Matteus Laurent, Johan Levinsson, Oscar Petersson, Erik Peyronsson

MatLabb - Designspecifikation

Högskoleingenjörsutbildning i datateknik, 180 hp

Designspecifikation - 19 oktober 2015
Programmeringsprojekt, HT15
TDDI02, Linköpings universitet

Handledare:
Johan Frimodig
Institutionen för datavetenskap

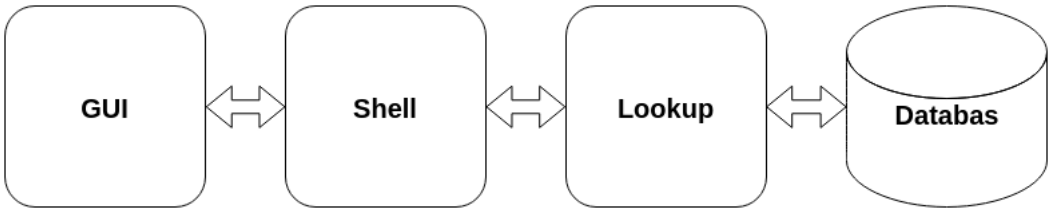
Innehåll

1. Inledning

Alla har vi någon gång stått framför vårat kylskåp och funderat över vad man kan hitta på för middag med det kylskåpsinnehåll vi konfronteras med. Projektet MatLabb är en interaktiv receptdatabas med grafiskt användargränssnitt. Dess syfte är att hjälpa användaren att organisera recept, söka recept baserat på tillgängliga ingredienser, samt att underlätta portions- och enhetsomvandling. Recepten kommer även att innehålla information om exempelvis näringsinnehåll och pris.

MatLabbs målbild presenteras närmare i dokumentet "MatLabb – Kravspecifikation", medan detta dokument närmare presenterar skalet och vad som finns under detsamma.

2. Arkitektur



Figur 2.1: Översikt över första lagrets moduler.

MatLabb har tre centrala delsystem samt en databas vilka initialt kan betraktas på följande sätt:

1. Ett användargränssnitt (**GUI**), baserat på biblioteket QT, som hjälper användaren att på ett intuitivt och välbekant sätt interagerar med receptmodulen, och i förlängningen databasen.
2. **Shell** - ett lager som närmast kan kallas för MatLabbs styrsystem.
3. **Lookup** - ett lager som huvudsakligen sköter kommunikationen mellan **Shell** och databasen.
4. En databas som lagrar all receptrelaterad information.

I detta kapitel ges en överblick över funktionaliteten hos dessa delsystem och hur dessa interagerar.

Databasen utgörs av en MySQL-databas där recept, ingredienser och relaterad data lagras. Exakt data som lagras framgår i kravspecifikationen, samt i kapitel 3 - *Detaljerad teknisk specifikation*.

Shell utgör det primära navet mellan databasen och användargränssnittet (**GUI**). Här behandlas aktuellt recept enligt de instruktioner som mottas via GUI:t. Modulen interagerar direkt med **GUI** och **Lookup** och indirekt med databasen via **Lookup**.

GUI är en abstraktion av det grafiska användargränssnittet och i förlängningen användaren. “Modulen” kommunicerar med **Shell**.

Lookup ingår egentligen i **Shell**, men betraktat som en separat modul sköter den kommunikationen mellan databasen och **Shell**s övriga interna submoduler.

