

# Account This!

---

## Group 9

**Johannes Edelstam**  
**Joakim Ekberg**  
**Kristoffer Renholm**  
**Jesper Skoglund**

## History

Version	Comment	Authors
1.0	Initial version.	Kristoffer Renholm, Johannes Edelstam, Joakim Ekberg, Jesper Skoglund

## Contents

1	Introduction.....	5
1.1	Concerning the design document .....	5
1.2	Glossary .....	5
1.3	Structure.....	6
2	System Overview.....	7
2.1	General Description.....	7
2.2	Overall Architecture Description.....	7
2.3	Detailed Architecture .....	7
2.3.1	Model View Controller .....	7
2.3.2	MVC implementation in Ruby on Rails.....	8
2.3.3	The file structure of a Ruby on Rails project .....	9
2.3.4	RESTful development .....	10
3	Design Considerations .....	11
3.1	Assumptions and Dependencies .....	11
3.2	General Constraints.....	12
4	Graphical User Interface.....	13
4.1	User Interface Overview.....	13
4.2	Concept and Functionality.....	13
4.2.1	Stage 1 – Welcome .....	13
4.2.2	Stage 2 – Launch (optional) .....	15
4.2.3	Stage 3 – Action.....	16
4.3	Detailed Mockups.....	17
4.3.1	Register new company .....	17
4.3.2	User Login.....	18
4.3.3	Update user information.....	18
4.3.4	Update company information .....	19
4.3.5	Create a new user in a company .....	19
4.4	Ask a question .....	20
4.4.1	Create new Fiscal Year.....	20
4.4.2	Change current Fiscal Year .....	21
4.4.3	Creating an accounting plan.....	21
4.4.4	Create a new Voucher .....	22
5	Design Details .....	23

5.1	Class Responsibility Collaborator (CRC) Cards.....	23
5.1.1	Models.....	23
5.1.2	Presentation .....	25
5.2	Class Diagram .....	27
5.2.1	Models.....	27
5.2.2	Controllers .....	27
5.3	State Charts .....	27
5.4	Interaction Diagrams .....	28
5.4.1	General HTTP GET Request Sequence.....	28
5.5	Detailed Design.....	28
5.5.1	Database.....	29
5.5.2	Models.....	30
5.5.3	Controllers .....	35
5.6	Package Diagram .....	48
6	Functional Test Cases .....	50

# 1 Introduction

A clarification of the purpose, the scope and the intended audience of this design document is seen below, prior to an explanation of the textual structure of the document and a brief glossary explaining the terms that are fundamental for understanding this project.

## 1.1 Concerning the design document

The purpose of this design document is to serve as a framework meant to support the development of the AccountThis! bookkeeping system. The design document can rightfully be seen as a collection of design considerations, including textual and visual clarifications on the many decisions concerning choice of software architecture and the thoughts underlying the graphical user interface.

The scope of the design document is to be focused on those aspects and factors that need to be mutually understood and agreed upon by the software developers. This will come to involve the planned system structure along with the functionality of the used classes and methods. But whilst connections and relations will be clarified, the document won't include any more than a few precise descriptions on how the different parts are to be built. The spotlight of this document is on the software design, the functionality and the logics that will serve as the backbone of AccountThis!, not on the specific layout of certain code. A software developer reading the document should be able to grasp how the system is intended to be developed, hence allowing him/her to analyze the written code or to contribute to the project by writing any of the missing parts.

System developers do, as have been indicated, constitute one group for whom the design document has been written. The design document will, by serving as a framework for the development, be able to assist us (or other programmers) in developing and testing the software. By examining this plan before the programming commences, it should be possible to detect any structural errors. And by examining and comparing the contents of this plan to the actual code during the development phase, one ought to be given better prospects of detecting bugs and glitches. This document is likely to be of interest for developers, customers and managers alike, and could during its establishment serve as a platform for knowledge transfer between these actors.

Those not familiar with the rationale of the AccountThis! bookkeeping system are advised to see the requirements document concerning the project. Whereas the plans seen in the requirements document serve as a basis for the project, the information seen here (in the design document) will rather function as a manual for the actual system development.

## 1.2 Glossary

Below is a list of abbreviations and terms we – the authors and developers of AccountThis! – regard as essential to grasp in case the contents of this design document are to be fully understood.

- Ajax – Asynchronous JavaScript and XML, Ajax, is a technique (based on a combination of languages) that can be used for creating web applications [source: Wikipedia].
- Apache – Is the name of a popular HTTP server application [source: Wikipedia].
- CGI – Common Gateway Interface, CGI, is a standard protocol for interfacing external application software with an information server, commonly a web server [source: Wikipedia].

- ERb – ERb is used to integrate Ruby with HTML and can therefore be used to generate HTML pages with dynamic content, using code written in Ruby.
- HTTP – Hypertext Transfer Protocol (HTTP) is a communications protocol used to transfer information on intranets and the World Wide Web [source: Wikipedia].
- InnoDB – InnoDB is a storage engine for MySQL, included as standard in all current binaries distributed by MySQL AB [source: Wikipedia].
- MIME – Multipurpose Internet Mail Extensions (MIME) is an Internet Standard that extends the format of e-mail [source: Wikipedia].
- Model View Controller (MVC) – a
- Mongrel – Mongrel is an open-source HTTP library and web server for Ruby web applications [source: Wikipedia].
- MySQL – MySQL is a multithreaded, multi-user SQL database management system (DBMS) which has, according to MySQL AB, more than 10 million installations [source: Wikipedia].
- REST – Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web [source: Wikipedia].
- Ruby on rails – Ruby on Rails is a free web application framework [source: Wikipedia].
- SQL – SQL is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management [source: Wikipedia].
- UI – The user interface (or Human Machine Interface) is the aggregate of means by which people interact with the system [source: Wikipedia].
- URL – Uniform Resource Locator (URL), still known as Universal Resource Locator, is a technical, Web-related term [source: Wikipedia].

### 1.3 Structure

Following the introductory chapter comes a part that will focus on the system itself. Through reading this, readers will be granted an opportunity to learn about our choice of system architecture; both in a more general and in a more detailed manner.

The third chapter will discuss and describe the design considerations, assumptions, dependencies and general constraints. This part tries to identify what needs to be thought of while devising the final design solution.

Chapter four portrays the graphical user interface. This part gives an overview of how the interface will be constructed, what it will look like and how it will communicate with (trigger) the system functionality.

The fifth chapter explains the design considerations in detail through the use of class responsibility collaborator cards, class diagrams, state charts, interaction diagrams and package diagrams, in combination with descriptive text.

Building on what's already been mentioned about the design is chapter six, where functional test cases are used to cover and describe the full range of functionality described in the requirements document.

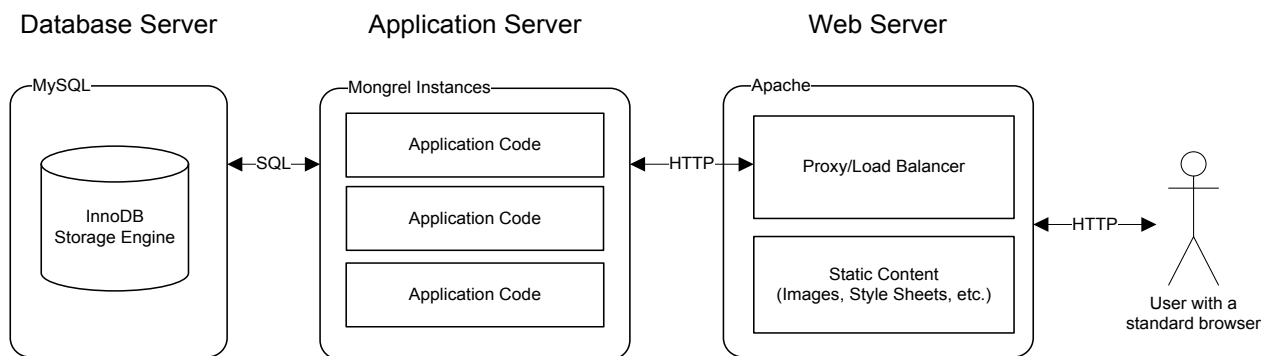
## 2 System Overview

### 2.1 General Description

Account This! aims to deliver a bookkeeping system suitable for small companies. Such companies could for example be hairdressers (almost every hairdresser in Sweden has a one-man-company which rents a hairdressers chair in a barbershop), consultants or similar small companies. Small companies often have small budgets where every cent matter. Account This! will be a system for double-entry booking; the standard used in most businesses and organizations. Unlike other, often large and complex, bookkeeping systems, ours will be web-based, easy to access and easy to use. Account This! should thus be able to help its intended users (mainly small companies) do, view and edit their bookkeeping.

### 2.2 Overall Architecture Description

The overall system architecture is divided into three major components: the database, the application and the web server. Each component has its own responsibilities and is communicating with other components through standardized protocols and communication channels.



*The web server* serves as the front-end towards the user. It is responsible for serving the user's request for static content like pictures and style sheets, and for forwarding the requests to the Application Server, that will generate a dynamic response or allow user input.

*The application server* is responsible for running the Account This! code base. Due to lack of threading in most Ruby applications there is no natural way of processing and responding to concurrent requests from users. Therefore, multiple instances of the special hosting server software for Ruby on Rails applications – called Mongrel – will be run concurrently on different TCP/IP ports. The web server is responsible for distributing the load over these instances in an even manner.

*The database server* is responsible for persisting data between user sessions. The database server uses a transactional storage engine with support for relations between tables. This reduces the risk of data corruption.

### 2.3 Detailed Architecture

#### 2.3.1 Model View Controller

Model View Controller (MVC) is a well known software design pattern. Its purpose is, like is for many other design patterns, to organize the code in a maintainable way.

