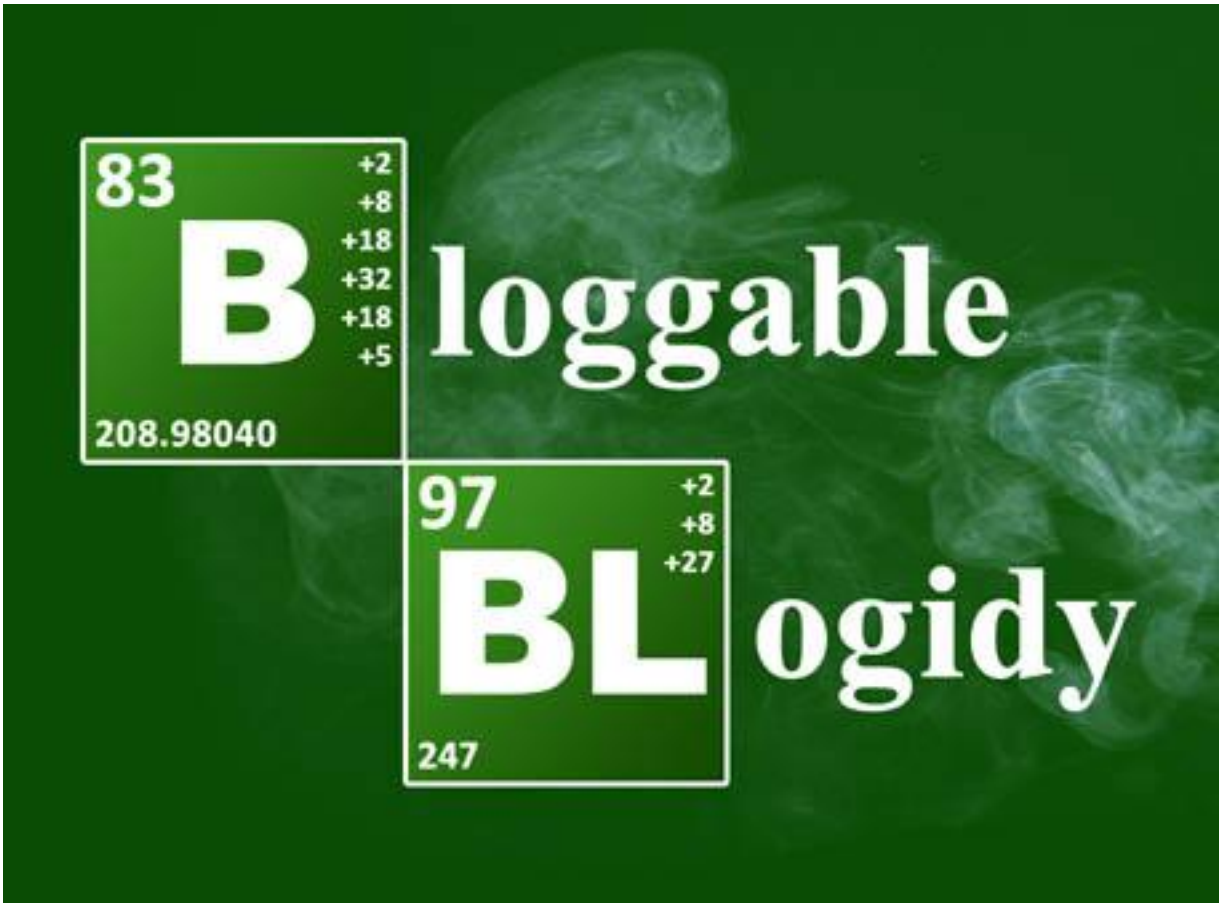
App Dev & Modeling

Continuous assignment 2 by: Anton Krug - 20062210



Contents

Introduction

In this assignment I attempted to recreate some features from generic Blog websites. Because I was implementing in Ember framework I had to recreate whole website from scratch. It turned out to be much more time consuming than anticipated and therefore I had to skip some features. The final result contains just few features of SpaceBook assignment. This would be like repeating the same task over again. The model is supporting much more features than website implemented. Deciding between embedded posts, or linked posts. Options between public and private posts. Sticky posts, different timestamps to show and different timestamp to order by. Users able to subscribe to other blogs and have their posts directly on own homepage. Some of the features wouldn't require much time to implement because there are good foundations and they are already in model. Even prefilled data with subscriptions so when it will be implemented there will be already some data to test it with.

Because I was starting from scratch I decided to couple things differently. For example friendships are not separate model, but they are just ArrayList inside User model, which is so much simpler. Different session management, I tried my own implementation which has nice features. For example it doesn't have to be supplied with cookies. This means that whole website can function without writing to hard drive and it doesn't conflict with the EU cookie law. It's not using any therefore it doesn't have to ask for permission. Side effect is that refreshing the browser will cause loosing session. To work around this limitation I implemented couple tweaks to fake sessions present when run from localhost server. This turned out as huge time saver while debugging and developing.

I decided to keep most of the server controller methods in one class file (Api.java). Normally it's not accepted behavior because of organization reasons, but in this case the methods are so small it was acceptable and even more organized than having many tiny files. More detail is described inside the Api.java file.

Stories

These stories were chosen long time before the Ember difficulty and time scope required was known. Therefore not all features are implemented fully (most of them are implemented at least on model/database level).

Story 0

Design graphics, decide on theme and basic layout. Create logo for website.

Story 1

Create welcome page, for unlogged user to sign in or sign up.

Story 2

Create login part of html and controller. Implement sessions.