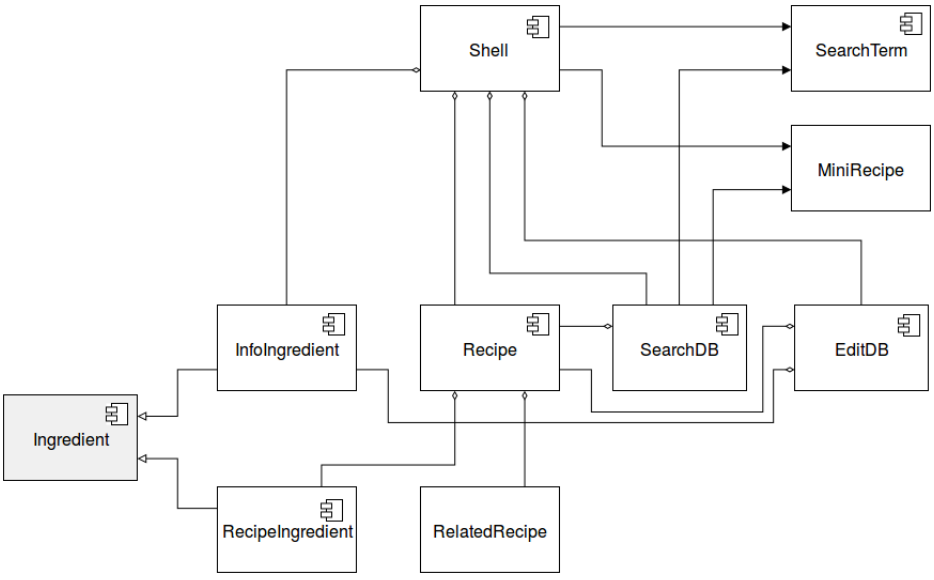
3. Detaljerad teknisk specifikation

Programmet MatLabb har i stort tre delsystem:

1. **GUI**, det grafiska användargränssnittet. Implementeras av klassen `GUI`.
2. **Shell**, eller, det inre skalet som håller centrala objekt och variabler, t.ex. aktivt recept och portionskalning. Motsvarar i stort klassen `Shell`.
3. **Lookup**, som tillhandahåller relevanta verktyg för kommunikation med databasen. Uppbygd av klasserna `SearchDB` och `EditDB`.

I detta kapitel tar vi ett steg ifrån modulbegreppen och detaljerar istället de klasser som bygger upp programmet MatLabb.

`GUI` ansvarar för att skriva ut data från det inre systemet på skärmen i grafisk tappning. Användaren styr även systemet genom att interagera med menyer, knappar och strängfält snarare än att ge skriftliga hänvisningar till kommandoprompten. Information presenteras enligt konceptbilderna i (figur 4.1 - 4.4). Det grafiska gränssnittet implementeras med hjälp av biblioteket Qt och kommer delvis designas i klienten Qt Creator.



Figur 3.1: Översiktsschema över MatLabbs klasser och structs. Se även figur 3.3 (s. 10) för närmare detaljer.

3.1 Shell

`Shell` innehåller objekt av klasserna `Recipe`, `InfoIngredient`, `SearchDB` och `EditDB` som datamedlemmar, där de två förstnämnda reflekterar det aktuella receptet och ingrediensen som vårt program

interagerar med. Utåt tillhandahåller `Shell` publikt endast funktioner som kan tänkas motsvara alla möjliga handlingar från användaren. Denna grupp av funktioner inkluderar, men är ej begränsade till, funktionerna listade i figur 3.2 (s. 4).

Figur 3.2: Funktioner för användarinputs

`addRecipe(string)`	Konstruerar ett nytt tomt `Recipe`-objekt för datamedlemmen `currentRecipe_`. Ett defaultargument av datatypen string existerar för att potentiellt tilldela ett namn.
`editRecipe()`	Kallar på `currentRecipe_.editRecipe()` för att kunna ändra på dess datamedlemmar.
`importTxt(string)`	Importering från textfil. Konstruerar ett nytt `Recipe`-objekt och försöker fylla i dess datamedlemmar enligt en standardmodell.
`exportTxt()`	Kallar på `currentRecipe_.exportTxt(string)` för att exportera till .txt. Kan modifieras för att först hämta ett annat recept från databasen för exportering.
`addIngredient(string)`	Motsvarande `addRecipe`.
`editIngredient(string)`	Motsvarande `addIngredient`.
`matchRecipe(string)`	Levererar en sträng till `Lookup`-objektet för att slå i databasen för exakt matchning.
`matchIngredient(string)`	Motsvarande `matchRecipe`.
`searchRecipe(cont<SearchTerm*>)`	Levererar söktermer i en godtycklig container till `Lookup`. `recipeSearchResults_` tilldelas det resultat som `Lookup` ger.
	get-funktioner som används av GUI:t och returnerar relevant data.

