

Project request handling system

Backendutveckling och systemintegration

Inledning

Om

Ett system för att hantera ärenden åt ett lägenhetskomplex. Programmet tillåter lägenhetsinnehavare att rapportera in fel och tekniker att lösa inkommande ärenden. Programmet är delat i två där den ena används för användaren och den andra för tekniker. Tekniker ska kunna ta emot en felanmälan, svara på den och sedan ska felanmälan kunna stängas.

Inledning/Syfte

Vår uppgift var att leverera ett projekt som visar på kunskap om att tillsammans kunna utveckla ett program som jobbar mot en databas (SQL) som backend. Programmet kunde vara textbaserat (CLI), grafiskt (Swing eller JavaFX) eller webbaserat (EE, Spring). Programmet behövde inte vara " snyggt", uppgiften handlade om att kunna använda SQL som backend. Programmet skulle skrivas i Java mot en databas (SQL) som skulle bestå av minst två tabeller som används av programmet. Källkoden skulle versionshanteras med git och projektet dokumenteras i rapportform och redovisas på plats i Piteå. Vi valde att bygga en app för ärendehantering i ett lägenhetskomplex.

Definitioner och förkortningar

Förklaringar till några av de begrepp och förkortningar som förekommer i rapporten ges i Tabell 1.

Tabell 1 Definitioner och förkortningar

Definition/Förkortning	Förklaring
AppearIn	Ett program för videomöten
API	Application Programming Interface (applikationsprogrammeringsgränssnitt)
fxml (JavaFx)	Layouten för en sida
Git	Verktyg för koddelning och versionshistorik

GUI	Graphical User Interface (grafiskt användargränssnitt)
nod (JavaFx)	Ett element på en sida
scene (JavaFX)	En sida
Slack	Searchable Log of All Conversation and Knowledge, ett verktyg för gruppkommunikation och samarbete
SQL	Structured Query Language
TDD	Test Driven Development (testdriven utveckling)
Trello	Verktyg för projektindelning
UTC	Coordinated Universal Time

Referenser

Länk till git-repository:

<https://github.com/gospelmalin/RequestHandlingApartments>

Länk till Trello-tavla:

<https://trello.com/invite/b/XjUVSqeX/1f2daa0616774d5d827e80f1a3948851/jtmm-project-databases>

JavaFX-dokumentation:

<https://www.oracle.com/technetwork/java/javase/documentation/javafx-docs-2159875.html>

Genomförande

Övergripande mål

Det övergripande målet var att skapa ett program för att hantera ärenden åt ett lägenhetskomplex och att samtidigt visa att vi kan utveckla ett program som jobbar mot en databas.

Avgränsningar

Projektet växte sig snabbt större än ursprungstanken var och då tiden var begränsad valde vi att inte implementera alla delar. Rollhanteringen är en sådan del. Requesthanteringen är inte heller implementerad fullt ut, men täcker det mest grundläggande och visar att vi kan hantera kommunikationen mellan app och databas. Vi har inte arbetat TDD-baserat, och testnings-nivån är inte alls så god som man skulle önska. Exceptionshanteringen och validering av indata skulle kunna förbättras.

Arbetsmetodik

Vi använde ett repository på github, <https://github.com/gospelmalin/RequestHandlingApartments>, där vi alla hade skrivrättigheter. Vi förde det mesta av diskussionerna via Slack, men provade även Appairn på slutet. Trello fick fungera som visualiseringsverktyg över projektets planering och status men i praktiken fick vi aldrig till något vettigt arbetsflöde där.

Initialt började vi med planering och databasdesign som vi lade ganska mycket krut på, innan vi gav oss på lämpliga delar av koden. Vi försökte i första hand att arbeta med olika delar, men på slutet gick det ihop en del. Vi hade ganska mycket tekniska problem (med ide och versionshanteringssystemet) vilket avspeglas i hur arbetsbelastningen blev i praktiken. Vi arbetade i varsin branch eller flera olika och mergade flitigt in de senaste ändringarna.

Ärendehanteringsappen

Översikt

I appen kan lägenhetsinnehavare rapportera in fel på sina lägenheter, och följa status på ärenden, och tekniker administrera såväl felrapporterna som uppgifter om områden, hus, lägenheter och boende.

Användarinterface

Användarinterfacet är byggt i JavaFx och inleds med möjligheten att välja om man vill rapportera ett nytt fel, titta på befintliga felrapporter eller - för tekniker - utföra andra administrativa uppgifter, se figur Programmetts startsida.

