platforms which in itself is a web based course enviroment for both students and teachers. The purpose of this project was to develop a learning management platform using the set of software specifications that is included in Java EE (Java Enterprise Edition) which is Oracles own collection of software used to distribute Enterprise application software. The developed web application uses Apache Derby as the database management system together with Java Persistence API to interact with it, Java Server Faces as user interface together with Primefaces component library, Context and Dependency Injection for instantiation of objects within the application and Enterprise Java Bean Technology for handling business logic within the application. The result became a functional web based learning management system were teachers can create a course with a syllabus and a student can join and look at the syllabus.

Förord

Jag skulle vilja rikta ett stort tack till Sakib Sistek för sitt otroliga stöd och vägledande under denna resan.

Innehållsförteckning

1 Introduktion **9**
- 1.1 Bakgrund . . . 9
- 1.2 Syfte . . . 9
- 1.3 Mål . . . 9
- 1.4 Avgränsningar . . . 1

2 Teknisk bakgrund **2**
- 2.1 Utvecklingsmiljö . . . 2
- 2.2 Maven . . . 2
  - 2.2.1 POM - Project Object Model . . . 2
- 2.3 Företagsapplikationer (Enterprise applications) . . . 2
- 2.4 Fler-nivå Webbapplikation . . . 3
- 2.5 Java EE Server . . . 3
  - 2.5.1 Payara Server . . . 4
- 2.6 JPA - Java Persistence API . . . 5
- 2.7 Crow's Foot Notation . . . 5

3 Metod **6**
- 3.1 Undersökning . . . 6
- 3.2 Planering . . . 6
- 3.3 Arbetsgång . . . 6
- 3.4 Implementering . . . 6

4 Systemkonstruktion **8**
- 4.1 Förberedelse av utvecklingsmiljön . . . 8
  - 4.1.1 Ansluta Server . . . 8
  - 4.1.2 Anslutning av Databas . . . 8
  - 4.1.3 Persistence enhet . . . 10
- 4.2 Affärslogiknivån . . . 10
  - 4.2.1 Databas . . . 10
  - 4.2.2 Modellering av Databasen . . . 11
  - 4.2.3 Lagring och Hämtning av Databasen . . . 12
  - 4.2.4 Testning av Databasen . . . 14
- 4.3 Webb Nivån . . . 16
  - 4.3.1 Java Server Faces . . . 16
  - 4.3.2 PrimeFaces . . . 17
- 4.4 Kommunikation mellan Webb - och Affärslogiknivån . . . 18
  - 4.4.1 Skapa Användare . . . 18
  - 4.4.2 Inloggning . . . 23
  - 4.4.3 Lärar instrumentbräda . . . 27
  - 4.4.4 Skapa kurs . . . 29
  - 4.4.5 Lärar kurssida . . . 31
  - 4.4.6 Student instumentbräda . . . 32

4.4.7 Student kurshemsida 34

5 Resultat **38**

6 Diskussion **40**
6.1 Utvärdering av metod 40
6.2 Utvärdering av Design 40
6.3 Utvärdering av Funktionalitet och Vidarutveckling 40
6.4 Utvärdering av Java EE och Implementering 41
6.5 Möjligheter att appliceras på mobil plattform 42
6.6 Miljö och Etiska frågor 42

1 Introduktion

Följande sektioner kommer att introducera projektet samt även motivering och syfte som resulterade i det grundläggande konceptet angående projektet.

1.1 Bakgrund

I gymnasiet finns en mängd av kurser som går parallelt vilket leder till att det skapas mycket material som en elev måste ha översikt över och hantera. För en vanlig gymnasielev kan det utgöra en stor börda för att navigera, organisera och strukturera sitt arbete under dessa förhållanden.

Med digitaliseringen har man utvecklat olika lärplattformar i syfte att underlätta undervisning och elevernas inlärning. En lärplattform, eller ”learning managment system” på engelska, är en webbaserad kursmiljö skapad för att underlätta kommunikation mellan lärare och elever. Canvas, moodle, och google classroom [1, 2, 3] är några av de mest förekommande.

Idag använder majoriteten av gymnasieskolor google classroom [4, 5, 6], vars ändamål är att skapa mer gynsamma förutsättningar för snabb och korrekt distribution av undervisningsmaterial från läraren till elev. Många goda aspekter finns i de nuvarande lärplattformerna och erbjuder lärare en rik mängd funktionalitet och frihet för att göra just detta. Nackdelen är att det lägger ett krav på en lärares informationshantering och tekniska kompetens, vilket kan utgöra en utmaning för lärare att designa och organisera sin utlärning som är lämpligt och tydligt för eleverna [7]. Vilket är väsentligt för att ge elever en möjlighet att effektivt erhålla sin inlärning.

1.2 Syfte

Syfte med arbetet är att utveckla en lärplattform utifrån de olika mjukvaruspecifikationerna som ingår i Java EE och samtidigt utvärdera de som appliceras. Plattformens funktionalitet kommer inspireras utifrån redan etablerade lärplattformar och försöka implementera de som anses gynna elever och lärare för översikt av diversa kurser, samt distrubering av kursmaterial.

1.3 Mål

Målet med projektet är att utveckla en funktionell lärplattform genom att undersöka och tillämpa de olika mjukvaruspecifikationerna inom Java EE. Ta reda på vad för funktionalitet oftast utgör lärplattformar för att i sin tur reda ut vilka tekniska aspekter krävs för att skapa och underhålla en lärplattform. Fortsättningsvis undersöka vilka mjukvaru specifikationer inom Java EE som kan anpassas för att stödja de aspekterna. Skapa en UI/UX design med hjälp av användarresor konstruerade med olika intressenter som översätts till use-cases. Bryt därefter ner projektet i olika ”user stories” och tillämpa scrum

metodiken för att erhålla översikt och spåra framsteg. Utforska avslutningsvis vilka möjligheter den utvecklade plattformen har att appliceras på andra plattformar.

1.4 Avgränsningar

Projektet kommer att inrikta sig på mjukvaruutveckling och de olika metoder tillämpade i anknytning till det. Utvecklingen kommer därför att ske enbart utifrån de olika mjukvaruspecifikationerna inom Java EE.

2 Teknisk bakgrund

I den här sektionen kommer de övergripande tekniska aspetkerna att tas upp för att underlätta förståelsen för implementeringen av projektet. Dessutom introduceras de flesta tekniska begreppen som upprepas igenom rapporten.

2.1 Utvecklingsmiljö

Apache Netbeans IDE valdes som utvecklingsmiljö. Den används primärt för att skapa Java plattformer i form av Webb, företags och mobila applikationer. Netbeans är en integrerande utvecklingsmiljö vilket innebär att den förser utvecklare med standard inbyggda program och programmeringsverktyg, som t.ex textredigerare, kompilator och avlusare (Debugger), för att underlätta utveckling av programvaror [8]. Liknande och kapabla utvecklingsmiljöer existerar som t.ex Eclipse och IntelliJ. Varje utvecklingsmiljö har självklart sina styrkor och svagheter i jämförelser med andra men i slutändan faller valet i smak. Netbeans stödjer dessutom full integration med Maven och Payaras applikations server som även kommer att användas genom utvecklingen.

2.2 Maven

Maven är ett projekthanteringsverktyg (project management and comprehension tool) vars uppgift är att bygga och underhålla Java projekt. Det här gör den genom dokumentation, hämtande och byggandet av olika externa beroenden (dependencies) [9]. Maven baseras på POM (Project Object Model) som är en XML-baserad konfigurationsmodell. Maven underlättar utvecklingen och förståelse för Java projekt och kommer att användas för att hämta externa beroenden inom projektet.

2.2.1 POM - Project Object Model

POM är en grundläggande arbetsenhet inom Maven. Det är en XML-fil som innehåller all information och konfigurationsdetaljer över hur projektet ska byggas [10]. XML-filen innehåller bland annat information om externa beroenden som används inom projektet.

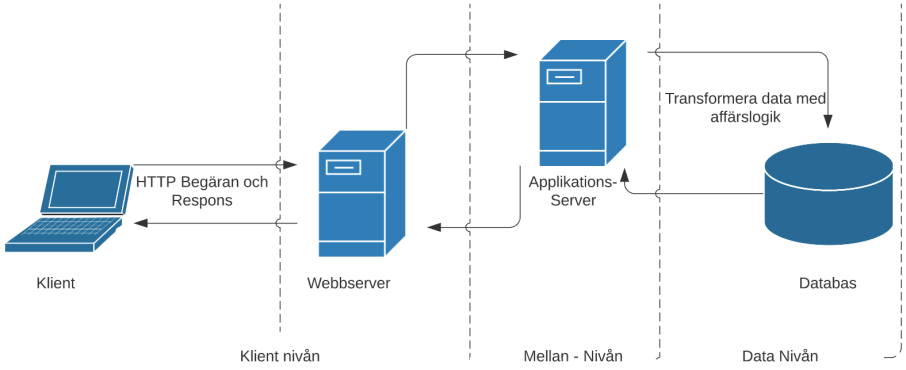
2.3 Företagsapplikationer (Enterprise applications)

En företgasapplikation (enterprise application) är en applikationsmjukvaru som är byggd för att underhålla storskaliga företag och lösa de problem som oftast de påträffar. Exempel på storskaliga företag är banker, byråer, regeringar och större företag [11]. Applikationsmjukvara bidrar med affärslogik och diverse olika verktyg för att hantera de olika processer som sker i ett storskaligt företag som t.ex faktureringssystem, e-handel, redovisning och banksystem.

Företagsapplikationer kräver att de är pålitliga, säkra och skalbara vilket är en bidragande faktor till ökad komplexitet för utvecklingen av denna form av applikationer. Java EE med bidrar med olika verktyg och API:er som hjälper till att hantera den komplexiteten och samtidigt möta de specifierade kraven i sin egna körningsmiljö, vilket leder till att utvecklare kan fokusera mer på funktionalitet än de underliggande systemkrav som t.ex distruberad databehandling och andra webbtjänster [11].

2.4 Fler-nivå Webbapplikation

I en fler-nivå webbapplikation (multi-tiered application) är applikationen indelad i olika funktionella delar så kallade nivåer: klientnivån, mellannivån och datanivån. Processen ser ut som följer. På klientnivån begär klienten åtkomst till servern. Klientens begäran skickas sedan till mellannivån. Här hanteras klientens begäran och bearbetar applikationens data. Ifall data behöver lagras permanent i applikationen görs det i datanivån[12]. JavaEE är försedd med diversa olika teknologier för att hantera mellannivån i en traditinell fler-nivå webbapplikation.



Figur 1: Arkitektur för Fler - nivå webbapplikation.

2.5 Java EE Server

Java EE servern är en applikationserver som bidrar med de olika API:erna och tjänsterna inom Java EE plattformen [12]. Arkitetkturen för Java EE servern är uppdelad som följer:

- **Klientnivån** består av olika applikationsklienter som får åtkomst till JavaEE servern. Klienterna skickar olika begäran till servern, där deras begäran bearbetas. Servern ger därefter respons tillbaka till klienten. Klienterna lokaliseras oftast på andra maskiner och kan ta kontakt med servern via en webbläsare eller en annan slags applikation[12].
- **Webbnivån** sköter själva interaktionen mellan klientnivån och affärslogiknivån. En av uppgifterna är att hantera input (data) från användaren via klientens

gränsnitt och sedan retunera ett resultat utformat av komponenterna i affärslogiknivån[12].

- **Affärslogiknivån** är den nivån där all affärslogik hanteras inom en applikation. Den utgörs av komponenter som har som uppgift att hantera de krävande uppgifterna inom applikationen [12].
- **Informationshanteringsnivån** är den nivå där databasen lokaliseras för att sköta lagring av data. Kommunikationen med databasen hanteras av de komponenterna som som ligger i affärslogiknivån[12].

I Java EE - applikationsutveckling läggs det fokus på utveckling av mellannivån som är uppdelad i webb och affärslogiknivån. Varje nivå i servern hanteras av olika containrar för att underhålla de olika komponenterna inuti de olika nivåerna. De tre olika containrarna är Webb container, EJB container och Applikationsklient container [13].

En **container** är själva gränsnittet mellan komponenterna i webbapplikationen och dess låg-nivå funktionalitet. Som nämndes tidigare kräver en fler-nivå webbapplikation en mängd komplex funktionalitet. Java EE låter containrarna hantera dessa tjänsterna och även ger tillgång till de olika API:erna inom Java EE. Genom Java annotationer specificeras vad för komponenttyp sådan att containern kan hantera komponenten korrekt [13].

Webb Container sköter de olika webb komponenternas exekvering. En webb komponent kan exmpelvis vara en servlet (en java klass som lyssnar på olika nätverks begäran t.ex HTTP) eller en webbsida som en Java Server Faces vy [13].

EJB (Enterprise Java Böna) containern hanterar och exekverar de olika Java Enterprise bönorna i applikationen. [13]. Via Java annotationer specifierar man vad för slags EJB komponent det är som exekveras sådan att containern kan hantera den.

Applikationsklient containern hanterar de Java EE komponenterna i en Java applikation som körs på en klients maskin. Containern öppnar kommunikation med servern och applikationen [14].