3.2 Recipe

Objekt av klassen `Recipe` kommer endast att existera i stabilt tillstånd som datamedlem i klassen `Shell`. Nya objekt skapas antingen i samband med t.ex. funktionen `addRecipe(string)` eller byggs upp och returneras av `SearchDB` som resultat av en matchning i databasen. Klassen `Recipe` innehåller främst fullständig data om ett specifikt recept, men även funktioner för implementeringen av `Shell`s "användarfunktioner", t.ex. åtkomst och redigering och för att hämta samt beräkna pris och kalorivärden. Lista över datamedlemmar i klassen `Recipe`:

- `string name_` - *Namn på receptet*
- `string description_` - *Beskrivning/utförande*
- `int minutesTime_` - *Tidsåtgång i minuter*
- `cont<string> comments_` - *Kommentarer i godtycklig container*
- `"referens" image_` - *Någon typ av referens för implentering av tillhörande bilder*
- `double grade_` - *Betyg*
- `cont<RecipeIngredient> ingredients_` - *Ingredienser som ingår*

- `cont<RelatedRecipe> relatedRecipes_` - *Besläktade recept*

3.2.1 MiniRecipe

Posten `MiniRecipe` har en begränsad men viktig uppgift; objekt av denna typ representerar en nerbantad version av ett specifikt recept och existerar för att påskynda framförandet av sökresultat. När `Shell` anropar `SearchDB::get_list(...)` förväntar den sig en container innehållande `MiniRecipe`s som returvärde, vilken den sparar i sin datamedlem `searchResults_`.

Då en väldigt generell sökning kan tänkas generera resultat som motsvarar en majoritet av recepten i databasen, så hjälper det om programmet begränsar sig till endast den information som för stunden är relevant. Därav innehåller ett `MiniRecipe`-objekt endast information om namn, tidsåtgång och betyg. Notera att inget släktskap existerar mellan `MiniRecipe` och `Recipe`.

3.3 Ingredient

`Ingredient` är en abstrakt klass ur vilken `InfoIngredient` och `RecipeIngredient` är härledda enligt figur 3.3.

3.3.1 InfoIngredient

`InfoIngredient` representerar en enskild ingrediens. Utöver det gemensamma arvet så utökas `InfoIngredient` med set-funktioner för att kunna redigera ingrediensens attribut.

3.3.2 RecipeIngredient

`RecipeIngredient` representerar en enskild ingrediens som del av ett recept. Klassen utökar sitt arv med två datamedlemmar – `double amount_` och `''unittype'' unit_` – för att hantera två ytterligare funktioner: `getKcal(double scaling)` och `getPrice(double scaling)`.

3.3.3 RelatedRecipe

`RelatedRecipe` är en struct med vissa särskilda krav, vars funktionalitet är avsedd för att agera datatyp för släktskap. Ett objekt av typen `RelatedRecipe` ska endast hänvisa släktskap med ett recept som hänvisar släktskap med det recept som objektet själv tillhör. Tanken är att strikt hålla kontroll så att inga enkelriktade släktskap ska kunna förekomma i databasen när vi väl tillför eller ändrar denna typ av information.

3.4 SerchDB

Endast ett objekt av klassen `SerchDB` existerar i programmet och då som datamedlem av `Shell`. `SerchDB`'s funktion är att skapa `Recipe`-objekt och `InfoIngredient`-objekt, samt att utföra sökningar och skapa listor av receptnamn utifrån sökningarna. Det finns även funktionalitet för att ta ut snitt union och komplement för att kunna kombinera sökresultat.

`SerchDB` har följande datamedlemmar

- `list_db_` ett objekt av typen QsqlQuery som används för att söka i databasen samt att hålla datan.
- `ingredient_db` enligt ovan men används endast för att skapa recept objekt.

- `list_pos_` Heltal som anger hur många recept som tidigare har hämtats i databasen av `query_list`

Följande funktioner kommer användas för att göra uppslagningar, samtliga använder `SerchDB`s datamedlem `DB_` och har således inte något returvärde.

- `query_list` inga parametrar, läser in 20 recept i `list_db_` och uppdaterar `list_pos_`.
- `query_ingredient_list()` tar en vektor med ingredienser som parameter sparar namnet på alla recept som innehåller sagda ingredienser i `list_db_`.
- `query_ingredient_list_explicit()` samma som ovan fast för recept som *endast* innehåller sagda ingredienser i `list_db_`.
- `query_allergy_list()` tar en allergen som parameter och läser in alla recept innehållande sagda allergen i `list_db_`.
- `query_price_list()` tar ett prisintervall som parameter och läser in alla recept i givet intervall i `list_db_`.
- `query_calory_list` tar ett kaloriintervall som parameter och läser in alla recept i givet intervall i `list_db_`.