The MVC principle divides the application into three separate subsystems called layers. These layers – Model, View and the Controller – can be described as follow:

- *Model* - The representation of the domain specific entries that builds up foundation of the application. This layer could consist of anything from users to shopping charts or accounts, and so on.
- *View* - The views purpose is to, as the name hints, present the data in different ways.
- *Controller* - The controller represents the glue in the application. The controller layer directs traffic inside the application and will thus handle everything from the querying of models (for information) to the rendering of end user views.

When building according to the MVC pattern, program code is separated in different layers. The code never floats around, i.e., the design pattern makes every line of code to live in one of the three layers.

### 2.3.2 MVC implementation in Ruby on Rails

This section of the document aims to describe how Ruby On Rails (Rails) implements the Model View Controller (MVC) design pattern.

When writing program code according to the MVC design pattern each line of code fits into one of the three layers of the application: model, view or controller.

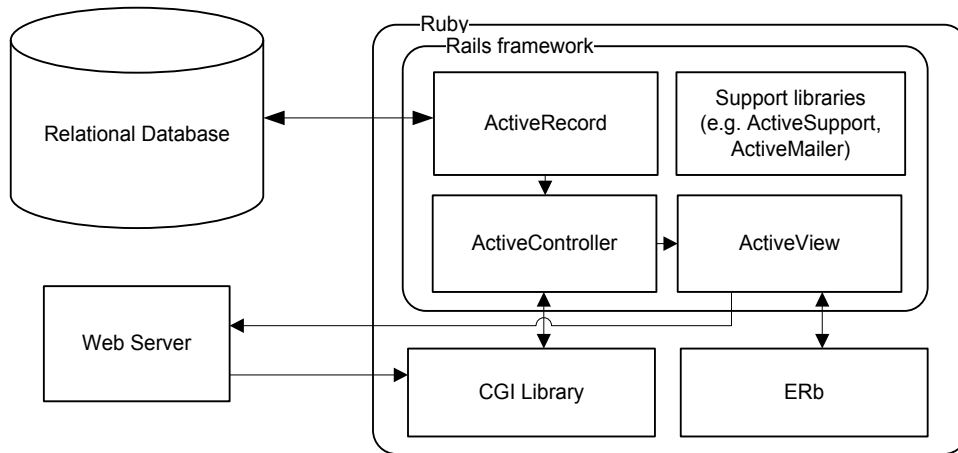
In accordance with its MVC foundations, Rails is made up of three different subsystems. Separate in each sense that they could be used individually. These are:

MVC Phase	Rails subsystem	Purpose
Model	ActiveRecord	The model provides the bridge between the database and the Ruby code that builds up the models. ActiveRecord provides several functions to read, manipulate, find, etc. to data. One important quality of ActiveRecord that it generates Ruby-methods from each field in a database column. Therefore it's very easy to translate the database to a set of model objects.
View	ActionView	The views of a Rails project usually contains of several XHTML documents with embedded ruby to display the dynamic content for the specific view.
Controller	ActionController	The ActionController represents the final glue needed to connect the view with the model. It handles forms and determine which view to render according to the data that the user inputs.

What is Rails then? We now that it's built up from three components that have different purpose according to the MVC pattern.

Rails is the only necessary infrastructure that is needed to connect these three different subsystems together to create a great web framework.





Please consult the glossary seen in chapter one for a description of terms as CGI and ERb. ActionView will use ERb to generate dynamic content from the controllers displaying views.

### 2.3.3 The file structure of a Ruby on Rails project

The file structure consists of more or less important directories. They are predefined by Ruby on Rails. This section describes them and how the files inside them will be named.

app/	Holds most of the source code. All the project specific code is placed here.
app/controllers/	This is where all controllers are placed. They will be named like vouchers_controller.rb to ensure that the paths will be recognized automatically by Rails.
app/models/	This is where all the models are placed. They will have names such as voucher.rb
app/views/	Holds all of the template files that will be rendered from the controllers, or by another template. They are named like vouchers/show.html.erb. Where 'html' could be any format and show is the name of the corresponding action for show in the vouchers controller. A template could also be named _voucher.html.erb, making it a partial. Partials are will be used to render rows describing the vouchers. A file could also be named vouchers/index.xml.builder, meaning that it is a template able to generate XML corresponding to the vouchers' controller index action.
app/views/layouts/	This is where the layouts will be stored. Layouts serve as templates and could (for example) be named application.html.erb. Each layout will contain html that can be rendered for each request (e.g. menus and such).
app/helpers/	This directory holds all the helpers. A helper is a View layer function that functions similar to a template and as such is likely to be used more than once, in a variety of instances and scenarios. A helper could be named as vouchers.rb. It will contain methods that the voucher template uses. There will also be a helper named application.rb which will contain methods available to all templates.

config/	This directory is very much described by its name. Config holds all configuration files, such as routes.rb which contains information about how rails should create paths. It also contains databases.yml which describes all databases that the project is using. Also all configuration files for the environments are kept here.
components/	This directory was used in previous versions of Rails, but not any more.
db/	Contains the auto generated file schema.rb which describes how the database tables were created.
db/migrate/	This directory holds all migration files. They describe how all the tables should be created. They are named as 009_create_vouchers.rb.
doc/	Auto generated project documentation ends up here. This documentation is generated (from comments in the source code) using RDOC, a utility used to produce an HTML-API covering each and every function included in the AccountThis! project.
lib/	Any extensions or classes that isn't models or controllers ends up here. It could i.e. be a parser or similar.
public/	This directory holds all files that should be directly available from a web browser. Like images, javascripts and CSS style sheets.
script/	Everything in this directory is auto generated when the project is created. It is different scripts to e.g. generate new controllers and models, or to destroy them and all the files that belong to them.
test/	Testing is important to every large software project. The test/ directory contain all functional and unit tests along with fixtures. Fixtures are files to load test data in the databases.
vendor/	Holds libraries that the project relies on. It also contains the plugins directory which holds all plugins used by the project.

#### 2.3.4 RESTful development

Account This! will be written with the quite new technique called RESTful Rails. REST is short for Representational State Transfer. The concept is to take advantage of the fact that the HTTP protocol standard uses more than just POST, and GET. It also uses the methods PUT and DELETE. Every URL should be mapped to a resource on which you can perform any of these methods instead of using URLs to trigger certain actions. The URL /vouchers/ can typically be called with the GET method that would correspond to get all vouchers. A POST to the very same URL would instead correspond to creating a new voucher. The new voucher would be accessed e.g. at the URL /vouchers/4. A call with the method PUT to that URL would result in an update of that specific voucher. A call with the method DELETE would result in destroying that voucher. RESTful also handles different formats in a very clever way. Since every URL is connected to a specific resource, every URL should theoretically be able to be called using any format. Like /vouchers.xml and /vouchers/4.xml. This would then result in getting the output from the voucher controller as XML.

HTTP Verb	REST-URL	Action	URL without REST
GET	/vouchers/4	show	/vouchers/show/4
DELETE	/vouchers/4	destroy	/vouchers/destroy/4
PUT	/vouchers/4	update	/vouchers/update/4
POST	/vouchers/	create	/vouchers/create

Why RESTful?

- Clean URLs. Every URL becomes very easy to understand.
- Format handling. Every resource can easily be requested with different formats.
- Clear code structure. When you open up a controller it is very easy to understand what happens on every request, thanks to the use of the HTTP methods.

Why not RESTful?

- Complications while using AJAX. Sometimes you would like to use the same method for several different outputs. That can however be solved by using formats that is defined by the developer, such as /vouchers.compact to get a compact list of vouchers, however the MIME type is still HTML.
- Sometimes these methods just isn't enough, you would like to create more actions. You can however do so, but it isn't totally by the book.

### 3 Design Considerations

This chapter intends to describe the major issues that must be taken into consideration when planning and realizing the implementation of the AccountThis! bookkeeping system.

#### 3.1 Assumptions and Dependencies

As AccountThis! is to demand no high-tech hardware nor no installation (except a functional web browser application) in order to be run, it seems fair to assume that most of the application's software dependencies will be based on the specific end-user characteristics. In order to understand how to successfully design the system, we must thus begin with making sure that we've grasped the demands of the system's primary audience; our customers.

The context of those using AccountThis! is likely to be all but consistent. While some may be employed for medium sized companies, others are likely to come from small firms and be bosses of their own. This poses both opportunities and risks. A person from one industry may not have the exact same preferences as one from another sphere and the same goes for the third and the fourth person. To fully satisfy all these desires may be impossible but through looking at what's important in ordinary accounting, we identify a few general pain points which the design must be able to cure. First, it's evident that no one would use a bookkeeping system that felt unreliable. Second, as bookkeeping can be a very tedious task, people are likely to refrain from using these systems in case they could be deemed as being inefficient. Third, systems that come with a steep and demanding

learning curve may not be well suited for smaller companies as their employees are likely to work rather as generalists than as experts.

Several helpful considerations can be tapped from the assumed pain points. To ensure that people feel reliable when using the system, AccountThis! must not only work as planned, it must also be equipped with a user interface that speaks of trustworthiness. The obtainment of trustworthiness may be ensured through adaptation of common design practices from popular sites and systems. Using a very ordinary design would however risk placing our system in the segment of being nothing but dull an ordinary, wherefore a certain balance between new and old ideas (that have proven to be successful) must be withheld. To ensure that our clients find AccountThis! useful, we must also strive to remove any unnecessary steps in the workflow of the application. It will hence be necessary to make sure that the system remains quick and easy to navigate, even as it may later come to be expanded in both size and functionality. To ensure that AccountThis! is easily learnt, it must also be designed with a variety of cognitive behavioural patterns in mind.

