

Projektuppgift

Deluppgift B – Fungerande prototyp

Programutveckling med Java II – D0024E

2018-05-18

Christoffer Eduards

Inledning

Uppgiften utförs i kursen D0024E och har bestått av två deluppgifter, dels en designdel och nu denna implementationsdel. Redovisningen av uppgiften sker med hjälp av en stor mängd screenshots på både gränssnitt och kod. Motiveringen till detta är att det upplevs som ett effektivt sätt att visa på förståelse för det objektorienterade synsättet.

Implementationen består av två delar, dels en webbklient (JSF) och dels en desktop klient (JFrame). Då uppgiften har programmerats enligt MVC var det möjligt att använda samma Model och Controller för båda klienterna, med reservation för att något extra SQL-kommando behövdes läggas till vid skapandet av webbklienten.

Därför kommer först Model och Controller presenteras på ett övergripande sätt genom skärmbild av kod och korta förklaringar av innehåll. Här presenteras även backing bean för webbklienten. Efter detta kommer specifika delar för de olika interfacen presenteras i kombination med screenshots på kod som är fundamental för motsvarande funktionalitet.

Uppgiften krävde en databas vilken presenteras i form av den logiska modellen samt skärmbilder av tabeller.

Resultatdelen innehåller även en redovisning av arbetsprocessen och vilka ändringar som har gjorts från design till implementation.

I diskussionsavsnittet beskrivs svårigheter med uppgiften och vad som hade kunnat förbättras.

Resultat

Arbetsprocessen

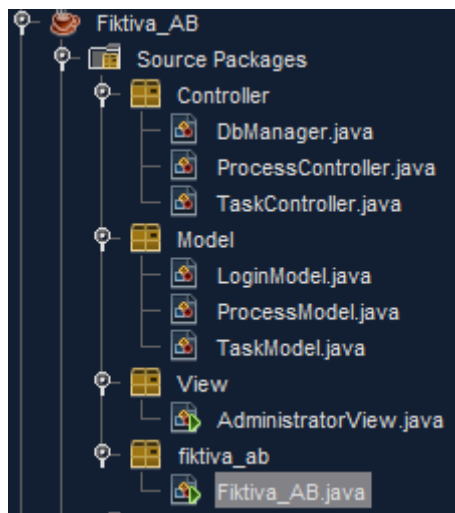
Arbetet påbörjades genom att först skapa desktopklienten. Detta gjordes genom att succesivt implementera ny funktionalitet. Innan implementation av funktionalitet gjordes små tester med mindre mängd data för att skapa förståelse för hur en tänkt funktionalitet programmeras. Detta byggdes sedan ut för att få den slutliga funktionaliteten och utseendet.

Sist skapades webbklienten. I detta projekt importerades Model och Controller från desktopklienten. Här ägnades mycket tid åt att förstå hur JSF och HTML fungerar innan uppgiftsspecifika saker kunde börja programmeras. När det väl klickade gick själva implementationen den funktionalitet som supportpersonalen behöver förhållandevis fort.

MVC

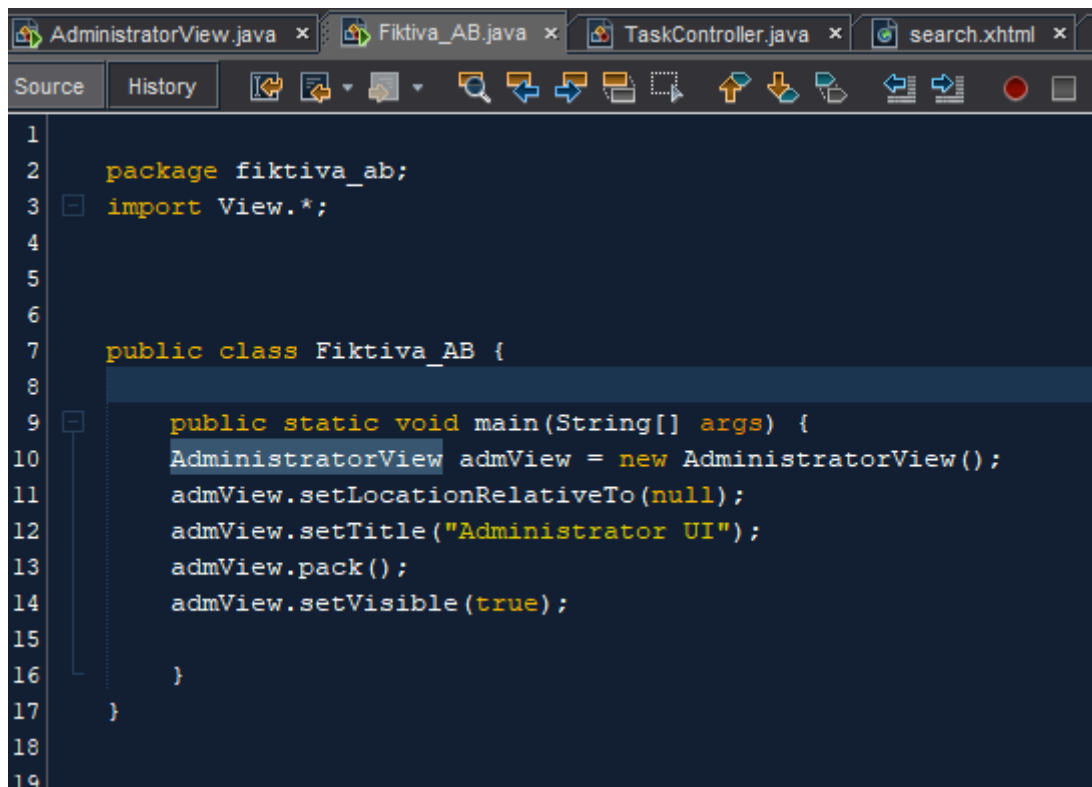
JFrame

I figur 1 nedan presenteras strukturen på koden för desktop klienten.



Figur 1

I figur 2 visas den klass där main-metoden finns för desktopklienten.

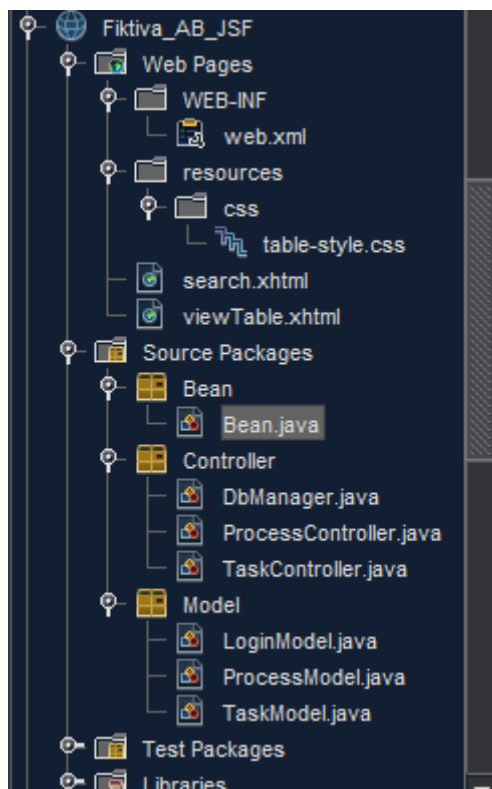


```
1 package fiktiva_ab;
2 import View.*;
3
4
5
6
7 public class Fiktiva_AB {
8
9     public static void main(String[] args) {
10         AdministratorView admView = new AdministratorView();
11         admView.setLocationRelativeTo(null);
12         admView.setTitle("Administrator UI");
13         admView.pack();
14         admView.setVisible(true);
15     }
16 }
17
18
19
```

Figur 2

JSF

I figuren nedan presenteras strukturen på koden för webbklienten.

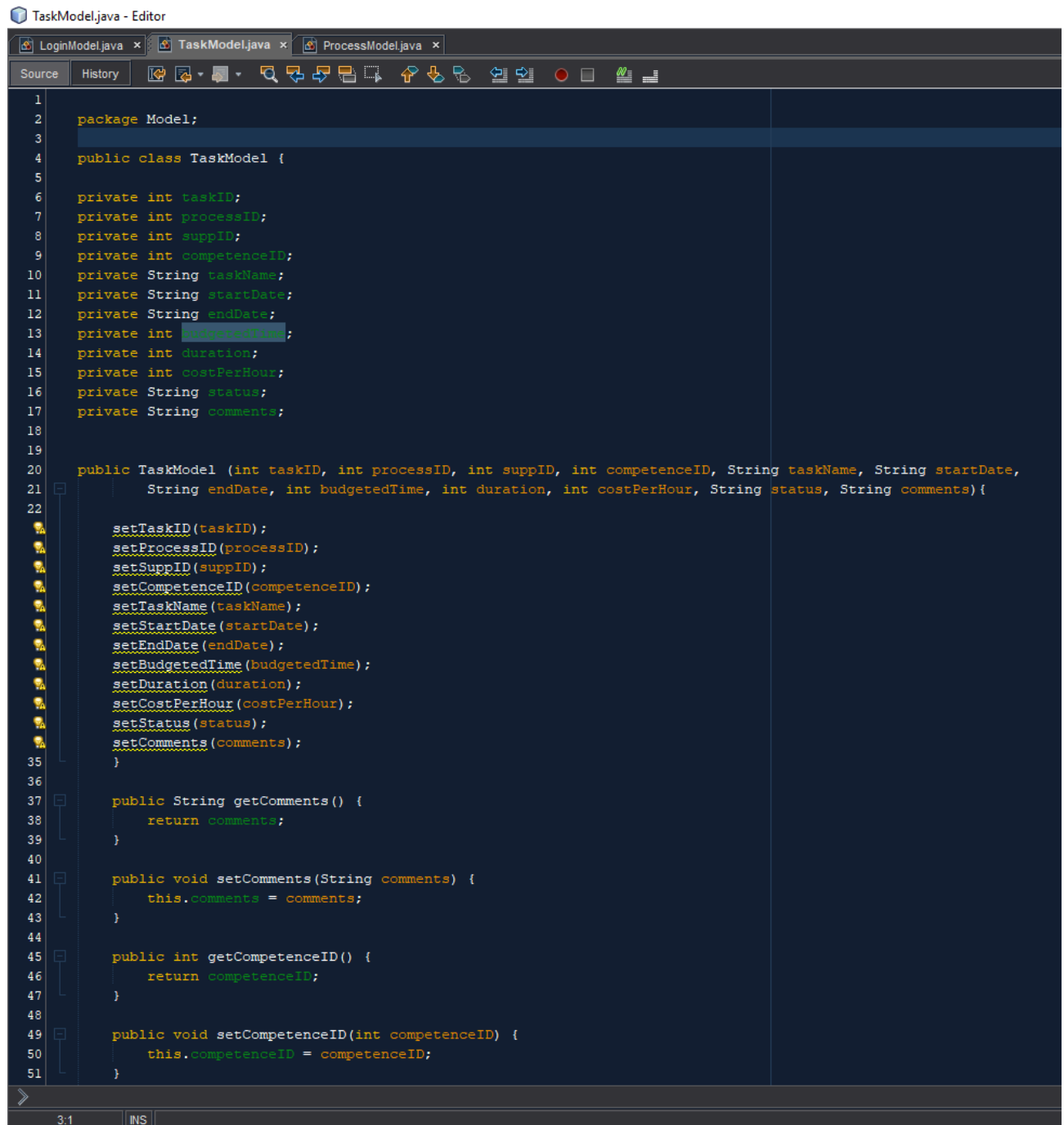


Figur 3

Model

TaskModel

Denna Model-klass innehåller setters och getters för all data som rör en task. Den motsvarar task-tabellen i databasen. Ett utdrag ur koden kan ses nedan.