2.5.1 Payara Server

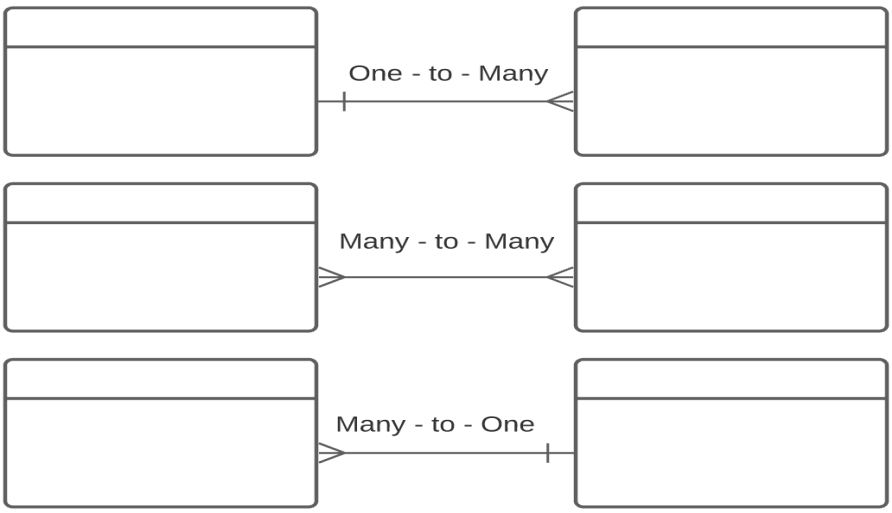
Applikationsservern som kommer att användas till projektet är Payara server. Det är en öppen mjukvaru Java EE applikationsserver som härstammar från Oracles egna Java EE applikationsserver Glassfish [15].

2.6 JPA - Java Persistence API

Eftersom Java är ett objektorienterat språk är det inte möjligt att direkt lagra objekten i en relationsdatabas. Java EE förser ett gränsnitt för att lösa det problemet. Java Persistence API (JPA) är ett gränsnitt som applicerar ORM (Object relational mapping). ORM är en teknik för att konvertera data för inkompatibla system med användning av objektorienterade språk. Med hjälp av Java annotationer kan man definera hur databasen ska vara strukturerad. Med hjälp av JPA kan man definera de olika tabellerna och hur de olika relationerna mellan de ska vara kopplade. Via Java annotationer kan man definer JPA entiteter som representerar en tabell i databasen.

2.7 Crow's Foot Notation

Crow's foot notation är ett ER-digrams notation för att definera de olika relationerna i ett ER-diagram. Notationen utnyttjades för att designa databasen. Ett exempel över relationerna illustreras i figur 2.



Figur 2: Crow's foot notation över ER relationer

3 Metod

Projektet delades in i olika faser för att kunna strukturera och följa en drivande arbetsgång under projektets utförande.

3.1 Undersökning

Det började med undersökningar för att ta reda på och skapa en lista på vad för funktionalitet dagens lärplattformar utgör samt även få inspiration från liknande platformar. Dessutom få en uppfattning om vilka funktionaliteter anses vara mest användbara, grundläggande och väsentliga för att sedan appliceras/förbättras inom plattformen. De lärplattformerna som låg mest i fokus under undersökning var Canvas, Google Classroom och Moodle. Utifrån dessa skapades en lista över vad som bör inkluderas i applikationen.

3.2 Planering

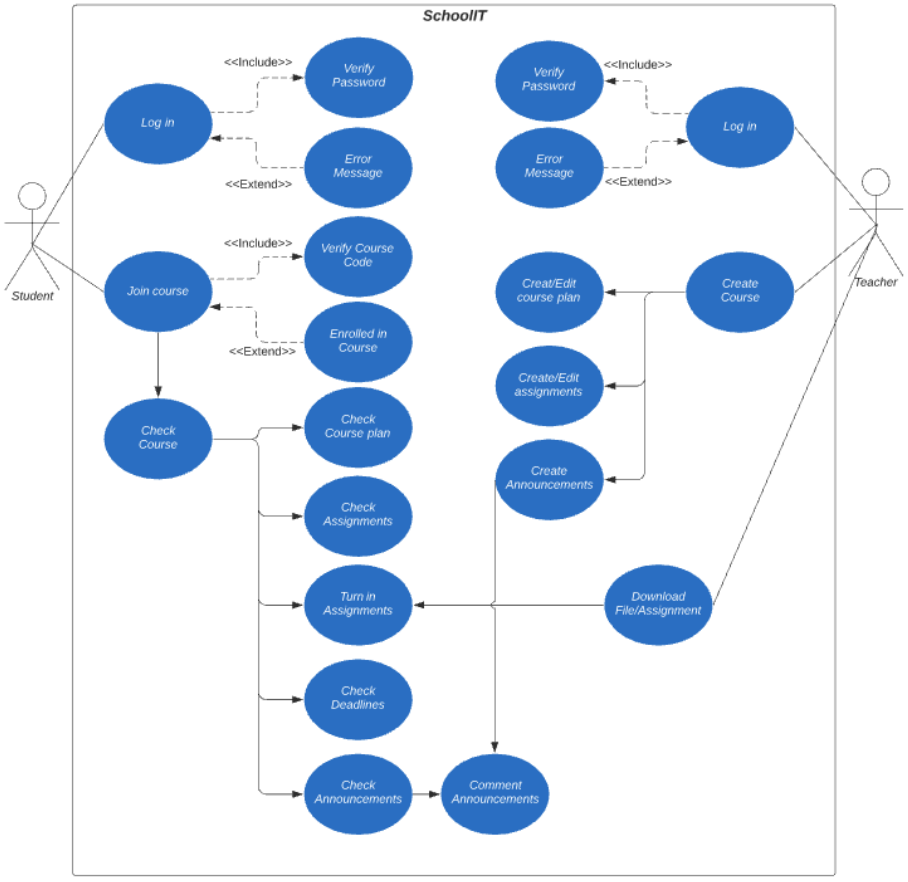
Under planeringsfasen skapades use-case och user-stories för plattformen utifrån grundfunktionaliteten som blev etablerad i undersökningsfasen. Use cases samlades och visualiserades i ett use-case diagram som visar vilka aktörer och användarresor som kommer att finnas tillgängliga i plattformen. Use-case diagrammet visualiseras i figur 3 och visar den etablerade funktionaliteten som eftersträvades. User-stories fylldes sedan in i produktbackloggen. Även ett första UI/UX design koncept togs fram i denna fas.

3.3 Arbetsgång

Arbetsgången som följdes var inspirerat av agilt product utveckling. Där user-stories fyllde produktbackloggen som sedan bröts ner i olika tasks där scrum metodiken applicerades för att hantera och följa de olika uppgifterna. Detta gjordes med hjälp av Trello.

3.4 Implementering

Implementeringen började med att välja vilket arkitekturmönster som skulle följas. Eftersom Java EE är byggd för att utveckla storskaliga webbapplikationer så är den designad för att följa en Fler-nivås arkitektur. Det betyder att det var den självklara arkitekturen att följa. Därefter implementerades funktionaliteten sucessivt.



Figur 3: Use - Case diagram över applikationen

4 Systemkonstruktion

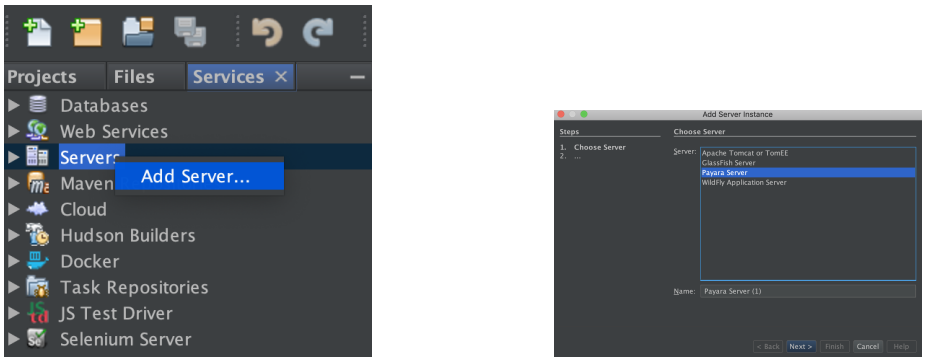
Följande del beskriver hur de olika komponenterna i applikationen implementerades, hur de olika teknologierna i Java EE utnyttjades för att framställa applikationen samt hur förberedelserna för utveckling gjordes.

4.1 Förberedelse av utvecklingsmiljön

Efter installation av utvecklingsmiljön NetBeans var första steget att lägga till applikationsservern Payara och sedan ansluta en databas till projektet.

4.1.1 Ansluta Server

NetBeans erbjuder ett simpelt sätt att ladda ner och aktivera servern. Efter att insallationen är klar kommer NetBeans att ha aktiverat support för Java EE i alla delar av utecklingsmiljön. Nedanför illustreras hur Payara ansluts i Netbeans.



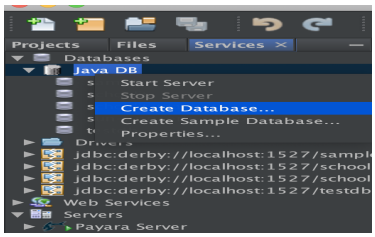
(a) Ansluta server i Netbeans

(b) Välj Payara Server

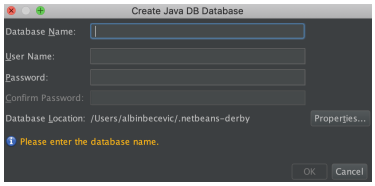
Figur 4: Anslutning av Payara Server i Netbeans

4.1.2 Anslutning av Databas

Efter att servern är ansluten är nästa steg att lägga till och konfigurera en databas till utveklingsmiljön. NetBeans har inbyggt stöd av Apache Derby, en relationsdatabas som är baserad på och stödjer JDBC och SQL standards [16]. Hur det gick till illustreras i figur 5.

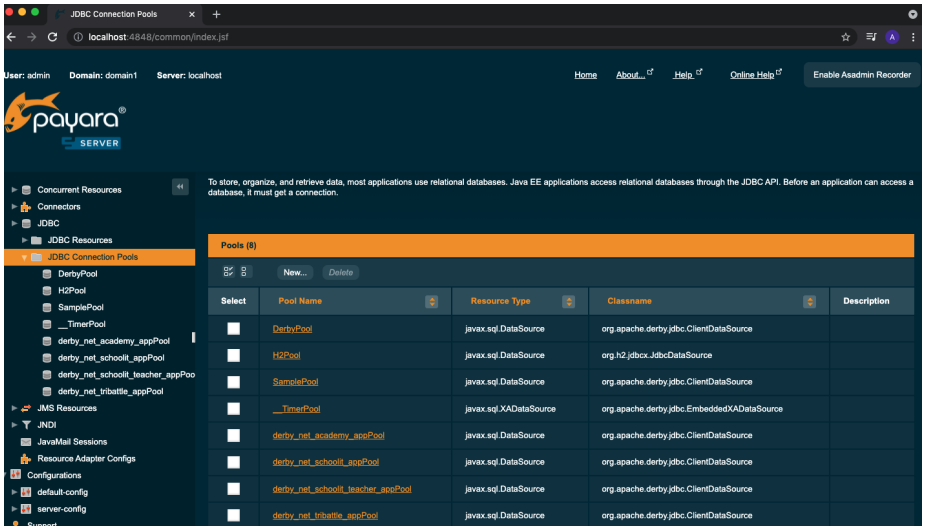


(a) Skapa databas i Neatbeans.



(b) Information som ska fyllas i.

Figur 5: Anslutning av databas.



Figur 6: Anslutning av en connection pool i payaras admin konsol.

Efter att en databas har skapats behöver en anslutningspool skapas och kopplas till databasen. Anslutningspool skapas genom att starta payara servern och sedan via admin konsolen konfigurera samt skapa den. Admin konsolen illustreras i figur 6. Skapning av anslutningpoolen är nödvändigt för att ens kunna få en koppling till databasen och köra applikationen på payara. En anslutningspool bidrar även med en del prestandaförbättringar. En anslutningspool är en lagring av databasanslutningar som kan användas och återanvändas för att ansluta till en databas. Databas anslutningar är kostsamma att skapa och underhålla. En anslutningspool förbättrar ett systems prestanda och skalbarhet [17].