While AccountThis! is currently able to handle most hardware (due to the software's low system requirements) and most operating systems, there's no guarantee that this will always be the case. It may become necessary to alter the system in the event of new technology or new customer demands, as to achieve better strategic alignment. The system should be designed with these considerations in mind. This means that: (1) ; AccountThis! must appeal to its users, and (2) the application must be extendable (from a coder's perspective).

### **3.2 General Constraints**

Due to differing legal requirements between different regions, it would be necessary to localize and adapt AccountThis! in case it was to be launched on a foreign market. While this poses a hinder for a rapid global expansion, it makes our current job a little easier. Instead of including multi-lingual support from the start, we can now focus on that of our core market; namely Swedish. While many multinational companies use similar bookkeeping standards, these are not universal and as lacking compliance could have fatal implications for the user of the bookkeeping system, we'll rather focus on making sure that the bookkeeping system meets all the local criteria. This will most likely improve the reliability of the functionality seen in AccountThis!, consequently reducing the risks that we, as developers and owners of the system, would be the subjects for possible lawsuits coming from disappointed customers.

Security is another issue. What the clients of AccountThis! are meant to enter into the system may be secret or highly sensitive financial data. The system must therefore – due to the risk of break-in attempts – not accept more customers and clients than it can possibly handle.

Assessing the quality goals and performance requirements of the AccountThis! system could take a tremendous amount of time. It will thus be difficult for us to guarantee that the application meets all of its requirements while it remains in beta phase.

## 4 Graphical User Interface

This chapter explains the logic, the concept and the functionality of the user interface that's been planned for use in the AccountThis! bookkeeping system.

### 4.1 User Interface Overview

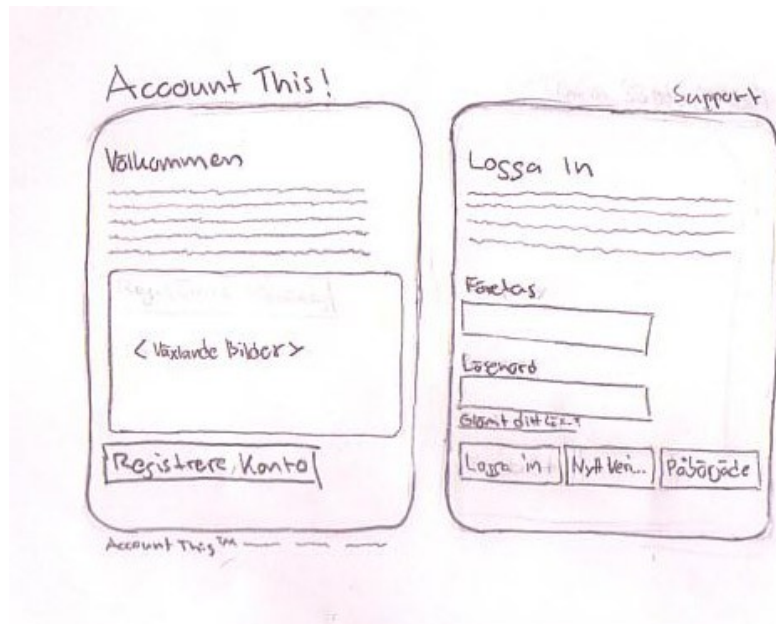
The AccountThis! user interface will serve as the link between the system and its clients. This implicates not only that the interface must – as the chapter concerning design considerations showed – be both easy and efficient to use, but also that it should be able to encourage potential users to sign-up. Attracting users can be done in several ways. While promotion and advertising are two such means, the establishment of a neat user interface is, in this case, a third. Potential clients that have managed to navigate to the AccountThis! webpage must: (1) get the impression that AccountThis! is a reliable and efficient bookkeeping system; (2) be able to retrieve more information about AccountThis!; and (3) find it easy to register for an account. By satisfying these criteria, the system will be much more likely to catch the attention of the masses than it otherwise would have been.

### 4.2 Concept and Functionality

A comprehensive discussion on alternative user interface designs led us to the solution portrayed below. To meet all the discussed business and clientele needs, we'll use a user interface that's divided into three stages. The first stage represents the opening page a user sees when he/she first enters the site and the two other stages follow in a chronological manner. While the first stage contains more visuals than the second stage, the second stage contains more visuals than the third ditto.

#### 4.2.1 Stage 1 - Welcome

It's obvious that visual design can be more or less appealing to people. Although there may be no universal good taste, a user interface may still be classified as being better than another. A good user interface should satisfy the needs of its users in an efficient manner. It would therefore be careless, if not entirely dumb, not to have a login function from the very first page of the system, given that users must login to be able to actually use the system. Any solution involving more steps would pose a clear obstruction to the user. From a business perspective, it's also of dire importance that the first page is able to help entice new clients. To solve this problem, and to satisfy both new and existing users, we've decided to divide the first page of the AccountThis! webpage in two. The leftmost part of the page will consist of flashy images on which short textual pitches have been applied. Whereas this may serve a medium for luring prospective customers to signup, the rightmost part of the page will rather focus on assisting already existing users, as this is where they login. The first fields of the login box (username and password) bring nothing new to the world of the web based systems but the buttons seen below have seldom been seen elsewhere. Next to the ordinary login button are login buttons that will serve as shortcuts, taking verified users straight from the opening page to the desired (pressed) system function; thereby avoiding the system's launch guide (stage 2).



UI concept image 1

The image above displays the AccountThis! opening page and how it's divided into two boxes. The first contains what could be described as AccountThis! advertisement and the second contains the system's login functionality; including account verification and shortcuts to the respective clients' instances of the bookkeeping system.

In case the image on the leftmost part of the opening page is pressed, the user will be transferred to a site (with a rather similar user interface) containing more detailed information about the system and the option of opening an AccountThis! account.

Below are sentences describing the functionality of the buttons that are displayed on the opening page of AccountThis! (these will be placed as can be seen on *UI concept image 1*).

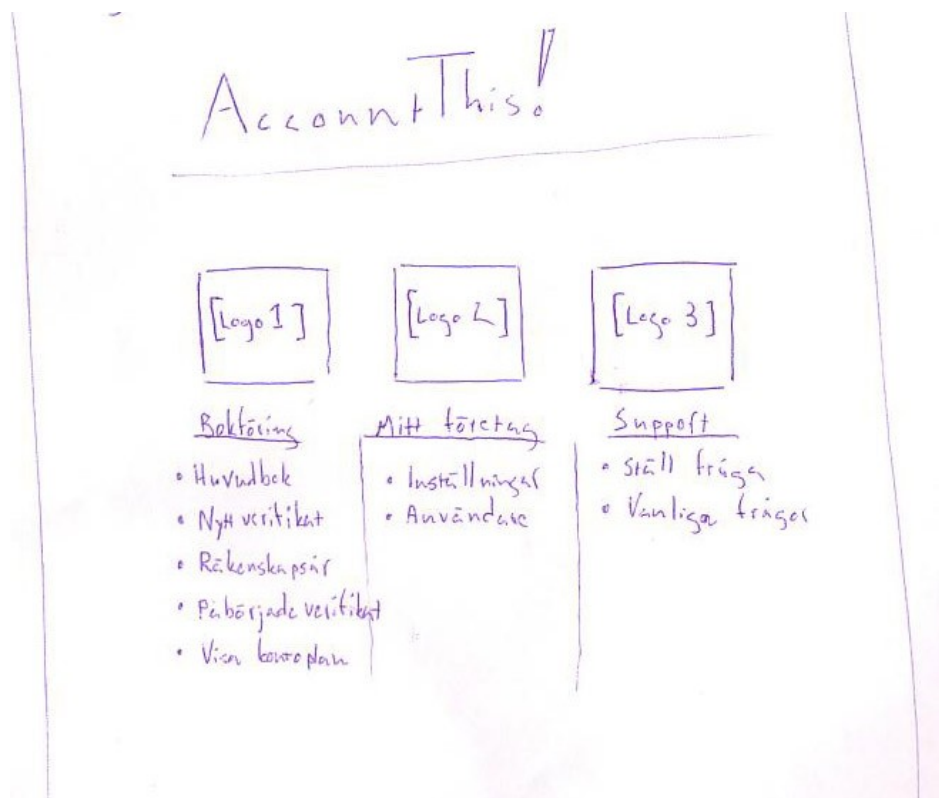
- *Registrera konto* – Pressing this button will take the user to another webpage, a form sheet, where he/she is able to fill in the personal information and specifics required to establish a new account.
- *Glömt ditt lösenord?* – Pressing this button take the user to another webpage where he/she can input an e-mail and trigger a function. This function will, in case the entered e-mail address is associated with an AccountThis! account, have the system send a message containing the related user password to the given e-mail address.
- *Logga in* – Pressing this button will have the system verify the entered user name and password, thereby rejecting or accepting the user to enter the system.
- *Nytt verifikat* – Pressing this button will have the system verify the entered user name and password, thereby rejecting or accepting the user to enter the system. If the user is accepted to proceed, he/she will immediately be transferred to the new voucher page.
- *Påbörjade verifikat* – Pressing this button will have the system verify the entered user name and password, thereby rejecting or accepting the user to enter the system. If the user is accepted to proceed, he/she will immediately be transferred to a page containing the user's voucher drafts.

#### 4.2.2 Stage 2 – Launch (optional)

Stage 2 is where users arrive when they use the ordinary login alternative. This, the launch page, presents an overview of the system's functionality that may assist users in quickly navigating to the functions they desire to use. The functions are categorized under relevant headings, and the headings in turn are to be associated with symbols to further increase the simplicity of learning AccountThis!.