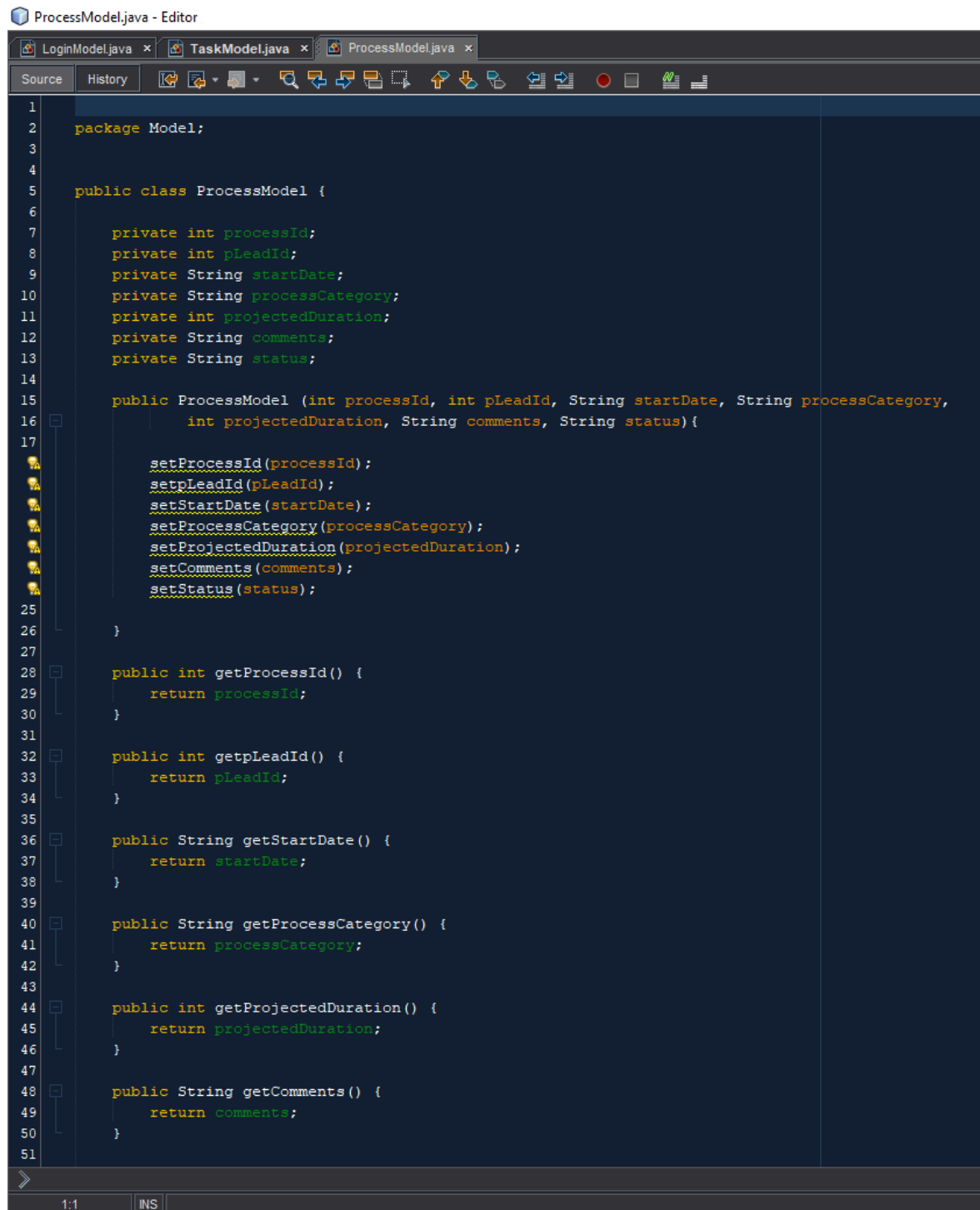


```
1 package Model;
2
3
4 public class TaskModel {
5
6     private int taskID;
7     private int processID;
8     private int suppID;
9     private int competenceID;
10    private String taskName;
11    private String startDate;
12    private String endDate;
13    private int budgetedTime;
14    private int duration;
15    private int costPerHour;
16    private String status;
17    private String comments;
18
19
20    public TaskModel (int taskID, int processID, int suppID, int competenceID, String taskName, String startDate,
21        String endDate, int budgetedTime, int duration, int costPerHour, String status, String comments){
22
23        setTaskID(taskID);
24        setProcessID(processID);
25        setSuppID(suppID);
26        setCompetenceID(competenceID);
27        setTaskName(taskName);
28        setStartDate(startDate);
29        setEndDate(endDate);
30        setBudgetedTime(budgetedTime);
31        setDuration(duration);
32        setCostPerHour(costPerHour);
33        setStatus(status);
34        setComments(comments);
35    }
36
37    public String getComments() {
38        return comments;
39    }
40
41    public void setComments(String comments) {
42        this.comments = comments;
43    }
44
45    public int getCompetenceID() {
46        return competenceID;
47    }
48
49    public void setCompetenceID(int competenceID) {
50        this.competenceID = competenceID;
51    }
52}
```

Figur 4

ProcessModel

Denna Model-klass innehåller setters och getters för all data som rör en process. Den motsvarar process-tabellen i databasen. Ett utdrag ur koden kan ses nedan.

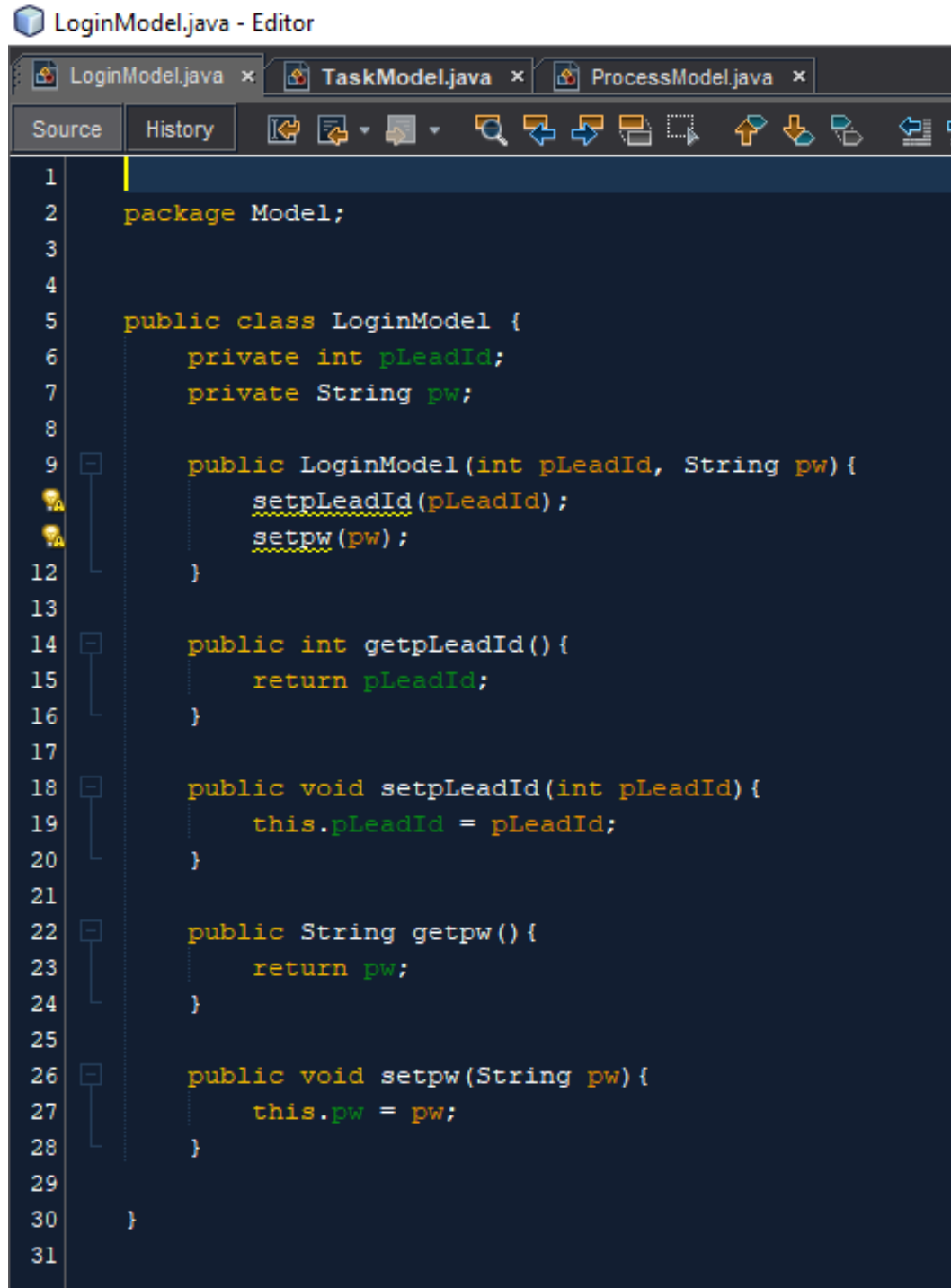


```
1 package Model;
2
3
4
5 public class ProcessModel {
6
7     private int processId;
8     private int pLeadId;
9     private String startDate;
10    private String processCategory;
11    private int projectedDuration;
12    private String comments;
13    private String status;
14
15    public ProcessModel (int processId, int pLeadId, String startDate, String processCategory,
16                        int projectedDuration, String comments, String status){
17
18        setProcessId(processId);
19        setpLeadId(pLeadId);
20        setStartDate(startDate);
21        setProcessCategory(processCategory);
22        setProjectedDuration(projectedDuration);
23        setComments(comments);
24        setStatus(status);
25    }
26
27
28    public int getProcessId() {
29        return processId;
30    }
31
32    public int getpLeadId() {
33        return pLeadId;
34    }
35
36    public String getStartDate() {
37        return startDate;
38    }
39
40    public String getProcessCategory() {
41        return processCategory;
42    }
43
44    public int getProjectedDuration() {
45        return projectedDuration;
46    }
47
48    public String getComments() {
49        return comments;
50    }
51
52 }
```

Figur 5

LoginModel

I uppgiften gjordes ett undantag från kravspecifikationen i desktopklienten och en form av en inloggning skapades. Inga lösenord finns i systemet utan klassen används i nuläget endast för att systemet skall spara processledarens ID medan denne är "inloggad". LoginModel som helhet kan ses i figuren nedan.



