

# Skyline™ - A blog community on the web

## Skyline (group 3)

Anton Palmqvist (198809064010)

Git name: epoxy

Gabriel Andersson (199110104131)

Git name: krabban91

Mike Phoohad (198908065637)

Git name: annoa

Tomas Selldén (199003072890)

Git name: tomsel

## 1. General overview

### 1.1. Introduction

Skyline is a free community for bloggers. Here one can post whatever experience he/she wants to share. Any member can read, comment and vote on any posts.

Skyline allows members to favourite other members to have a customized Favourite-wall of posts when they only want to read from specific users.

The whole idea with this community is to let the blogger feel that he/she has a more intimate audience (other members) and still have the possibility to let outsiders read his/her posts as well as have content related to his/her interests by allowing everyone to vote posts so that they get more or less exposure.

Skyline

Wall

Profile

Favorites

Search...

Log out

About

Add post

A cool video I found

Krabban

Posted on: 2013-10-23 00:00

This video is very funny!

Java Forever And Ever Movie (JAVA vs

Scala Johansson

0:00 / 3:14

YouTube

Show comments

Reply


4

A nice little text

Anno

Posted on: 2013-10-23 00:00

This is a text I wrote when I was travelling in Australia.



Show comments

Reply

2

1.1.1 Screenshot of Skyline

## 1.2. Roles and use cases

### 1.2.1 Roles

#### **Common user**

Someone who hasn't logged onto it's account in the community.

#### **Skyline member**

Someone that has logged onto it's account in the community. Possesses all rights that the Common user has plus the rights to add information to the community.

### 1.2.2 Use cases

See [3.4 Use cases](#) for more an elaborate description.

Use cases that are specific for the Skyline member role are

UC2: Log in

UC3: Log out

UC7: Favorite member

UC8: Unfavorite member

UC10: View posts written by the members you have favorited

UC11: Write post

UC12: Vote on post

UC13: Write comment

UC15: Vote on Comment

Use cases that are possible to perform as any kind of user:

UC1: Become a member

UC4: View Wall-feed

UC5: Read post

UC6: Read profile

UC9: Read post

UC14: Read comment

UC16: Search for user

UC17: Try to comment when logged out

UC18: Try to log in with wrong password

UC19: Try to become member with already existing name

UC20: Read the About-section

## 2. Technical design of system

### 2.1. Physical setup

#### Client

Uses a web browser that supports HTML5, CSS3 and Javascript.

#### Server

Is written using Java with Java EE 7.3 and Maven.

#### Datastorage

The ORM we use is JPA, and the database is created using Derby (Java DB).

As a physical setup between many users on different computers each user will have a unique member id. The users will get access to the data of the other users through the database.

See [3.1 Layers & Tiers](#) to get a better visual of the project.

### 2.2. Modules & technologies of components

#### 2.2.1. Service based

We started off by getting our RESTful service from the server using JAX-RS in Java EE 7 to work in the blog. Data of members, posts and comments were possible to be inserted in HTML (the GUI), manipulated via javascript with help of JQuery and put into proxies (as a part of the controller). The data would then be transferred using AJAX as JSON and put into the REST resource (being the model part). From there the data would be transferred, by middleware, using the ORM into the Java Derby database. These steps would be in the opposite way when getting data from the database and rendering it in the GUI. We have support for JAXB so it's possible to test resource classes by curl and also in Netbeans with marshal objects.

#### 2.2.2. Request based

The authentication of the app is done with the request based component. A log in-servlet is triggered by an url and passing the given username and password as request attributes to see if they matches and exists in the database. When logged in a session attribute of the logged in member's is created. At logout this id is set to null and therefore it is being checked not to be null every time the user wants to do an operation that requires log in, as for example putting up a post on the wall. To do that we use a filter that will be invoked when we do a HTTP POST-method when adding posts and comments.

### 2.2.3. Component based

All applications normally have an about-section, more or less extensive. So we thought, why shouldn't we? But there is no point putting energy on something that is mostly text and almost no functionality. Therefore the "About-section" has been implemented in a component based approach. By using JSFs and EJB (Enterprise Java Beans) with different CDI annotations this section is easy to expand with new modules and also be easy to navigate through. At the moment there aren't any manuals about the application but there is a module about the contributors ready to fill with information.