The planned contents of the launch site are (where font in bold represents headings):

- **Bokföring**
  - Huvudbok
  - Nytt verifikat
  - Räakenskapsår
  - Påbörjade verifikat
  - Visa kontoplan
- **Mitt företag**
  - Inställningar
  - Användare
- **Support**
  - Ställ fråga
  - Vanliga frågor



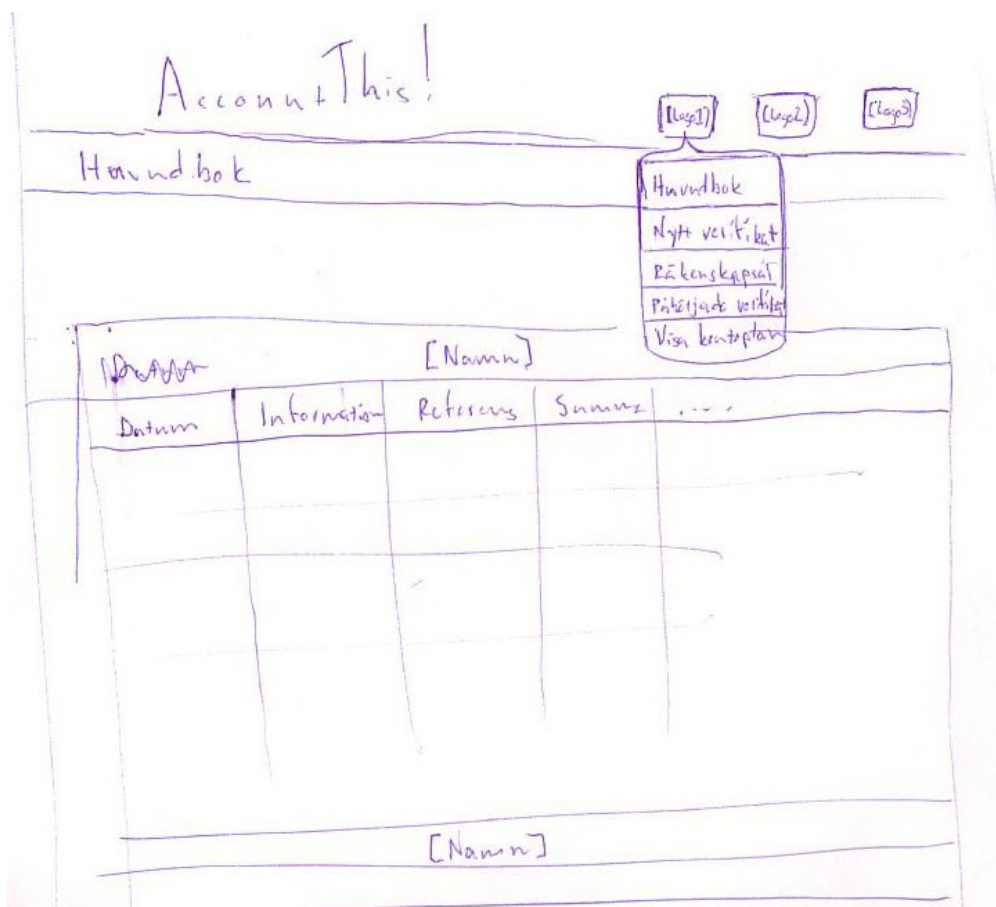
UI concept image 2

The area below the navigation alternatives displayed above may come to include system notifications, news or usage history (these alternative changes to the stage 2 user interface are still under consideration).



### 4.2.3 Stage 3 – Action

The symbols shown above the respective headings on the launch page, stage 2, will remain visible (but in an altered form) from the stage 3 user interface. As this is the user interface people will encounter when they perform tasks with AccountThis!, space must be made for the important content (such as the information contained in the general ledger). The navigation symbols will, for this reason, be placed in the top right corner of the webpage, where they'll act as dropdown menus when highlighted. The concept image seen below (*UI concept image 3*) displays how the dropdown menus may look, given that a fictive user are currently highlighting the symbol associated with the first heading. Note that the appearing alternatives are the same alternatives/functions as those that were displayed in conjunction with the heading before, when shown on the launch site.



UI concept image 3



## 4.3 Detailed Mockups

This section contains more detailed mockups of the Account This! user interface. Text with yellow background describes the name of the form element.

### 4.3.1 Register new company

**Requirement:** User – 1

**Use Case:** User – Register new company

## Registrera företag

### Företagsdetaljer

Här fyller ni i era detaljer för vad som rör erat företag.

Namn (@company.name)

Företagsform (@company.form)

Organisationsnummer (@company.organisation\_number)

Gata (@company.adress.street)

Postnummer (@company.adress.postnumber)

Ort (@company.adress.city)

### Användardetaljer

Här fyller ni i detaljerna för vad som rör den första användaren i företaget.

Användarnamn (@user.username)

Namn (@user.username)

Lösenord (@user.password)

Repetera Lösenord (@user.repeat\_password)

Telefonnummer (@user.phone\_number)

Email (@user.email)

Gata (@user.adress.street)

Postnummer (@user.adress.postnumber)

Ort (@user.adress.city)

Registrera

### 4.3.2 User Login

**Requirement:** User – 2

**Use Case:** User – User login

## Logga in

Namn (@user.username)

Lösenord(@user.password)

Logga in

### 4.3.3 Update user information

**Requirement:** User – 4

**Use Case:** User – Update user information

## Redigera användare

Namn (@user.username)

Telefonnummer (@user.phone\_number)

Email (@user.email)

Gata (@user.adress.street)

Postnummer (@user.adress.postnumber)

Ort (@user.adress.city)

Redigera

#### 4.3.4 Update company information

**Requirement:** User – 2

**Use Case:** Company – Update the company information

### Redigera Exempelföretaget AB

Namn (@company.name)

Företagsform (@company.form)

Organisationsnummer (@company.organisation\_number)

Gata (@company.adress.street)

Postnummer (@company.adress.postnumber)

Ort (@company.adress.city)

Redigera

#### 4.3.5 Create a new user in a company

**Requirement:** Company – 1

**Use Case:** Company – Create a new user in a company

### Ny användare i Exempelföretaget AB

Namn (@user.username)

Telefonnummer (@user.phone\_number)

Email (@user.email)

Gata (@user.adress.street)

Postnummer (@user.adress.postnumber)

Ort (@user.adress.city)

Redigera

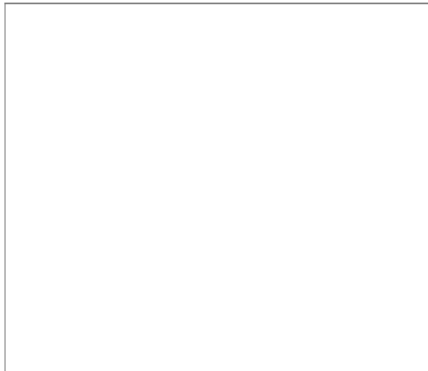
## 4.4 Ask a question

**Requirement:** User – 1, User – 2

**Use Case:** Support – Ask a question

### Skicka fråga till supporten

Fråga (@question.content)



Skicka

### 4.4.1 Create new Fiscal Year

**Requirement:** Fiscal Years 1

**Use cases:** Fiscal Years 1

## Nytt bokföringsår

År

2008

Använd det här bokföringsåret istället för nuvarande?

- Ja, ändra så att det här blir det nuvarande.
- Nej, skapa det bara.

Create

[Back](#)

#### 4.4.2 Change current Fiscal Year

Use cases: Fiscal Years 1

##### Bokföringsår

2008

2007  
Ändra till nuvarande?

2006  
Nuvarande

2005

2004

2003

[New fiscal year](#)

#### 4.4.3 Creating an accounting plan

The interface will look very similar when editing an accounting plan.

**Requirements:** Accounting Plans 1, 2, 5-7

**Use cases:** Duplicating an accounting plan, Create accounting plan, Adding accounts to an accounting plan, Removing accounts from an accounting plan

##### Ny kontoplan

Namn

##### Konton

Kopiera konton från

BAS 2007

##### Konto

- Checkkonto
- Fodringar
- Förbrukningsvaror
- Maskiner
- Moms 25%
- Moms 12%

Lägg till konto

[Back](#)

#### 4.4.4 Create a new Voucher

**Requirements:** Vouchers 1, 5, 6.

**Use cases:** Create a new voucher, Edit a saved voucher, Create a new voucher row, Edit a saved voucher row, Remove a saved voucher row.