Story 3

Design signup part of form and create controller for it

Story 4

Refactor signup part as component so it can be used later for “edit your details”

Story 5

Allow users to have multiple Blogs. Allow to have private and public (and maybe even unlisted) options for blog. (maybe private/public, unlisted/promote options).

Story 6

Each blog should have own home edit page with some options to alter (title, graphical tweeking). And then the preview to see it in final form so owner can see what he is creating.

Story 7

Each blog can have multiple Posts and each post should allow multiple comments. Design model and base controllers for it.

Story 8

Allow comments to be as reply to other comments.

Story 9

Implement deepness checking and limit the level how deep the the comment replies can go.

Story 10

Allow uploading pictures as attachment to bottom of the posts

Story 11

Allow anonymous people to comment on public blogs (not sure if it's good idea), Then allow to disable anonymous people comment on each post.

Story 12

Blogs should contain 2 different dates, date by which it will be ordered and date it will show it was published. This way user can have some blog always sticking on the top

Story 13
Each post should be able to block/allow comments

Story 14

If embedding will be used to display inline posts then allow to see all comments there as well.

Story 15

User can upload 1 picture as their avatar/profile (it will be used in the comments)

Story 16

Allow images for blogs, separate custom blog background image per each blog.

Story 17

Allow yourself to subscribe to some blogs (add is as reading list) so on dedicated webpage or your dashboard you can see other blog posts

Story 18

When creating Post you have option to not stream it to others (for example small unimportant blog post, or sticky post which should not propagate.

Story 19

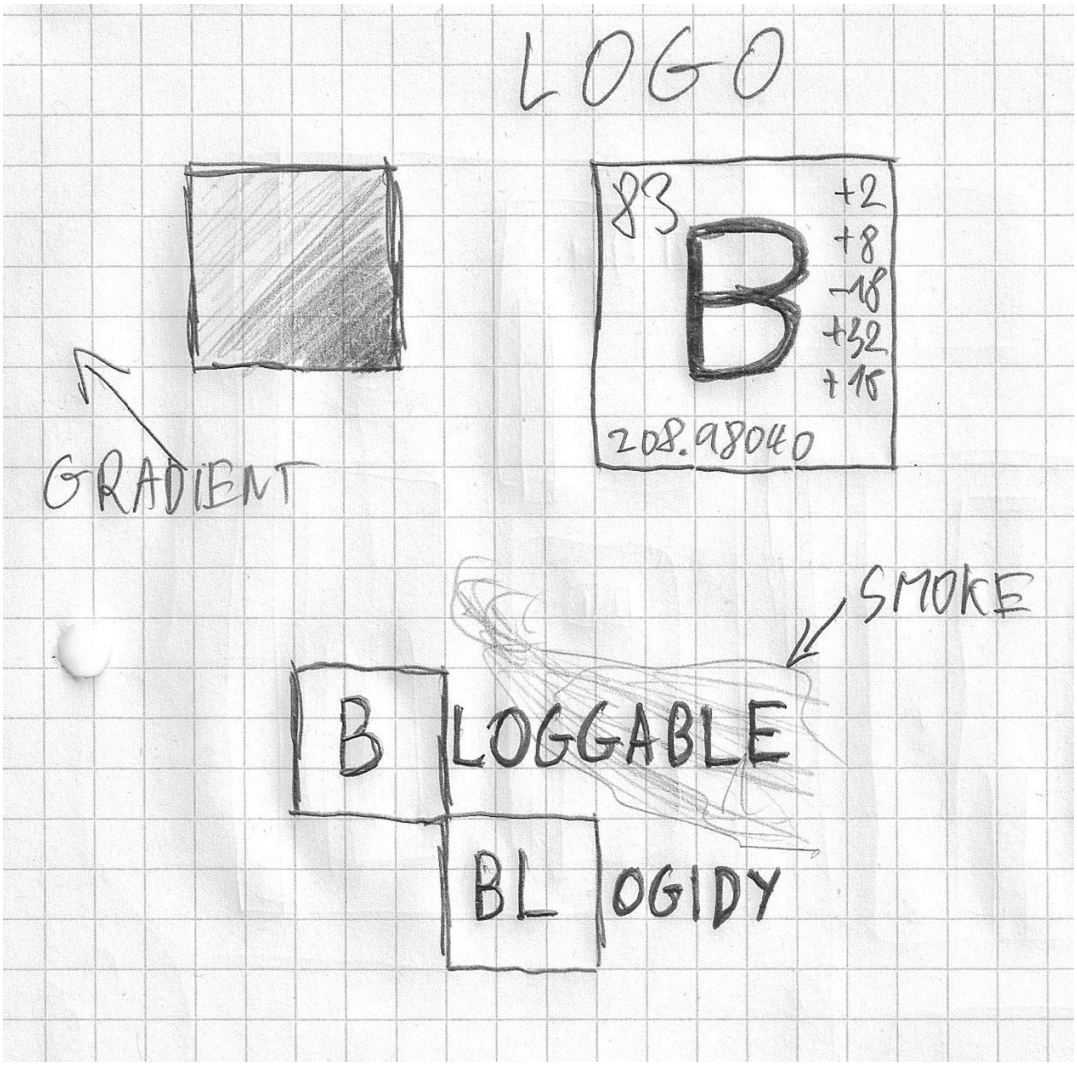
Allow markup language inside comments, posts. So user can create links, and do some simple formatting.

Story 20

To delete comment you have to own the comment, or own the whole blog. On foreign blog posts you can remove only yours comments. On your own blog posts you can remove anybodies comments.