4.1.3 Persistence enhet

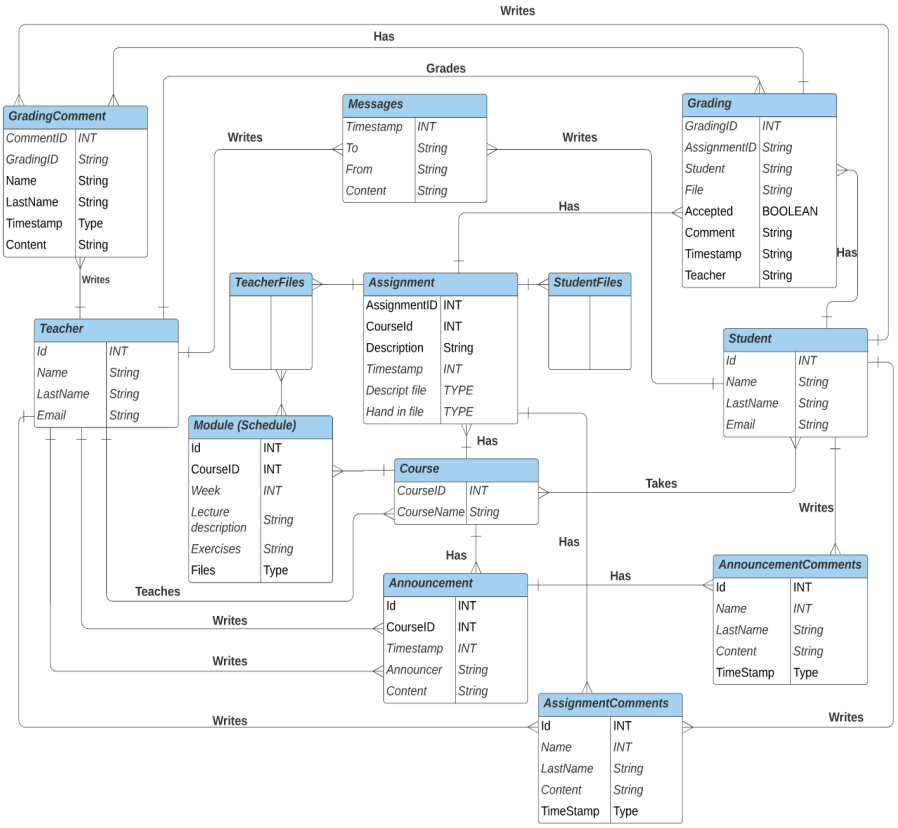
En persistence enhet måste sedan defineras. Persistence enheten säger var i applikationsservern databasen finns och hur den får åtkomst till den när applikationen körs på payara [18]. Persistence enheten är definerad i persistence.xml filen.

4.2 Affärslogiknivån

Som tidigare nämndes, affärslogiknivån sköter själva logiken av applikationen och även där ligger och underhålls representation av databasen.

4.2.1 Databas

Innan någon form av implementering gjordes etablerades ett databasschema för designen av databasen. Databasschemat visas nedanför i figur 7 och implementerades i systemet med JPA.



Figur 7: ER - diagram över databasen

4.2.2 Modellering av Databasen

I figur 8 visas en JPA entitet. JPA annotationen @Entity specificerar att den underliggande klassen representerar en entitet. Varje instans av det här objektet som skapas kommer sedan att representeras som en rad i databastabellen. De resterande JPA annotationerna @ManyToMany, @OneToMany, @ManyToOne specifierar de olika relationerna mellan databastabellerna. @ManyToMany hanterar tabellrelationen many-to-many där @JoinTable specifierar en ny tabell där den slår ihop två kolumner från två olika tabeller. @OneToMany specifierar tabellrelationen one-to-many där mappedBy specifierar instansvariabeln som är definerad i den entiteten som relationen relaterar till.

```java
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
public class Student implements Serializable{
    @Id
    private String studentID;
    private String name;
    private String lastName;
    private String password;
    private String email;
    private String userType;

    @ManyToMany
    @JoinTable(name = "studentCourse", joinColumns = @JoinColumn(name = "studentID"), inverseJoinColumns = @JoinColumn(name = "course_ID"))
    private List<Course> courses;

    @OneToMany
    @JoinColumn(name = "assigmentID")
    private List<Assignment> assignments;

    @OneToMany(mappedBy = "student")
    private List<AnnouncementComment> announcementComments;

    @OneToMany(mappedBy = "student")
    private List<GradingComment> gradingComments;

    @OneToMany(mappedBy = "student")
    private List<AssignmentComment> assignmentComments;
```

Figur 8: Java Persistence Entitet för en student.

```java
@ManyToMany
@JoinTable(name = "studentCourse", joinColumns = @JoinColumn(name = "studentID"), inverseJoinColumns = @JoinColumn(name = "course_ID"))
private List<Course> courses;
```

(a) Many to many relation i JPA.

```java
@OneToMany(mappedBy = "student")
private List<AssignmentComment> assignmentComments;
```

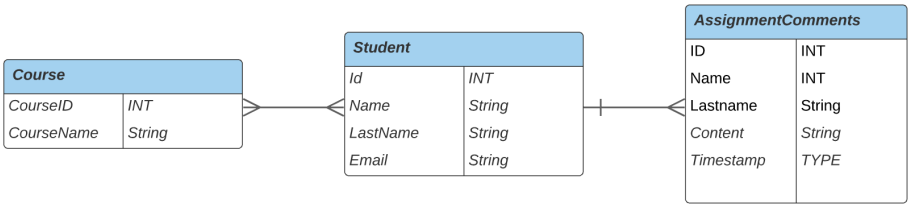
(b) One to many relation i JPA.

Figur 9: Relations mappning i JPA

Figur 9 visar två exemplen på relationer från student entiteten, där instans variablerna courses och assignments specifierar vilken entitet relationen är kopplade till. Ifall en dubbelriktad relation önskas måste också motsvarande entitet definera en relation som kopplar tillbaka enligt figur 10. Exemplet från figur 9 och figur 10 kommer att skapa en databas mappning enligt figur 11.

```
@ManyToMany(mappedBy = "courses")
private List<Student> students;
```

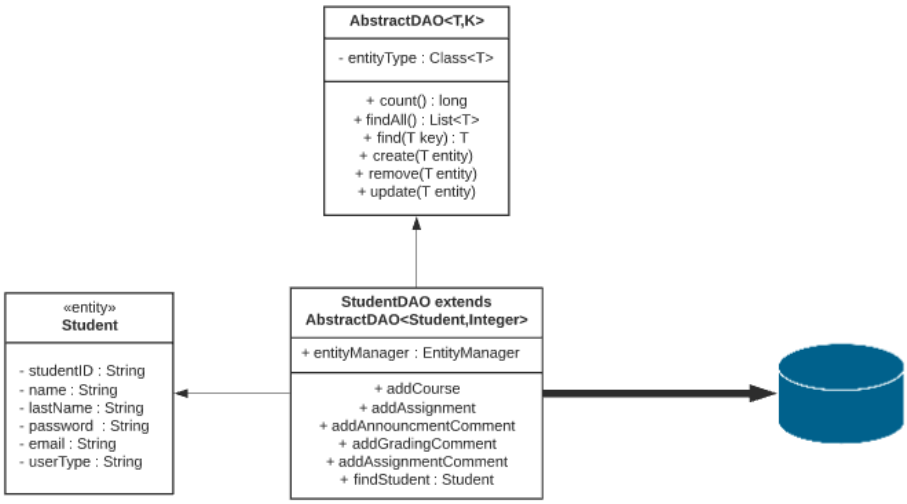
Figur 10: Motsvarande relation i en dubbelriktad mappning



Figur 11: ER digram för motsvarande exempel från figur 10 och figur 11

4.2.3 Lagring och Hämtning av Databasen

För att hantera lagring och hämtning av data användes designmönstret Data Access Object (DAO). DAO designmönstret hjälper till att isolera logiken i affärslogiknivån och separera den från kommunikationen med databasen. Enligt ett arkitekturperspektiv kan det beskrivas som att lägga till en ny nivå som sitter mellan affärslogiknivån och informationshanteringsnivån. UML diagrammet för designmönstret illustreras i figur 12. Varje entitet har ett så kallad DAO objekt som sköter själva kommunikationen med databasen. I DAO:en implementeras de olika metoder för att hämta och lagra data i databasen.



Figur 12: UML diagram för DAO - Designmönster

```java
@Stateless
public class StudentDAO extends AbstractDAO<Student, Integer>{
    @Getter @PersistenceContext(unitName = "schoolit_teacher")
    private EntityManager entityManager;

    public StudentDAO(){
        super(Student.class);
    }

    public void addCourse(Student s, Course c){
        c.addStudent(s);
        update(s);
    }
```

Figur 13: StudentDAO som kommunicerar med databasen för student entiteten

Figur 13 visar en del av implmentering för Student entitetens DAO. Instansvariabeln entityManager definerad i StudentDAO sköter själva läsning och skrivningen till databasen. Den måste tillsammans defineras med en @PersistenceContext annotation som laddar själva persistence enheten sådan att läsning och skrivning kan göras till korrekt databas. Varje DAO använder också arv till den abstrakta DAO klassen AbstractDAO för att implentera de mer vanliga metoderna som alla DAO klasser bör innehålla. Figur 14 illustrerar den Abstrakta acess klassen med en metod count() som räknar antal element av ett objekt i databasen.

```java
@RequiredArgsConstructor
public abstract class AbstractDAO <T, K> {
    private final Class<T> entityType;
    protected abstract EntityManager getEntityManager();

    public long count() {
        final CriteriaBuilder builder = getEntityManager().getCriteriaBuilder();
        final CriteriaQuery cq = builder.createQuery();
        final Root<T> rt = cq.from(entityType);
        cq.select(builder.count(rt));
        final Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult());
    }
```

Figur 14: Abstrakta klassen AbstractDAO.

Querydsl lades till i projektet för mer elegant query skrivning för sökning genom databasen. Java EE erbjuder deras Criteria API för att skriva queries, dock är det inte en elegant lösning på det. Querydsl är byggt ovanpå Criteria API och erbjuder en elegant och bekant metod för att skriva queries. Querydsl adderas i projektet genom att lägga in det som ett beroende i pom.xml enligt figur 15 som säger åt maven att hämta det till projektet. Ett exempel på en query skriven i querydsl ser ut som följande i figur 16 och hämtar en student i databasen baserat på email. En query skriven i Criteria API visas i figur 14 som räknar antal element i en tabell. En jämförelse visar att mindre kod behöver skrivas i querydsl, dessutom efterliknar uppbyggnaden av en query mer en standard SQL query.

```xml
<dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
    <version>${querydsl.version}</version>
</dependency>
```

Figur 15: Querydsl beroende som läggs in i pom.xml.

```java
public Student findStudent(String email){
    JPAQueryFactory queryFactory = new JPAQueryFactory(entityManager);
    QStudent student = QStudent.student;
    Student s = queryFactory.selectFrom(student).where(student.email.eq(email)).fetchOne();
    return s;
}
```

Figur 16: Exempel query skriven i querydsl.

4.2.4 Testning av Databasen

Testning applicerades i projektet för att försäkra att databasen fungerade enligt dess implementering. Testningen gjordes med Arquillian kombinerat med Junit tester. Arquillian är en test plattform som tillåter en att köra en komplett testmiljö som stödjer all funktionalitet i en applikationsserver inuti en vanlig

unittest [19]. För att implementera Arquillan i projektet behövdes flera beroenden adderas till pom.xml. De tre beroenden arquillian-junit-container, arquillian-payara-server-embedded och payara-embedded-all bidrar med att Arquillian och Payara kan integreras med Junit. Därefter måste Maven veta vilken version ska användas och det görs genom att lägga in beroendet arquillian-bom med en specifierad version i pom.xml. Sist adderas Junit in i projektet precis som föregående beroenden. Tillsammans med de olika etablerade beroenden stödjer projektet Junit tester med användning av Java Persistence API, Arquillian och Payara. Nedanför i figur 17 illustreras ett exempel på hur test klasser och test metoder implementerades.

```java
@RunWith(Arquillian.class)
public class CourseDAOTest {
    @Deployment
    public static WebArchive createDeployment(){
        return ShrinkWrap.create(WebArchive.class)
                .addClasses(TeacherDAO.class, Teacher.class, CourseDAO.class, Course.class, StudentDAO.class,
                .addAsResource("META-INF/persistence.xml")
                .addAsManifestResource(EmptyAsset.INSTANCE,"beans.xml");
    }

    @EJB
    private TeacherDAO teacherDao;

    @EJB
    private CourseDAO courseDao;

    @EJB
    private StudentDAO studentDao;

    @EJB
    private CourseModuleDAO courseModuleDao;

    @EJB
    private TeacherFileDAO teacherFileDao;
```

(a) Arquillian test klass för CourseDAO.

```java
@Test
public void checkAddCourse(){
    Course c = courseDao.find(1);
    CourseModule cm = courseModuleDao.find(1);

    courseDao.addCourseModule(c, cm);

    Assert.assertTrue(c.getCourseModules().contains(cm));
    Assert.assertFalse(cm.getCourse() == null);
    Assert.assertTrue(cm.getCourse().equals(c));
}

@Test
public void checkAddStudent(){
    Course c = courseDao.find(1);
    Student s = studentDao.find(1);

    courseDao.addStudents(c, s);

    Assert.assertTrue(c.getStudents().contains(s));
    Assert.assertFalse(s.getCourses().get(0) == null);
    Assert.assertTrue(s.getCourses().contains(c));
}
```

(b) Test metoderna för CourseDAO.

Figur 17: Arquillian Testning.

4.3 Webb Nivån

I webbnivån sköts kommunikationen mellan klientnivån och affärslogiknivån. Kommunikationen sker i form av webbkomponenter som t.ex är dynamiska webbsidor eller servlets som exekveras inuti webb containern.

4.3.1 Java Server Faces

Java EE erbjuder Java Server Faces API (JSF) vilket kan ses som ett frontend ramverk. En mer korrekt definition är att det är ett server liggande komponentbaserat användargränsnitt ramverk. JSF är inte en traditionell frontend i sig, utan det ligger på applikationsservern och exekveras i webb containern.