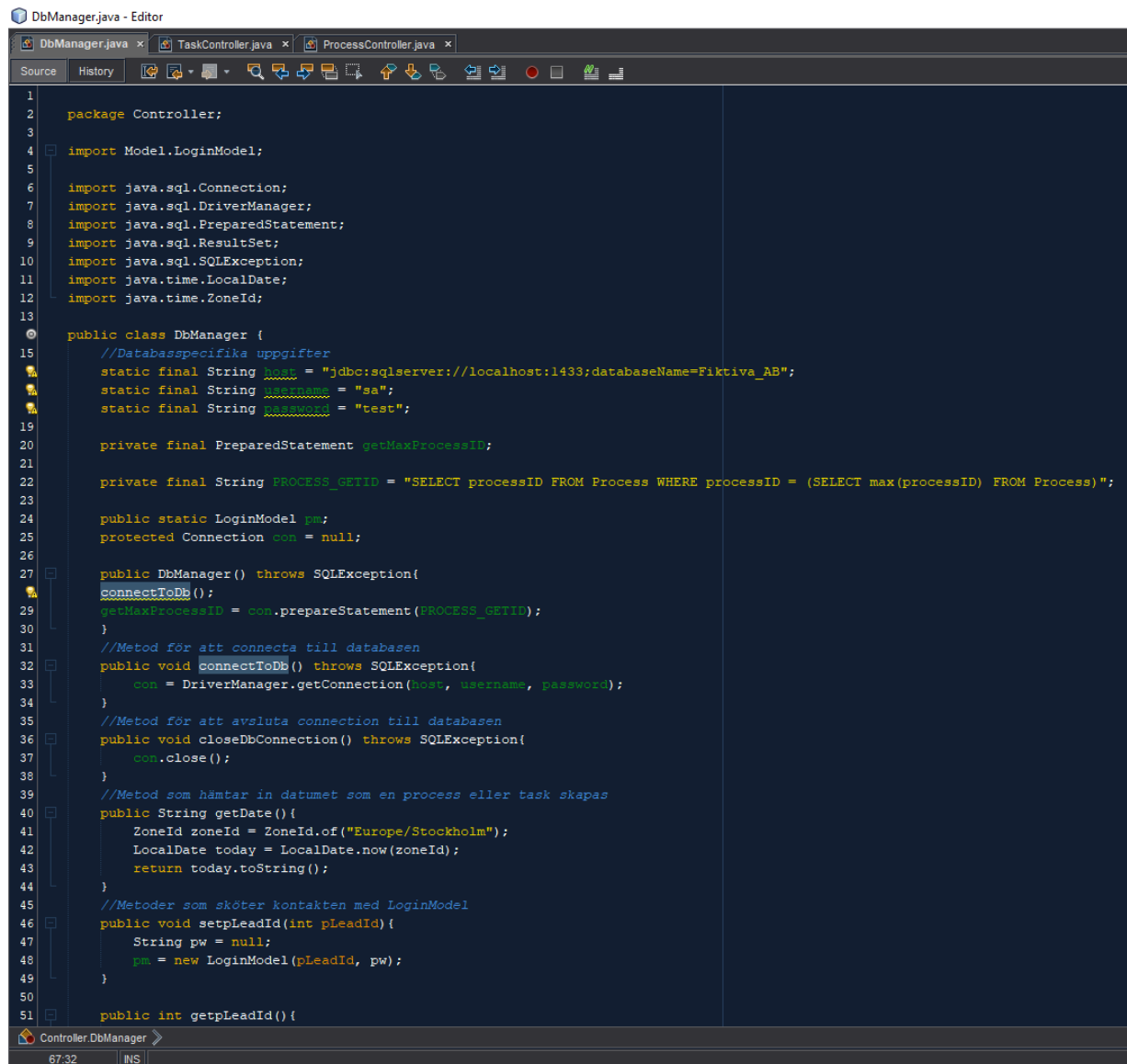
```
1
2 package Model;
3
4
5 public class LoginModel {
6     private int pLeadId;
7     private String pw;
8
9     public LoginModel(int pLeadId, String pw){
10         setpLeadId(pLeadId);
11         setpw(pw);
12     }
13
14     public int getpLeadId(){
15         return pLeadId;
16     }
17
18     public void setpLeadId(int pLeadId){
19         this.pLeadId = pLeadId;
20     }
21
22     public String getpw(){
23         return pw;
24     }
25
26     public void setpw(String pw){
27         this.pw = pw;
28     }
29
30 }
31
```

Figur 6

Controller

DbManager

I denna klass finns all information som krävs för att programmet skall få kontakt med rätt databas, vilket TaskController och ProcessController sedan ärver. På grund av detta arv innehåller klassen även metoder som krävs i både TaskController och ProcessController. Ett utdrag från kod av DbManager kan ses i figuren nedan.

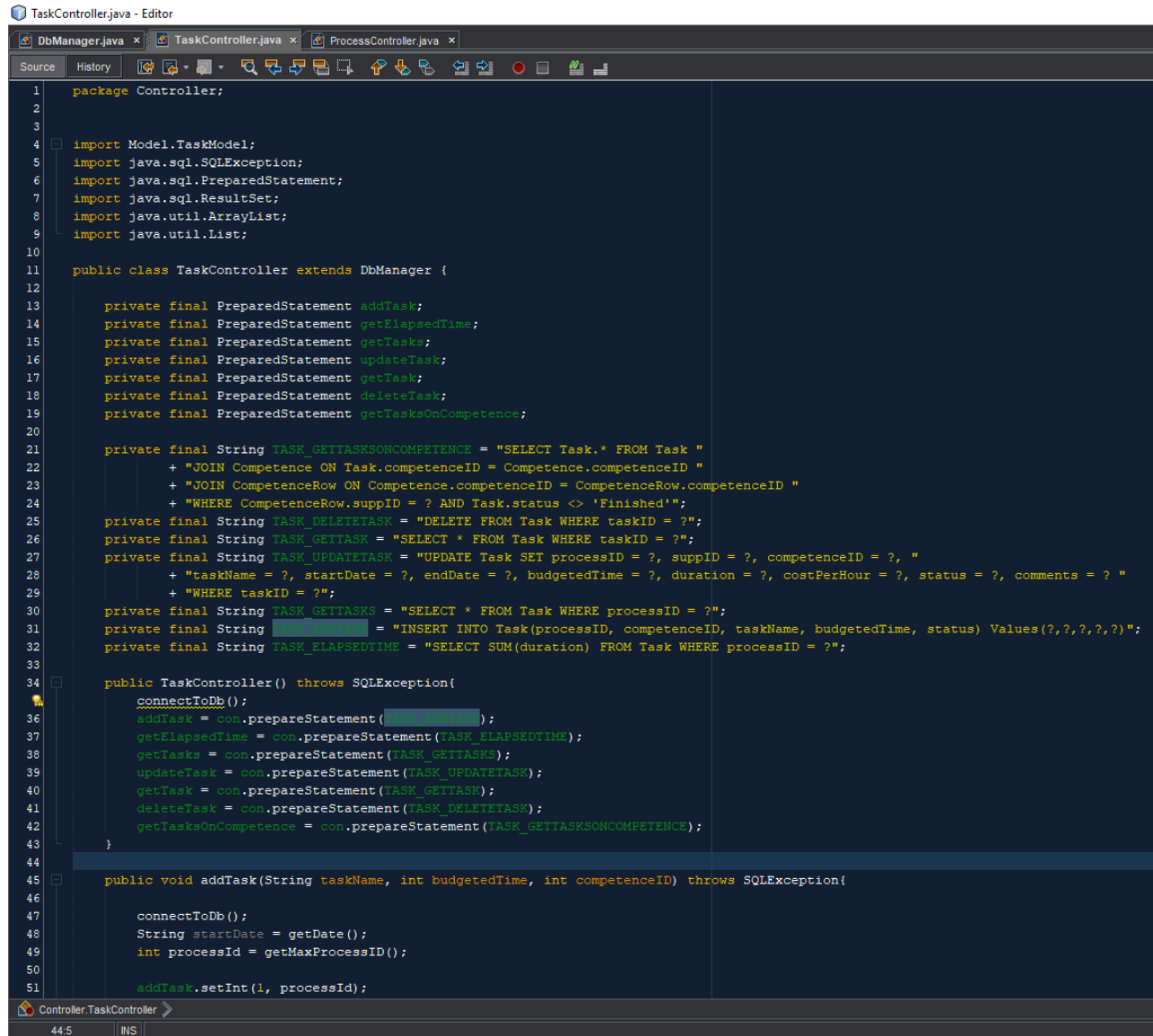


```
1 package Controller;
2
3
4 import Model.LoginModel;
5
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.PreparedStatement;
9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import java.time.LocalDate;
12 import java.time.ZoneId;
13
14 public class DbManager {
15     //Databasspecifika uppgifter
16     static final String host = "jdbc:sqlserver://localhost:1433;databaseName=Fiktiva_AB";
17     static final String username = "sa";
18     static final String password = "test";
19
20     private final PreparedStatement getMaxProcessID;
21
22     private final String PROCESS_GETID = "SELECT processID FROM Process WHERE processID = (SELECT max(processID) FROM Process)";
23
24     public static LoginModel pm;
25     protected Connection con = null;
26
27     public DbManager() throws SQLException{
28         connectToDb();
29         getMaxProcessID = con.prepareStatement(PROCESS_GETID);
30     }
31     //Metod för att connecta till databasen
32     public void connectToDb() throws SQLException{
33         con = DriverManager.getConnection(host, username, password);
34     }
35     //Metod för att avsluta connection till databasen
36     public void closeDbConnection() throws SQLException{
37         con.close();
38     }
39     //Metod som hämtar in datumet som en process eller task skapas
40     public String getDate(){
41         ZoneId zoneId = ZoneId.of("Europe/Stockholm");
42         LocalDate today = LocalDate.now(zoneId);
43         return today.toString();
44     }
45     //Metoder som sköter kontakten med LoginModel
46     public void setpLeadId(int pLeadId){
47         String pw = null;
48         pm = new LoginModel(pLeadId, pw);
49     }
50
51     public int getpLeadId(){
```

Figur 7

TaskController

Här finns all logik som krävs för CRUD-operationer i databasen när det gäller Tasks. Klassen ärver från DbManager. Ett utdrag av koden kan ses nedan.