Programmets startsida

The screenshot shows the main interface of the 'Request Handling System for Apartments - an App by JTMM'. The window has a title bar with the application name and standard Windows window controls. Below the title bar is a menu bar with 'File', 'Edit', and 'Help'. The main content area is divided into two columns. The left column is titled 'What do you want to do?' and contains two buttons: 'Add new request' and 'View request(s)'. The right column is titled 'Admins only' and contains one button: 'Administrative tasks'. At the bottom of the window is a 'Message console' area, which is a large rectangular box with a vertical line indicating where messages would appear.

Formuläret för nytt ärende

The screenshot shows the 'Request form' interface of the 'Request Handling System for Apartments - an App by JTMM'. The window has a title bar with the application name and standard Windows window controls. Below the title bar is a menu bar with 'File', 'Edit', and 'Help'. The main content area is titled 'Request form'. It contains four input fields: 'First name' (with the value 'Miriam'), 'Last name' (with the value 'Shell'), 'Apartment' (a dropdown menu with the value '5'), and 'House' (a dropdown menu with the value '3510'). Below these fields is a 'Problem description' text area with the value 'Strange sounds from freezer'. To the right of the text area is a 'Submit' button. At the bottom of the window is a 'Message console' area, which is a large rectangular box with a vertical line indicating where messages would appear. A 'Home' button is located at the bottom left of the window.

Vyn för att titta på ärenden i databasen

Request Handling System for Apartments - an App by JTMM

Request information

Apartment: Request id: Resolvers only

Status: OR

Number of results to show:

Request id	Reported by	Request date	Address	House No	Apartment	District	Description	Status	Completion date
20	Rico Medina	2018-11-11	Duck Cree...	10A	25	Duck Creek	New problems in this house	Not started	
21	Robert Wright	2018-11-11	Duck Cree...	10A	35	Duck Creek	Window won't open	Not started	
22	Robert Wright	2018-11-11	Duck Cree...	10A	35	Duck Creek	more and more and more problems	Not started	
23	Robert Wright	2018-11-11	Duck Cree...	10A	35	Duck Creek	ANd more problems	Not started	
24	Miriam Shell	2018-11-12	Greater Fo...	3510	5	The Forest	No water	Not started	
25	Rico Medina	2018-11-12	Duck Cree...	10A	25	Duck Creek	Noise from the walls	Not started	
26	Miriam Shell	2018-11-12	Greater Fo...	3510	5	The Forest	Tired of strange sounds during night	Not started	
27	Miriam Shell	2018-11-19	Greater Fo...	3510	5	The Forest	Strange sounds from freezer	Not started	

Result console

Administratörsvyn

Request Handling System for Apartments - an App by JTMM

Admin form

Handle requests

Select request:

Other administrative tasks

Message console

Ändra ett ärende

Request Handling System for Apartments - an App by JTMM

Edit request

To edit, select a request in the table

Selected Request: Status: Update status: Allocated to/Resolver name: Update resolver: Update request:

Request id	Reported by	Request date	Address	House No	Apartment	District	Description	Status	Completion date	Allocated to
6	Rico Medina	2018-11-09	Duck Cree...	10A	25	Duck Creek	Something wr...	Completed	2018-11-21	
7	Rico Medina	2018-11-10	Duck Cree...	10A	25	Duck Creek	Something wr...	Completed	2018-11-21	
8	Rico Medina	2018-11-10	Duck Cree...	10A	25	Duck Creek	Something wr...	In progress		
9	Miriam Shell	2018-11-10	Greater Fo...	3510	5	The Forest	This is a real er...	Not started		
10	Miriam Shell	2018-11-10	Greater Fo...	3510	5	The Forest	This is a real er...	In progress		
11	Rico Medina	2018-11-11	Duck Cree...	10A	25	Duck Creek	This is too bad	Not started		
12	Miriam Shell	2018-11-11	Greater Fo...	3510	5	The Forest	This is the first...	Not started		

Result console

Redigera lägenheter

Request Handling System for Apartments - an App by JTMM

Edit apartments

Apartment number:

House:

id	Apartment number	House number
1	15	10A
2	25	10A
3	35	10A
4	45	10A
5	4	3510
6	5	3510
7	6	3510
8	7	3510
9	31	2045
10	32	2045
11	33	2045
12	34	2045

Redigera district

Om vi tar District som exempel, så kan man lägga till ett nytt område genom att skriva in ett namn i District name-fältet och klicka på Create new district. Då ser `handleMouseClicked`-metoden i `DistrictController` till att skapa ett district-object och anropa `add(district)`-metoden i `DistrictRepository`, som i sin tur hämtar en `databasconnection` och skickar in ett prepared statement med insert-queryn i `executeUpdate`-metoden varvid databastabellen `district` uppdateras med en ny rad. `handleMouseClicked`-metoden ser också till att anropa metoden `updateTable()` som via `DistrictRepository` skickar en `select` till databasen så att tabellen i användarinterfacet uppdateras och visar den nya posten.