För datatillgång finns följande funktioner

- `get_list()` levererar en lista över receptnamn som finns i `list_db_`
- `get_recipie()` tar ett receptnamn som parameter och returnerar ett objekt av typen `recipe`.
- `get_info_ingredient()` tar ett ingrediensnamn som parameter och returnerar ett objekt av typen `info_ingredient`.
- `get_recipe_ingredient()` hjälpfunktion till `get_recipe`.

`SearchDB` kommer alltså inte att klara av alla olika kombinationer av sökningar på egen hand då detta skulle vara komplicerat att implementera utan kommer istället utgå från de sökningar som finns och sedan med hjälp av följande funktioner slå ihop listor för att nå fram till det resultat som önskas. Samtliga tar två listor som parametrar och returnerar en sammanslagen lista.

- `union()` returnerar ihop unionen av två listor.
- `intersect()` returnerar snittet av två listor.
- `complement()` returnerar två listors komplement.

3.5 EditDB

`EditDB` är den klass som används för att skriva data till databasen. Den skall kunna hantera tilläggning och borttagning av ingredienser samt recept. Den skall även kunna Lägga till kommentarer till ett recept samt lägga till redskap i databasen.

För att klara detta använder den sig av följande funktioner för insättning

- `add_recipe()` Tar emot ett `recipie`-objekt som parameter och updaterar databasen ifall receptet fanns sedan innan eller lägger till ifall det inte fanns.
- `add_ingredient()` Tar emot ett `ingredient`-objekt som parameter och updatera databasen om ingrediensen fann sedan innan eller lägger till om det inte fanns.
- `add_comment()` Tar emot ett receptnamn och en kommentar som parametrar och lägger till kommentaren till motsvarande recept
- `add_tool()` Tar emot ett namn på ett redskap och lägger till det i databasen.

Och följande funktioner för borttagning

- `remove_recipe()` Tar emot ett receptnamn som parameter och raderar motsvarande recept ur databasen.
- `remove_ingredient()` Tar emot ett ingrediensnamn som parameter och raderar motsvarande ingrediens ur databasen.
- `remove_tool()` Tar emot ett redskapsnamn och tar bort motsvarande redskap i databasen.
- `remove_comment()` Tar emot ett kommentars-id och tar bort motsvarande kommentar.

Endast en datamedlem finns och är av typen QsqlQuery vid namn `db_` och används för att sköta kontakten med databasen

3.6 SearchTerm

`SerchTerm` är en klass som sparar information om och utför en kombinerade sökningar i databasen. När en sökning skall genomföras skapas ett `SearchTerm`objekt och de medlemsfunktioner som krävs för de delsökningar som skall genomföras anropas. Därefter kallar `SearchTerm` på de sök- och sammanslagningsfunktioner SearchDB erbjuder för att kombinera ihop ett sökresultat.

följande datamedlemmar finns:

- `ingredients_` En vektor innehållande de ingredienser som skall ingå.
- `not_ingredients_` en vektor av ingredienser som inte får ingå.
- `allergies_` en vektor av de allergier som inte får ingå.
- `price_` en struct av typen Price innehållande undre och övre gräns på pris
- `cal_` en struct av typen Cal innehållande undre och övre gräns på kaloriinehåll.
- `time_` en struct av typen Time innehållande undre och övre gräns på tidsåtgång.
- `serch_result_` en vector av mini-recipies motsvarande resultatet.

Och följande medlemsfunktioner för dataåtkomst finns:

- `set_ingredients()` Tar en vektor av ingrediensströngar och lagrar dom i `ingredients_`.
- `set_not_ingredients()` Tar en vektor av ingrediensströngar och lagrar dom i `not_ingredients_`.
- `set_price()` Tar en struct av typen Price och lagrar den i `price_`.
- `set_cal()` Tar en struct av typen Calory och lagrar den i `cal_`.
- `set_time()` Tar en stryct av typen Time och lagrar den i `time_`.
- `set_allergy()` Tar en vector av allergiströngar och lagrar den i `allergies_`
- `push_ingredient()` Tar emot en sträng och lägger till den i `ingredients_`
- `push_not_ingredient()` Tar emot en sträng och lägger till den i `not_ingredient_`
- `push_allergy()` Tar emot en sträng och lägger till den i `allergies_`
- `get_result()` Hämtar relevant data från databasen med avseende på datamedlemmarnas innehåll och sammanställer en lista med hjälp av SearchDB och returnerar en vektor av `mini_recipie`.