Java server Faces API är redan direkt integrerat med Java Servlet API [20]. Det innebär att alla begäran som skickas till webbapplikationen hanteras av JSF i bakgrunden sådan att en utvecklare inte behöver implementera egna servlets till applikationen. Innan teknologier som JSF var en traditionell Java applikation uppbyggd av html kod, servlets och Java bönor. Med JSF används xhtml och managed Java bönor istället och låter utvecklare att fokusera på implmentering av webbsida och affärslogiken. JSF i sig själv följer MVC (Model - View Controller) arkitekturen.

JSF hanterar klient begäran i form av olika faser i en livscykel för att generera response till klienten. Livscykeln börjar när klienten skickar en HTTP förfrågan för en vy och slutar efter att servern har levererat den. JSF livscykel består av nio faser. De här olika faserna sker i flera steg för att bygga komponenterna, validera data given från klienten, generera felmeddelanden, updatera servern och anropa applikationen för att sköta affärslogik [21].

JSF erbjuder en massa fördelar. Med JSF behövs inte fokuset längre ligga på att hantera komponenter. JSF erbjuder ett API för att själv hantera validering av komponenter, data konvertering och tillägg av andra externa komponenter (t.ex Primefaces). JSF ramverket hanterar livscykeln automatiskt och även erbjuder manuell hantering av livscykeln för mer komplex livscykel hantering [21]. Figur 18 visar hur en JSF sida kan se ut. Inuti visas även primefaces komponenter och en EJB (Enterprise Java Böna) som hanterar logik.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:p="http://primefaces.org/ui">
    <f:view>
        <h:head>
            <title>Facelet Title</title>
            <h:outputStylesheet library="css" name="main.css"/>
        </h:head>
        <h:body styleClass="backgroundImage">
            <div id="schoolitText">
                <h1>SchoolIT</h1>
            </div>


            <div id="welcomeText">
                <h1>Welcome</h1>
            </div>

            <h:form>
                <p:messages>
                    <p:autoUpdate/>
                </p:messages>

                <div id="loginSquare">
                    <div id="loginInput">
                        <p:inputText placeholder="E-mail" style="height: 25px" value="#{loginBackingBean.email}"/>
                    </div>
                    <div id="passwordInput">
                        <p:password placeholder="Password" style="height: 25px" value="#{loginBackingBean.password}"/>
                    </div>
                    <div id="loginButton">
                        <p:commandButton value="Login" update="@form" style="background-image: none; background-color: white"
                    </div>
                    <div id="createAccButton">
                        <p id='createAccButtonText'>Don't have an account?</p>
                        <p:button value="Create Account" outcome="selectTypePage"/>
                    </div>
                </div>

            </h:form>
        </h:body>
    </f:view>
</html>
```

Figur 18: Exempel på en JSF vy i xhtml (Extensiable HyperText Markup Language).

4.3.2 PrimeFaces

PrimeFaces är ett komponentbibliotek ramverk för JSF som tillåter utvecklare att lägga till diversa olika UI komponenter till en vy [22]. Primefaces underlättar utvecklingen av frontenden och designen av applikationen. För att addera PrimeFaces till projektet läggs det till som en dependency in pom.xml som tidigare. Komponenter kan bestå allt från knappar till panel menyer. Primefaces ebjuder en mängd komponenter av olika syften. Figur 19 visar ett simpelt exempel på en primeface komponent.

(a) Knapp komponenter från primefaces.

```
<p:button value="Submit" />
<p:button value="Disabled" disabled="true" />
```

(b) Implementering av komponenterna.

Figur 19: Primefaces exempel.

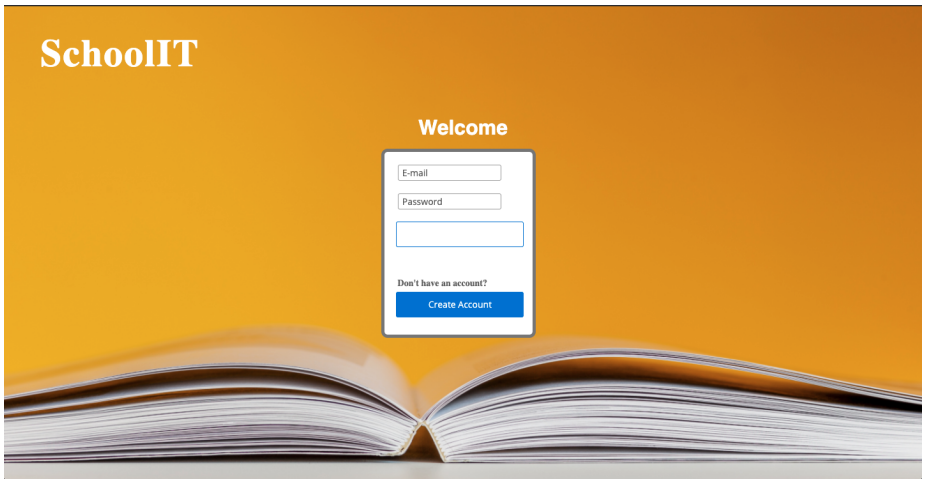
4.4 Kommunikation mellan Webb - och Affärslogiknivån

En klient kan kommunicera med applikationen direkt genom affärslogiknivån eller via en webbläsare som kommunicerar med affärslogiknivån genom webbsidor och servlets som ligger inom webbnivån. Eftersom applikationen är en webb-applikation så utnyttjas den sistnämnda.

Webbnivån kan innehålla Java Bönor för att hantera input från klienten som sedan skickar den här informationen till så kallade Enterprise Java Bönor (EJB) som ligger i affärslogiknivån för att sedan hantera logiken eller lagra och hämta data i informationshanteringsnivån. I den här applikationen sköter dock själva Persistencenivån kommunikationen med databasen via DAO objekten. Följande sektioner kommer gå igenom implementeringen av de olika delarna i applikationen där kommunikation mellan webb, affärslogik och informations-hanteringsnivån sker.

4.4.1 Skapa Användare

En användare blir först introducerad till välkomstvyn i figur 20 som är en standard inloggningssida. I välkomstvyn kan klienten välja att göra två saker. Antigen logga in eller skapa en användare.



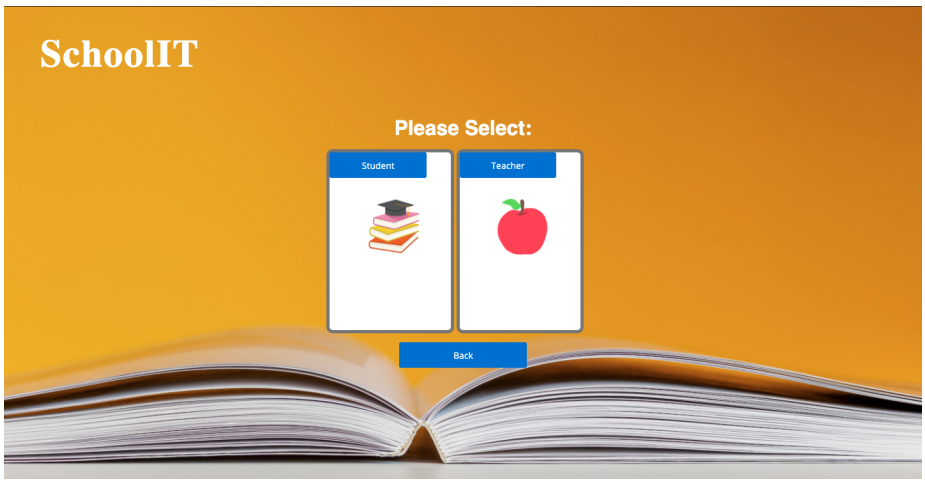
Figur 20: Inloggninssida

Efter att klienten väljer att skapa ett konto kommer vyn att dirigera klienten till en vy som låter klienten att välja vad för slags användare som ska skapas. Själva dirigerigen av vyn sköts med en primefaces knappkomponent i figur 21 där outcome är värdet på vart den ska navigera till via en GET - begäran.

```
<div id="createAccButton">
    <p id='createAccButtonText'>Don't have an account?</p>
    <p:button value="Create Account" outcome="selectTypePage"/>
</div>
```

Figur 21: Knapp för att skapa konto

Vyn för val av användare är uppdelat i två paneler enligt figur 22. När klienten gör ett val genom att trycka på någon av de två knapparna redigeras klienten till en utav de två olika vyerna för skapning av användare. Där kan klienten sedan lägga in information som används för att skapa sitt användarkonto.



Figur 22: Väljarsida för val av slags användare.

De två panelerna hanteras av en selectionBackingBean enligt figur 23 som består av två metoder, returnStudent och returnTeacher. De två metoder retunerar bara en sträng som är navigeringen till de två olika vyerna för att skapa användare. Klassen selectionBackingBean illustreras i figur 24.

```
<div id="parentDiv">
    <div class="square">
        <h:form>
        <h:graphicImage styleClass="selectTypePicture" name="images/book-2841867_1920.jpg" style="height: 110px; width: 115px"/>
        <p:commandButton styleClass="selectTypeButton" value="Student" action="#{selectTypeBackingBean.returnStudent}"/>
        </h:form>
    </div>
    <div id="square2">
        <h:form>
        <h:graphicImage styleClass="selectTypePicture" name="images/teacher-1202735_1920.png" style="height: 110px; width: 110px;"/>
        <p:commandButton styleClass="selectTypeButton" value="Teacher" action="#{selectTypeBackingBean.returnTeacher}"/>
        </h:form>
    </div>
</div>
```

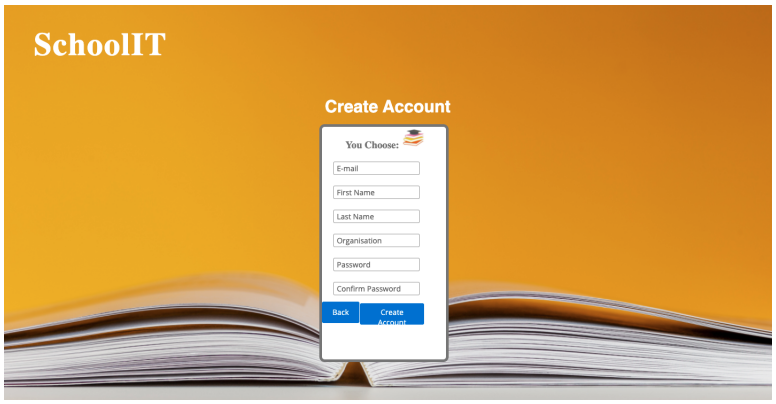
Figur 23: xhtml för panelerna av väljarsida

```java
package controller;

import java.io.Serializable;
import javax.annotation.PostConstruct;
import lombok.Data;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

/**
 *
 * @author albinbecevic
 */

@Data
@Named
@RequestScoped
public class SelectTypeBackingBean implements Serializable{

    @PostConstruct
    public void init(){

    }

    public String returnTeacher(){
        return "createTeacherPage";
    }

    public String returnStudent(){
        return "createStudentPage";
    }
}
```

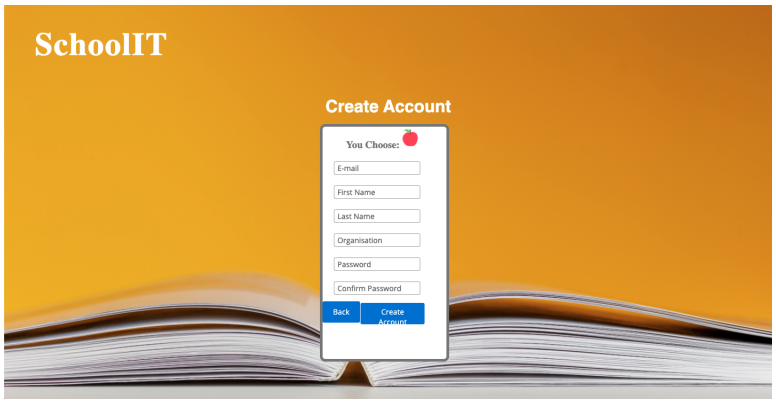
Figur 24: SelectTypeBackingBean som hanterar väljarsidan

Beroende på vad klienten valde för användare så redigeras klienten till en av de följande två vyerna i figur 25. Både vyerna hanteras av två olika Java bönor som är createTeacherBackingBean och createStudentBackingBean. Båda bönorna hanterar logiken på vyerna på liknande sätt, enda skillnaden är att de skapar två olika användarkonton.

På figur 26 illustreras hur fälten på vyn är implementerade. Fälten på vyn tar in informationen från klienten genom att varje fält har attributen required = True, det här ser till att alla fält måste vara ifyllda för att kalla på metoder inom Java bönan. Eftersom Java bönan skapas när vyn är levande sätts informationen i fälten in i instansvariabler i Java bönan. Java bönan createStudentBackingBean och dess variabler illustreras i figur 27.



(a) Skapa konto för student.



(b) Skapa konto för lärare.

Figur 25: Skapa konto sidor.