### 2.2.4. Communication between components

Generally the components only get contact with each other through the server. There are however some exceptions in the filter part. When we invoke a HTTP POST-method from the posts and comments (service based) we check if there are a user logged in.

### 2.2.5. Included packages

JQuery, JQuery UI, JQuery Cookie, Bootstrap, Typeahead

JQuery is used for most of the Javascript. It just makes it a whole lot easier to work with Javascript.

JQuery UI is used to add functionality for easy GUI-editing using JQuery.

JQuery Cookie is used to have easy access to cookies, and is used to remember what the user has customized.

Bootstrap is used to make it easier for us to make a prettier GUI.

Typeahead is used to make the search-field interactive and autocomplete what you type for you.

## 2.3. Layered view of Skyline

The layered view ([3.1 Layers & Tiers](#)) of Skyline has as you can see three layers. The representation lies on both the client and the server. The client side is basically the browser and the current html and script files. The server side of the representation consists the files in the system and the transactions between them.

The business logic layer are the JEBs for the about section and the blog-model for the rest of the application. The model is quite straight forward and is mentioned in the next caption [2.4](#) Object oriented model.

The persistence layer are the persistent entities and the entity resources of the model. These together represent the ORM used to communicate with the database. More about these will be mentioned in [2.5 Application](#).

## 2.4 Object oriented model

Our model ([3.2 Object oriented model](#)) is built in a way that is similar to the shop model in the workshops. We use one registry/container for each kind of entity to persist every instance. As you can see in the persistent model (See [3.3 Persistent model](#)) The entities are in a way aware of each other. In Skyline we have designed this top-down. The member needs to know which posts and comments it has submitted. It also has a list of favorites to know whom to favourite in a less anonymous way. The post needs to know which comments that are made on it. And the comment does not know who the author is, neither is it aware of which post it was made on. Although, a comment can have other comments on itself and as therefor an awareness of it's "children".

## 2.5 Application

The graphical interface is comprised of some parts, but mainly follows the SPA-pattern. The navigation and support libraries (and some functionality) is loaded through **navigation.xhtml** coupled with **navigation.js**. The **index.xhtml** and **home.xhtml** loads navigation and then controls most of the functionality, and loads the correct content. The reason they are separated is that **index.xhtml** (paired with **index.js**) is the window for "not-logged-in" users and **home.xhtml** (paired with **home.js**) is used when we are sure the user is logged in. The navigation in index and in home is a bit different. Home adds some more options for the user through insertions done in **home.js**. The other html-files are mostly used to insert content into div-containers inside index and home. **mainabout.xhtml** is the about-page, where information from different components are included. This file has a EJB called **AboutBean.java** to control the content of the page. Uses only **authors.xhtml** at the moment. Which is the section about the contributors. The page gets information from the previously mentioned bean and uses **ViewAuthorBB.java** to help represent it. **Author.java** and **Authors.java** are only content objects to hold information about the contributors.

The server application is done in Java and is connected to the ORM and Database through **Blog.java**. The three proxy classes for the entities in the ORM are **CommentProxy.java**, **MemberProxy.java** and **PostProxy.java**. The RESTful service that the client interface is connected through is three files as well: **CommentResource.java**, **MemberResource.java** and **PostBoxResource.java**. The files responsible for the log-in functionality is **AuthenticationFilter.java** and **LoginServlet.java**.