3.7 Import och export av recept

Import och export av recept sker via funktioner i `Shell`, vilka skriver datan relaterad till ett specifikt `Recipe`-objekt till antingen en xml-fil eller en txt-fil (export), eller parsar en xml/txt-fil och konverterar denna till ett `Recipe`-objekt (import). Formatet för txt- respektive xml-filerna visas nedan:

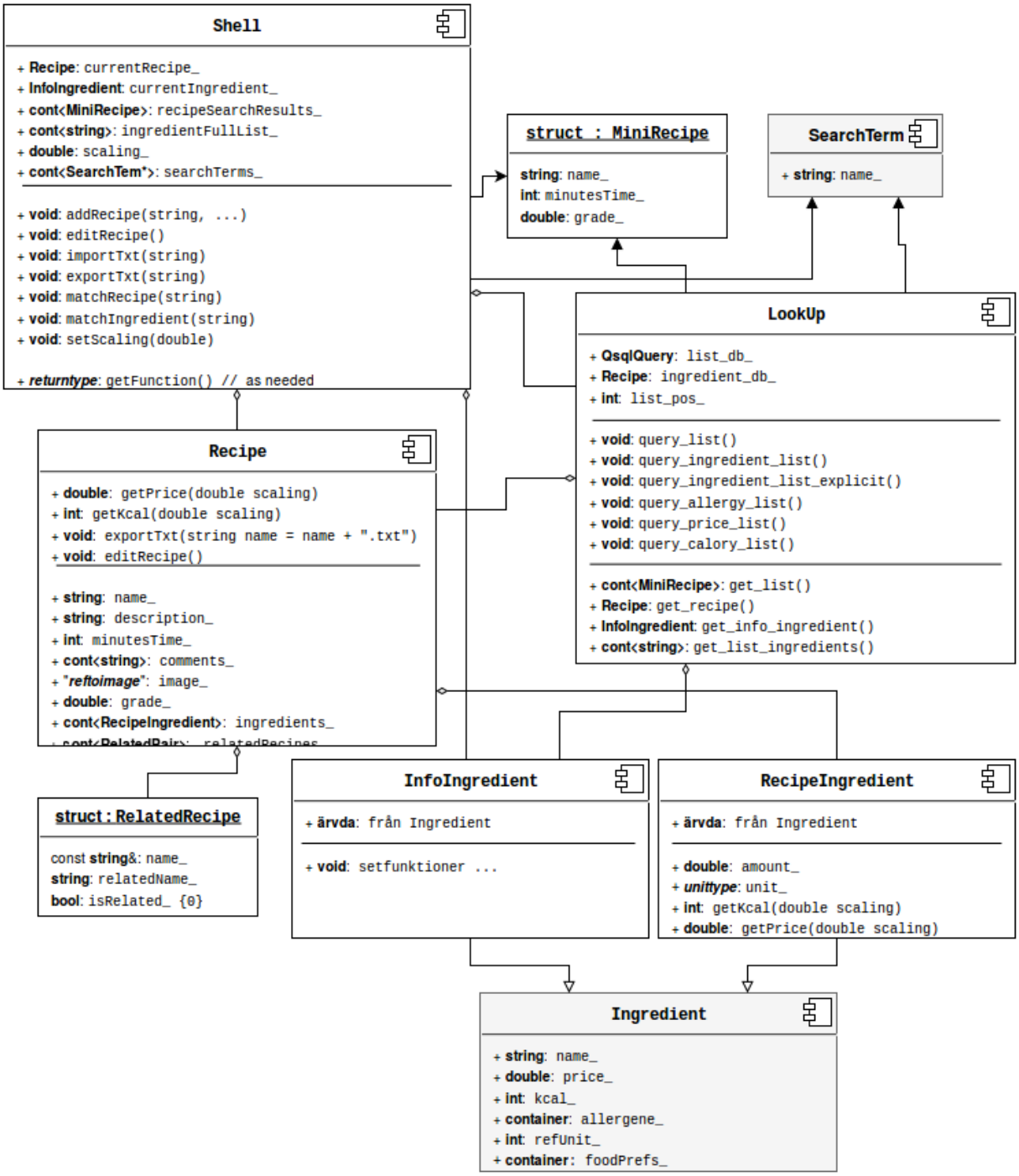
```
<?xml version="1.0"?>
<!-- recipe.xml -->
<recipe>
  <name>Köttbullar</name>
  <portionsize>4</portionsize>
  <ingredient>köttfärs<amount>500<unit>g</unit></amount></ingredient>
  <ingredient>ströbröd<amount>0,5<unit>dl</unit></amount></ingredient>
  <ingredient>grädde<amount>1<unit>dl</unit></amount></ingredient>
  <ingredient>lök<amount>2<unit>msk</unit></amount></ingredient>
  <ingredient>ägg<amount>1<unit>st</unit></amount></ingredient>
  <ingredient>salt<amount>1<unit>tsk</unit></amount></ingredient>
  <ingredient>svartpeppar<amount>1<unit>krm</unit></amount></ingredient>
  <ingredient>olja<amount>2<unit>msk</unit></amount></ingredient>
  <instruction>Blanda ströbröd och grädde. Låt svälla 10 min. Lägg i färs, lök, ägg,
  salt och peppar. Rör till en jämn smet. Forma färsen till jämna bullar. Stek dem i
  smör-&amp;amp; rapsolja på medelvärme 3-5 min.</instruction>
  <time>40</time>
  <tool>stekpanna</tool>
  <energy>375</energy>
  <rating>3</rating>
</recipe>
<!-- EOF -->
```

```
<?xml version="1.0"?>
<!-- ingredient.xml -->
<ingredient>
  <name>köttfärs</name>
  <stdunit>kg</stdunit>
  <energy>1910</energy>
  <price>89</price>
  <egg>0</egg>
  <milk>0</milk>
  <celiac>0</celiac>
  <nuts>0</nuts>
  <animalic>1</animalic>
</ingredient>
<!-- EOF -->
```

```
<!-- recipe.txt (kommentarer kommer ej med vid export) -->
# Köttbullar (4 portioner)
Tillagningstid: c:a 40 min
==========================
köttfärs 500 g
ströbröd 0,5 dl
grädde 1 dl
lök 2 msk
ägg 1 st
salt 1 tsk
svartpeppar 1 krm
olja 2 msk
```