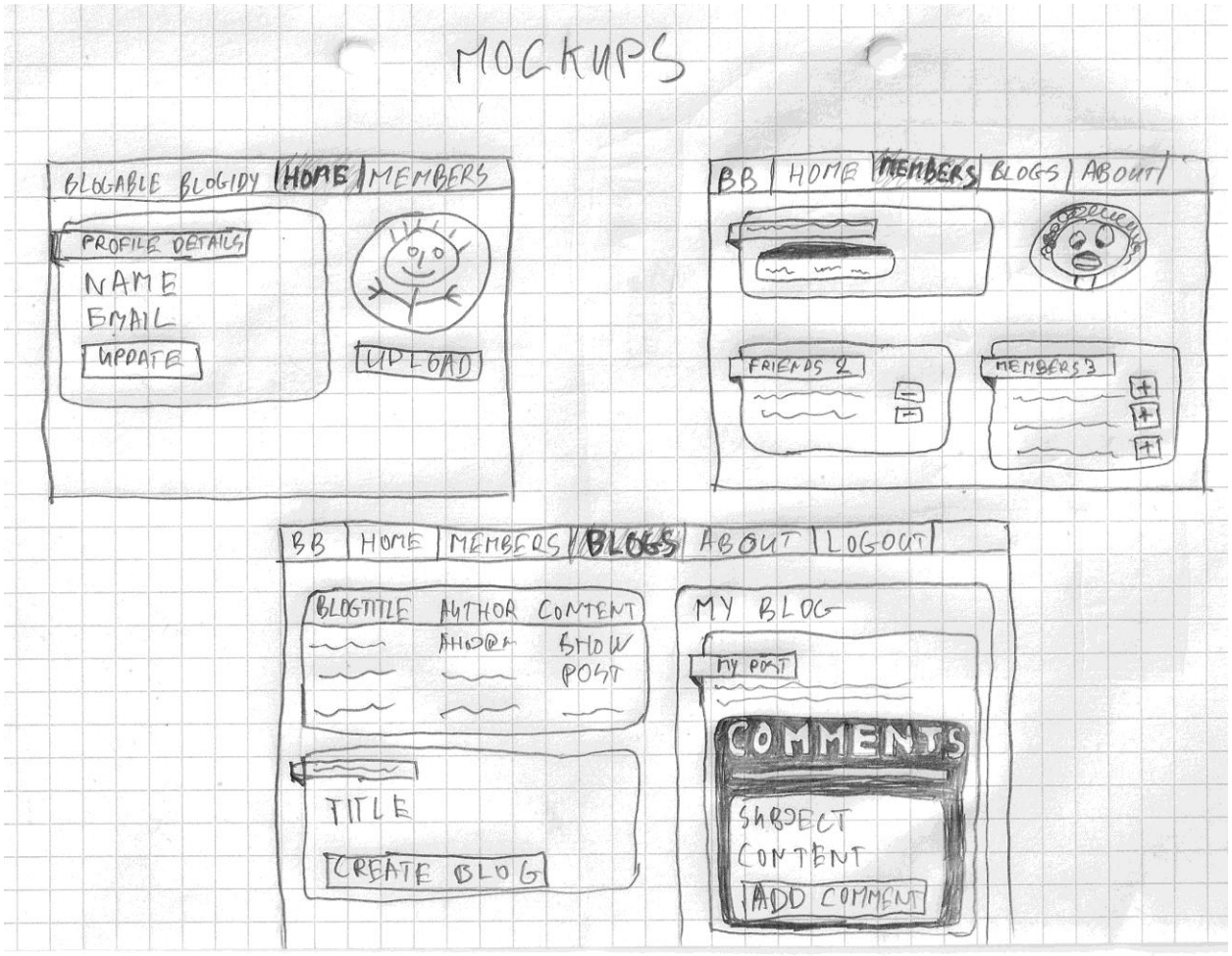
Sketches and mockups

Logo sketch

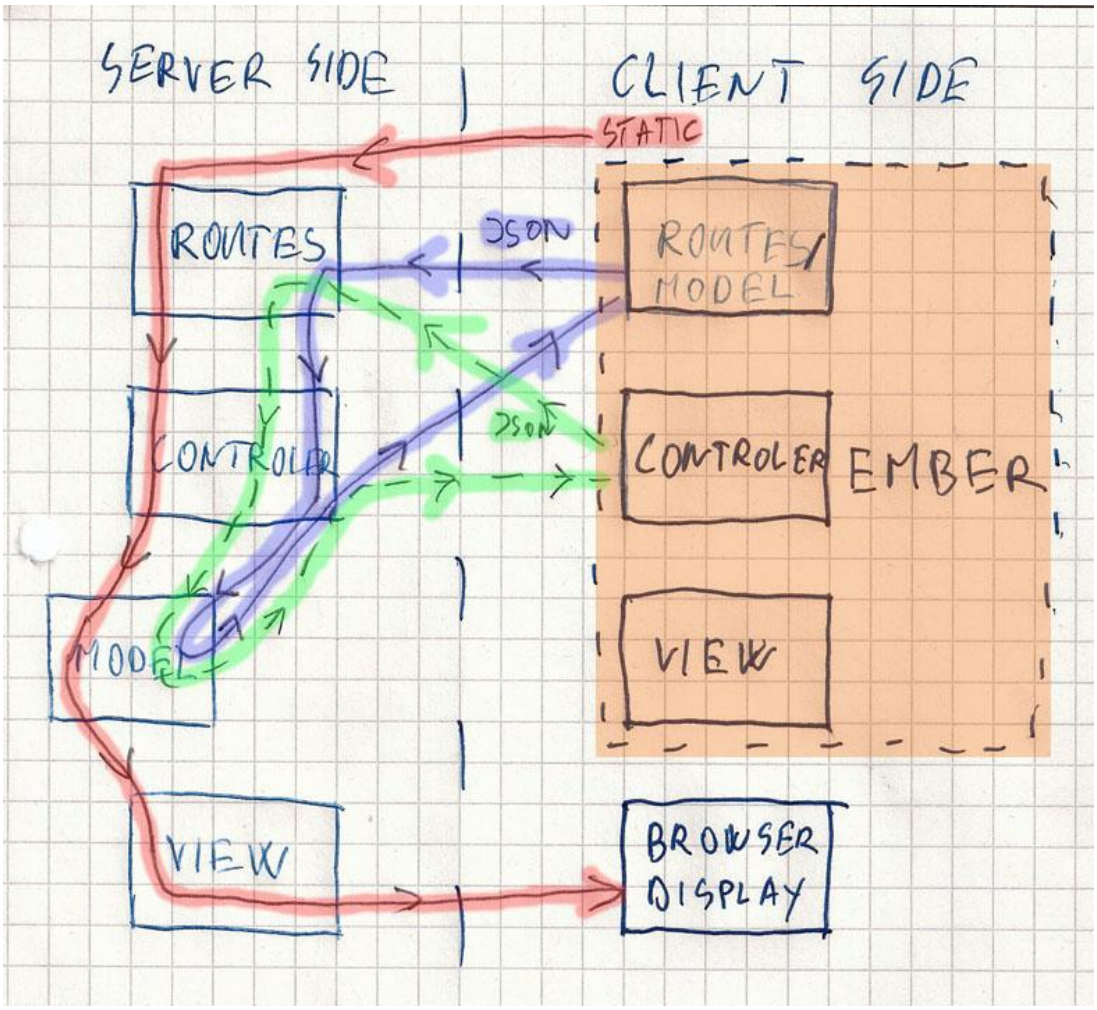


Inspired by Breaking Bad theme, all Bootstrap data contains characters from this show with their quotes and references.