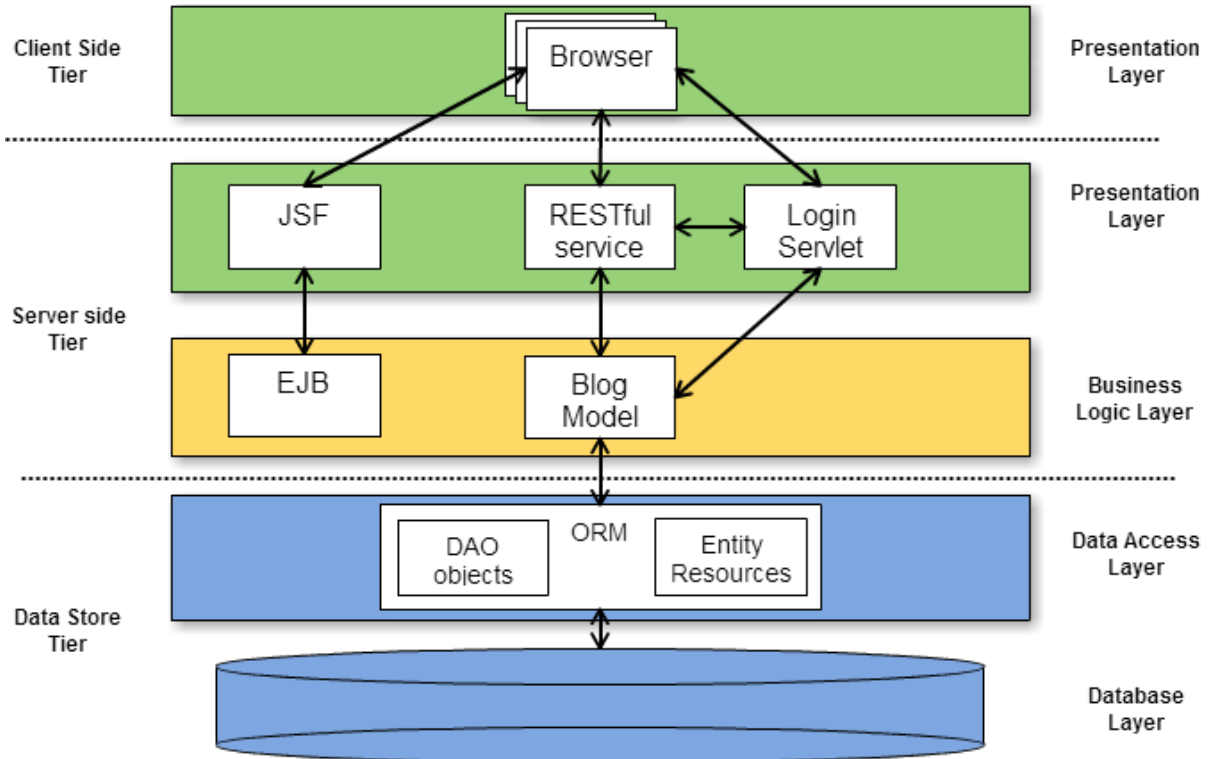
The back-end is comprised of the ORM and the Database. The using of JPA makes it possible to not have to write as much database related stuff, and mostly correct tagging is enough. Although some methods in the DAO-type classes use JPQL and related classes in order to get the data in a specific way, e.g. sorting. The DAO-type classes are **MemberRegistry.java**, **PostContainer.java** and **CommentContainer.java**. The entities that are stored and their relationships can be viewed in the files **Member.java**, **Post.java** and **Comment.java**. We decided to store the votes in a class the posts and comments can embed called **VotingSystem.java**. This is not entirely necessary, however it lets us customize the equals of each entity and have methods that are related to voting without copy-pasting it into both post and comment, since they both use the same voting system. We use the same type of interfacing and abstracting as the course materials did in the workshops: **IDAO.java**, **AbstractDAO.java**, **AbstractEntity.java**.

The MVC-pattern is used in the REST-part as much as possible. All model related javascripts (the ones getting access to the model through AJAX) are collected in the core folder. The javascripts handling the view-part are put in the gui-folder. The controller-folder is covering the controller parts concerning the adding of a new post. However there is an exception from the MVC-pattern since the comment's and the post's listeners are unfortunately set inside **wall.js** which is more or less required, since the listeners are set right after the buttons and links are inserted into the DOM.

Concerning to the log in part where we use Request based approach we use the PRG-pattern. We send a post method but in the servlet and filter we stop the POST method and resend a GET method so every time a user refresh the homepage it only makes a GET-request so the user won't be disturbed.