Man kan också editera en befintlig post i databasen genom att markera en rad i tabellen, i det här exemplet Greater Valley. Då visas namnet i fältet ovanför, och när man ändrat det väljer man Update varvid handleUpdateOnMouseClicked-metoden i kontrollern hämtar det inmatade värdet och skickar in det i update(district) via repositoryet. En connection till databasen upprättas, raden uppdateras i databasen och på samma sätt som när man gjorde create new district uppdateras tabellen i användarinterfacet då kontrollern anropar updateTable().

Delete fungerar på motsvarande sätt, bara att det är andra metoder som är involverade.


district_id	district_name
1	Duck Creek
2	The Forest
3	Hampton Meadows
4	Summertown
5	Greater Valley
6	Happy Homes District














































Projektstruktur

Vi har använt ett model-view-controller-repository-pattern för vårt projekt. Model speglar de olika tabellerna i databasen, och tar även hänsyn till relevanta kopplingar till andra tabeller.

Exempelvis ingår i request-modellen namn för requestern och inte bara requester-id. View är själva GUI:t och är kopplad till kontrollern. Kontrollern för respektive view håller våra views åtskilda från såväl model som repository och utgör kopplingen dem emellan.

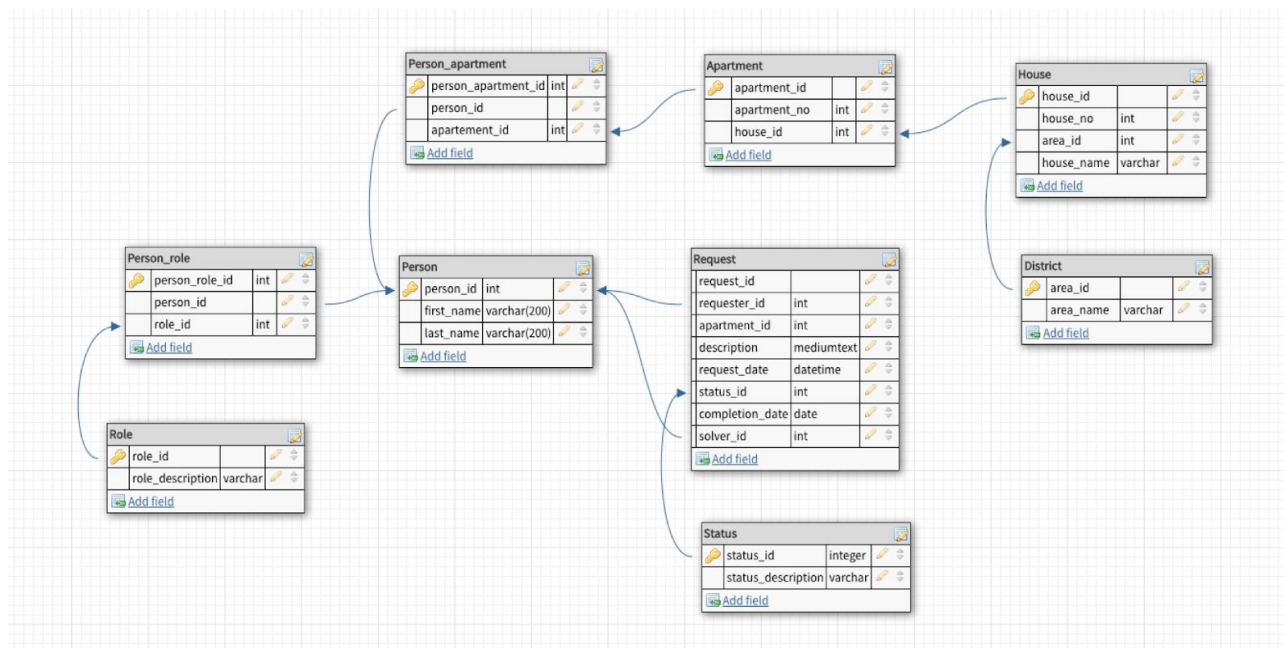
Klasserna i repositories hanterar själva databasfrågorna. Dessa klasser bygger på ett gemensamt interface. Vi har också ett paket för "utilities" där klassen Database sköter själva kommunikationen med databasen, och där Util-klassen håller metoder för datumomvandling.