### Nytt verifikat

Datum


2008 February 4 - 16 : 34

Beskrivning

### Verifikatsrader

Konto	Debet	Kredit
 Checkkonto	125 kr	
 Förbrukningsmaterial		100 kr
 Moms 25%		25 kr

Moms 25%  kr  kr

 Lägg till verifikatsrad

[Back](#)

## 5 Design Details

### 5.1 Class Responsibility Collaborator (CRC) Cards

#### 5.1.1 Models

<b>User</b>	
Responsibilities	Collaborators
Represent a user in the system.  Knows the required information for a user to login into the system. Knows information about the user used across the system.	Associate

<b>Associate</b>	
Responsibilities	Collaborators
Represent the relation between a user and a company in the system.  Knows of which kind the relation is.	User Company

<b>Company</b>	
Responsibilities	Collaborators
Represent a company in the system.  Knows information about the company used across the system.	Associate FiscalYear

<b>Voucher</b>	
Responsibilities	Collaborators
<p>Representing a collection of voucher rows which together forms a voucher.</p> <p>Knows when it was created.</p> <p>Can be commented.</p> <p>Knows which fiscal year (therefor even which company) it belongs to.</p>	<p>VoucherRow</p> <p>FiscalYear</p>

<b>VoucherRow</b>	
Responsibilities	Collaborators
<p>Knows which voucher it belongs to.</p> <p>Knows when it was created.</p> <p>Knows which account that has been used.</p> <p>Knows amount.</p>	<p>Voucher</p> <p>Account</p>

<b>FiscalYear</b>	
Responsibilities	Collaborators
<p>Knows which accounts it has.</p> <p>Knows which company it belongs to.</p> <p>Knows which vouchers it has.</p> <p>Knows which accounting plan template was used, if there is one.</p>	<p>Voucher</p> <p>Account</p> <p>AccountingPlanTemplate</p> <p>Company</p>

<b>Account</b>	
Responsibilities	Collaborators
<p>Represents a bookkeeping account.</p> <p>Knows name</p> <p>Knows description</p> <p>Knows number</p> <p>Knows parent accounting plan</p> <p>Knows voucher rows</p>	<p>AccountingPlan</p> <p>VoucherRow</p>



<b>AccountingPlanTemplate</b>	
Responsibilities	Collaborators
Represents a template collection of accounts that can be used as a base for an AccountingPlan.  Knows name Knows description Knows number Knows parent accounting plan Knows accounting plans that implements this template	AccountingPlan AccountTemplate

<b>AccountTemplate</b>	
Responsibilities	Collaborators
Represents a bookkeeping account template.  Knows name Knows description Knows number Knows parent accounting plan template	AccountingPlanTemplate

### 5.1.2 Presentation

<b>UserController</b>	
Responsibilities	Collaborators
Represents the MVC's controller layer for working with a user in the system.	User Associate View

<b>CompanyController</b>	
Responsibilities	Collaborators
Represents the MVC's controller layer for working with a company in the system.	Company Associate View

<b>VouchersController</b>	
Responsibilities	Collaborators
Representing the MVC's controller layer for working with vouchers in the system.	Voucher VoucherRow

<b>VoucherRowsController</b>	
Responsibilities	Collaborators
Representing the MVC's controller layer for working with voucher rows in the system.	VoucherRow Account

<b>FiscalYearController</b>	
Responsibilities	Collaborators
Representing the MVC's controller layer for working with fiscal years in the system.	FiscalYear Company User

<b>AccountingPlanTemplateController</b>	
Responsibilities	Collaborators
Representing the MVC's controller layer for working accounting plan templates in the system.	AccountPlanTemplate AccountTemplate

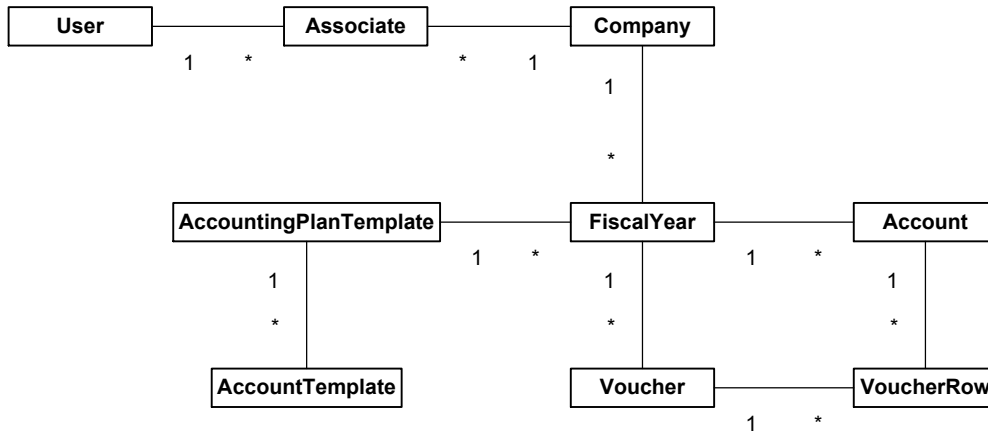
<b>AccountTemplateController</b>	
Responsibilities	Collaborators
Representing the MVC's controller layer for working account templates in the system.	AccountPlanTemplate AccountTemplate

<b>SupportController</b>	
Responsibilities	Collaborators
Responsible for sending user support question to the staff support mailbox.	User

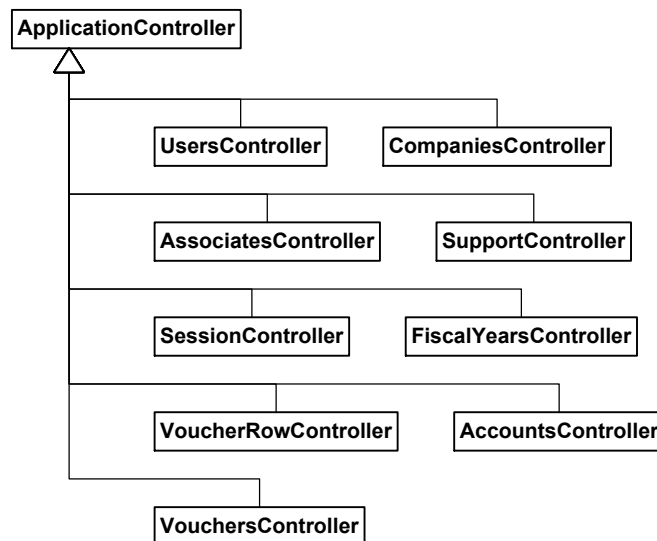
## 5.2 Class Diagram

The below seen class diagrams have been constructed using the Standard Skylight Model Entity Relations Method (SSMERM).

### 5.2.1 Models



### 5.2.2 Controllers



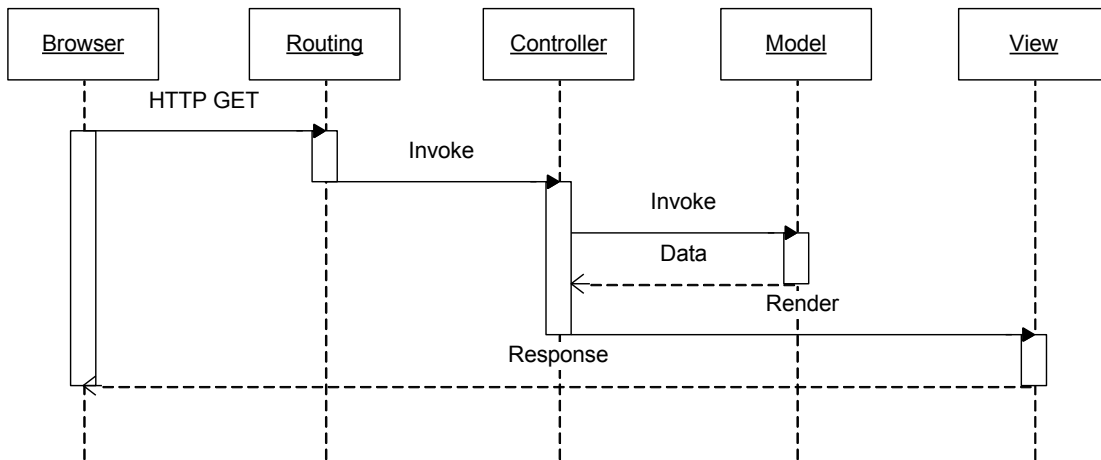
## 5.3 State Charts

Considering whether to use state charts or not to, we decided on the latter alternative. While state charts may generally be an efficient tool for depicting complex relationships and flows between (or within) different actions along with their underlying processes, they couldn't contribute with much to this specific design document. This is since the previous chapters have already described how AccountThis! is intended to function and how it is meant to be used.

## 5.4 Interaction Diagrams

### 5.4.1 General HTTP GET Request Sequence

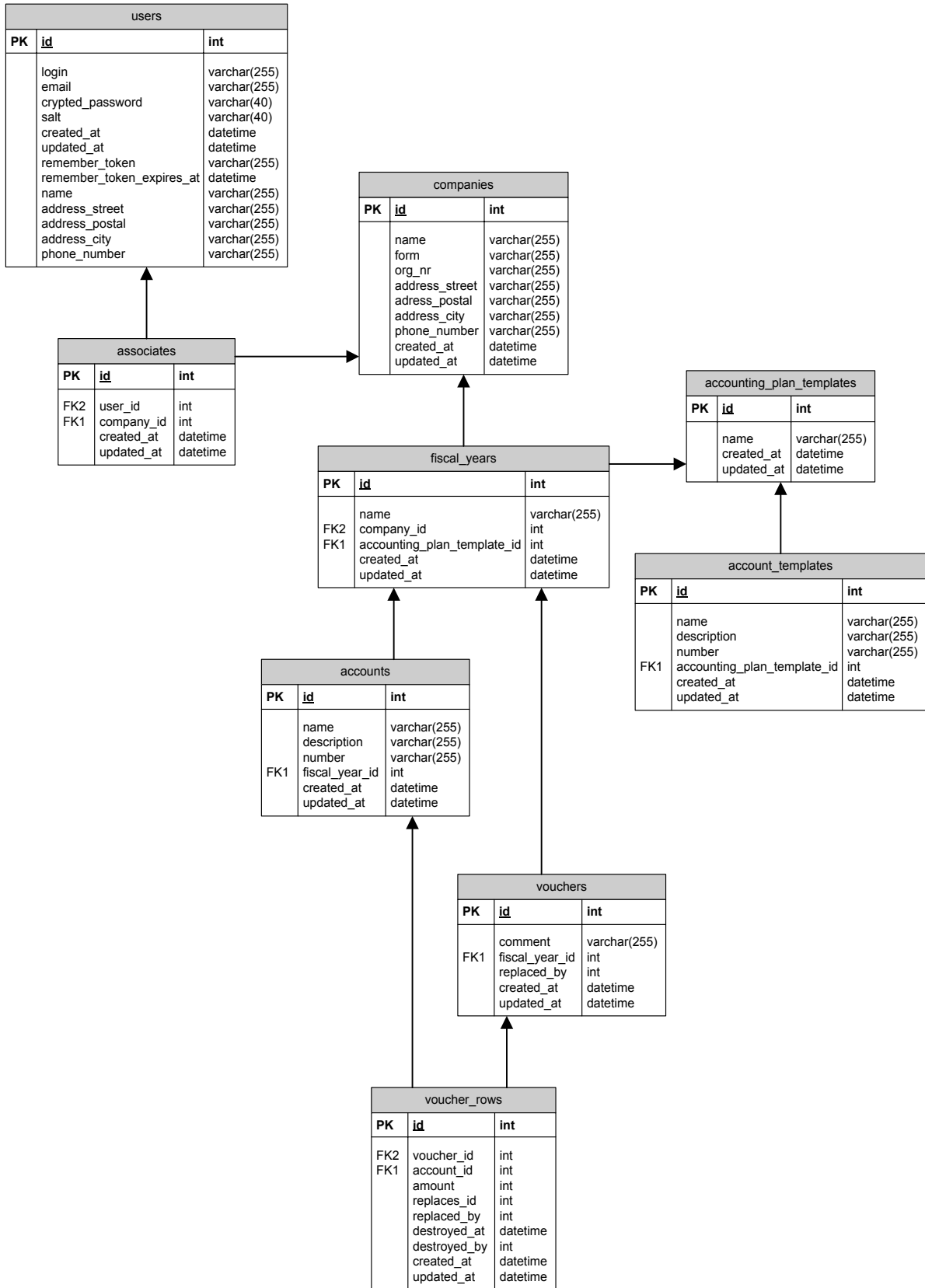
Sequence of a typical HTTP GET request.