```
<div id="loginInput">
    <p:inputText placeholder="E-mail" style="height: 25px" value="#{createStudentBackingBean.email}" required="true" lab
</div>

<div id="loginInput">
    <p:inputText placeholder="First Name" style="height: 25px" value="#{createStudentBackingBean.firstName}" required="t
</div>

<div id="loginInput">
    <p:inputText placeholder="Last Name" style="height: 25px" value="#{createStudentBackingBean.lastName}" required="tru
</div>

<div id="loginInput">
    <p:inputText placeholder="Organisation" style="height: 25px" value="#{createStudentBackingBean.organisation}" requir
</div>

<div id="passwordInput">
    <p:password placeholder="Password" style="height: 25px" value="#{createStudentBackingBean.password}" required="true"
</div>

<div id="passwordInput">
    <p:password placeholder="Confirm Password" style="height: 25px" value="#{createStudentBackingBean.confirmPassword}"
```

Figur 26: Xhtml fälten för skapning av konto

```java
@Data
@Named
@RequestScoped
public class CreateStudentBackingBean implements Serializable{
    private String email;
    private String firstName;
    private String lastName;
    private String organisation;
    private String password;
    private String confirmPassword;
    private boolean accountExist;

    @EJB
    private StudentDAO studentDao;

    @Inject
    Pbkdf2PasswordHash passwordHasher;

    @PostConstruct
    public void init(){

    }
```

Figur 27: CreateStudentBackingBean som hanterar student sidan för att skapa konto.

Instansvariabeln studentDao är ett DAO objekt som kommunicerar med databasen, den sköter två jobb. Se till att användaren inte redan finns och skapa användaren. Objektet passwordHasher som skapas via en Dependency Injektion kommer ifrån Java EE säkerhets API och används för att hasha lösernordet som klienten skrivit in. Efter att alla fält har fyllts i och användaren trycker på knappen Create Account i figur 28 kallar JSF på metoden createAccount inuti Java bönan createStudentBackingBean. Metoden createAccount visas i figur 29.

```
<p:commandButton styleClass="creationButton2" value="Create Account" action="#{createStudentBackingBean.createAccount}"/>
```

Figur 28: Knapp för att skapa konto.

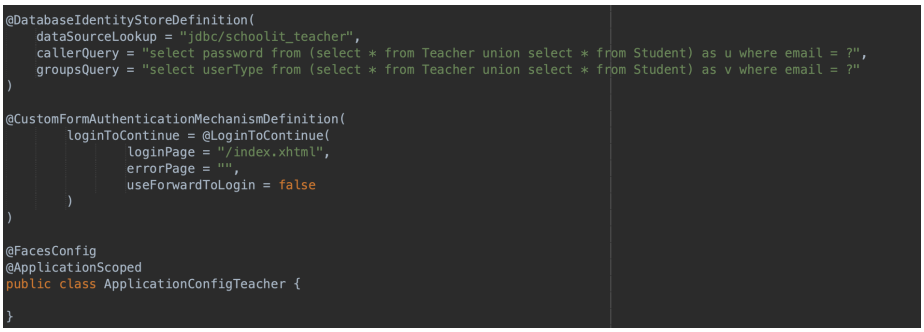
Figur 29: Metod createAccount.

Metoden CreateAccount kontrollerar att alla fält är ifyllda, om inte genreras ett felmeddelande som visas upp på vyn. I nästa steg kontrolleras ifall kontot redan existerar, det här sköter studentDao i metoden checkExistingAcc. Om kontot existerar genreras ett felmeddelande på liknande sätt som innan. Om allt går igenom genereras ett ID för användaren och en student entitets objekt skapas med det genrerade ID:et tillsammans med lösenordet som blir hashat. Därefter placerar studentDao den skapaden entiteten som en instans i databasen. Ifall en lärare skapas sker det på liknande sätt, skillnaden är att teacherDao kommunicerar med databasen istället för studentDao.

4.4.2 Inloggning

För att skapa ett säker inlogg med autentisering i JSF krävs det tre steg. Implementera en så kallad Identitystore, som säger vart informationen som ska autentiseras ligger nånstans. Nästa är att skapa en autentiseringsmekanism och sist specifiera auktoriseringsroller.

För att upfylla de tre stegen behöver först en Applikationskonfig klass skapas. Klassen kommer vara tom och används för att annotera säkerhetskomponenterna. Klassen och säkerhetskomponenterna illustreras i figur 30.



Figur 30: Konfigureringsklassen för att annotera säkerhetskomponenter

Första komponenten är Identitystore som visas i figur 30. Eftersom användarna är lagrade i databasen används annotationen @DatabaseIdentityStore tillsammans med dess tre attributer. Attributen dataSourceLookup specifierar vilken datakälla som kommer användas för att hämta användarna. Nästa attribut callerQuery specifierar hur identifieringen av en användare sker, den här queryn pratar direkt med databasen och är byggd som en standard query. Lösenordet används för identifieringen. Eftersom att de två olika slags användarna är två entiteter i applikationen krävs det att queryn kollar i båda datatabellerna, därför sker det en union på de två. GroupsQuery hämtar vilken grupp den här användaren tillhör. (Kan diskuteras ifall det här ens behöver användas eftersom student och teacher är två olika entiteter och inte en samma entitet deklarerad som user). Annotationen @DatabaseIdentityStore förväntar att lösenorden i databasen är i deras hashat tillstånd, därför krävs det också att de redan är hashade i databasen när de hämtas.

Nästa komponent som behöver annoteras är autentiseringsmekanismen @CustomFormAuthenticationMechanism, den annoteringen säger till servern vad den ska göra efter att en klient har blivit autentiserad. Annotationen visas också i figur 30. I autentiseringsmekanismen behövs en inloggningsida specifieras, sedan en felsida ifall autentiseringen misslyckas. Den lämnas blank för att användaren ska vara kvar på inloggninssidan om autentiseringen misslyckas. Attributen UseForwardToLogin är falsk för att användaren ska omdirigeras till inloggningssidan.

Inloggningssidan består av en form som tar input från klienten som sedan hanteras av LogingBackinBean som sköter hela inloggninsmekanismen. Login formen illustreras i figur 31. Login knappen består av tre attributer, attributen update specifierar vilken section av vyn som ska updateras. Med värdet @form specifieras att uppdateringen sker bara på formen där knappen ligger inuti. Via action kallar JSF login metoden som sköts av loginBackinBean och sist måste ajax flaggan vara falsk eftersom en full HTTP begäran ska skickas till servern. Ifall en delvis begäran skickas kommer inte applikationen att autentisera korrekt.

Figur 32 visar de variablerna LoginBackingBean behöver för att sköta autentiseringen.

```
<h:form>
    <p:messages>
        <p:autoUpdate/>
    </p:messages>

    <div id="loginSquare">
        <div id="loginInput">
            <p:inputText placeholder="E-mail" style="height: 25px" value="#{loginBackingBean.email}"/>
        </div>
        <div id="passwordInput">
            <p:password placeholder="Password" style="height: 25px" value="#{loginBackingBean.password}"/>
        </div>
        <div id="loginButton">
            <p:commandButton value="Login" update="@form" style="background-image: none; background-color: white" action="#{loginBackingBean.login}" ajax="false"/>
        </div>
        <div id="createAccButton">
            <p id='createAccButtonText'>Don't have an account?</p>
            <p:button value="Create Account" outcome="selectTypePage"/>
        </div>
    </div>

</h:form>
```

Figur 31: Login form

```
@Data
@Named
@RequestScoped
public class LoginBackingBean implements Serializable{

    @NotEmpty
    private String email;

    @NotEmpty
    private String password;

    @EJB
    private TeacherDAO teacherDao;

    @Inject
    FacesContext facesContext;

    @Inject
    SecurityContext securityContext;
```

Figur 32: LoginBackingBean klassen

Informationen klienten skriver in lagras i variablerna email och password, annoteringen @NotEmpty är en försäkring att inte börja försöka autentisera ifall något av dessa skulle saknas. TeacherDAO används för att kolla ifall informationen given tillhör en lärare eller inte. FacesContext hjälper oss att hantera klient begäran och ge response på den vilket som sköts av den underliggande containern, via FacesContext kan vi hämta den begäran som klienten gör. Sist används Javas egna enterprise säkerhetsramverk, SecurityContext för att exekvera själva autentiseringsprocessen.

För att hantera autentisering av klienten behövs det två hjälpmetoder. Första hjälpmetoden är getExternalContext. Det metoden gör är att den tillåter

åtkomst till de underliggande servlets i JSF där klienten har skickat sin begäran. Metoden visas i figure 33.

```
private ExternalContext getExternalContext(){
    return facesContext.getExternalContext();
}
```

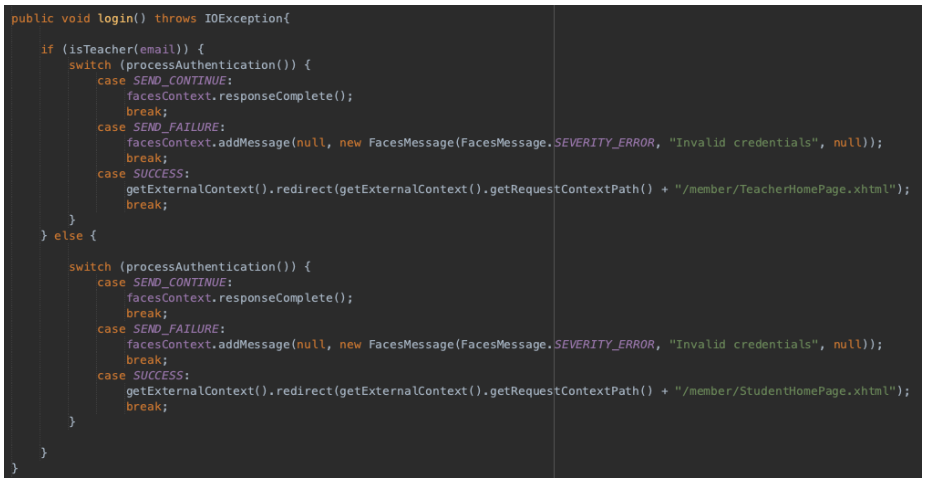
Figur 33: GetExternalContext metoden

Nästa hjälpmetod är processAuthentication, implementationen visas i figur 34. Metoden processAuthentication exekverar securityContexts autentisering. Autentiseringen kräver klientens begäran, dess response och själva parametrerna som skickades från klienten. Parametrarna måste göras om till ett Authentication-Parameters objekt för att SecurityContext ska kunna exekvera autentiseringen. Med den informationen klienten givit byggs en UserCredential objekt, med det objektet byggs det sedan ett AuthenticationParameters objekt som sedan autentiseras via securityContext. Metoden kommer sedan att retunera tre möjliga autentiserings svar. SEND CONTINUE, SEND FAILURE och SUCCESS. Beroende på vad autentiseringen retunerar hanterars det och applikationen agerar därefter.

```
private AuthenticationStatus processAuthentication(){
    ExternalContext ec = getExternalContext();

    return securityContext.authenticate((HttpServletRequest)ec.getRequest(),
                                        (HttpServletResponse)ec.getResponse(),
                                        AuthenticationParameters.withParams().credential(new UsernamePasswordCredential(email, password)));
}
```

Figur 34: ProcessAuthentication metoden

Login metoden kollar vad för slags användare klienten vill autentisera och agerar enligt vad processAuthentication retunerar. Metoden illustreras i figur 35.



```
public void login() throws IOException{

    if (isTeacher(email)) {
        switch (processAuthentication()) {
            case SEND_CONTINUE:
                facesContext.responseComplete();
                break;
            case SEND_FAILURE:
                facesContext.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_ERROR, "Invalid credentials", null));
                break;
            case SUCCESS:
                getExternalContext().redirect(getExternalContext().getRequestContextPath() + "/member/TeacherHomePage.xhtml");
                break;
        }
    } else {

        switch (processAuthentication()) {
            case SEND_CONTINUE:
                facesContext.responseComplete();
                break;
            case SEND_FAILURE:
                facesContext.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_ERROR, "Invalid credentials", null));
                break;
            case SUCCESS:
                getExternalContext().redirect(getExternalContext().getRequestContextPath() + "/member/StudentHomePage.xhtml");
                break;
        }