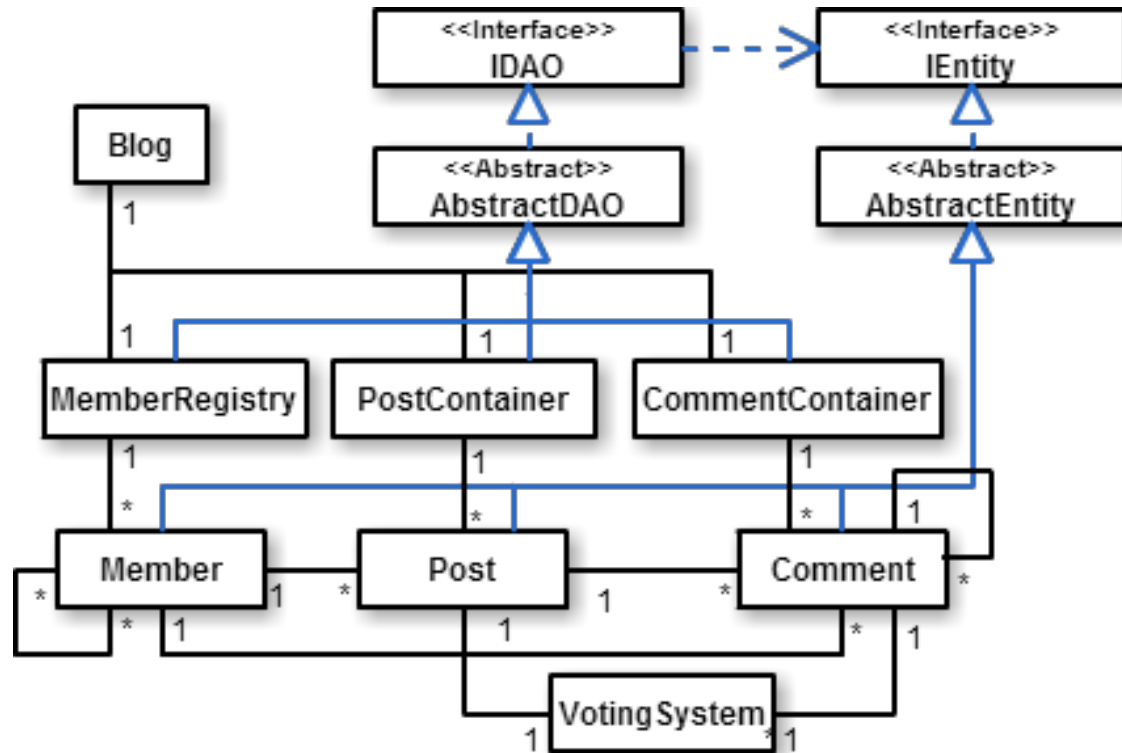
## 3. Appendix

### 3.1 Layers & Tiers

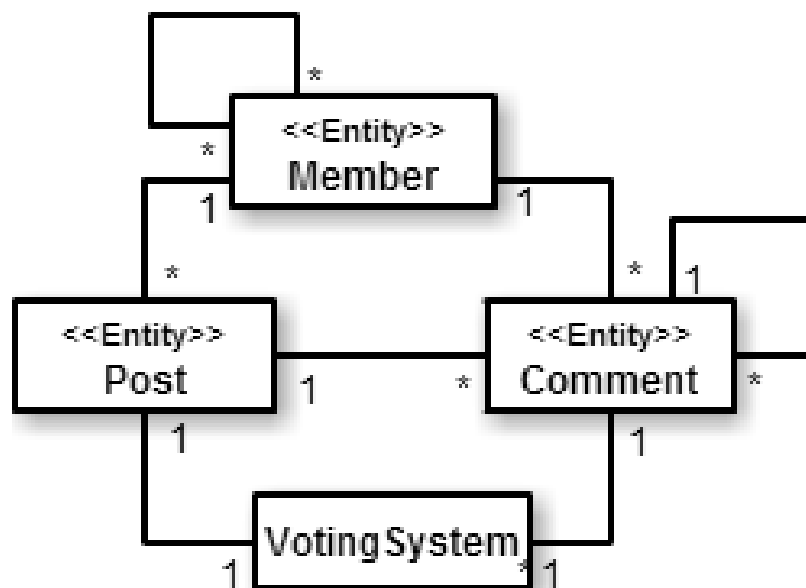




### 3.2 Object oriented model



### 3.3 Persistent model



### 3.4 Use cases

#### UC1: Become a member

Standard flow:

1. From anywhere, navigate to "Login" with the navigation bar
2. Click "Become a member"
3. enter new account-information and click "Submit"

Special cases:

1. Username is already taken, return to step 3.

#### UC2: Log in

Standard flow:

1. From anywhere, navigate to "Login" with the navigation bar
2. enter username and password and click "Submit"
3. You are now logged in and have more access in the application

Special cases:

1. Already logged in at step 1.
  - a. If logged in at correct account; done.
  - b. If logged in at someone elses account; Perform Log out (UC3) and continue and begin at step 1.
2. Not a member of Skyline at step 2.
  - a. Become a member (UC1)

#### UC3: Log out

Standard flow:

1. From anywhere, click "Logout" on the navigation bar.
2. confirm log out when prompted.

#### UC4: View Wall-feed

Standard flow:

1. From anywhere, navigate to "Wall" with the navigation bar

#### UC5: Read post

Standard flow:

1. Perform View wall-feed (UC4)
2. When posts are shown, pick and read one of choice.

#### UC6: Read profile

Standard flow:

1. Perform View wall-feed (UC4)
2. Click on the author of a post or a comment
3. Feel free to read content of the members profile

Alternative flow:

1. Enter members username in search-field.
2. Press enter
3. Feel free to read content of the members profile

Special Case:

1. Want to read your own profile
  - a. Navigate to "Profile" in navigation bar

**UC7: Favorite member**

Standard flow:

1. Perform Read profile (UC6)
2. Click on the "Favorite member"-star

**UC8: Unfavorite member**

Standard flow:

1. Perform Read profile (UC6)
2. Click on the "Unfavorite member"-star

**UC9: read post**

Standard flow:

1. Perform View wall-feed (UC4)
2. Read one of the posts that are shown.

**UC10: View posts written by the members you have favorited**

Standard flow:

3. Navigate to "Favorites" with the navigation bar
4. All posts written by the favoured members will be displayed as a wall

**UC11: Write post**

Standard flow:

1. Perform Log in (UC2)
2. Perform View Wall-feed (UC4)
3. Click "Add post"
4. Enter title and text of the post
5. Press "Save"-button.

**UC12: Vote on post**

Standard flow:

1. Perform Read post (UC6)
2. Vote up/down on post.

**UC13: Write comment**

Standard flow:

1. Perform Log in (UC2)
2. Perform Read post (UC9)
3. Click "Reply" to create comment form.
4. Write comment
5. Click on "Save"-button

**UC14: Read comment**

Standard flow:

1. Perform Read post (UC9)
2. Click button "Show comments" on a post of your choice.
3. Comments are shown; pick one to read.

Special case:

1. No comments are shown at step 3
  - a. there are no comments on this post. Back to step 1.

**UC15: Vote on Comment**

Standard flow:

1. Perform Read Comment (UC14)
2. Vote up/down on Comment.

**UC16: Search for user**

Standard flow:

1. Locate the input field in the navigation bar
2. Type the name of the user you want to find

Special case:

1. If member not found, try step 2 with other name.

**UC17: Try to comment when logged out**

1. Log out
2. Comment on a post
3. Nothing happens

**UC18: Try to log in with wrong password**

1. Press Log in
2. Type "Tomas" as user name, password "qqq"
3. Gets to main page, without being logged in

**UC19: Try to become member with already existing name**

1. Press Log in
2. Press Become a member
3. Type "Tomas" as user name
4. "not a valid user name" since user already exists

**UC20: Read the About-section**

Standard flow:

1. From anywhere, navigate to "About" with the navigation bar
2. Feel free to access different parts of this section.