## 5.5 Detailed Design

Below are detailed design descriptions for each class, method and accessor (for information concerning accessors, please see: <http://en.wikipedia.org/wiki/Accessor>) that is to be implemented in the final version of AccountThis!. This subchapter is based on the requirements document along with the overall design information and will serve as a general template (or a backbone) to which the actual program code can be compared and evaluated. By analyzing the detailed design, one should easily be able grasp what need to be done to build the system. This provides an overview of the system, granting us – as developers – better odds of discovering mistakes and logical errors before they appear in the fully implemented code. Discovering errors in an early phase of development is deemed to be less costly than having errors appear in the near complete version.

## 5.5.1 Database



## 5.5.2 Models

### 5.5.2.1 User

#### 5.5.2.1.1 Accessors

Name	Type	Description
id	Fixnum	The user's internal identification.
name	String	The user's full name.
login	String	The user's login name.
address_street	String	The user's street address.
address_postal	String	The user's postal address.
address_city	String	The user's city.
email	String	The user's email.
phone_number	String	The user's phone number.
associates	Array	The associates that this user has.
companies	Array	The companies that this user belongs to.
updated_at	Time	User updated time.
created_at	Time	User creation time.

#### 5.5.2.1.2 Methods

<b>self.authenticate</b>	
Authenticates the user with their login name and unencrypted password.	
Parameters	login, password
Return Value	Returns the User with the provided login if login and password matches, otherwise nil.
Pre-condition	
Post-condition	

<b>self.encrypt</b>	
Encrypt the password with the given salt (for information concerning salt, please see: <a href="http://en.wikipedia.org/wiki/Salt_(cryptography)">http://en.wikipedia.org/wiki/Salt_(cryptography)</a> ).	
Parameters	password, salt
Return Value	Returns the given password in an encrypted form.
Pre-condition	N/A (not applicable)

Post-condition	That the password was encrypted with an SHA1 hex digest algorithm using the given salt.
----------------	---

<b>encrypt</b>	
Encrypt the password with the user's salt (for information concerning salt, please see: <a href="http://en.wikipedia.org/wiki/Salt_(cryptography)">http://en.wikipedia.org/wiki/Salt_(cryptography)</a> ).	
Parameters	password
Return Value	Returns the given password in an encrypted form.
Pre-condition	N/A
Post-condition	That the password was encrypted with an SHA1 hex digest algorithm using the user's salt.

<b>authenticated?</b>	
Checks if the user is authenticated with the provided password.	
Parameters	password
Return Value	Returns true if the user is authenticated otherwise false.
Pre-condition	N/A
Post-condition	That the password was encrypted with an SHA1 hex digest algorithm using the user's salt.

<b>encrypt_password</b>	
Before the use is saved the password will be encrypted so the saved password will be in an encrypted form.	
Parameters	
Return Value	Returns nil.
Pre-condition	N/A
Post-condition	That the password is encrypted and stored in the database.

<b>password_required?</b>	
Checks whatever a password is required for the current user.	
Parameters	
Return Value	Returns true if no password is required for the current user, otherwise false.

Pre-condition	N/A
Post-condition	N/A

### 5.5.2.2 Associate

#### 5.5.2.2.1 Accessors

Name	Type	Description
id	Integer	Internal identification.
user	User	The user that the associate has.
company	Company	The company that the associate has.
updated_at	Time	The associate's updated time.
created_at	Time	The associate's creation time.

### 5.5.2.3 Company

#### 5.5.2.3.1 Accessors

Name	Type	Description
id	Integer	Internal identification.
name	String	The company's name.
form	String	The company's form.
associates	Array	The associates this company has.
users	Array	The users that belongs to this company.
org_nr	String	The company's organizational number.
address_street	String	The company's street address.
address_postal	String	The company's postal address.
address_city	String	The company's city.
phone_number	String	The company's phone number.
updated_at	Time	Company updated time.
created_at	Time	Company creation time.

### 5.5.2.4 FiscalYear

#### 5.5.2.4.1 Accessors

Name	Type	Description
id	Fixnum	Internal identification of the fiscal year



company	Company	The company that the fiscal year belongs to.
voucher_rows	Array	The vouchers associated with the fiscal year.
created_at	Time	The date and time when the fiscal year was created.
accounting_plan_template	AccountingPlanTemplate	The accounting plan template that was initially used to create the accounts that belongs to the fiscal year.
accounts	Array	The accounts associated with the fiscal year.
name	String	The name of the fiscal year (i.e. "2007")

### 5.5.2.5 Account

#### 5.5.2.5.1 Accessors

Name	Type	Description
id	Fixnum	Internal identification of the account.
name	String	Name of the account.
description	String	Description of the account.
number	String	The account number.
created_at	Time	Date and time when the instance was created.
updated_at	Time	Date and time when the instance was updated.
fiscal_year	FiscalYear	The accounting plan template in which the account template is created.

### 5.5.2.6 Voucher

#### 5.5.2.6.1 Accessors

Name	Type	Description
id	Fixnum	Internal identification of the voucher.
voucher_rows	Array	The voucher rows that belongs to a voucher.
fiscal_year	FiscalYear	The fiscal year that the voucher belongs to.
created_at	Time	The date and time when the voucher was created.
comment	String	An optional comment to the voucher.
replaces	Voucher	The voucher that this voucher replaces (if any).
replaced_by	Voucher	The voucher that replaces this voucher (if any).

### 5.5.2.7 VoucherRow

#### 5.5.2.7.1 Accessors

Name	Type	Description
------	------	-------------

id	Fixnum	The id.
voucher	Voucher	The voucher that the voucher row belongs to.
account	Account	The account that this voucher row refers to.
created_at	Time	The date and time when the voucher row was created.
amount	Fixnum	The amount specified by a voucher row, can be negative.
replaces	VoucherRow	The voucher row that this voucher row replaces (if any).
replaced_by	VoucherRow	The voucher row that replaces this voucher row (if any).
destroyed_at	Time	The time and date when the voucher row was destroyed, if it is destroyed.
destroyed_by	User	The user that destroyed the voucher row, if it is destroyed.

### 5.5.2.8 AccountingPlanTemplate

#### 5.5.2.8.1 Accessors

Name	Type	Description
id	Fixnum	Internal identification of accounting plan templates.
name	String	Name of the accounting plan.
accounts	Array	Set of accounts for the accounting plan.
created_at	Time	Date and time when the instance was created.
updated_at	Time	Date and time when the instance was updated.

#### 5.5.2.8.2 Methods

use_in_fiscal_year	
Export accounts from the accounting plan to a fiscal year and sets the fiscal year's account template accessor.	
Parameters	fiscal_year
Return Value	N/A
Pre-condition	Parameter fiscal_year cannot be null.
Post-condition	All or none accounts have been copied to the fiscal year.

### 5.5.2.9 AccountTemplate

#### 5.5.2.9.1 Accessors

Name	Type	Description
id	Fixnum	Internal identification of the account template.
name	String	Name of the account template.
description	String	Description of the account template.
number	String	The account number.

created_at	Time	Date and time when the instance was created.
updated_at	Time	Date and time when the instance was updated.
accounting_plan_template	AccountingPlanTemplate	The accounting plan template in which the account template is created.

### 5.5.3 Controllers

#### 5.5.3.1 UsersController

The controller is reached by /companies/:company\_id/users/

##### 5.5.3.1.1 Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of users for the selected company as rows in HTML.
Preconditions	A user must be logged in.
Postconditions	

<b>new</b>	
The action for rendering a form for a new user in the selected company.	
Parameters	
Return value	A form as HTML for a new user in the selected company.
Preconditions	A user must be logged in.
Postconditions	

<b>edit</b>	
The action for rendering a form to update a user.	
Parameters	
Return value	A form as HTML for updating a user.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>create</b>
---------------

The action for saving a new user.	
Parameters	
Return value	A redirect to the index action or the form for creating a new user with error messages if errors occurred.
Preconditions	A user must be logged in. Post parameters from the new action must be present.
Postconditions	A new fiscal year is created or an error message is returned.