    }
}
```

Figur 35: Login metoden

Tre olika agerande kan inträffa. SEND FAILURE retunerar bara ett felmeddelande till vyn. SUCCESS betyder att autentiseringen har fullföljts och vi dirigerar klienten till korrekt användarhemsida beroende på vad för användare. Ifall processAuthentication retunerar SEND CONTINUE betyder det att autentiseringen har redan gjorts och en flerstegs autentisering med klienten har påbörjats. Eftersom det inte önskas att applikationen autentisera klienten i flera olika steg blir JSF tillsagd att denna begäran har redan gjorts och att cykeln borde termineras direkt efter att första fasen är klar.

Sist specifieras säkerhetsbegränsningar för auktoriseringsrollerna. Det görs för att specifiera vilka vyer de olika rollerna har tillgång till. Det görs i web.xml där vi adderar två säkerhetsbegränsningar för både en student och lärare. Specifieringen av säkerhetsbegränsningar görs på liknande som att definera beroenden, men det görs i en annan fil med en annan syntax. Säkerhetsbegränsningarna i det här fallet är simpla. Där varje roll har tillåtelse till alla vyer som ligger inuti medlemsmappen.

4.4.3 Lärar instrumentbräda

Efter att en klient har blivit autentiserad som en lärare blir användaren introducerad till hemsidan för en lärare. Lärarhemsidan illustreras i figur 36.

Figur 36: Lärar instrumentbräda sida

Instumentbrädan visar alla kurser läraren har hand om och kan interagera med. För att ge en bättre användarupplevelse används JSF templating på instrumentbrädan för att inte behöva uppdatera hela vyn när en begäran skickas. Med JSF template behöver bara en del av vyn uppdateras. JSF Templating sker genom att man specifierar en rad ui:insert med ett referens namn i xhtml vyn. När uppdatering av vyn ska ske med en annan vy behöver bara namnet refereras, då kommer sektionen inom ui:insert att updateras till den nya vyn. Standard vyn som visas innan någon updatering sker är teacherDashboard.xhtml som representerar instumentbrädan.

Till instrumentbrädan användes primefaces dashboard komponent, det gjordes genom att referera till p:dashboard. Hela komponenten sköts av teacherDashboard-View bönan. Instrumentbrädan fylls på med primefaces paneler. Den första panelen tar läraren vidare till en vy där en lärare kan skapa en kurs. Om en lärare har kurser kommer de resterande panelerna att fyllas på därefter och deras panelvariabler sätts.

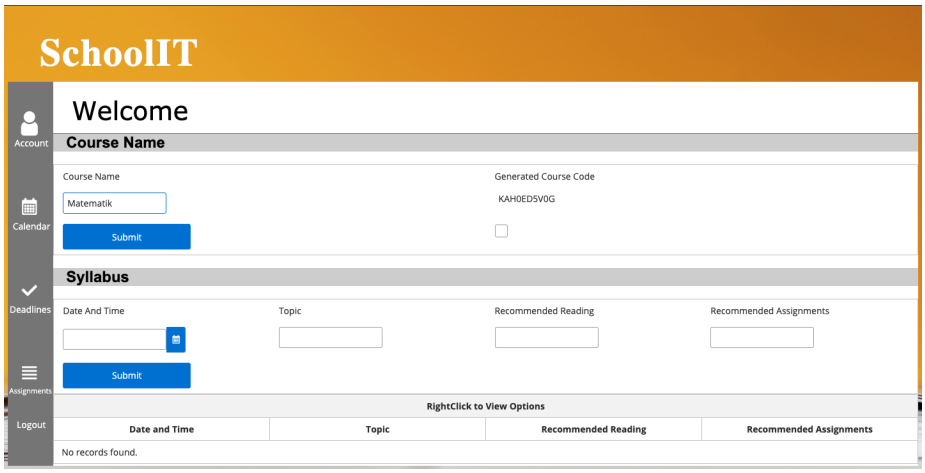
```java
@Data
@Named
@RequestScoped
public class TeacherDashboardView implements Serializable {

    @Inject
    private TeacherDAO teacherDao;

    @Inject
    private CourseDAO courseDao;

    @Inject
    private SecurityContext securityContext;

    private Teacher currentTeacher;
    private List<Course> courseList;
    private DashboardModel model;


    @PostConstruct
    public void init() {
        String email = securityContext.getCallerPrincipal().getName();
        userTypeSetup(email);
        courseListSetup();
        dashBoardSetup();
    }
```

Figur 37: TeacherDashbordView klassen som hanter instumenrbrädan för lärare.

Följande variabler och metoder som ser till att hantera instumentbrädan visas i figur 37. Via SecurityContext kan den nuvarande inloggade användarens id (email) hämtas. Det kan bara göras ifall en användare har blivit autentiserad. Med id:et hämtas lärar entiteten och sätter den till currentTeacher, det görs i userTypeSetup() med hjälp av teacherDao. Efter currentTeacher är satt används den för att hämta alla attributer som tillhör den användaren. Den attributen som är av intresse är kurserna som läraren äger. Det hämtas i courseListSetup() och sätter sedan variabeln courseList. Metoden dashBoardSetup() sätter upp hela strukturen för instrumentbrädan (antal rader och kolumner) och sedan populerar den med alla objekt (kurser) som finns i courseList.

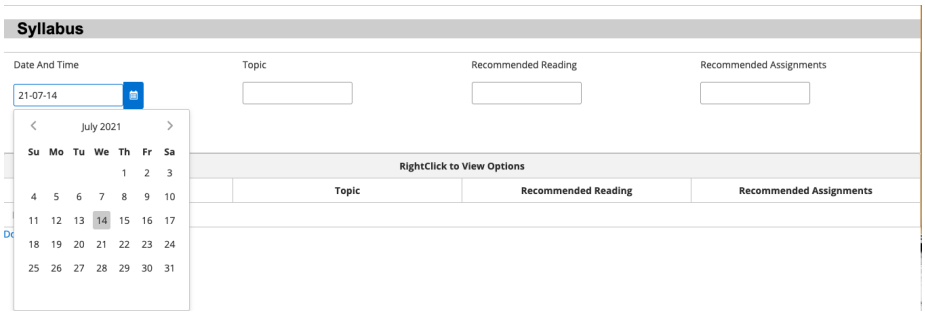
4.4.4 Skapa kurs

Efter att läraren trycker på create course panelen updateras vyn till följande vy i figur 38.

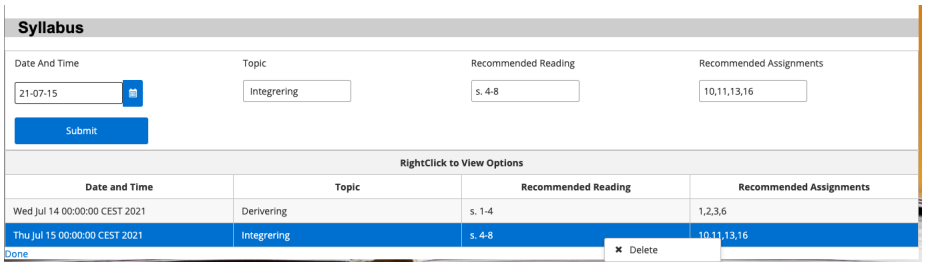


Figur 38: Kurskapar sida för lärare

Direkt när läraren tar sig till vyn genereras en slumpmässig kurskod som kan användas av eleverna för att gå med i kursen. Första sektionen skapas själva kursen och dess namn, nästa sektion skapas en kursplan till kursen. Succesivt kan man addera en rad in i datatabellen nedanför enligt figur 39 och 40.



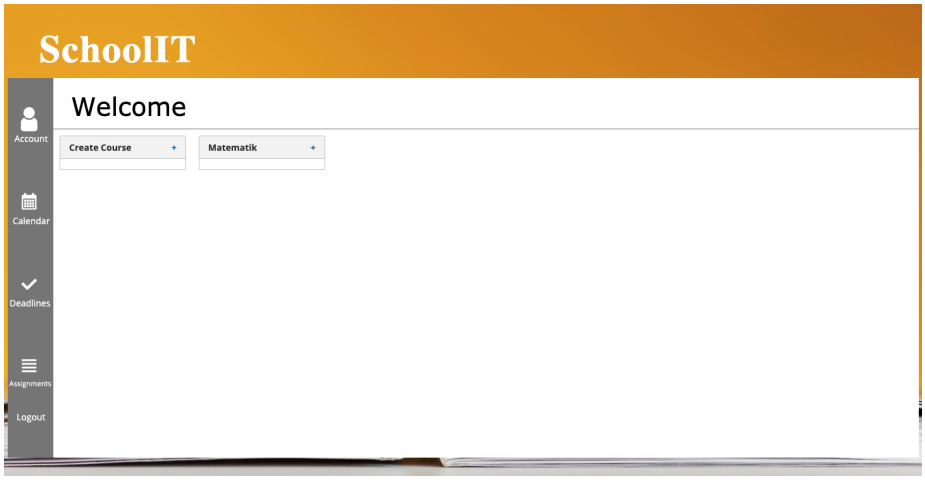
Figur 39: Data tabell för kursplan



Figur 40: Resultat efter skapad kursplan

Kommunikationen mellan webbnivån och affärslogiknivån sker på liknande sätt som tidigare beskrivit i sektionen Skapa användare. Vyn hanteras här av CreateCourseBackingBean med diversa olika variabler som är kopplade till olika primefaces komponenter för reprsentering av data. DAO:arna som sköter kommunikationen med databasen är TeacherDAO, CourseDAO och CourseModuleDAO.

Efter att läraren klickar på Done knappen kommer den tas sig tillbaka till en uppdaterad instrument bräda enligt figur 41.



Figur 41: Updaterad instrumentbräda för lärare

4.4.5 Lärar kurssida

Efter att en kurs är skapad kan läraren navigera till sin kurshemsida, det här görs genom att klicka på den panelen som representerar den önskade kurssidan. Eftersom varje panel som laddas upp på instrumentbrädan blir tilldelad sitt kurs id enligt figur 42 kan id:et skickas som en faces parameter enligt raden f:param. Genom FacesContext enligt figur 43 kan parametrarna som skickades begäras för att sedan använda de för att hämta relevant data angående kursen.

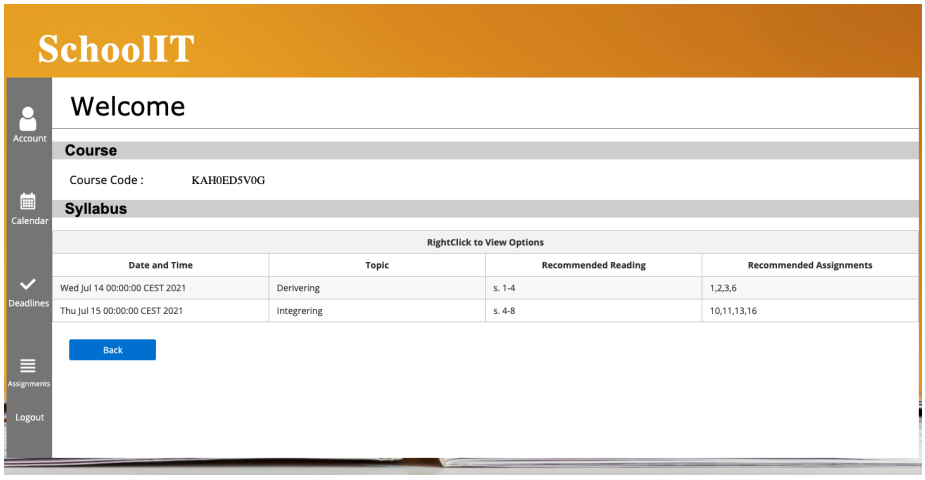
Tillsammans kommer läraren att bli representerad följande vy som illustreras i figur 44.

```
<p:panel id="#{course.courseID}" header="#{course.courseName}">
    <f:facet name="actions">
        <p:link id="course" outcome="/member/TeacherCoursePage">
            <h:outputText value="+"/>
            <f:param name="courseID" value="#{course.courseID}"/>
        </p:link>
    </f:facet>
</p:panel>
```

Figur 42: Kurs panel