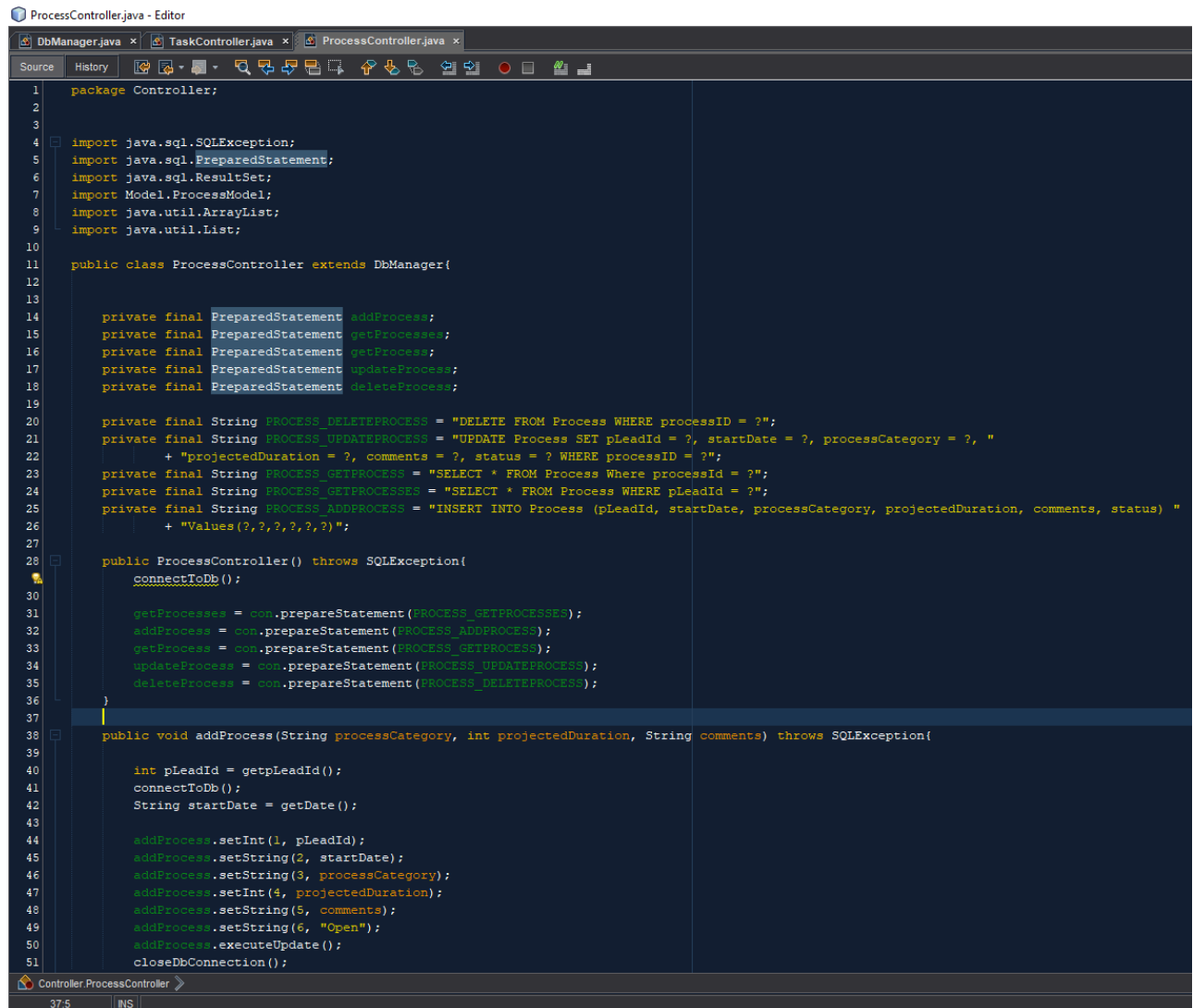


```
1 package Controller;
2
3
4 import Model.TaskModel;
5 import java.sql.SQLException;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class TaskController extends DbManager {
12
13     private final PreparedStatement addTask;
14     private final PreparedStatement getElapsedTime;
15     private final PreparedStatement getTasks;
16     private final PreparedStatement updateTask;
17     private final PreparedStatement getTask;
18     private final PreparedStatement deleteTask;
19     private final PreparedStatement getTasksOnCompetence;
20
21     private final String TASK_GETTASKSONCOMPETENCE = "SELECT Task.* FROM Task "
22         + "JOIN Competence ON Task.competenceID = Competence.competenceID "
23         + "JOIN CompetenceRow ON Competence.competenceID = CompetenceRow.competenceID "
24         + "WHERE CompetenceRow.supplID = ? AND Task.status <> 'Finished'";
25     private final String TASK_DELETETASK = "DELETE FROM Task WHERE taskID = ?";
26     private final String TASK_GETTASK = "SELECT * FROM Task WHERE taskID = ?";
27     private final String TASK_UPDATETASK = "UPDATE Task SET processID = ?, supplID = ?, competenceID = ?, "
28         + "taskName = ?, startDate = ?, endDate = ?, budgetedTime = ?, duration = ?, costPerHour = ?, status = ?, comments = ? "
29         + "WHERE taskID = ?";
30     private final String TASK_GETTASKS = "SELECT * FROM Task WHERE processID = ?";
31     private final String TASK_ADDTASK = "INSERT INTO Task(processID, competenceID, taskName, budgetedTime, status) Values(?,?,?,?,?)";
32     private final String TASK_ELAPSEDTIME = "SELECT SUM(duration) FROM Task WHERE processID = ?";
33
34     public TaskController() throws SQLException{
35         connectToDb();
36         addTask = con.prepareStatement(TASK_ADDTASK);
37         getElapsedTime = con.prepareStatement(TASK_ELAPSEDTIME);
38         getTasks = con.prepareStatement(TASK_GETTASKS);
39         updateTask = con.prepareStatement(TASK_UPDATETASK);
40         getTask = con.prepareStatement(TASK_GETTASK);
41         deleteTask = con.prepareStatement(TASK_DELETETASK);
42         getTasksOnCompetence = con.prepareStatement(TASK_GETTASKSONCOMPETENCE);
43     }
44
45     public void addTask(String taskName, int budgetedTime, int competenceID) throws SQLException{
46
47         connectToDb();
48         String startDate = getDate();
49         int processID = getMaxProcessID();
50
51         addTask.setInt(1, processID);
```

Figur 8

ProcessController

Här finns all logik som krävs för CRUD-operationer i databasen när det gäller Processes. Klassen ärver från DbManager. Ett utdrag av koden kan ses nedan.




```
1 package Controller;
2
3
4 import java.sql.SQLException;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import Model.ProcessModel;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class ProcessController extends DbManager{
12
13     private final PreparedStatement addProcess;
14     private final PreparedStatement getProcesses;
15     private final PreparedStatement getProcess;
16     private final PreparedStatement updateProcess;
17     private final PreparedStatement deleteProcess;
18
19     private final String PROCESS_DELETEPROCESS = "DELETE FROM Process WHERE processID = ?";
20     private final String PROCESS_UPDATEPROCESS = "UPDATE Process SET pLeadId = ?, startDate = ?, processCategory = ?, "
21         + "projectedDuration = ?, comments = ?, status = ? WHERE processID = ?";
22     private final String PROCESS_GETPROCESS = "SELECT * FROM Process Where processId = ?";
23     private final String PROCESS_GETPROCESSES = "SELECT * FROM Process WHERE pLeadId = ?";
24     private final String PROCESS_ADDPROCESS = "INSERT INTO Process (pLeadId, startDate, processCategory, projectedDuration, comments, status) "
25         + "Values(?, ?, ?, ?, ?, ?)";
26
27     public ProcessController() throws SQLException{
28         connectToDb();
29
30         getProcesses = con.prepareStatement(PROCESS_GETPROCESSES);
31         addProcess = con.prepareStatement(PROCESS_ADDPROCESS);
32         getProcess = con.prepareStatement(PROCESS_GETPROCESS);
33         updateProcess = con.prepareStatement(PROCESS_UPDATEPROCESS);
34         deleteProcess = con.prepareStatement(PROCESS_DELETEPROCESS);
35     }
36
37     public void addProcess(String processCategory, int projectedDuration, String comments) throws SQLException{
38
39         int pLeadId = getpLeadId();
40         connectToDb();
41         String startDate = getDate();
42
43         addProcess.setInt(1, pLeadId);
44         addProcess.setString(2, startDate);
45         addProcess.setString(3, processCategory);
46         addProcess.setInt(4, projectedDuration);
47         addProcess.setString(5, comments);
48         addProcess.setString(6, "Open");
49         addProcess.executeUpdate();
50         closeDbConnection();
51 }
```