<b>update</b>	
The action for saving an updated user.	
Parameters	
Return value	A redirect to the index action or the form for updating the user with error messages if errors occurred.
Preconditions	A user must be logged in. Post parameters from the edit action must be present.
Postconditions	The user is updated.

<b>destroy</b>	
The action for destroying a user.	
Parameters	
Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in.
Postconditions	The user is destroyed or an error message is returned.

<b>show</b>	
The action for rendering a user.	
Parameters	
Return value	HTML for a user.
Preconditions	A user must be logged in.
Postconditions	

### 5.5.3.2 *SessionController*

The controller is reached by /session/

### 5.5.3.2.1 Methods

<b>create</b>	
The action for logging a user into the system.	
Parameters	The username, the password and a URL address.
Return value	The user will be redirected, and given authorization, to another webpage in case the username and password matches. If it doesn't, he or she will automatically be brought back to the original webpage, where a message stating that <i>the entered username or password was incorrect</i> is now to be shown.
Preconditions	That the user isn't logged in.
Postconditions	

<b>destroy</b>	
The action for logging out a user from the system.	
Parameters	
Return value	A redirect to the start page.
Preconditions	That the user is logged in.
Postconditions	

### 5.5.3.3 *AssociatiesController*

The controller is reached by `/users/:user_id/associates` and `/companies/:company_id/associates`

#### Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of associates rows as HTML.
Preconditions	A user must be logged in.
Postconditions	

<b>edit</b>	
The action for editing an associate.	
Parameters	
Return value	A form as HTML for updating associates.

Preconditions	A user must be logged in.
Postconditions	

<b>update</b>	
The action for saving an updated associate.	
Parameters	
Return value	A redirect to the index action or the form for updating the associate with error messages if errors occurred.
Preconditions	A user must be logged in. Post parameters from the edit action must be present.
Postconditions	The associate is updated.

<b>destroy</b>	
The action for destroying a associate.	
Parameters	
Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in.
Postconditions	The associate is destroyed or an error message is returned.

### 5.5.3.4 *CompaniesController*

The controller is reached by /companies/.

#### 5.5.3.4.1 Methods

<b>new</b>	
The action for rendering a form for a new company and the initial user.	
Parameters	
Return value	A form as HTML for a new company and the initial user.
Preconditions	
Postconditions	

<b>edit</b>	
The action for rendering a form to update a company.	
Parameters	

Return value	A form as HTML for updating a fiscal year.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>create</b>	
The action for saving a new company.	
Parameters	
Return value	A redirect to the index action or the form for creating a company with error messages if errors occurred.
Preconditions	A user must be logged in. Post parameters from the new action must be present.
Postconditions	A new company is created or an error message is returned.

<b>update</b>	
The action for saving an updated company.	
Parameters	
Return value	A redirect to the index action or the form for updating the company with error messages if errors occurred.
Preconditions	A user must be logged in. Post parameters from the edit action must be present.
Postconditions	The company is updated.

<b>show</b>	
The action for rendering a company.	
Parameters	
Return value	HTML for a company.
Preconditions	A user must be logged in.
Postconditions	

### 5.5.3.5 *SupportController*

The controller is reached from /support/.

#### 5.5.3.5.1 Methods

<b>index</b>
--------------

The default action for this controller.	
Parameters	
Return value	A HTML form for a new support request.
Preconditions	
Postconditions	

<b>new</b>	
The action for rendering a form for a new support request.	
Parameters	
Return value	A HTML-form for a new support request.
Preconditions	
Postconditions	

<b>submit</b>	
The action for submitting a support request through e-mail to the support staff.	
Parameters	
Return value	A confirmation of the submitted support request.
Preconditions	Post parameters from the new action must be present.
Postconditions	

### 5.5.3.6 *FiscalYearsController*

The controller is reached by `/fiscal_years/`

#### 5.5.3.6.1 Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of voucher rows as HTML.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>new</b>	
The action for rendering a form for a new fiscal year.	
Parameters	



Return value	A form as HTML for a new fiscal year.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>edit</b>	
The action for rendering a form to update a fiscal year.	
Parameters	
Return value	A form as HTML for updating a fiscal year.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>create</b>	
The action for saving a new fiscal year.	
Parameters	
Return value	A redirect to the index action or the form for creating a new fiscal year with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year must have been chosen. Post parameters from the new action must be present.
Postconditions	A new fiscal year is created or an error message is returned.

<b>update</b>	
The action for saving an updated fiscal year.	
Parameters	
Return value	A redirect to the index action or the form for updating the fiscal year with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year must have been chosen. Post parameters from the edit action must be present.
Postconditions	The fiscal year is updated.

<b>destroy</b>	
The action for destroying a fiscal year.	
Parameters	

Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	The fiscal year is destroyed or an error message is returned.

<b>show</b>	
The action for rendering a fiscal year.	
Parameters	
Return value	HTML for a fiscal year.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

### 5.5.3.7 AccountsController

The controller is reached by `/fiscal_years/:fiscal_year_id/accounts/`.

#### 5.5.3.7.1 Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of accounts as HTML.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>new</b>	
The action for rendering a form for a new account.	
Parameters	
Return value	A HTML-form for a new account.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>edit</b>	
The action for rendering a form to update an account.	
Parameters	
Return value	A form as HTML for updating an account.

Preconditions	A user must be logged in and a fiscal year and an account must have been chosen.
Postconditions	

<b>create</b>	
The action for saving a new account.	
Parameters	
Return value	A redirect to the index action or the form for creating a new account with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year must have been chosen. Post parameters from the new action must be present.
Postconditions	A new account is created or error messages are returned.

<b>update</b>	
The action for saving an updated account.	
Parameters	
Return value	A redirect to the index action or the form for updating the account with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year and an account must have been chosen. Post parameters from the edit action must be present.
Postconditions	The account is updated.

<b>destroy</b>	
The action for destroying an account.	
Parameters	
Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in and a fiscal year and an account must have been chosen. The account cannot be referred to from any voucher row.
Postconditions	The selected account is destroyed or the user has been notified of the failure of the operation.

<b>show</b>	
The action for rendering displaying an account	

Parameters	
Return value	HTML for an account.
Preconditions	A user must be logged in and a fiscal year and an account must have been chosen.
Postconditions	

### 5.5.3.8 *VouchersController*

The controller is reached by `/fiscal_years/:fiscal_year_id/vouchers/`.

#### 5.5.3.8.1 Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of vouchers as HTML.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>new</b>	
The action for rendering a form for a new voucher.	
Parameters	
Return value	A form as HTML for a new voucher.
Preconditions	A user must be logged in and a fiscal year must have been chosen.
Postconditions	

<b>edit</b>	
The action for rendering a form to update a voucher.	
Parameters	
Return value	A form as HTML for updating a voucher.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen.
Postconditions	

<b>create</b>	
The action for saving a new voucher.	

Parameters	
Return value	A redirect to the index action or the form for creating a new voucher with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year must have been chosen. Post parameters from the new action must be present.
Postconditions	A new voucher is created or an error message is returned.

<b>update</b>	
The action for saving an updated voucher.	
Parameters	
Return value	A redirect to the index action or the form for updating the voucher with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen. Post parameters from the edit action must be present.
Postconditions	A new voucher is created which overrides the old one.

<b>destroy</b>	
The action for destroying a voucher.	
Parameters	
Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen.
Postconditions	A new voucher is created which overrides the old one. All voucher rows are marked as destroyed.

<b>show</b>	
The action for rendering a voucher.	
Parameters	
Return value	HTML for a voucher.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen.
Postconditions	

### 5.5.3.9 *VoucherRowsController*

The controller is reached by `/fiscal_years/:fiscal_year_id/vouchers/:voucher_id/voucher_rows/`

### 5.5.3.9.1 Methods

<b>index</b>	
The default action for this controller.	
Parameters	
Return value	A list of voucher rows as HTML.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen.
Postconditions	

<b>new</b>	
The action for rendering a form for a new voucher row.	
Parameters	
Return value	A form as HTML for a new voucher row.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen.
Postconditions	

<b>edit</b>	
The action for rendering a form to update a voucher row.	
Parameters	
Return value	A form as HTML for updating a voucher row.
Preconditions	A user must be logged in and a fiscal year, a voucher and a voucher row must have been chosen.
Postconditions	

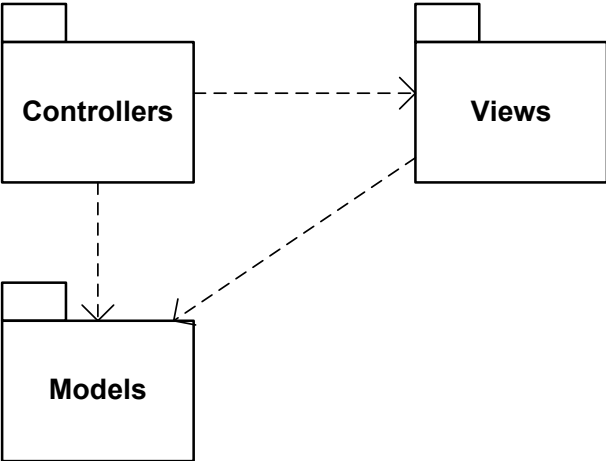
<b>create</b>	
The action for saving a new voucher row.	
Parameters	
Return value	A redirect to the index action or the form for creating a new voucher row with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year and a voucher must have been chosen. Post parameters from the new action must be present.
Postconditions	A new voucher row is created or error messages is returned.