```
@PostConstruct
public void init(){
    Map<String, String> params = FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap();
    this.courseID = params.get("courseID");
    this.course = courseDao.find(courseID);
    this.moduleList = course.getCourseModules();
}
```

Figur 43: Hämtnig av data genom FacesContext

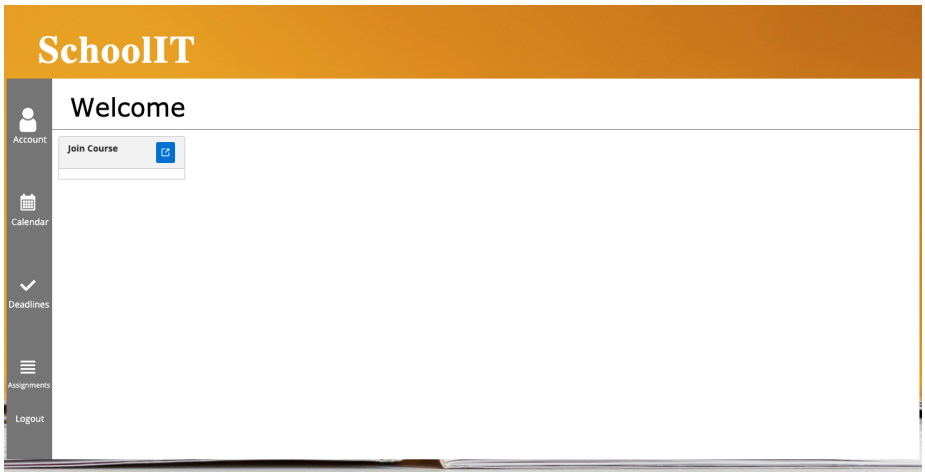


Figur 44: Kurshemsida för lärare

4.4.6 Student instumentbräda

När en klient har blivit autentiserad som en student dirigeras den till en liknande vy som en lärare. Instrumentbrädan fungerar på liknande sätt som innan, enda skillnaden är panelen Join course i figur 45. Genom Join course panelen kan

en student gå med i en kurs. Studenten kan klicka på panelen som visar en primefaces dialogruta, här kan studenten sedan skriva in kurs id:et för att gå med i en kurs. Implementeringen för en primefaces dialogkomponent visas i figur 46.



Figur 45: Student hemsida

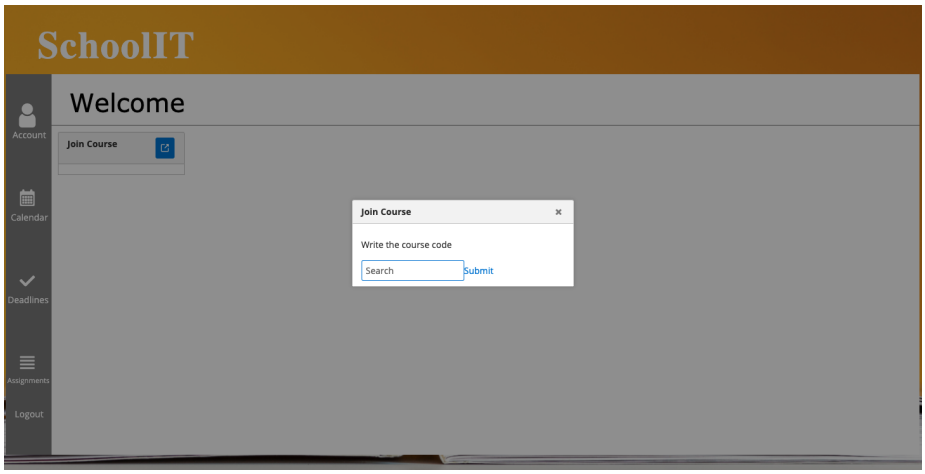
```
<p:dialog header="Join Course" widgetVar="dlg2" minHeight="40" width="350" showEffect="fade" modal="true">
            <p class="p-m-0">Write the course code</p>
            <p:inputText placeholder="Search" value="#{studentDashboardView.courseCodeSearch}"/>
            <p:commandLink value="Submit" action="#{studentDashboardView.joinCourse}" ajax="false"/>
</p:dialog>
```

Figur 46: Primefaces dialog komponent

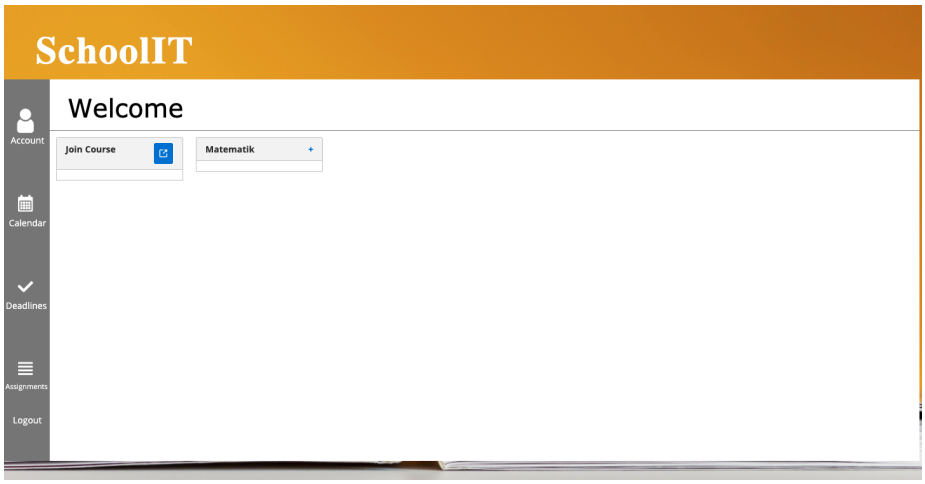
Hela vyn hanteras av StudentDashboardView bönan som kallar på metoden joinCourse som visas i figur 47. Metoden exekveras efter att en student trycker på länken submit. Metoden kollar ifall kursen existerar, om den finns läggs den till i student användarens kurslista, och sedan uppdateras instrumentbrädan. Ifall den inte finns genereras ett felmeddelande. Processen illustreras i figur 48 och figur 49.

```
public String joinCourse(){
    if(courseDao.find(courseCodeSearch) != null){
        Course c = courseDao.find(courseCodeSearch);
        studentDao.addCourse(currentStudent, c);
        return "/member/StudentHomePage.xhtml?faces-redirect=true";
    }else{
        FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "No course with that code exist"
        return null;
    }
}
```

Figur 47: Metod för att gå med i en kurs



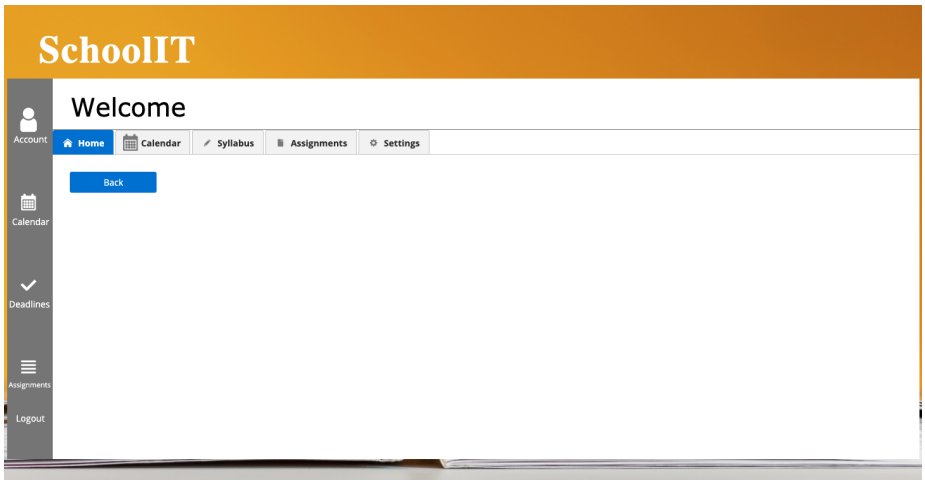
Figur 48: Dialog ruta för att gå med i kurs



Figur 49: Updaterad instrumentbräda för student

4.4.7 Student kurshemsida

Efter att en student har gått med i en kurs kan den ta sig vidare till student kurshemsidan. Det görs på samma sätt som tidigare beskrivet i lärare instrumentbrädan. När studenten klickar på kursen uppdateras intstumentbrädan till följande vy i figur 50.



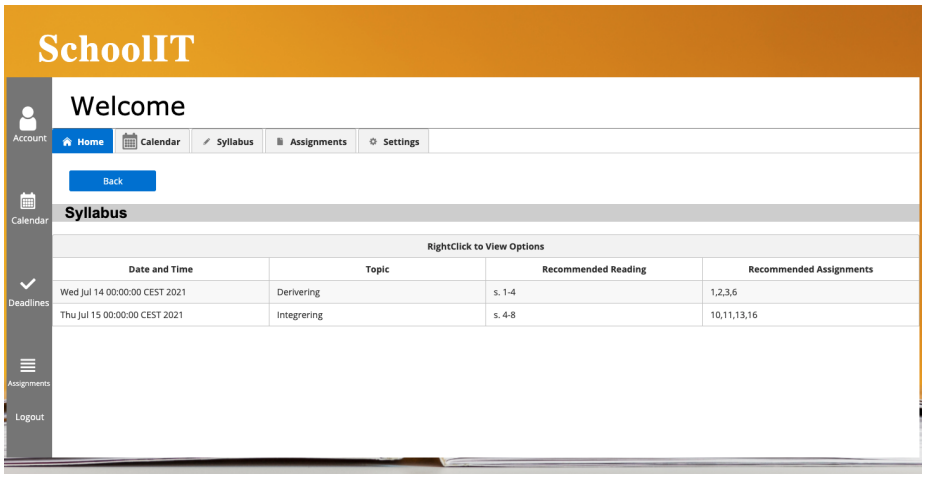
Figur 50: Kurshemsida för student

Studenten kan nu se en meny med olika kategorier. För tillfället är det bara Syllabus och Calendar som har en funktion. Sektionen under menyn implmenteras med en ny JSF template för att kunna uppdatera den delen av vyn för de olika kategorierna inuti kurshemsidan. Menyn är en primefaces tabmenu komponent och varje komponent inuti menyn är en primefaces menuitem komponent. Omredigeringen gör på samma sätt som beskrevs i sektion 4.4.5 där kurs id:et skickas som en parameter för att få fram relevant data. Samma koncept appliceras nu till de olika menuitem komponenterna och det illustreras i figur 51. Varje sektion hämtar parametern via FacesContext och hanteras sedan av sin egen Java böna.