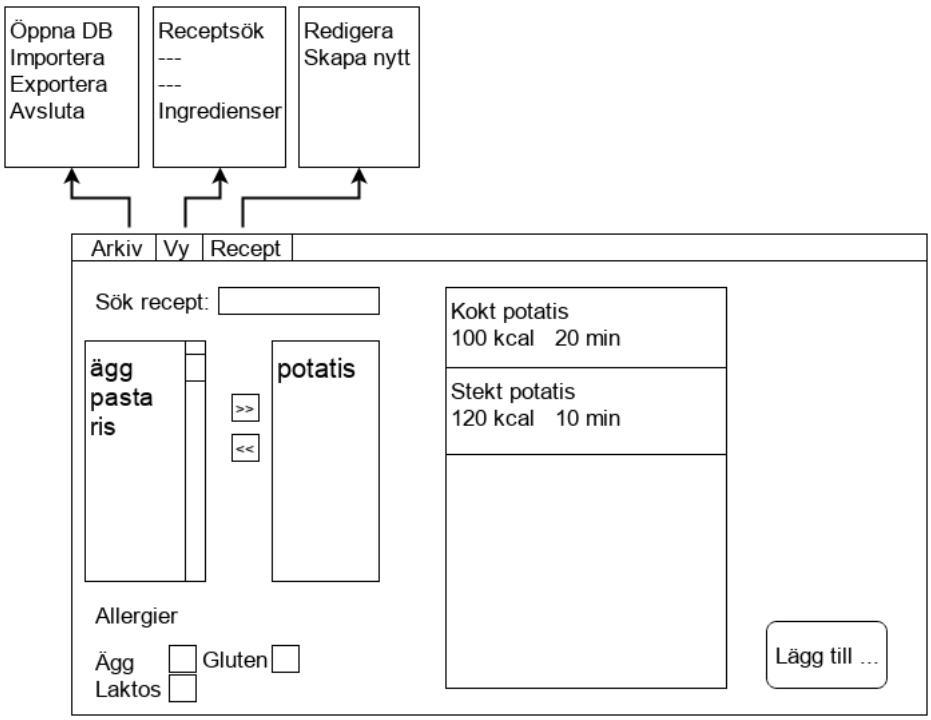
```
Blanda ströbröd och grädde. Låt svälla 10 min. Lägg i färs, lök, ägg, salt och peppar.
Rör till en jämn smet. Forma färsen till jämna bullar. Stek dem i smör-&rapsolja på
medelvärme 3-5 min.

Husgeråd: stekpanna
Uppskattat energiinnehåll: 375 kcal
Betyg: 3/5
<!-- EOF -->
```