Figur 9

Bean

I Bean finns all JSF-logik som kopplar samman supportpersonalens View med databasen. Koden som helhet kan ses i kommande två figurer.



```
1 package Bean;
2
3
4 import Controller.TaskController;
5 import Model.TaskModel;
6 import java.io.Serializable;
7 import java.sql.SQLException;
8 import java.util.List;
9 import javax.faces.bean.ManagedBean;
10 import javax.faces.bean.SessionScoped;
11
12 @ManagedBean
13 @SessionScoped
14 public class Bean implements Serializable{
15
16     //En lista med taskmodels, behövs för att skriva ut alla tasks i dataTabeln
17     private List<TaskModel> list;
18     private TaskModel tm;
19     //Variabel som behövs för att kunna editera en task
20     private boolean edit;
21     //Sparar vilket suppID som är angivet för att få fram tasksen
22     private Integer suppID;
23
24     public Bean(){
25     }
26     //Söker fram de tasks som en viss supportperson har kompetens för och som inte redan är utförda
27     public String search(){
28         try {
29             TaskController tc = new TaskController();
30             list = tc.getTasksOnCompetence(suppID);
31         } catch (SQLException ex) {
32             System.out.println("Fel vid initieringen: " + ex.getMessage());
33         }
34         return "viewTable";
35     }
36
37     public String getSearchPage(){
38         return "search";
39     }
40
41     public void edit(TaskModel tm){
42         this.tm = tm;
43         edit = true;
44     }
45     //Metod som uppdaterar arbetsuppgiften i databasen med angivna värden
46     public void save(){
47         try {
48             TaskController tc = new TaskController();
49             tc.updateTask(tm);
50             edit = false;
51         } catch (SQLException ex) {
```

Figur 10

```

44     }
45     //Metod som uppdaterar arbetsuppgiften i databasen med angivna värden
46     public void save() {
47         try {
48             TaskController tc = new TaskController();
49             tc.updateTask(tm);
50             edit = false;
51         } catch (SQLException ex) {
52             System.out.println("Fel vid uppdateringen: " + ex.getMessage());
53         }
54     }
55
56
57     public List<TaskModel> getList() {
58         return list;
59     }
60
61     public Integer getSuppID() {
62         return suppID;
63     }
64
65     public void setSuppID(Integer suppID) {
66         this.suppID = suppID;
67     }
68
69     public TaskModel getTm() {
70         return tm;
71     }
72
73     public boolean isEdit() {
74         return edit;
75     }
76 }
77

```

Figur 11

View – Grafiskt och kod

Processledarens View

I figur 12 ses startsidan för processledarens desktop-klient. Denna symboliserar en inloggningssida och hade kunnat fungera som en sådan då systemet sparar både ID och password i LoginModel. Detta hade exempelvis kunnat kontrolleras mot inloggningsuppgifter sparade i databasen. Så som systemet fungerar nu är att det sparar processledaren ID när programmet körs för att ha koll på vilken person som använder systemet.



Figur 12

När Login trycks körs kod som kan ses i figur 13 nedan. Felhantering av felaktigt inslaget ID sker i tryParse.

```
969 private void login_jbActionPerformed(java.awt.event.ActionEvent evt) {  
970  
971     CardLayout card = (CardLayout) mainPanel.getLayout();  
972     card.show(mainPanel, "mainMenu");  
973  
974     initDbManager();  
975     int pLeadId = dbm.tryParse(id_field.getText());  
976     dbm.setpLeadId(pLeadId);  
977 }
```

Figur 13

Som koden ovan visar fås cardet "mainMenu" när Login trycks. Denna meny kan ses i figur 14.

Menyn består av tre val, antingen att skapa en ny process, se de processer med tillhörande tasks som tillhör den processledare som är inloggad, eller "logga ut".

ÄHS - Fiktiva AB

Create new Process

View my Processes

Sign Out

Figur 14

När "Create new Process" aktiveras körs koden som kan ses i figuren nedan.

```
898 private void createProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {  
899     initDbManager();  
900     //Gör så att cardet createProcess visas när knappen trycks  
901     CardLayout card = (CardLayout) mainPanel.getLayout();  
902     card.show(mainPanel, "createProcess");  
903     //På kommande createProcess-sidan skrivs korrekt process-id ut genom nedanstående kod  
904     try {  
905         int processID = dbm.getMaxProcessID();  
906         preprintID_field.setText(Integer.toString(processID + 1));  
907     } catch (SQLException ex) {  
908         System.out.println("Fel i hämtningen av maxProcessID: " + ex.getMessage());  
909     }  
910 }
```

Figur 15

På följande sida har processledaren möjlighet att skapa en ny process. Processnummret (processID) hämtas från databasen och skrivs automatiskt ut. Processledaren ombeds sedan att fylla i kategori, arbetsuppgifter och eventuella kommentarer. Gällande arbetsuppgifterna skriver processledaren in namnet, den budgeterade tiden och vilken form av kompetens som uppgiften kräver i form av ett ID som hämtas från databasen. I efterhand hade det varit snyggare om kompetensen hade kunnat väljas genom exempelvis en drop-down-meny med alternativ istället för ett ID.

Den budgeterade tiden för de ingående uppgifterna sammanställs vilket per automatik blir den budgeterade tiden för hela processen.

Process number:

Enter category:

Specify tasks:

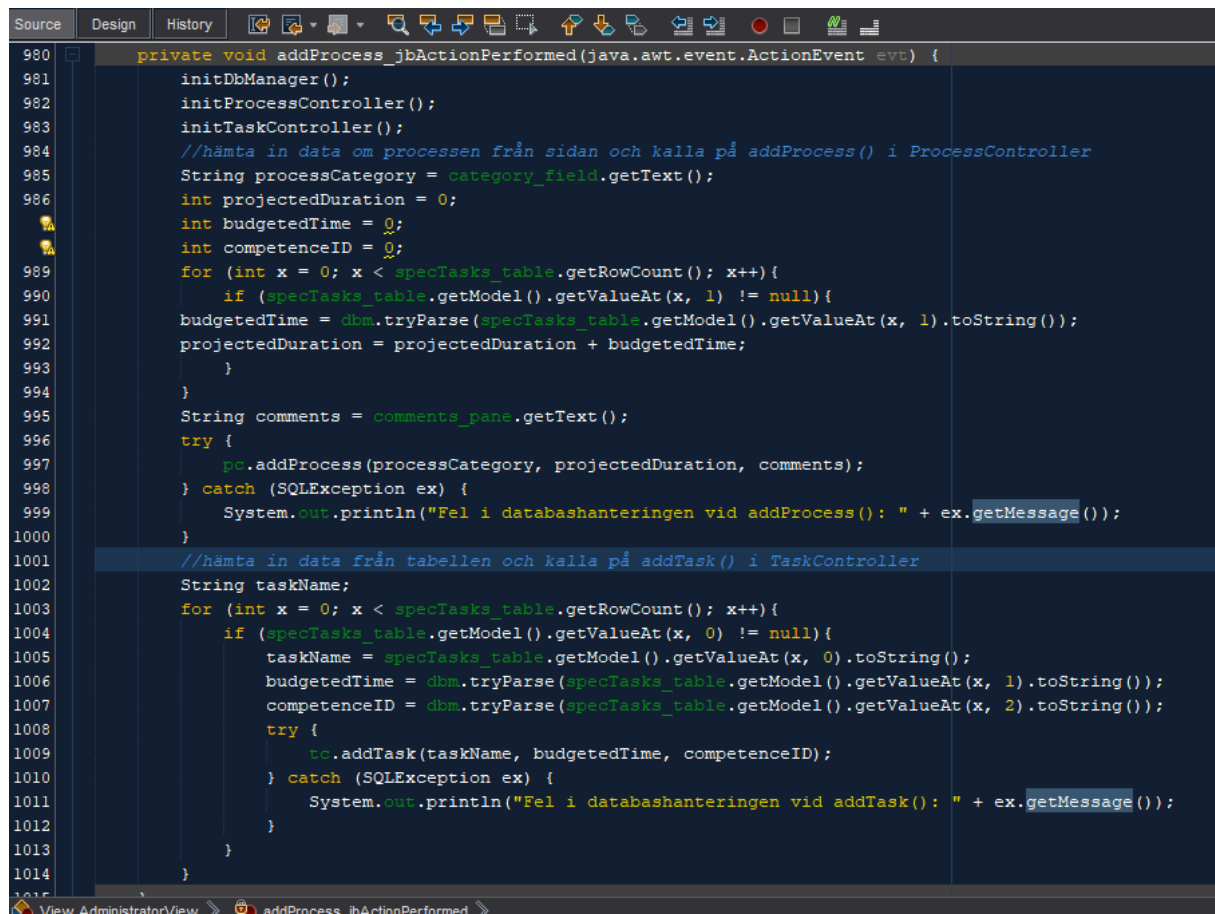
Name	Budgeted time	CompetenceID
Unable to login	10	2
Program freezing	5	1

Additional comments:

This is some example comments

Figur 16

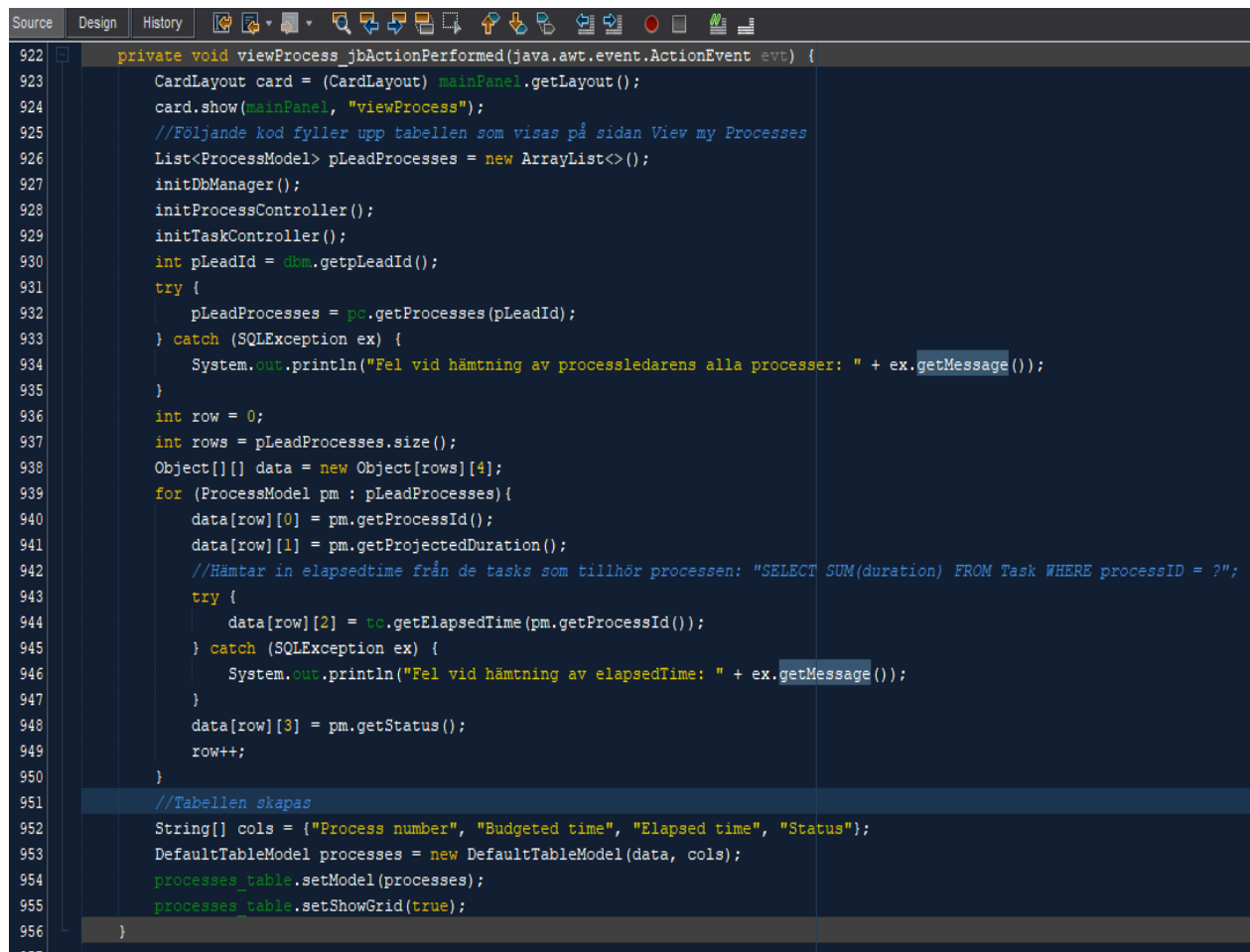
Följande kod som kan ses i nedanstående figur körs när "Add process" trycks.

The image is a screenshot of an IDE window showing a Java source file. The code is for a method named `addProcess_jbActionPerformed` which takes a `java.awt.event.ActionEvent` parameter. The code is written in a dark-themed editor. It includes comments in Swedish. The code logic involves initializing managers, getting data from a `category_field`, calculating `projectedDuration` and `budgetedTime` by iterating through `specTasks_table`, and then calling `pc.addProcess` and `tc.addTask` with the calculated values. It also includes exception handling for `SQLException`.

```
980 private void addProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {  
981     initDbManager();  
982     initProcessController();  
983     initTaskController();  
984     //hämta in data om processen från sidan och kalla på addProcess() i ProcessController  
985     String processCategory = category_field.getText();  
986     int projectedDuration = 0;  
987     int budgetedTime = 0;  
988     int competenceID = 0;  
989     for (int x = 0; x < specTasks_table.getRowCount(); x++){  
990         if (specTasks_table.getModel().getValueAt(x, 1) != null){  
991             budgetedTime = dbm.tryParse(specTasks_table.getModel().getValueAt(x, 1).toString());  
992             projectedDuration = projectedDuration + budgetedTime;  
993         }  
994     }  
995     String comments = comments_pane.getText();  
996     try {  
997         pc.addProcess(processCategory, projectedDuration, comments);  
998     } catch (SQLException ex) {  
999         System.out.println("Fel i databashanteringen vid addProcess(): " + ex.getMessage());  
1000     }  
1001     //hämta in data från tabellen och kalla på addTask() i TaskController  
1002     String taskName;  
1003     for (int x = 0; x < specTasks_table.getRowCount(); x++){  
1004         if (specTasks_table.getModel().getValueAt(x, 0) != null){  
1005             taskName = specTasks_table.getModel().getValueAt(x, 0).toString();  
1006             budgetedTime = dbm.tryParse(specTasks_table.getModel().getValueAt(x, 1).toString());  
1007             competenceID = dbm.tryParse(specTasks_table.getModel().getValueAt(x, 2).toString());  
1008             try {  
1009                 tc.addTask(taskName, budgetedTime, competenceID);  
1010             } catch (SQLException ex) {  
1011                 System.out.println("Fel i databashanteringen vid addTask(): " + ex.getMessage());  
1012             }  
1013         }  
1014     }  
1015 }
```

Figur 17

Om processledaren navigerar från startmenyn till "View my Processes" körs nedanstående kod.

The image is a screenshot of an IDE's source code editor. It shows a Java method named `viewProcess_jbActionPerformed` which is triggered by an `ActionEvent`. The code starts by showing a `CardLayout` panel named "viewProcess". It then initializes a `List<ProcessModel>` and calls `getDbManager()`, `initProcessController()`, and `initTaskController()`. A `pLeadId` is retrieved from `dbm`. A `try` block attempts to get processes from `pc` using `pLeadId`. If an `SQLException` occurs, it prints an error message. The code then iterates over the `pLeadProcesses` list, filling a `data` array with process details: `processId`, `projectedDuration`, `elapsedTime` (retrieved from a database query), and `status`. After the loop, a `DefaultTableModel` is created from the `data` array and the `processes_table` is updated with this model and grid lines are shown.

```
922 private void viewProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {
923     CardLayout card = (CardLayout) mainPanel.getLayout();
924     card.show(mainPanel, "viewProcess");
925     //Följande kod fyller upp tabellen som visas på sidan View my Processes
926     List<ProcessModel> pLeadProcesses = new ArrayList<>();
927     initDbManager();
928     initProcessController();
929     initTaskController();
930     int pLeadId = dbm.getpLeadId();
931     try {
932         pLeadProcesses = pc.getProcesses(pLeadId);
933     } catch (SQLException ex) {
934         System.out.println("Fel vid hämtning av processledarens alla processer: " + ex.getMessage());
935     }
936     int row = 0;
937     int rows = pLeadProcesses.size();
938     Object[][] data = new Object[rows][4];
939     for (ProcessModel pm : pLeadProcesses){
940         data[row][0] = pm.getProcessId();
941         data[row][1] = pm.getProjectedDuration();
942         //Hämtar in elapsedtime från de tasks som tillhör processen: "SELECT SUM(duration) FROM Task WHERE processID = ?";
943         try {
944             data[row][2] = tc.getElapsedTime(pm.getProcessId());
945         } catch (SQLException ex) {
946             System.out.println("Fel vid hämtning av elapsedTime: " + ex.getMessage());
947         }
948         data[row][3] = pm.getStatus();
949         row++;
950     }
951     //Tabellen skapas
952     String[] cols = {"Process number", "Budgeted time", "Elapsed time", "Status"};
953     DefaultTableModel processes = new DefaultTableModel(data, cols);
954     processes_table.setModel(processes);
955     processes_table.setShowGrid(true);
956 }
```

Figur 18

Detta fyller upp data i den tabellen som kan ses i figur 19.

Processes:

Process number	Budgeted time	Elapsed time	Status
37	12	0	Open
38	22	0	Open
48	8	0	Open
49	15	0	Open

Corresponding tasks:

Task ID	Name	Responsibi...	Cost/h	Elapsed time	Status

Figur 19

På denna sida har processledaren flera alternativ.

Select process: processledaren klickar först på en rad i tabellen och sedan på knappen. Då fylls tabellen "Corresponding tasks" upp med viktiga värden för en enkel överblick över information om de arbetsuppgifter som tillhör processen. Nedanstående kod körs:

```
1017 private void selectProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {
1018     //Metoden fyller upp tabellen "corresponding tasks"
1019     initTaskController();
1020     List<TaskModel> tasksInProgress = new ArrayList<>();
1021     //Hämtar in de tasks som tillhör en process
1022     int row = processes_table.getSelectedRow();
1023     int processID = parseInt(processes_table.getModel().getValueAt(row, 0).toString());
1024     try {
1025         tasksInProgress = tc.getTasks(processID);
1026     } catch (SQLException ex) {
1027         System.out.println("Fel vid inhämtning av de tasks som tillhör en process: " + ex.getMessage());
1028     }
1029     //Fyller upp Object-arrayen med viktiga värden från arbetsuppgifterna
1030     row = 0;
1031     int rows = tasksInProgress.size();
1032     Object[][] data = new Object[rows][6];
1033     for (TaskModel tm : tasksInProgress){
1034         data[row][0] = tm.getTaskID();
1035         data[row][1] = tm.getTaskName();
1036         data[row][2] = tm.getSuppID();
1037         data[row][3] = tm.getCostPerHour();
1038         data[row][4] = tm.getDuration();
1039         data[row][5] = tm.getStatus();
1040         row++;
1041     }
1042     //Tabellen skapas
1043     String[] cols = {"TaskID", "Name", "Responsibility", "Cost/h", "Elapsed time", "Status"};
1044     DefaultTableModel tasks = new DefaultTableModel(data, cols);
1045     tasks_table.setModel(tasks);
1046     tasks_table.setShowGrid(true);
1047 }
```

Figur 20

Detta leder till att sidan ser ut som i figur 21.