<b>update</b>	
The action for saving an updated voucher row.	
Parameters	
Return value	A redirect to the index action or the form for updating the voucher row with error messages if errors occurred.
Preconditions	A user must be logged in and a fiscal year, a voucher and a voucher row must have been chosen. Post parameters from the edit action must be present.
Postconditions	A new voucher row is created which overrides the old one.

<b>destroy</b>	
The action for destroying a voucher row.	
Parameters	
Return value	A redirect to the index action with or without error messages.
Preconditions	A user must be logged in and a fiscal year, a voucher and a voucher row must have been chosen.
Postconditions	The voucher row is marked as destroyed or an error message is returned.

<b>show</b>	
The action for rendering a voucher row.	
Parameters	
Return value	HTML for a voucher row.
Preconditions	A user must be logged in and a fiscal year, a voucher and a voucher row must have been chosen.
Postconditions	

5.6 Package Diagram







## Functional Test Cases

#	Reference	Test	Expected Result	Pass	Fail	Comment
1.1	RD Users 1	<ol style="list-style-type: none"> <li>1. Browse to the Register page.</li> <li>2. Fill out the form with all the required information.</li> <li>3. Press the create button.</li> </ol>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The login screen is shown.</li> <li>• The user is able to login to the system using his new username and password.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
1.2	RD Users 2	<ol style="list-style-type: none"> <li>1. Browse to the login page.</li> <li>2. Fill out the form with your username and password.</li> <li>3. Press of the quicklinks or just press login.</li> </ol>	<ul style="list-style-type: none"> <li>• The user is logged in.</li> <li>• On 3, if a quicklink is pressed the user is forwarded to the selected function in an logged in state.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
1.3	RD Users 3	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Click the logout link.</li> </ol>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The user is presented with the startpage again.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
1.4	RD Users 4	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Navigate to the page where your information is shown.</li> <li>3. Click the link where it says that you can edit your information.</li> <li>4. Edit your information after your taste.</li> <li>5. Click the save button.</li> </ol>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The users updated information shows at the users page.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	

#	Reference	Test	Expected Result	Pass	Fail	Comment
2.1	RD Company 1	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Navigate to the page where you can add new users to your company.</li> <li>3. Fill out the with the email of the new user.</li> <li>4. Click the create button.</li> <li>5. The new user receives an email with an link.</li> <li>6. The user clicks the link and is presented with a form for all the additional information for the new user.</li> <li>7. The new user clicks create.</li> </ol>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The user should have received an email with the register link</li> <li>• When the new user has filled out the form and clicked create he should be able to login into the system.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
2.2	RD Company 2	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Navigate to the page where your company's information is shown.</li> <li>3. Click the link where it says that you can edit your information.</li> <li>4. Edit your information after your taste.</li> <li>5. Click the save button.</li> </ol>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The company's updated information shows at the company page.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
3.1	RD Fiscal years 1	<ul style="list-style-type: none"> <li>-Login to the system.</li> <li>-Navigate to the Fiscal years page.</li> <li>-Click start new fiscal year.</li> </ul>	<ul style="list-style-type: none"> <li>• A confirm question appears which asks you if you are certain.</li> <li>• A success notification is shown.</li> <li>• The fiscal years page appears.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	

#	Reference	Test	Expected Result	Pass	Fail	Comment
4.1	RD Vouchers 1,5	<ul style="list-style-type: none"> <li>-Login to the system.</li> <li>-Navigate to the vouchers page.</li> <li>-Click create new voucher.</li> <li>-Set required fields.</li> <li>-Click add new voucher row.</li> <li>-Fill required fields.</li> <li>-Repeat 5-6.</li> <li>-Click delete next to one of the voucher rows.</li> <li>-Change the filled fields of the other voucher row.</li> <li>-Click save</li> </ul>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The voucher appears in the list.</li> <li>• The voucher has the voucher rows that where supposed to be created.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
4.2	RD Vouchers 2,6	<ul style="list-style-type: none"> <li>-Login to the system.</li> <li>-Navigate to the vouchers page.</li> <li>-Make sure that there is at least one voucher with at least 2 voucher rows.</li> <li>-Click edit next to a voucher.</li> <li>-Alter values in the fields.</li> <li>-Click delete next to a voucher row.</li> <li>-Click new voucher row.</li> <li>-Fill out the required fields.</li> <li>-Click save.</li> </ul>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The voucher appears in the list.</li> <li>• The voucher has the voucher rows that where supposed to be created.</li> <li>• The voucher has the voucher rows that where supposed to be created.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
4.3	RD Vouchers 3	<ul style="list-style-type: none"> <li>-Login to the system.</li> <li>-Navigate to the vouchers page.</li> <li>-Make sure that at least 1 voucher appears in the list.</li> <li>-Click replace next to a voucher.</li> <li>-Preform test 4.1.4-1.1.10</li> </ul>	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The voucher appears in the list.</li> <li>• The voucher has the voucher rows that where supposed to be created.</li> <li>• The old voucher that you clicked replace next to is marked as deleted and marked as replaced by the new one.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	

#	Reference	Test	Expected Result	Pass	Fail	Comment
4.4	RD Vouchers 4	-Login to the system. -Navigate to the vouchers page. -Make sure that at least 1 voucher appears in the list. -Click delete next to a voucher.	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The voucher appears in the list, but is marked as removed.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
5.1	RD Support 1	1. Enter the send questions page. 2. Input a textual query into the text field. 3. Press the send button.	<ul style="list-style-type: none"> <li>• A success notification is shown.</li> <li>• The query is stored and made available to the support staff.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
5.2	RD Support 2	1. Enter the send questions page. 2. Input a textual query into the text field. 3. Press the send button.	<ul style="list-style-type: none"> <li>• A reference containing time and user information shall be attached to the query.</li> <li>• The support staff must be able to find information on whom the question was sent from, and when it was sent.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
6.1	RD Accounting plans 1	1. Login to the system. 2. Browse to the page handling accounting plans. 3. Produce an accounting plan using the options shown. 4. Press the create button.	<ul style="list-style-type: none"> <li>• The new accounting plan shall be stored in the system.</li> <li>• The new accounting plan shall be made accessible and ready for use by the user/firm that produced the plan.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
6.2	RD Accounting plans 2	1. Login to the system. 2. Browse to the page handling accounting plans. 3. Use the option showing existing accounting plans. 4. Pick and edit one of the shown accounting plans. 5. Press the create button.	<ul style="list-style-type: none"> <li>• On 3. All plans that have been created by the user or are predefined by the system itself shall be shown in the list.</li> <li>• On 4. Picking an existing accounting plan shall alter the accounting plan creation options so that they correlate with the chosen design.</li> <li>• On 5. The new accounting plan shall be stored in the system.</li> <li>• On 5. The new accounting plan shall be made accessible and ready for use by the user/firm that produced the plan.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	

#	Reference	Test	Expected Result	Pass	Fail	Comment
6.3	RD Accounting plans 3	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Browse to the page handling accounting plans.</li> <li>3. Use the option showing existing accounting plans.</li> </ol>	<ul style="list-style-type: none"> <li>• All plans that have been created by the user or are predefined by the system itself shall be shown in the list.</li> <li>• No accounting plans created by other firms shall be shown.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
6.4	RD Accounting plans 4	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Browse to the page handling accounting plans.</li> <li>3. Use the option showing existing accounting plans.</li> <li>4. Pick a plan.</li> <li>5. Press the remove button.</li> </ol>	<ul style="list-style-type: none"> <li>• In case the accounting plan is not being used in a fiscal year associated with the user, the accounting plan shall: (1) be removed from the system; (2) no longer show in the user's list of existing accounting plans.</li> <li>• In case the accounting plan is being used in a fiscal year associated with the user, a message saying so shall appear. Nothing will be removed if this is the case.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
6.5	RD Accounting plans 5	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Browse to the page handling the currently used accounting plans.</li> <li>3. Pick an account.</li> <li>4. Press the add button to add the account to the accounting plan.</li> <li>5. Save or create the accounting plan.</li> </ol>	<ul style="list-style-type: none"> <li>• The just added account shall show in the accounting plans list of active accounts.</li> <li>• The added account shall, when the change is saved (or the accounting plan is created), be shown among the other accounts belonging to the accounting plan.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	
6.6	RD Accounting plans 6	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Browse to the page handling the currently used accounting plans.</li> <li>3. Pick an account.</li> <li>4. Press the edit button to edit the account to the accounting plan.</li> <li>5. Edit the account.</li> <li>6. Add the edited account.</li> <li>7. Save or create the accounting plan.</li> </ol>	<ul style="list-style-type: none"> <li>• On 4. Customization menus shall appear when the edit button is pressed.</li> <li>• On 5. It shall be possible to edit the specifics of the account by altering the information that's shown in the now activated edit menus.</li> <li>• On 6. The just added account shall show in the accounting plans list of active accounts.</li> <li>• On 7. The added account shall, when the change is saved (or the accounting plan is created), be shown among the other accounts belonging to the accounting plan.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	

#	Reference	Test	Expected Result	Pass	Fail	Comment
6.7	RD Accounting plans 7	<ol style="list-style-type: none"> <li>1. Login to the system.</li> <li>2. Browse to the page handling the currently used accounting plans.</li> <li>3. Pick an account.</li> <li>4. Press the remove button to remove the account from the accounting plan.</li> <li>5. Save or create the accounting plan.</li> </ol>	<ul style="list-style-type: none"> <li>• The added account shall, when the change is saved (or the accounting plan is created), no longer be shown among the other accounts belonging to the accounting plan.</li> </ul>	<input type="checkbox"/>	<input type="checkbox"/>	