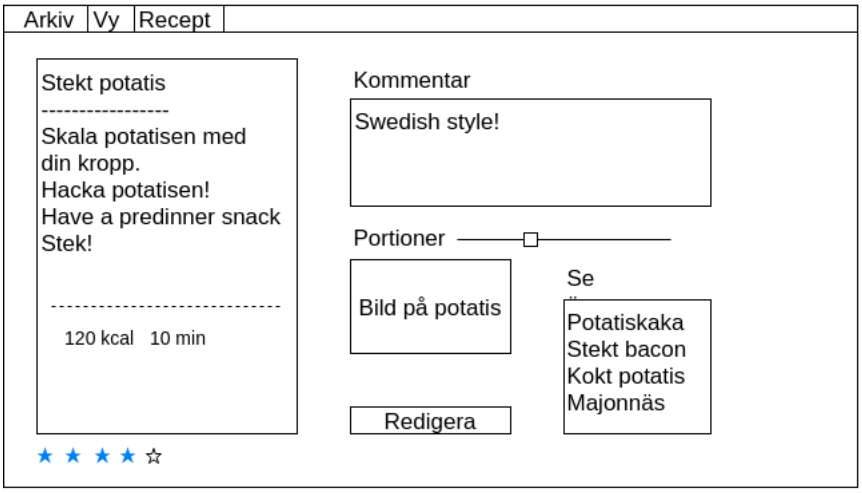


Figur 3.3: Klass-schema

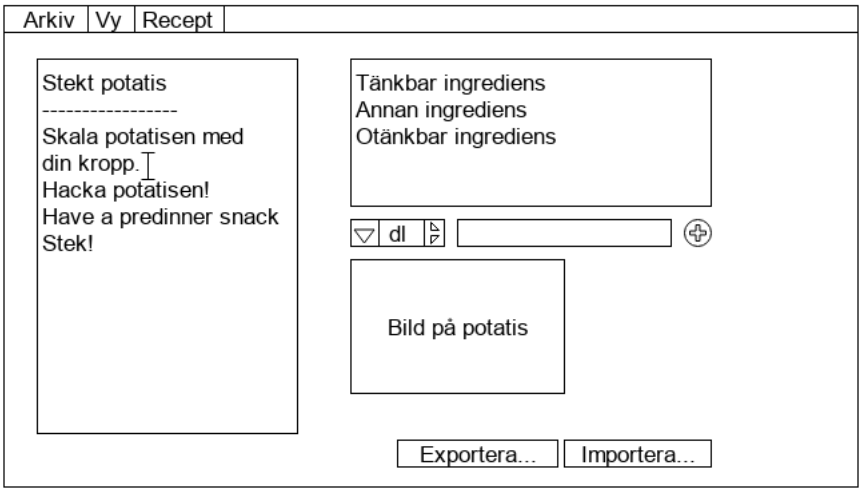
4. Design av användargränssnitt



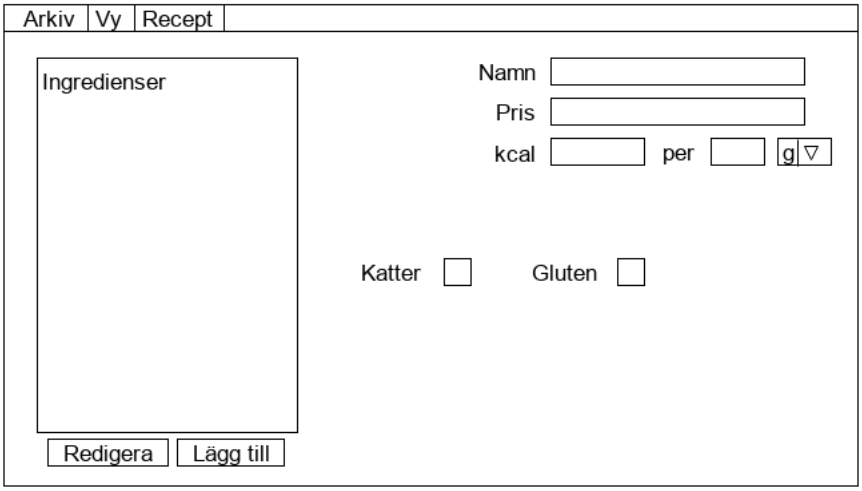
Figur 4.1: Sökfönster



Figur 4.2: Receptfönster



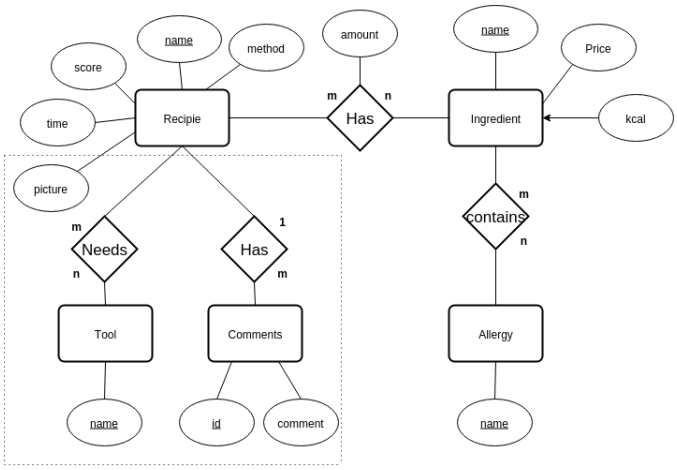
Figur 4.3: Redigering av recept



Figur 4.4: Redigering av ingredienser

5. Design av databas

Då all receptinforamtion behöver sparas mellan körningar av programmet behöver den lagras externt. I det här programmet kommer en databas användas med hjälp av MySQL. Databasen kan överskådas i EER-diagramet i 5.1. Den del som ligger inom de streckade området hör till funktionalitet som endast kommer implementeras i mån av tid.



Figur 5.1: Entity-Relationship diagram

Databasen kommer bestå av fem entiteter `Recipe` som innehåller information unik för ett enskilt recept. `Ingredient` som är en lista över de olika ingredienser som databasen innehåller, `Allergy` som är en lista över allergier, `Tool` som är en lista över allergier samt `Comment` som är en lista över kommentarer till varje recept.

Entiteten `Recipe` är den entiteten som lagrar namn, beskrivning, bild tillagningsmetod och tidsåtgång. Då recept skall ha unika namn för att särskilja dem åt är det receptets namn som agerar primärnyckel.

`Recipe` har en m-n relation till `Ingredient` vilket ger oss en lista på ingredienser till vajre recept. Genom att ha attributen `kcal` och `price` på entiteten `Ingredients` istället för `Recipe` behöver unik information om portionspris och närings-inehåll till varje recept inte sparas utan kan räknas ut beroende på vilka ingredienser