Processes:

Process number	Budgeted time	Elapsed time	Status	
37	12	0	Open	▲
38	22	0	Open	≡
48	8	0	Open	
49	15	0	Open	▼

Select process

Update selected process

Delete process

Corresponding tasks:

TaskID	Name	Responsibi...	Cost/h	Elapsed time	Status
54	Unable to l...	0	0	0	Pending
55	Program fr...	0	0	0	Pending

Update selected task

Delete task

Back to main menu

Figur 21

Update selected process: Sidan som kan ses i figur 23 fås. Härifrån kan processledaren uppdatera en process värden. Nedanstående kod körs när knappen trycks.

```
1049 private void updateProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {
1050     //Gör så att updateProcess visas när knappen trycks
1051     CardLayout card = (CardLayout) mainPanel.getLayout();
1052     card.show(mainPanel, "updateProcess");
1053
1054     initProcessController();
1055     int row = processes_table.getSelectedRow();
1056     int processID = parseInt(processes_table.getModel().getValueAt(row, 0).toString());
1057     processNum_field.setText(Integer.toString(processID));
1058     //Fyller upp alla fält på nästkommande sida
1059     try {
1060         ProcessModel pm = pc.getProcess(processID);
1061         up_pLeadID_field.setText(Integer.toString(pm.getpLeadId()));
1062         up_startDate_field.setText(pm.getStartDate());
1063         up_processCategory_field.setText(pm.getProcessCategory());
1064         up_projectedDuration_field.setText(Integer.toString(pm.getProjectedDuration()));
1065         up_comments_pane.setText(pm.getComments());
1066         up_status_field.setText(pm.getStatus());
1067     } catch (SQLException ex) {
1068         System.out.println("Fel vid hämtning av processID: " + ex.getMessage());
1069     }
1070 }
```

Figur 22

Process ID:	<input type="text" value="49"/>
Process Leader ID:	<input type="text" value="3"/>
Start Date:	<input type="text" value="2018-05-18"/>
Process Category:	<input type="text" value="Microsoft Excel"/>
Projected Duration:	<input type="text" value="15"/>
Status:	<input type="text" value="Open"/>
Comments:	<div>This is some example comments</div>
	<input type="button" value="Update"/> <input type="button" value="Back"/>

Figur 23

När önskade värden är ifyllda trycks "Update". Då exekveras nedanstående kod.

```

1131 private void updatePr_jbActionPerformed(java.awt.event.ActionEvent evt) {
1132     initProcessController();
1133     //Hämtar in data från fälten på sidan
1134     ProcessModel pm = new ProcessModel(parseInt(processNum_field.getText()),
1135     parseInt(up_pLeadID_field.getText()), up_startDate_field.getText(),
1136     up_processCategory_field.getText(), parseInt(up_projectedDuration_field.getText()),
1137     up_comments_pane.getText(), up_status_field.getText());
1138     //Kallar på uppdateringsmetoden i ProcessController
1139     try {
1140         pc.updateProcess(pm);
1141     } catch (SQLException ex) {
1142         System.out.println("Fel i uppdateringen" + ex.getMessage());
1143     }
1144 }
1145

```

Figur 24

Delete process: Tar bort en process. Nedanstående kod körs.

```
1146 private void deleteProcess_jbActionPerformed(java.awt.event.ActionEvent evt) {  
1147     initProcessController();  
1148     //Hämtar in den markerade radens processID  
1149     int row = processes_table.getSelectedRow();  
1150     int processID = parseInt(processes_table.getModel().getValueAt(row, 0).toString());  
1151     //Kallar på deleteProcess i ProcessController  
1152     try {  
1153         pc.deleteProcess(processID);  
1154     } catch (SQLException ex) {  
1155         System.out.println("Fel vid borttagning av process: " + ex.getMessage());  
1156     }  
1157 }
```

Figur 25

Update selected task: Både den kod som körs när knappen trycks och efterföljande sida är uppbyggd på samma sätt som Update selected Process. Sidan kan ses i figur 26.

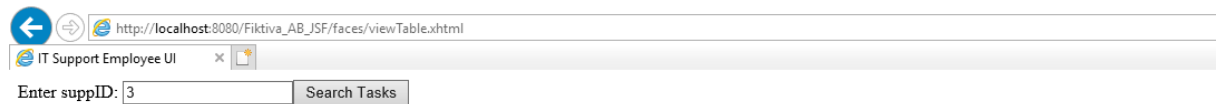
Task ID:	<input type="text" value="55"/>
Process ID:	<input type="text" value="49"/>
Support Employee ID:	<input type="text" value="0"/>
Competence ID:	<input type="text" value="1"/>
Task Name:	<input type="text" value="Program freezing"/>
Start Date:	<input type="text"/>
End Date:	<input type="text"/>
Budgeted Time:	<input type="text" value="5"/>
Duration:	<input type="text" value="0"/>
Cost Per Hour:	<input type="text" value="0"/>
Status:	<input type="text" value="Pending"/>
Comments:	<div><div></div></div>

Figur 26

Delete task: Även denna knapp fungerar på motsvarande sätt som "Delete process".

Supportpersonalens View

På förstasidan för supportpersonalen finns en sökfunktion där supportpersonalen skriver in sitt suppID. Detta kan ses i figuren nedan.



The screenshot shows a web browser window with the URL `http://localhost:8080/Fiktiva_AB_JSF/faces/viewTable.xhtml`. The page title is "IT Support Employee UI". Below the title, there is a text input field labeled "Enter suppID:" with the value "3" entered. To the right of the input field is a button labeled "Search Tasks".

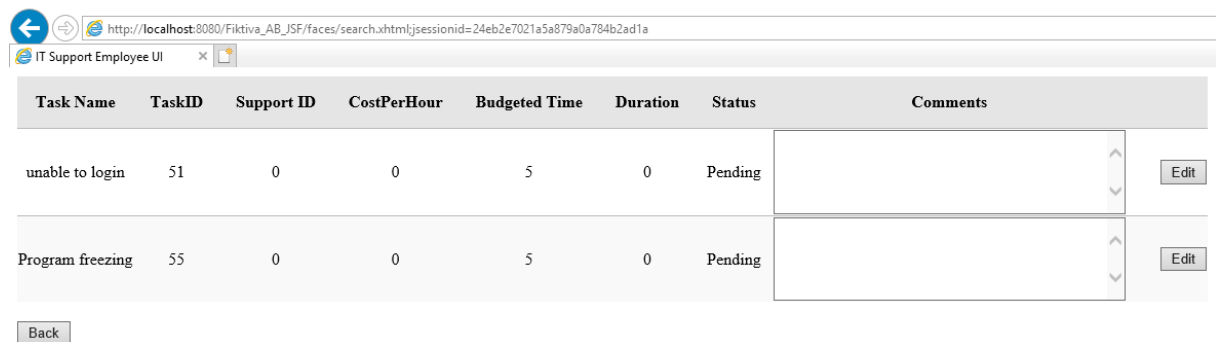
Figur 27

När "Search Tasks" trycks görs en sökning i databasen enligt koden nedan.

```
private final String TASK_GETTASKSONCOMPETENCE = "SELECT Task.* FROM Task "
+ "JOIN Competence ON Task.competenceID = Competence.competenceID "
+ "JOIN CompetenceRow ON Competence.competenceID = CompetenceRow.competenceID "
+ "WHERE CompetenceRow.suppID = ? AND Task.status <> 'Finished'";
```

Figur 28

Detta resulterar i de tasks som den angivna supportpersonalen har rätt kompetens för att kunna utföra och som inte redan är färdiga. Detta visas på nästkommande sida "viewTable" som ses i figuren nedan. Sidan använder sig av ett Stylesheet för att få önskat utseende på dataTabeln.



The screenshot shows the search results page with the URL `http://localhost:8080/Fiktiva_AB_JSF/faces/search.xhtml?sessionId=24eb2e7021a5a879a0a784b2ad1a`. The page title is "IT Support Employee UI". Below the title, there is a table with the following columns: Task Name, TaskID, Support ID, CostPerHour, Budgeted Time, Duration, Status, and Comments. The table contains two rows of data:

Task Name	TaskID	Support ID	CostPerHour	Budgeted Time	Duration	Status	Comments
unable to login	51	0	0	5	0	Pending	
Program freezing	55	0	0	5	0	Pending	

Below the table, there is a "Back" button. To the right of each row in the table, there is an "Edit" button.

Figur 29

Koden för den dataTable som skrivs ut på sidan kan ses i figuren nedan.

```
11 <h:body>
12 <h:form rendered="#{not empty bean.list}">
13 <h:dataTable value="#{bean.list}" var="item"
14 styleClass="order-table"
15 headerClass="order-table-header"
16 rowClasses="order-table-odd-row,order-table-even-row">
17 <h:column><f:facet name="header">Task Name</f:facet>#{item.taskName}</h:column>
18 <h:column><f:facet name="header">TaskID</f:facet>#{item.taskID}</h:column>
19 <h:column><f:facet name="header">Support ID</f:facet>#{item.suppID}</h:column>
20 <h:column><f:facet name="header">CostPerHour</f:facet>#{item.costPerHour}</h:column>
21 <h:column><f:facet name="header">Budgeted Time</f:facet>#{item.budgetedTime}</h:column>
22 <h:column><f:facet name="header">Duration</f:facet>#{item.duration}</h:column>
23 <h:column><f:facet name="header">Status</f:facet>#{item.status}</h:column>
24 <h:column><f:facet name="header">Comments</f:facet>
25 <h:inputTextarea rows="5" cols="40" value="#{item.comments}" readOnly="true"></h:inputTextarea>
26 </h:column>
27 <h:column><h:outputText value="#{item.duration}" /></h:column>
28 <h:column><h:commandButton value="Edit" action="#{bean.edit(item)}" /></h:column>
29 </h:dataTable>
30 </h:form>
```

Figur 30

Trycker supportpersonalen på "Edit" dyker följande alternativ upp.

The screenshot shows a web browser window with the URL `http://localhost:8080/Fiktiva_AB_JSf/faces/viewTable.xhtml`. The page title is "IT Support Employee UI". Below the browser window is a table with the following columns: Task Name, TaskID, Support ID, CostPerHour, Budgeted Time, Duration, Status, and Comments. The table contains two rows: "unable to login" with TaskID 51 and "Program freezing" with TaskID 55. Both tasks have a Status of "Pending". To the right of each row is an "Edit" button. Below the table, there is a section titled "Edit task with taskID: 55". This section contains a form with the following fields: "Support ID:" with a text input containing "0", "CostPerHour:" with a text input containing "0", "Duration:" with a text input containing "0", "Status:" with a dropdown menu showing "Pending", and "Comments:" with a text area. Below the form are "Save" and "Back" buttons.

