

1 Developers Guide

This introductory developers guide, aims to make a developer able to set up Stedr so that the developer gets an overview of how the system works and also how the developer can get started programming. The first part of the guide is about the backend while the second part is about the frontend. Both of the guides needs to be completed to get the example program running, and the backend part has to be completed before the frontend part.

This guide is **not** meant as a tool guide, so some parts of the guide is superficial and it is left to the reader to study the tools closer.


1.1 Backend

The backend is a Java program using the Play Framework, and the backend is deployed to the Heroku, a cloud platform hosting applications as services. The source code itself maintained on a GitHub account provided by Sintef called TagCloud. Before continuing this means that a couple of prerequisites has to be fulfilled by the developer.

The developer should have:

- Installed an updated version of JDK
- Installed a code editor (Eclipse will be used in the tutorial)
- A working GitHub-account
- Installed git
- Cloned TagCloud/StedR_server with the help of Git from GitHub
- Installed the Typesafe Framework from Play Framework
- An Heroku account

On your computer, open up a terminal of your choice (cmd, bash, ...) and navigate to the folder where you have extracted Typesafe Activator (Play Framework). Depending on your platform type the command which will execute activator. It is possible to use Typesafe Activator with a graphical user interface by passing ui as a parameter. This will look something like:



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

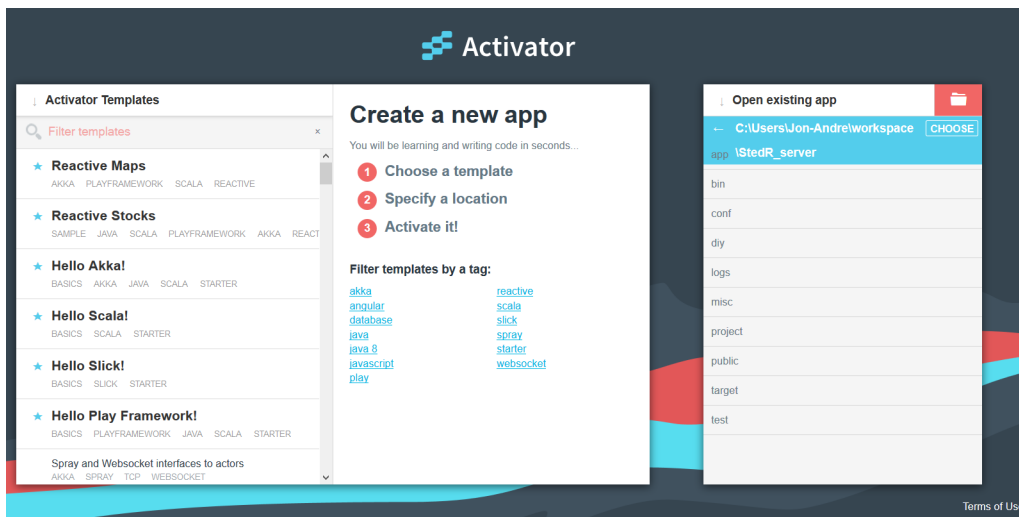
C:\Users\Jon-Andre>cd workspace

C:\Users\Jon-Andre\workspace>cd activator-1.0.13

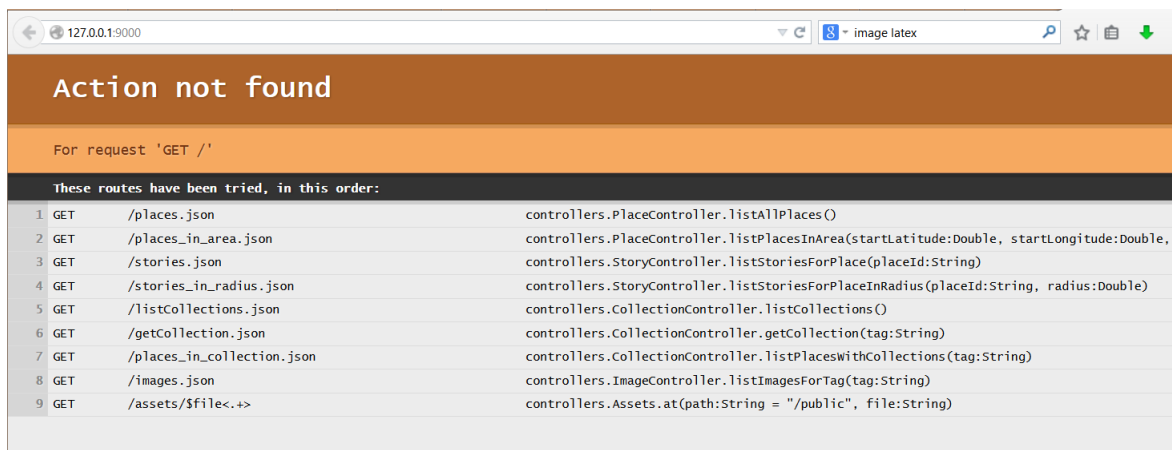
C:\Users\Jon-Andre\workspace\activator-1.0.13>activator ui
```

The graphical user interface should then open automatically in a browser view. If this doesn't happen check the terminal for error messages.

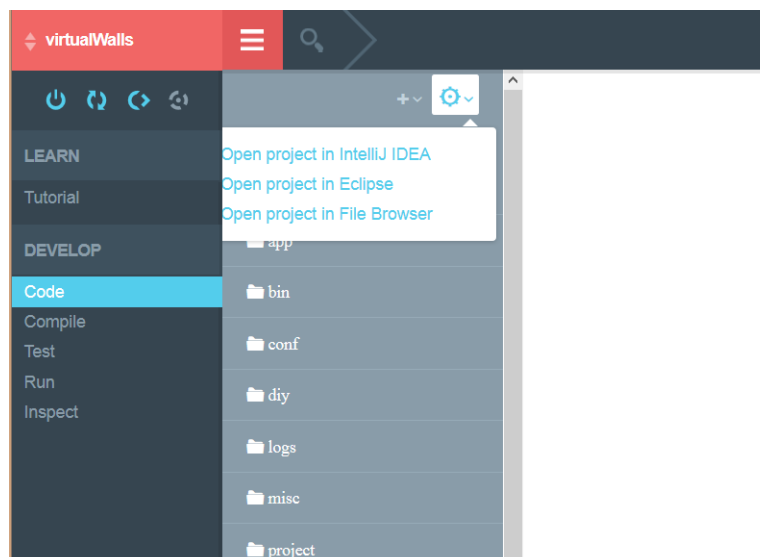
In the right sidebar navigate to the folder where you have cloned StedR_server from GitHub. Click choose. Now the program is starting to compile, and the server will try to run as a local instance on localhost port 9000.