▼  > RequestHandlingApartments [RequestHandlingApartments New]lokkeMalinMerge]

- ▼  > src
 - ▼  > (default package)
 - ▼  Main.java
 - >  Main
 - ▼  controller
 - >  AddRequestController.java
 - >  AdminViewController.java
 - >  DistrictController.java
 - >  EditApartmentController.java
 - >  EditRequestController.java
 - >  RequestViewController.java
 - >  RootLayoutController.java
 - >  StartViewController.java
 - >  ViewController.java
 - ▼  model
 - >  Apartment.java
 - >  District.java
 - >  House.java
 - >  Person.java
 - >  Request.java
 - >  Role.java
 - >  Status.java
 - ▼  repository
 - >  ApartmentRepository.java
 - >  DistrictRepository.java
 - >  HouseRepository.java
 - >  IRepository.java
 - >  PersonRepository.java
 - >  RequestRepository.java
 - >  RoleRepository.java
 - >  StatusRepository.java
 - >  SQL_Files
 - ▼  > util
 - >  > Database.java
 - >  Util.java
 - ▼  > view
 - >  AddEditViewPerson.fxml
 - >  AddRequestLayout.fxml
 - >  AdminFormView.fxml
 - >  DistrictView.fxml
 - >  EditApartmentLayout.fxml
 - >  EditRequestLayout.fxml
 - >  RequestViewLayout.fxml
 - >  RootLayout.fxml
 - >  StartView.fxml

Databas

Databasstrukturen ser ut som följande



Dom flesta av våra förfrågningar utgår från person-tabellen och person_id. Då det är en person i systemet som ska felanmäla sin lägenhet behöver vi kunna knyta person_id till tabellerna **Person_apartment**, **Request** samt **Person_role**. Utöver det har dessa tre tabeller i sin tur egna tabeller som man kopplar samman.

Person_apartment tabellen har tre foreign keys, först till **Apartment** tabellen, där lagrar vi lägenhetsnummer, **Apartment** har i sin tur en foreign key mot **house** där husets nummer blir lagrat och tillslut mot **District** där namnet på området där huset finns ligger.

Request-tabellen har en foreign key mot **status**-tabellen som håller vilken status ärendet har, alltså om det är nytt, jobb pågår eller stängt och avslutat.

Person_role-tabellen har en foreign key mot tabellen role. Role innehåller en text om vad användaren har för roll.

Teknisk lösning

JavaFX

Vi har använt oss av JavaFX som GUI till vårt projekt. För att bygga upp våra scenes i applikationen har vi använt oss av fxml-filer. Fxml-filen är upplagd som en kombination av xml och html. Varje fil innehåller element som representerar noder på sidan, och precis som i html går det att sätta css-style på dessa. I Fxml-dokumentet anger vi även en controller som knyter

filen till en java-klass. Noderna når vi sedan i vår controller-klass genom att använda @FXML-annoteringen följt av nodens unika id som vi satt på elementet i fxml-dokumentet. Alla knapphändelser och datainhämtning till javafx vyn startas också från controller klassen.

Utvecklingsmiljö och utvecklingsverktyg

Vi har använt följande miljöer och verktyg för realisering och byggande.

Eclipse	Programmeringsverktyg (IDE)
JavaFX Scene Builder 2.0	Byggande av fxml filer
MariaDB	Databas
Centos på virtuell Linux-maskin	För att köra databasen
My SQL Workbench	SQL sökningar
mariadb-java-client-2.3.0.jar	Driver för mariaDB

Felhantering

Javas exceptions används för felhantering i applikationen. Inträffar ett fel ska problemet åtgärdas. Om rutinen inte klarar det ska felet kastas vidare. För att dölja implementationsdetaljer utåt och öka underhållbarheten bör de fel som kastas vara egendefinierade i första hand.

Säkerhet

Prepared Statements

Vår tanke var att använda Prepared statements överallt där användaren gör inmatningar, för att skydda oss mot sql-injektioner. Det är implementerat överlag, men saknas fortfarande på vissa ställen. Dock görs databasfrågan med statements istället för prepared statements, på vissa ställen även där prepared statements definitivt vore lämpligt. Det beror på att vi nästan in i det sista funderade över vilken lösning vi skulle ha för databasconnection när det gällde prepared statements, och under tiden ville vi se att frågorna fungerade. Tidsbrist gjorde sedan att vi inte hann ändra överallt.