som ingår. Genom att tillföra attributet `amount` behöver varje ingrediens endast lagras en gång per recept och enhetsomvandling och portionsskalning kommer vara möjlig.Den har även en 1-n relation till `Comment` vilket resulterar i att alla recept får en lista med komentarer skrivna av användaren. samt en m-n relation till `Tool` som ger en lista över de redskaps som behövs.

För att hålla ordning på de vanligaste matallergierna (samt kött mejeri och fisk för veganer/veganer) finns entiteten `Allergy`.

Genom att utforma databasen enligt sagda modell kommer programmet kunna utföra olika sökningar och filtreringar på ingredienser som ingår, inte ingår, eventuella allergener etc.

6. Exempel

Detta avsnitt utgör ett exempel på användarinmatning och vad som händer bakom kulisserna till följd av denna.

1. Användaren står i sökfönstervyn (fig. 4.1) och klickar på *"Lägg till"*.
2. Vyn ändras till redigeringsvyn (fig. 4.3).
3. Användarinmatning av receptdetaljer i respektive fält. Avslutar med *"Spara recept"*.
4. Skapa tomt `Recipe`-objekt och anropa dess `Recipe:editRecipe`-funktion med fältenas värden som argument.
5. Ett anrop `Shell::addRecipe(Recipe)` görs.
6. Anrop: `EditDB::addRecipe(Recipe)`
7. `EditDB` använder receptets namn och slår i databasen. Om det redan existerar skrivs informationen över. Om inte så läggs raden till i tabellen med recept.
8. Returvärde returneras beroende av utfall.
9. Nivå `Shell`: `currentRecipe_` uppdateras, returvärdet ovan returneras uppåt.
10. Nivå GUI: GUI ger feedback baserat på returvärde (recept {tillagt,uppdaterat}).
11. Byt vy till receptvy (fig. 4.2).