```
<p:menuitem value="Calendar" outcome="/member/StudentCourseCal" icon="pi pi-fw pi-calendar">
    <f:param name="courseID" value="#{studentScheduleBackingBean.courseID}"/>
</p:menuitem>
<p:menuitem value="Syllabus" outcome="/member/StudentCourseSyll" icon="pi pi-fw pi-pencil">
    <f:param name="courseID" value="#{studentSyllabusBackingBean.courseID}"/>
</p:menuitem>
```

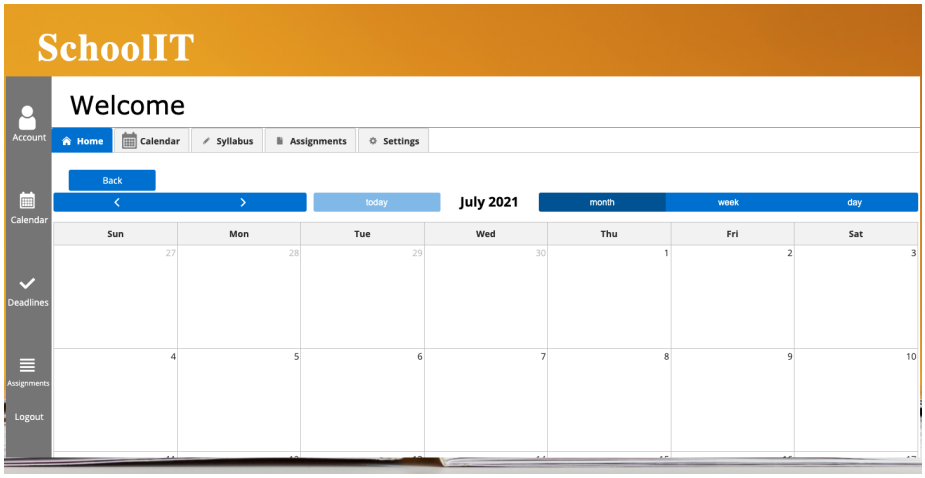
Figur 51: Primefaces menu komponenterna

Trycker studenten på Syllabys knappen updateras en sektion av vyn som sedan representerar vyn i figur 52. Informationen hämtas precis som i sektion 4.4.5 och representeras i en datatabell för studenten.

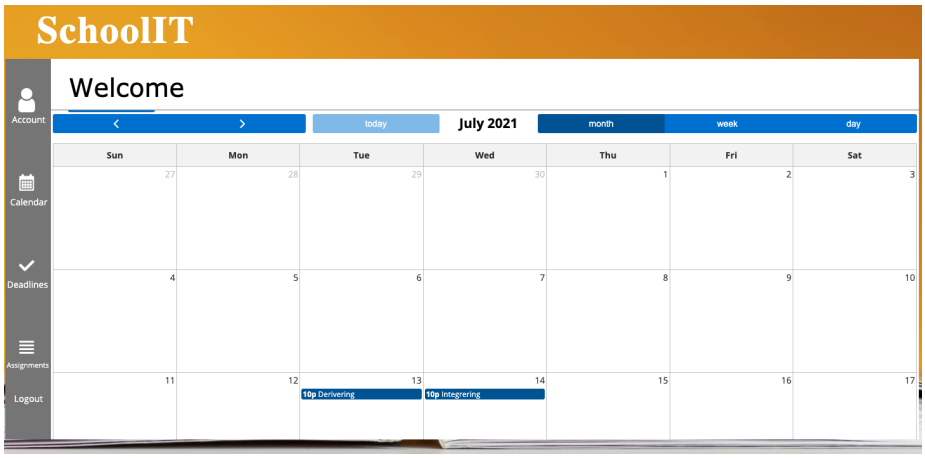


Figur 52: Kursplan för student kurshemsida

Nästa sektion som en student kan tas sig till är kalender schemat. Studenten kan klicka på Calender knappen och vyn kommer att updateras till följande vy i figur 53. Kalendern är en primfaces komponent som hanteras av StudentSchdule-BackingBean som bygger upp, omformaterar och lägger till datan till kalendern. Kurskalendern visar kursmodulerna från kursplanen. När studenten tar sig ner på vyn kommer den se de två modulerna som är tillagda i kursen enligt figur 54.



Figur 53: Kalender vy för student kurhemsida



Figur 54: Kursmodulern uppvisade i kurs kalendern

5 Resultat

Arbetet resulterade i en funktionell enterprise webbapplikation som byggdes med de olika mjukvaruspecifikationerna inom Java EE. De olika Java EE mjukvaru specifikationerna som tillämpades var Java Persistence API, Java Server Faces, Java EE security API, Context och dependency injection och Enterprise Java Bön teknologi.

För att hantera lagring av objekt i databasen samt strukturera den användes Java Persitence API. DAO - desing mönstret användes för att separera logiken och kommunikationen med databasen. Quersydsl användes för att kunna skriva mer eleganta queries för att kommunicera med databasen. Java Server Faces användes som användargränsnitt för att hantera interaktioner med klienter. Primefaces komponent biliotek användes för att använda klient komponenter som klienter kunde interagera med. Enterpirse Java bön teknologi användes för att hantera affärslogik och dessutom hantera komponenter på olika JSF vyer. Samt även utnyttjades Context och Dependency injektion för att instantiera olika objekt inom programmet. Java Security API utnyttjades för säker autentisering av klienter.

Figur 55: Funktionaliteten som uppfylldes i applikationen är markerat med rött

6 Diskussion

I följande sektionen kommer projektets arbetsgång att utvärderas samt de olika målen som sattes i början av projektet, vidarutveckling och design av applikationen.

6.1 Utvärdering av metod

Metoden som utfördes bidrog till en bra grund till att faställa en visualisering och grundfunktionalitet som skulle implementeras. Att följa ett agilt arbetssätt med hjälp av trello hjälpte till att ha översikt över all implementering som skulle appliceras, dock under projektets gång drevs arbetsättet mer ifrån scrum metodiken och projektet implementerades utifrån egen insikt. Anledningen till det här var för att lägga mer fokus på imlementering, dessutom är metodiken mer avsett för mjukvarutveckling inom en team baserad miljö vilket också även var en bidragande faktor till att man sakta började avvika från metodiken.

6.2 Utvärdering av Design

Inspirationen för mock-up designen inspirerades från andra lärplattformer och gav en visualisering av vad som skulle eftesträvas i applikationen. Däremot gjordes mock-ups bara i början av projektet för några vyer eftersom alltför mycket tid bör inte gå åt design. Tyvärr kunde inte design koncept dras ifrån olika intressenters perspektiv eftersom det skulle bli för tidskrävande. Tack vare primefaces kunde man använda avancerade komponenter tillsammans med JSF för att skynda på utvecklingen av frontenden med design och komponent interaktion. Primefaces erbjöd också teman till sina komponenter vilket insågs senare i projektet när en del implementering redan hade gjorts, när temat lades till i projektet skedde det komplikationer med css:n för just de komponenterna, därför kan knappar och textfält i figurerna se missplacerade ut, men förmodligen är det en snabb css lagning i projektet.

6.3 Utvärdering av Funktionalitet och Vidarutveckling

Funktionaliteten inspererades utifrån andra lärplattformar i undersökningsfasen som tillslut utgjordes till grundfunktionaliteten som skulle implementeras. Lärplattformar brukar oftast innehålla en oerhört stor mängd olika funktioner vilket inte hade varit rimligt inom en viss tidsram. Därför avgränsades applikationen med avseende för gymnasienivå eftersom mängden funktioner inte är lika nödvändig som på högre utblidningsnivå.

En del funktionalitet lyckades åstadkommas i applikationen. Tyvärr lyckades inte allt implementeras såsom skapande av uppgifter och filhantering. Databasen är implementerad för att hantera sådan data men tyvärr lyckades projektet inte komma så långt att implementera det vilket innebär att det kan inte uteslutas ifall det hade fungerat eller inte. Dock ska filuppladdning och lagring av filer vara möjligt inom Java EE, med JPA används det en byte array variable med

annotation @lob för att hantera fillagring. Dessutom har primefaces också komponenter som stödjer filupladdning. Kännedom om det tyder på att det finns stöd för att att tillämpa filuppladdnig i systemet.

6.4 Utvärdering av Java EE och Implementering

Apache derby är förmodligen inte det bästa när det kommer till att distrubera företagsövergripande program till en produktionsmiljö eftersom att den inte är tillräckligt skalbar för att hantera sådana applikationer. Men eftersom Java Persistence API är databasagnostisk (teknisk obunden), vilket innebär att det är kompatible med andra databassystem, så är det simpelt att byta ifall det skulle behövas. Apache Derby räcker väl för att användas i en utvecklingsmiljö.

Java Persistence API underlättar databasprogrammering när man jobbar med objektorienterat. Dock kan relations mappningen bli väldigt komplicerad vilket kan resultera i skapandet av extra tabeller istället för att addera en extra kolumn till en databastabell, vilket kan vara väldigt offektiv. Tur nog erbjöd netbeans en databas utforskare vilket bidrog till undersökning över hur databasen betedde sig som t.ex genererade tabeller och datalagring. Därför användes Arquillian testning i början av projektet bara på en liten del av databasen för att kolla att relations mappningen betedde sig enligt förväntat. Det innebär att det är möjligt att datalagring i en annan del av databasen som inte utnyttjas kan bete sig underligt eftersom det inte har fullt ut blivit testat.

Eftersom JSF är byggd ovanför Java Servlet API slipper man själv implementera servlets för att hanter klient begäran. Nackdelen det tillför är att det är svårt att problemsöka när vyn inte beter sig som den ska. En annan nackdel är att JSF är ett komponentbaserat ramverk vilket restrikterar utvecklare och tvingar de att följa deras komponent standard. Eftersom det är komponentbaserad kan man lätt lägga till komponentbibliotek, vilket gör att man kan lägga in komplexa komponenter utan att själv behöva implementera de. Enda nackdelen med det här är att komponenter kommer oftast med sin egna inbyggda css som måste överskrivas. Det här gör att design ändringar kan bli oerhört mer komplicerat än vad det behöver vara.

Java Enterprise Bönteknologi är själva komponenterna som sköter logiken i applikationen. Hanterat korrekt, vilket innebär att man låter all logik ligga på rätt nivå i applikationen, ska det vara väldigt skalbart. En miss som gjordes i projektet däremot är att man missupfattade de olika benämningarn som förekommer oftast i Java EE utveckling. Dessa var backingbean, managedbean och enterprise java bean. Managedbeans och backingbeans är oftast sedd som att vara samma sak i att de uppfyller samma syfte. Managedbean hanteras av JSF ramverket och defineras i faces-congi.xml medan en backingbean backar up själva vyn precis som det gjordes i systemkonstruktionen. Det diskuteras även att backingbean hanterar bara själva komponenterna i vyn. Missupfattningen skedde mellan Enterprise Java bönan (EJB) och backingbean/managedbean. EJB:s ska sköta

det tunga arbetet i affärslogiknivån, medan backingbean/managedbean ligger i webbnivån och ska bara ta emot och överföra data till EJB:s så de kan utföra de krävande jobben. Därför kan det ligga lite logik som backingbeans sköter. Eftersom det är i webbnivån där klienterna interagerar med servern och skickar olika begäran är det osäkert ifall det här lägger för mycket tryck på den delen av applikationen ifall det sköts logik bakom JSF vyerna.

Java Security API användes för autentisering vid inlogg. Implementeringen som följdes verkar vilja ha en entitet som användare med en användarroll som attribut för att kunna sedan specifiera säkerhetsroller. Men eftersom databasen var implementerad med två olika användare som entiteter (student och lärare), fick varsin entitet lägga till en ny attribut som är användarroll. Det är därför callerQuery i annotationen @DatabaseIdentityStore slår ihop både tabellerna vid autentiseringen. Det är inte undersökt ifall det här leder till några komplikationer i applikationen, men allt verkar fungera som det ska och den inloggade användare kan hämtas när den skickar begäran via securitycontext.

6.5 Möjligheter att appliceras på mobil plattform

Möjligheterna för att applicera webbapplikationen på en mobil plattform ser ut att vara begränsade. Det hittades en tredje part mjukvaru för att applicera Java EE server applikationer till mobila front - end. Dock är det här varken testat i projektet, och det verkar inte finnas mycket underliggande stöd för det. Det verkar dessutom inte finnas ett direkt standardiserat sätt att bygga ut till en multi - plattform applikation. En metod är att begära webbapplikationen via en mobil webbläsare. Dock för att få en bra klient upplevelse måste css:n har byggts med en responsiv design eftersom att en mobilskärm är mycket mindre än en standard dataskärm, vilket helt och hållet kan förstöra designen för en mindre skärm. Tyvärr kommer upplevelsen inte riktigt kännas som en ren mobil applikation.

6.6 Miljö och Etiska frågor

Många teknologier kan bidra till en effekt på miljön, både negativt eller positivt. Applikationer som utnyttjas inom en företagsmiljö har reducerat pappers avfallet i en stadig mängd [23]. Lärplattformer är inte ett undantag inom den kategorin. Eftersom lärare kan utnyttja digitala format för att distrubera sitt kursmaterial leder det till minskad kopiering och distribution av papper. Däremot leder det till ett mer beroende för teknologiska produkter och energikonsumering för att driva dem. Samtidigt som papper är ett utav det material som återvinns mest [24]. Att avgöra vad som är mest hållbart är inte helt klart.

Eftersom utveckling i form av applikationer kan bidra till ett ständingt mänskligt beroende av tekniska produkter kan det leda till mer utsatthet för skärmtid, vilket kan leda till olika former av hälsorisker [25]. Studier har även visat att skärmtiden bland unga har ökat. Det här väcker frågan över om det

är rätt att försöka introducera lärplattformer på lägre akademiska nivåer för att försöka skapa en möjlighet till att underlätta undervisningen.

References

[1] Instructure Global Ltd. *Your classroom. To the power of Canvas LMS.* URL: https://www.instructure.com/en-gb/product/canvas/schools/lms. (acessed: 17 - 01 - 2022).

[2] Moodle. *About Moodle.* URL: https://docs.moodle.org/311/en/About_Moodle. (acessed: 17 - 01 - 2022).

[3] Google. *Where teaching and learning come together.* URL: https://edu.google.com/intl/en_ALL/products/classroom/. (acessed: 17 - 01 - 2022).

[4] Julia Rydstrand. *Från klassrum till distans – så går du tillväga.* URL: https://skola.gymnasium.se/artiklar/fr%C3%A5n-klassrum-till-distans. (acessed: 17 - 01 - 2022).

[5] Lenita Jällhage. *Checklista: Så förbereder du dig bäst för distansundervisning.* URL: https://www.lararen.se/nyheter/distansundervisning/checklista-sa-forbereder-du-dig-bast-for-distansundervisning. (acessed: 17 - 01 - 2022).

[6] Stefan Pålsson. *Distansundervisning på Lerums gymnasium.* URL: https://skolahemma.se/distansundervisning-pa-lerums-gymnasium/. (acessed: 17 - 01 - 2022).

[7] Dr. Sherimon P.C Buthaina Almur Salim Al Handhali Anwar Taeeb Najim Al Rasbi. *Advantages and Disadvantages of Learning Management System (LMS) at AOU Oman.* URL: https://arivjournal.com/technology/advantages-and-disadvantages-of-learning-management-system-lms-at-aou-oman/. (acessed: 17 - 01 - 2022).

[8] Netbeans. *Oracle Proposes Netbeans As Apache Incubator Project.* URL: https://netbeans.org/community/apache-incubator.html. (accessed: 08 - 06 - 2020).

[9] Apache Software Foundation. *Welcome To Apache Maven.* URL: https://maven.apache.org/. (accessed: 08 - 06 - 2020).

[10] Apache Software Foundation. *Introduction To The POM.* URL: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html. (accessed: 08 - 06 - 2020).

[11] Oracle. *Overview of Enterprise Applications.* URL: https://docs.oracle.com/javaee/7/firstcup/java-ee001.htm. (accessed: 9 - 8 - 2021).

[12] Oracle. *Your First Cup.* URL: https://docs.oracle.com/javaee/6/firstcup/doc/gcrky.html. (accessed: 10 - 7 - 2020).

[13] Oracle Corporation. *Java EE Containers.* URL: https://javaee.github.io/tutorial/overview005.html#BNABO. (acessed: 07 - 01 - 2022).

[14] Oracle. *Your First Cup.* URL: https://docs.oracle.com/javaee/6/firstcup/doc/gcrkq.html. (accessed: 10 - 7 - 2020).

[15] Payara Services Ltd. *Payara Server Enterprise*. URL: https://www.payara.fish/products/payara-server/. (acessed: 18 - 08 - 2021).

[16] Apache Software Foundation. *Apache Derby*. URL: https://db.apache.org/derby/. (accessed: 16 - 06 - 2021).

[17] Payara Services Ltd. *Configure a connection pool*. URL: https://docs.payara.fish/community/docs/documentation/user-guides/connection-pools/connection-pools.html. (accessed: 16 - 06 - 2021).

[18] Oracle Corporation. *Persistence Units*. URL: https://docs.oracle.com/cd/E19798-01/821-1841/bnbrj/index.html. (acessed: 19 - 08 - 2021).

[19] Red Hat Inc. *Aruquillian Invasion!* URL: https://arquillian.org/invasion/. (acessed: 24 - 06 - 2021).

[20] Oracle Corporation. *JavaServer Faces Technology Benefits*. URL: https://javaee.github.io/tutorial/jsf-intro003.html. (acessed: 19 - 08 - 2021).

[21] Oracle Corporation. *The Lifecycle of a JavaServer Faces Application*. URL: https://javaee.github.io/tutorial/jsf-intro007.html#BNAQQ. (acessed: 19 - 08 - 2021).

[22] PrimeTek. *Leading Provider of Open Source UI Component Libraries*. URL: https://www.primefaces.org/. (acessed: 07 - 01 - 2022).

[23] Jane Wakefield. *Can paper survive the digital age?* URL: https://www.bbc.com/news/technology-32130647. (acessed: 07 - 01 - 2022).

[24] Alison Moodie. *Is digital really greener than paper?* URL: https://www.theguardian.com/sustainable-business/digital-really-greener-paper-marketing. (acessed: 07 - 01 - 2022).

[25] Jon Johnson. *Negative effects of technology: What to know*. URL: https://www.medicalnewstoday.com/articles/negative-effects-of-technology. (acessed: 07 - 01 - 2022).