Task Name	TaskID	Support ID	CostPerHour	Budgeted Time	Duration	Status	Comments
unable to login	51	0	0	5	0	Pending	
Program freezing	55	0	0	5	0	Pending	

Edit task with taskID: 55

Support ID:

CostPerHour:

Duration:

Status:

Comments:

Figur 31

Härifrån kan supportpersonalen ändra på information om en arbetsuppgift enligt kravspecifikationen. Kod för den panelGroup som dyker upp kan ses i figuren nedan.

```
31 <h:panelGroup rendered="#{bean.edit}">
32   <h3>Edit task with taskID: #{bean.tm.taskID}</h3>
33   <h:form>
34     <p>Support ID: <h:inputText value="#{bean.tm.suppID}" /></p>
35     <p>CostPerHour: <h:inputText value="#{bean.tm.costPerHour}" /></p>
36     <p>Duration: <h:inputText value="#{bean.tm.duration}" /></p>
37     <p>Status: <h:selectOneMenu value = "#{bean.tm.status}">
38       <f:selectItem itemValue = "Pending" itemLabel="Pending"></f:selectItem>
39       <f:selectItem itemValue = "Started" itemLabel="Started"></f:selectItem>
40       <f:selectItem itemValue = "Finished" itemLabel="Finished"></f:selectItem>
41     </h:selectOneMenu></p>
42     <p>Comments: <h:inputTextarea rows="5" cols="40" value="#{bean.tm.comments}" /></p>
43     <p><h:commandButton value="Save" action="#{bean.save}" /></p>
44   </h:form>
45 </h:panelGroup>
```

Figur 32

Följande figur visar på när supportpersonalen fyller i värden. Detta resulterar i utseendet som kan ses i figur 33.

Task Name	TaskID	Support ID	CostPerHour	Budgeted Time	Duration	Status	Comments
unable to login	51	0	0	5	0	Pending	
Program freezing	55	0	0	5	0	Pending	

Edit task with taskID: 55

Support ID:

CostPerHour:

Duration:

Status:

Comments:

Figur 33

När Save sedan trycks fås utseendet av sidan som kan ses i figuren nedan.

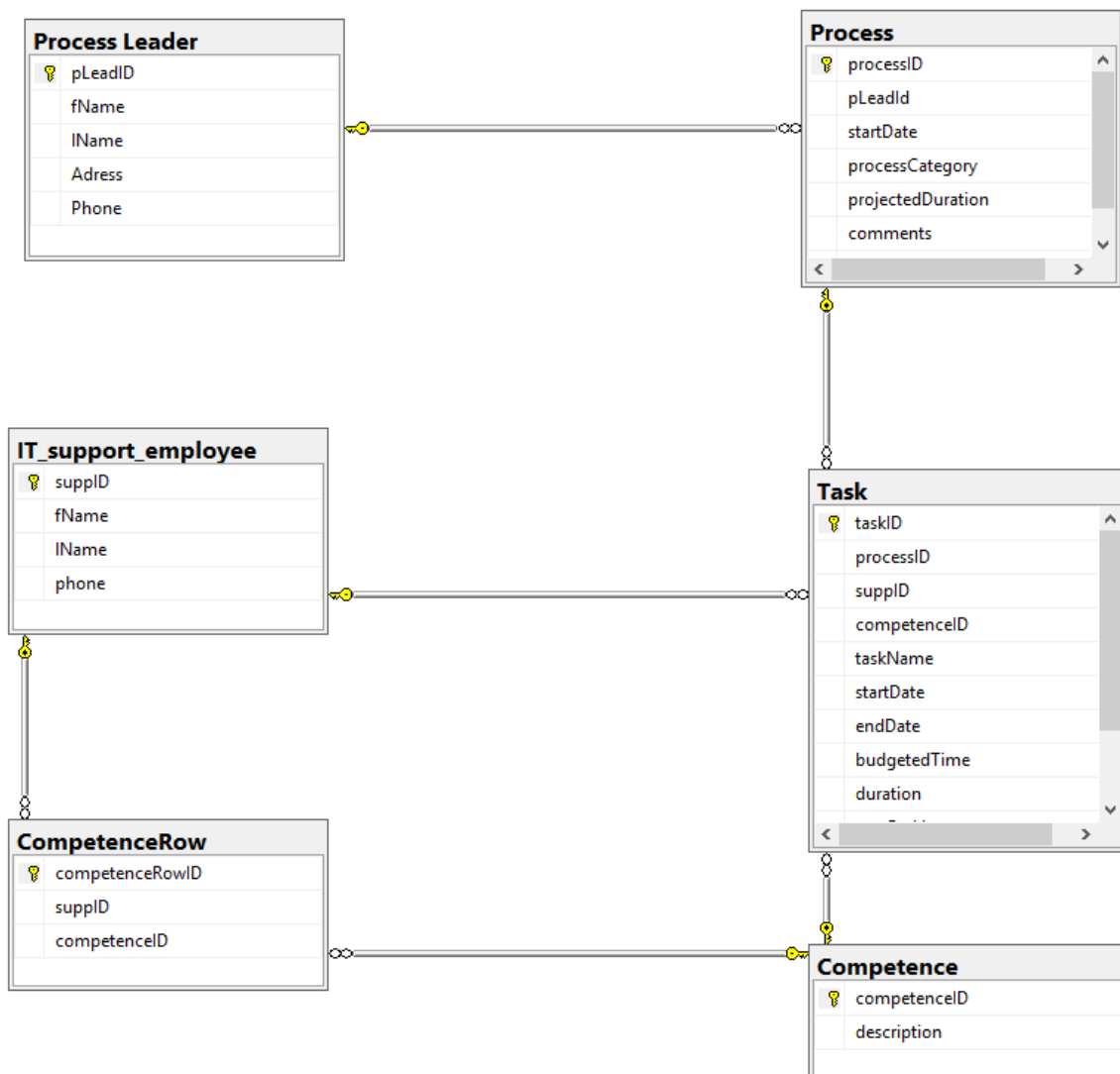
Task Name	TaskID	Support ID	CostPerHour	Budgeted Time	Duration	Status	Comments
unable to login	51	0	0	5	0	Pending	
Program freezing	55	3	15	5	10	Finished	This task is finished

Figur 34

Databas

Logisk modell

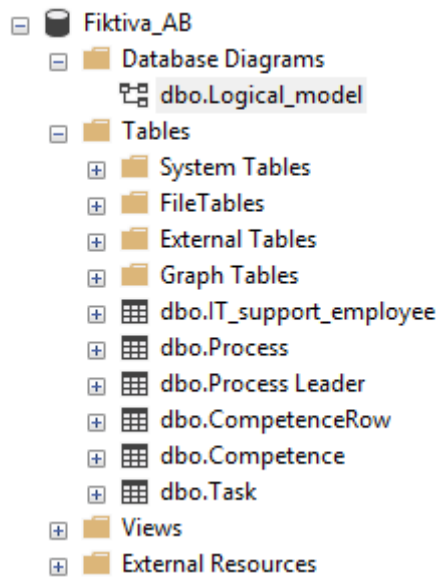
I figur 35 visas den logiska modellen för den databas som ligger till grund för informationen som hanteras av de bägge klienterna. Den skiljer sig en del från den logiska modell som gjordes i deluppgift A. De största skillnaderna är att de tidigare "Support Role"- och "Role Row"-tabellerna har bytt namn till de bättre lämpade namnen "Competence" och "CompetenceRow". Competence är kopplad till en task-tabellen då varje task kräver en specifik kompetens. Då de support-anställda har olika kompetenser möjliggör det för att söka fram tasks som kräver kompetens som den anställda besitter. En annan skillnad från designuppgiften är att "TaskRow" har tagits bort. Detta då en task endast utförs av en anställd. I övrigt motsvarar modellen den tidigare gjorda designen.



Figur 35

Tabeller

Nedan visas strukturen för databasen och ett antal viktiga tabeller med korta förklaringar.



Figur 36

Task

Eftersom en task skapas av processledaren utan att en specifik support-personal tillsätts den behöver ett antal kolumner tillåta nullvärden. Med andra ord är de kolumner som tillåter null samma som de kolumner som en supportanställd har möjlighet att redigera efter att tasken har skapats.

	Column Name	Data Type	Allow Nulls
🔑	taskID	int	<input type="checkbox"/>
	processID	int	<input type="checkbox"/>
	supplID	int	<input checked="" type="checkbox"/>
	competenceID	int	<input type="checkbox"/>
	taskName	varchar(30)	<input type="checkbox"/>
	startDate	date	<input checked="" type="checkbox"/>
	endDate	date	<input checked="" type="checkbox"/>
	budgetedTime	int	<input type="checkbox"/>
	duration	int	<input checked="" type="checkbox"/>
	costPerHour	int	<input checked="" type="checkbox"/>
	status	varchar(15)	<input type="checkbox"/>
	comments	varchar(MAX)	<input checked="" type="checkbox"/>
▶			<input type="checkbox"/>

Figur 37

CompetenceRow

Denna tabell möjliggör för en supportanställd att ha flera kompetenser.

	Column Name	Data Type	Allow Nulls
🔑	competenceRowID	int	<input type="checkbox"/>
	supplD	int	<input type="checkbox"/>
	competenceID	int	<input type="checkbox"/>
			<input type="checkbox"/>

Figur 38

Process

Denna tabell håller all data som rör en process.

	Column Name	Data Type	Allow Nulls
🔑	processID	int	<input type="checkbox"/>
	pLeadId	int	<input type="checkbox"/>
	startDate	date	<input type="checkbox"/>
	processCategory	varchar(15)	<input type="checkbox"/>
	projectedDuration	int	<input type="checkbox"/>
	comments	varchar(MAX)	<input checked="" type="checkbox"/>
	status	varchar(10)	<input type="checkbox"/>
			<input type="checkbox"/>

Figur 39

Diskussion

Uppgiften var mycket utmanande för mig då jag saknade kunskap inom flera viktiga områden. Jag hade aldrig innan gjort varken HTML, JSF, JFrame eller kodat mot en databas. Dessutom med begränsad erfarenhet av MVC och skapande av klassdiagram blev det en komplicerad uppgift. Allt som allt gjorde detta att jag utgick förhållandevis minimalt utefter mitt uppsatta klassdiagram utan fokuserade mest på att koda på ett korrekt sätt enligt MVC. Därför upplever jag att skapandet av klassdiagrammet mest gav en djupare förståelse för uppgiften utan att spela en speciellt stor roll vid själva implementeringen. Alla dessa nya delar gjorde att jag fick lägga väldigt mycket tid på varje nytt moment för att få det att fungera. Detta gjorde att viss funktionalitet och utseende blev lidande.

De fyra saker som jag hade fokuserat på om jag hade haft mer tid eller gjort uppgiften på nytt är en utökad felhantering, att integervärden som är null i databasen inte skrivs ut som 0 och att få snyggare och smartare användargränssnitt, via exempelvis dialogrutor. Till sist hade jag ändrat så att min "DbManger" endast består av saker som hanterar databaskopplingen och inga övriga metoder. Trots detta är jag mycket nöjd med att ha lyckats skapa två användargränssnitt som enligt mig uppfyller den givna kravspecifikationen.