Website layout mockup



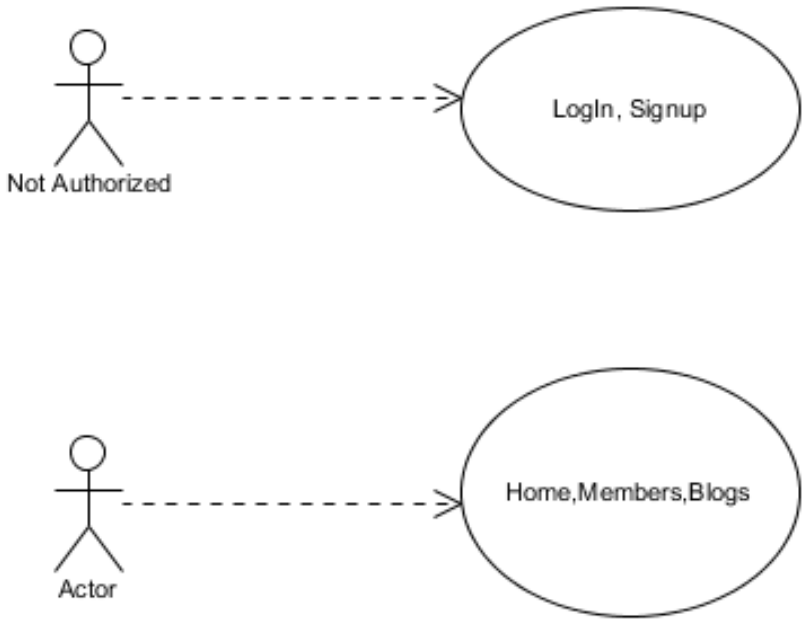
Network request diagram sketch



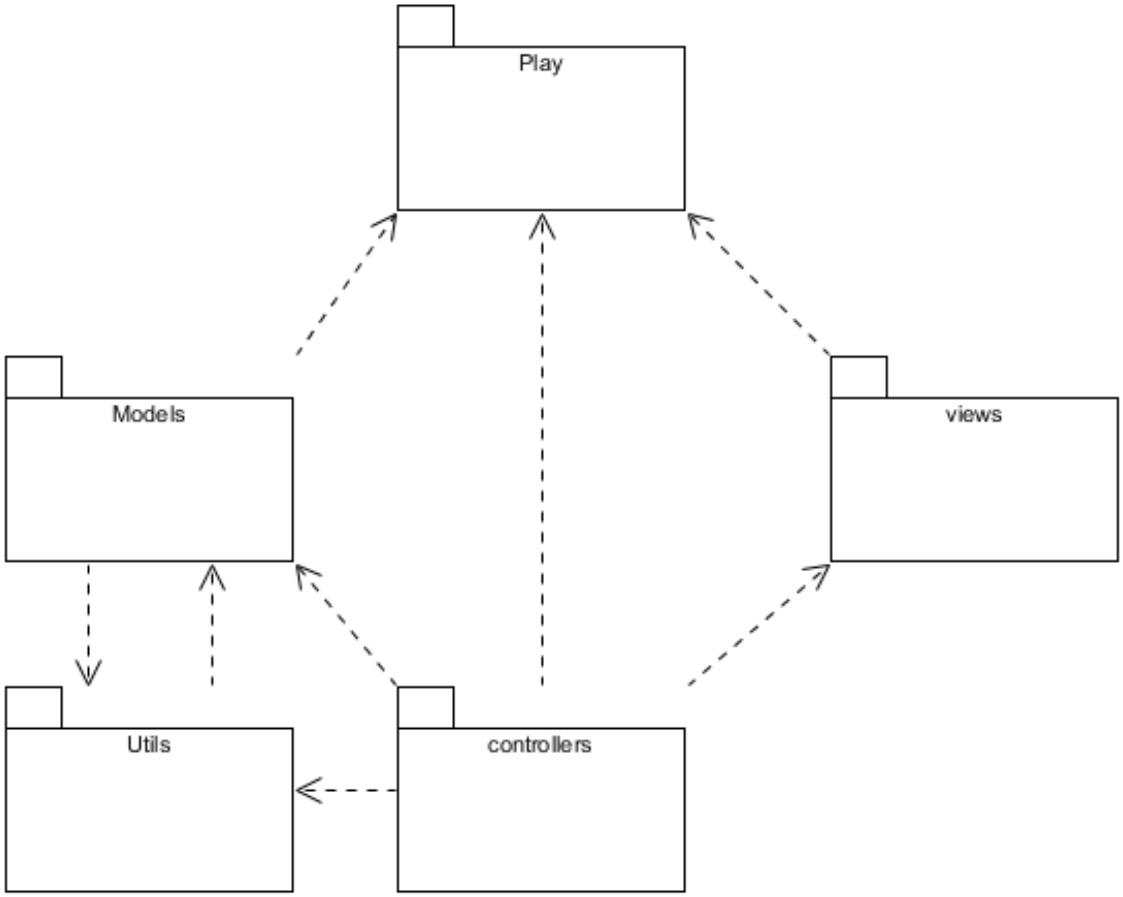
First request is highlighted in red where first static request is made. It goes through whole server side and goes into browser on client side. Browser just received whole ember framework which he will fire up. This creates another set of routes, model, controller and view inside the browser on client side. When the website is running each event is triggering some JavaScript and depending of type of the action I will originate from model or controller. Both