Inloggning och accessnivåer

I en produktionsversion skulle vi absolut ha implementerat användarinloggning och försäkrat oss om att lägenhetsinnehavarna bara kommer åt de funktioner de ska komma åt (felanmälan och att titta på befintliga ärenden), medan teknikerna även ska komma åt de administrativa delarna. För det här projektet nöjde vi oss med att implementera tabell och fält för roll i databasen, vilka på sikt kan användas för att validera vilken access användaren ska ha. Övriga tabeller och funktioner för användaraccess är inte varken inkluderade eller implementerade.

Slutsatser/Diskussion

Appens implementering

Vi har tydligt visat att vi kan använda SQL som backend från Java. Vårt program har följande funktionalitet implementerad från ett GUI skrivet i JavaFx till databas:

- Lägga till en request i databasen (lägger till data i tabellen request)
- Visa vilka hus som finns i databasen (visar alla rader från tabellen house)
- Visa vilka lägenheter som finns (visar alla rader från tabellen apartment)
- Visa alla requests som finns i databasen (visar alla rader från tabellen request joinad med tabellerna apartment, person, status, district, house)
- Uppdatera status och completion date för utvald request (tabell request)
- Visa alla lägenheter som finns för respektive hus (tabellerna apartment, house)
- Lägga till lägenheter i databasen (tabell apartment)
- Ta bort lägenheter ur databasen (tabell apartment)
- Visa alla områden i databasen (tabell district)
Lägga till nya områden i databasen (tabell district)
Uppdatera befintliga områden i databasen (tabell district)
Ta bort områden från databasen (tabell district)

Sammanfattningsvis visar vi att vi kan hämta/visa upp data från såväl en som flera joinade tabeller, lägga till, uppdatera och ta bort data ur tabeller med hjälp av SQL från ett Java-program

Vi har också visat att vi vet hur man planerar en väl designad databas där dubbellagring undviks.

På grund av tekniska problem och att vi inte fick till ett optimalt arbetsflöde i gruppen hann vi inte implementera riktigt alla delar vi planerat att prioritera fullt ut eller riktigt så genomarbetat som man skulle önska för att gå i produktion. Självklart bör resolver kunna sättas när man editerar en request. Det är dock enkelt att lägga till denna funktionalitet då GUI:t och kontrollern är förberedda för det. Det enda som krävs är att lägga till metoden i repositoriet och att lägga till anropet av den i kontrollerns metod. Lika självklart bör man kunna välja att visa bara en viss request. Dock har man möjligheten att sortera data i tabellerna i GUI:t som man själv önskar genom att klicka på kolumnrubrikerna, vilket kan fungera som workaroud. Kod för att ta bort en request finns klar i repositoriet om vi anser att det är funktionalitet som ska finnas i appen

(alternativt kan man tänka sig att det inte ska vara tillåtet att ta bort requests, då det kan finnas ett värde i att se vad som hänt i en viss lägenhet över tid).

Att tabellerna i GUI:t ska uppdateras när man editerat något är en självklarhet. Den funktionaliteten var inte korrekt implementerad när det gäller Request då vi höll redovisningen (den försvann efter en inte helt lyckad lösning av mergekonflikter) men är nu tillagd igen. Funktionaliteten finns sedan tidigare också för apartment och district.

Vi borde ha haft inloggningsuppgifter, ip-adress etc i en Properties-fil istället för direkt i programmet. Det vore lämpligt både ur arbets- och säkerhetssynpunkt.

Vi har lagt vikt vid att hålla olika delar av koden väl åtskilda, för att få ett mer flexibelt och lättunderhållet system. Det är förhållandevis enkelt att byta databas - vilket vi också praktiskt provade under projektet. Det bör också vara relativt enkelt att byta ut användarinterfacet då all kommunikation med detta sker i controller-klasserna.

Arbetsmetodik

Arbetsmetodikmässigt hade vi vunnit på att försöka ha lite fler online-möten, även om det är en utmaning när man har olika dagar och tider då man kan arbeta med projektet. Regelbundna avstämningar/avrapporteringar hade också varit ett alternativ, gärna utifrån modellen "Vad gjorde du igår - vad ska du göra idag - vad hindrar dig? Nu hördes vi löpande via en egen grupp på Slack, men mer oregelbundet, och det kunde vara svårt att hitta statusuppdateringarna i flödet. Hade vi använt Trello mer aktivt, som tanken ursprungligen var, hade vi nog också fått ett bättre flyt.

Resultat

Vi har väl uppfyllt projektets mål, och dessutom den hårda vägen lärt oss att man aldrig kan vara för tydlig i sin kommunikation.