of them are JSON type of requests, they are highlighted with green and purple. Request will go down to server controller, getting data from model, from model back to controller to pre-process the data. Here templating is done on client side inside the ember view and therefore before sending the data away from the server they need to be processed and clean sensitive data. This problem wasn't so huge when templating was done on server side.

UML diagrams

Use case diagram



Package diagram

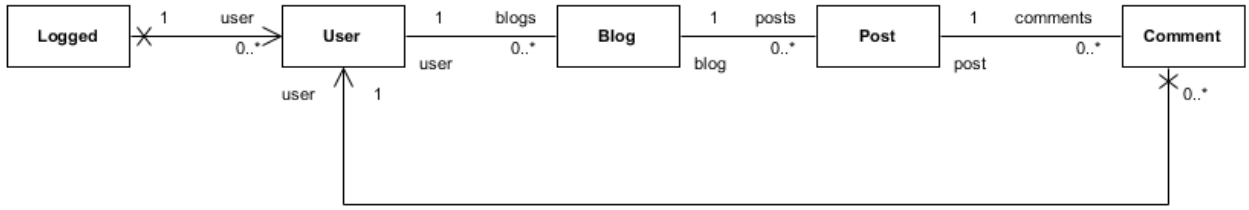


Class diagrams

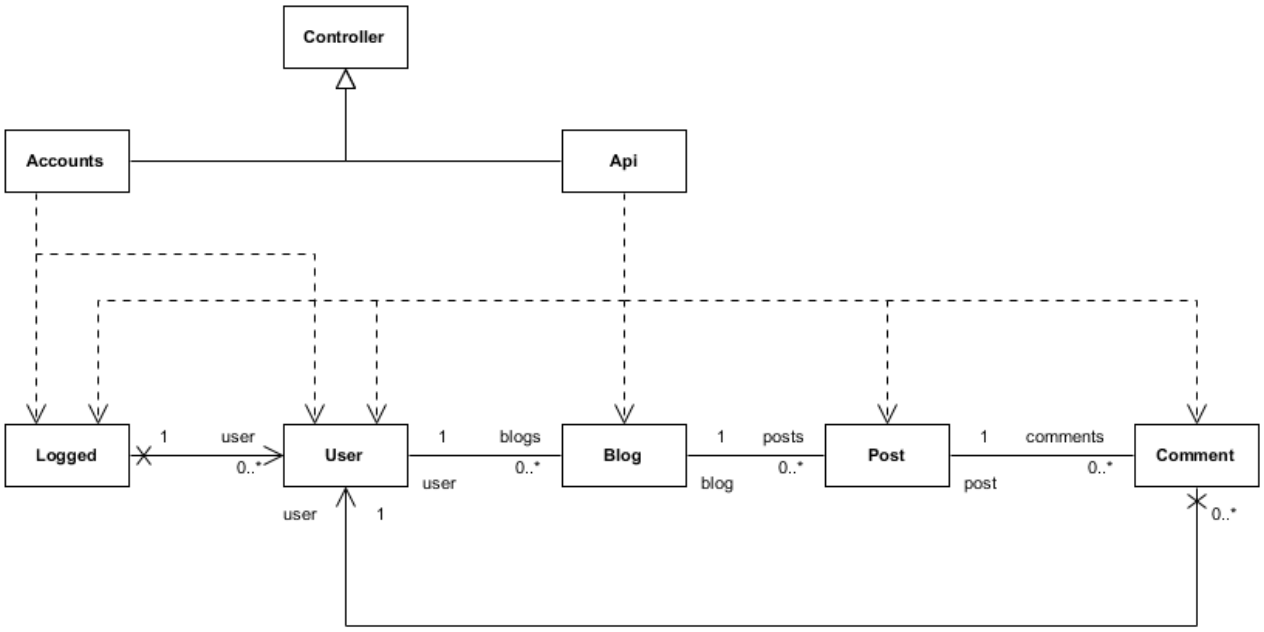
Play:

Utils:

Models:



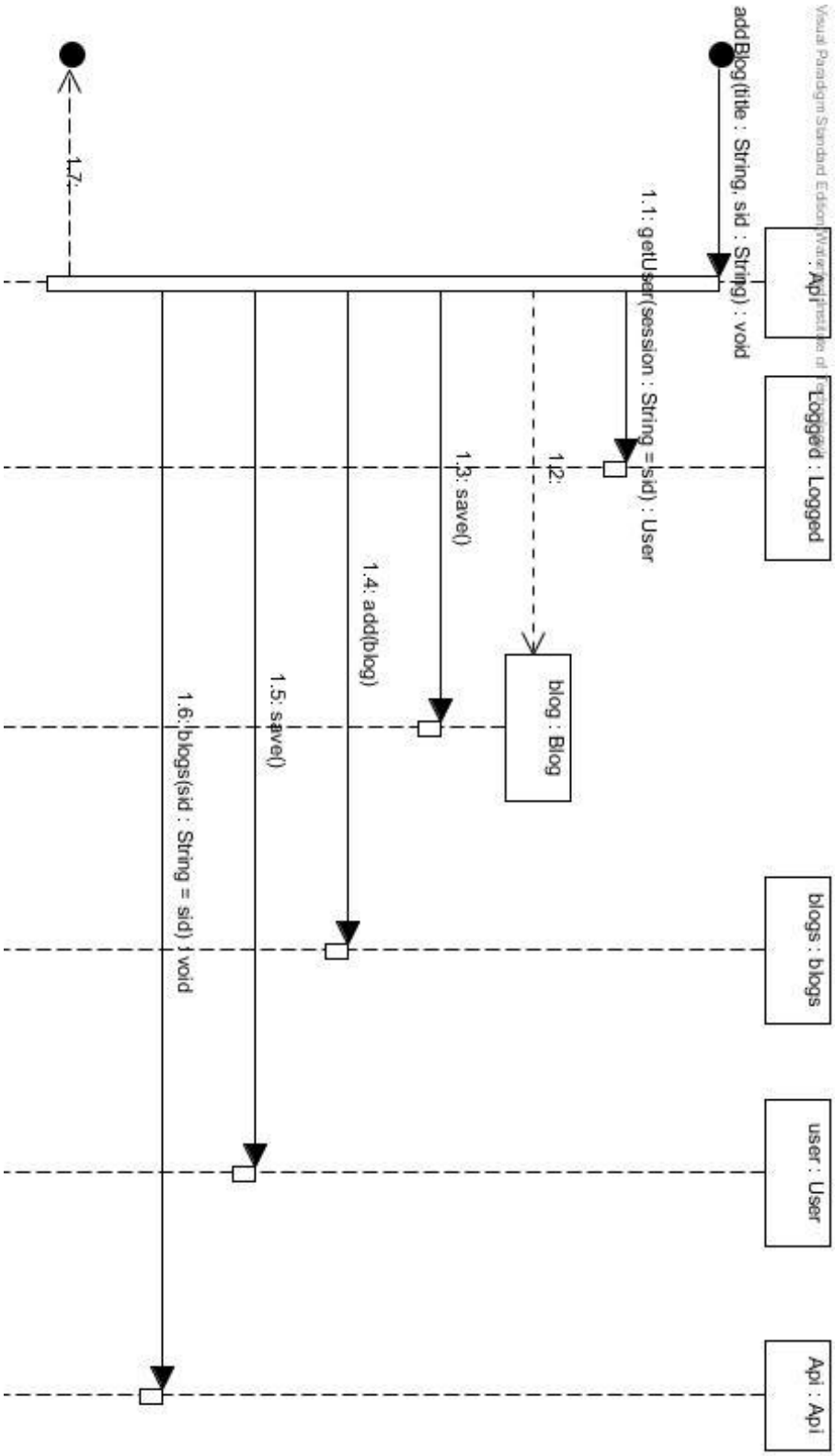
Controllers:



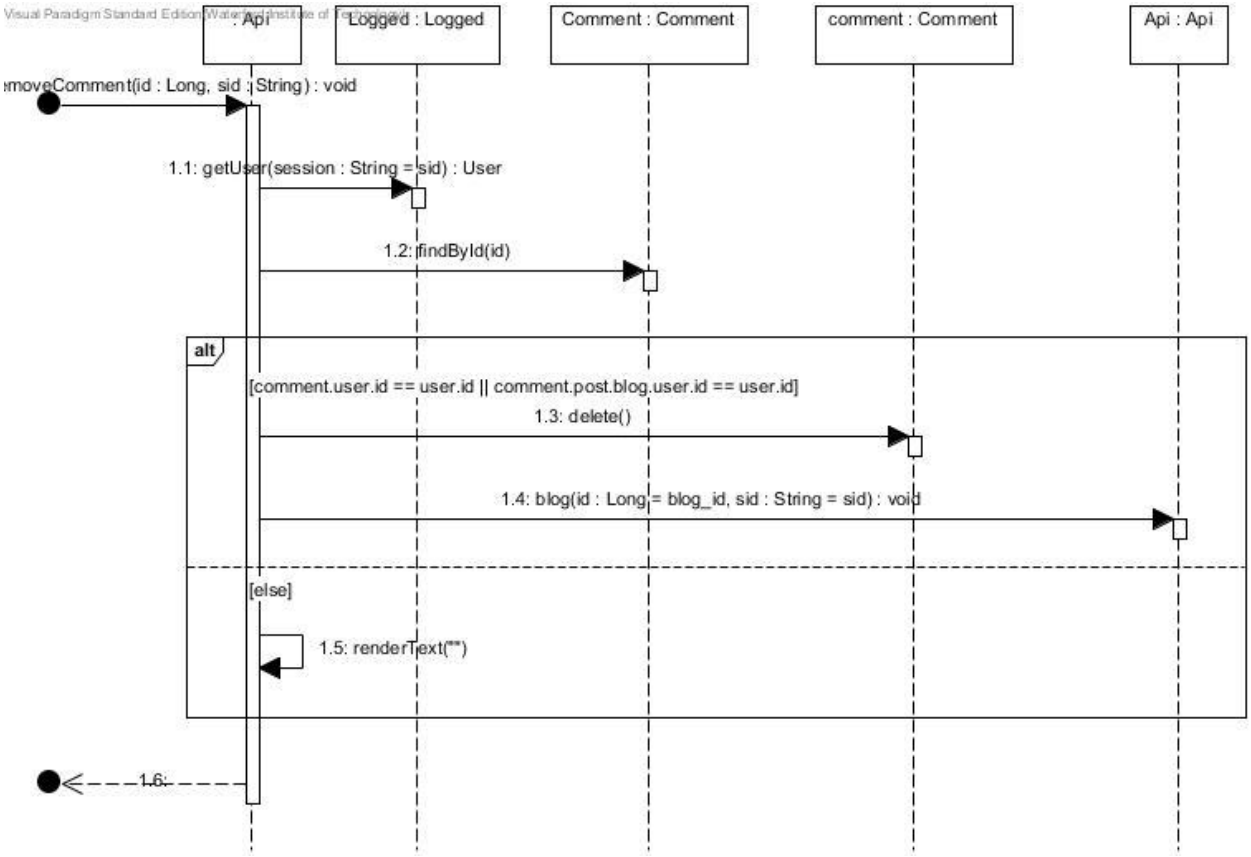
Sequence diagrams:

addComment()

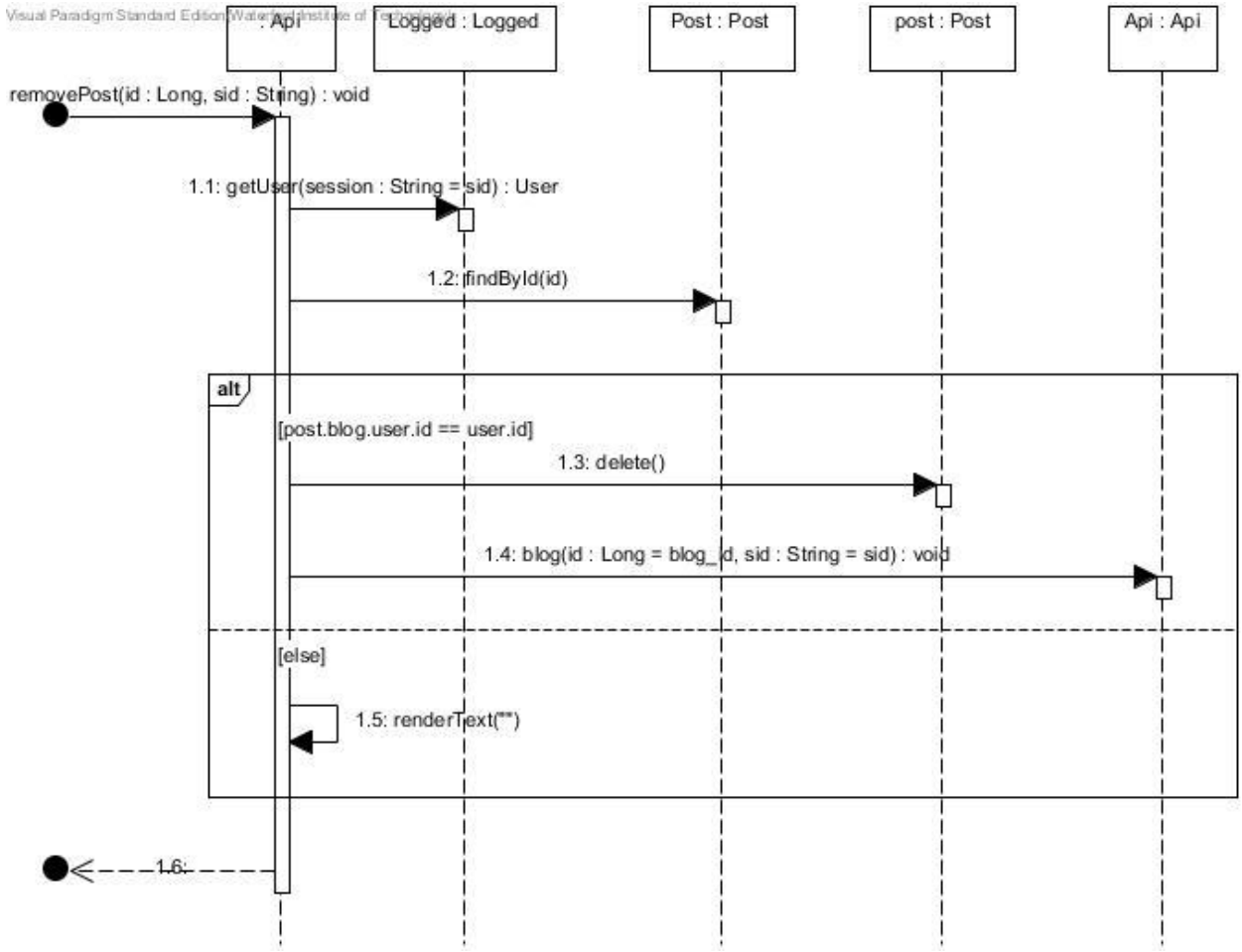
addBlog()


addBlog(title : String, sid : String) : void
1.1: getUser(session : String = sid) : User
1.2:
1.3: save()
1.4: add(blog)
1.5: save()
1.6: blogs(sid : String = sid) : void
1.7:
: Api
Logged : Logged
blog : Blog
blogs : blogs
user : User
Api : Api

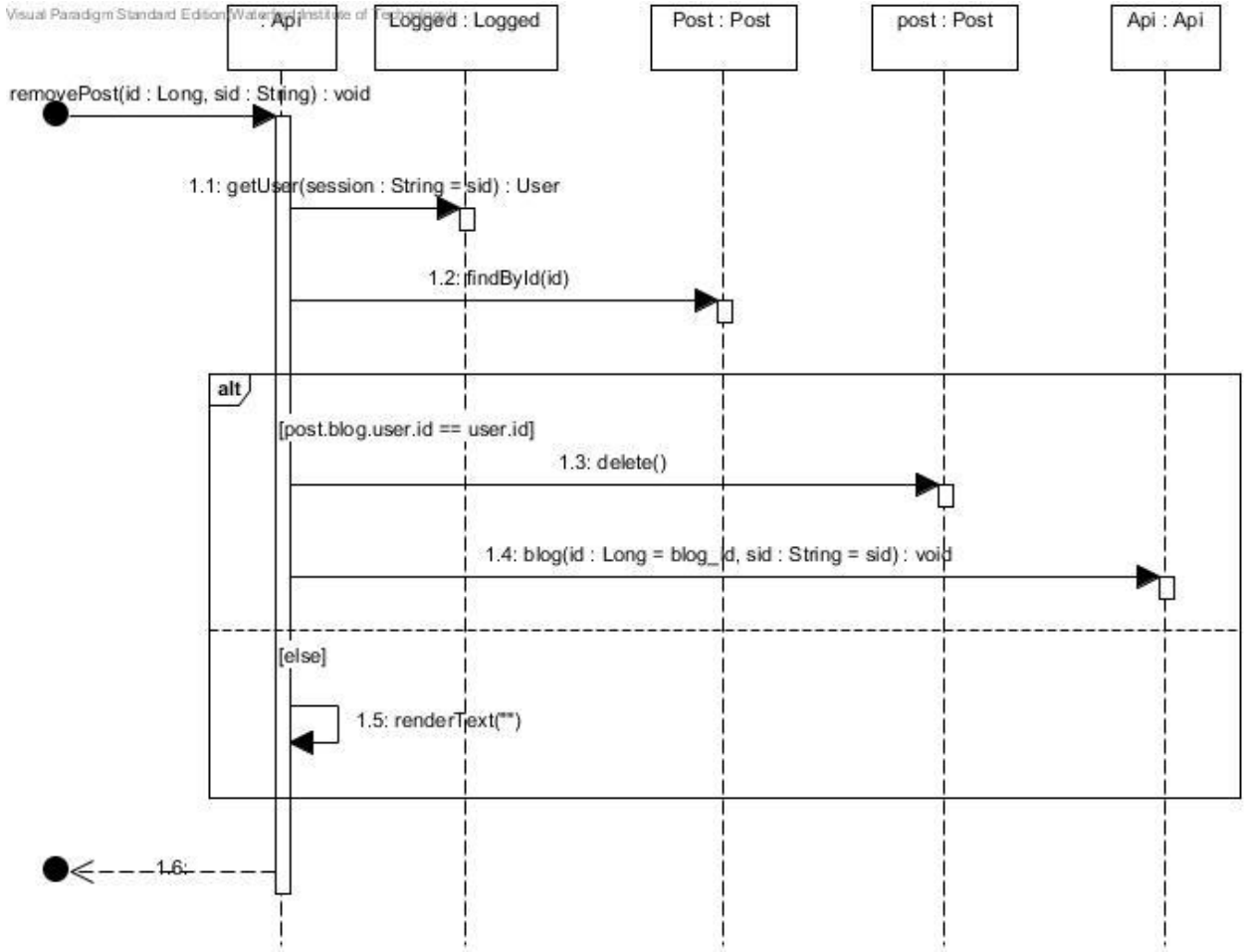
removeComment()



removeBlog()


: Api
Logged : Logged
Post : Post
post : Post
Api : Api
removePost(id : Long, sid : String) : void
1.1: getUser(session : String = sid) : User
1.2: findById(id)
alt
[post.blog.user.id == user.id]
1.3: delete()
1.4: blog(id : Long = blog_id, sid : String = sid) : void
[else]
1.5: renderText("")
1.6:

removePost


: Api
Logged : Logged
Post : Post
post : Post
Api : Api
removePost(id : Long, sid : String) : void
1.1: getUser(session : String = sid) : User
1.2: findById(id)
alt
[post.blog.user.id == user.id]
1.3: delete()
1.4: blog(id : Long = blog_id, sid : String = sid) : void
[else]
1.5: renderText("")
1.6:

Deployment

It's deployed on Cloudbees server, the URL is:

http://bb.antonwit.cloudbees.net/

To find passwords to all users go to /utils/Helpers.java and find method fillData();

Mostly it was tested with this user:

Login name: "skyler@white.com" and password: "hope"

Any user can be used to login, but this one has mostly prefilled data so it's most to see and the experience should be most balance.