Notice that Typesafe Activator itself is running on port 8888. During the compiling of the system problems may occur. Often this is related to a mismatch between Typesafe and Java, for example an updated version of Java and an unupdated version of Typesafe often leads to issues. If the program is compiled successfully, something like this should appear at localhost:

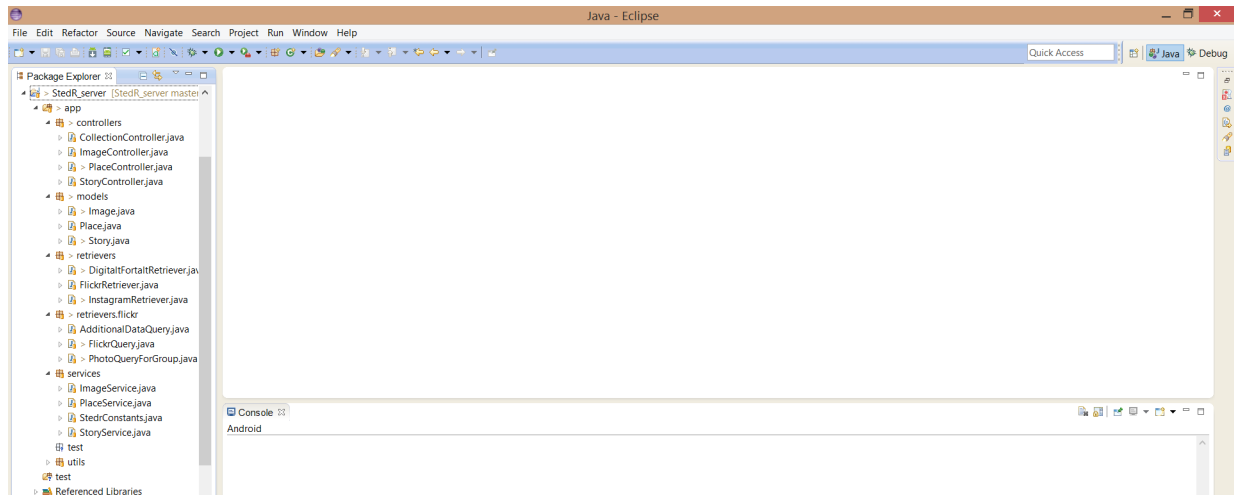


If you already have tried importing the folder with the source code to an editor like Eclipse, you may have noticed a lot of errors appears. To import the program and its dependencies as a project: In the left sidebar of Typesafe, click on Code, then the gear-icon. Here you can choose between IntelliJ and Eclipse, and Typesafe will then generate project files and guide you through how to open the program as a project.

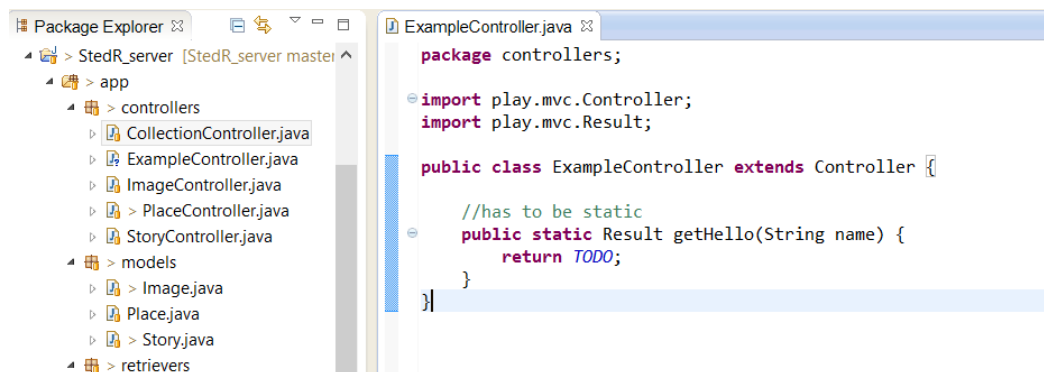


Now you should have a running instance of the server locally, and you're also ready to code.

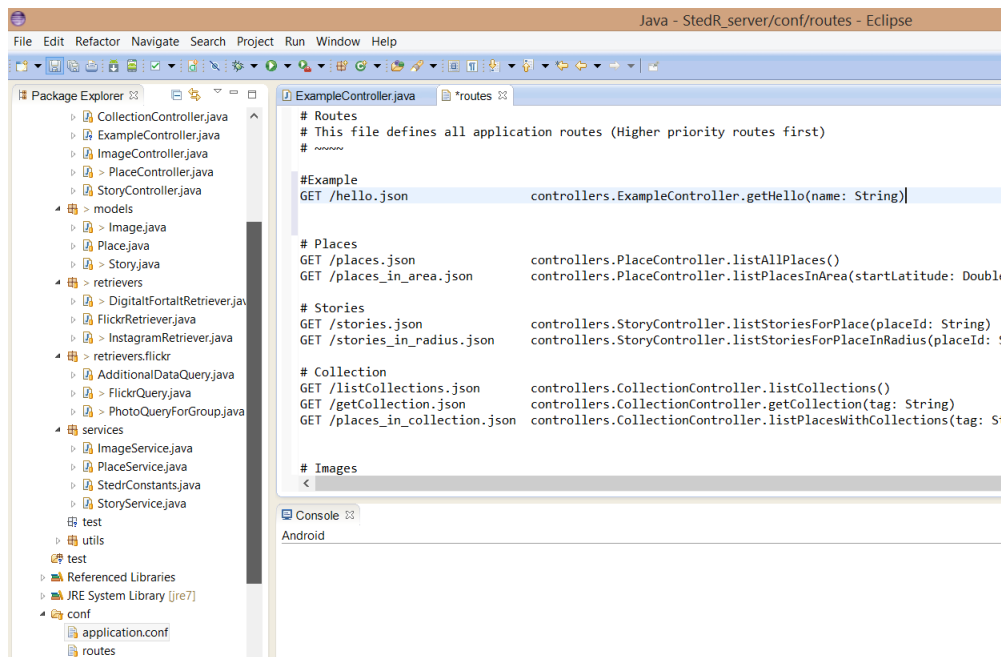
In Eclipse you will get an overview of the different source files and source packages. In the Controllers package you will find controllers that take care of identifying queries (sent as an URL), creating Retrievers that process the queries and at last returning a response to the query. All of the Controllers are written by former Stedr-developers, but all of the Controllers extends a Controller from the Play Framework.



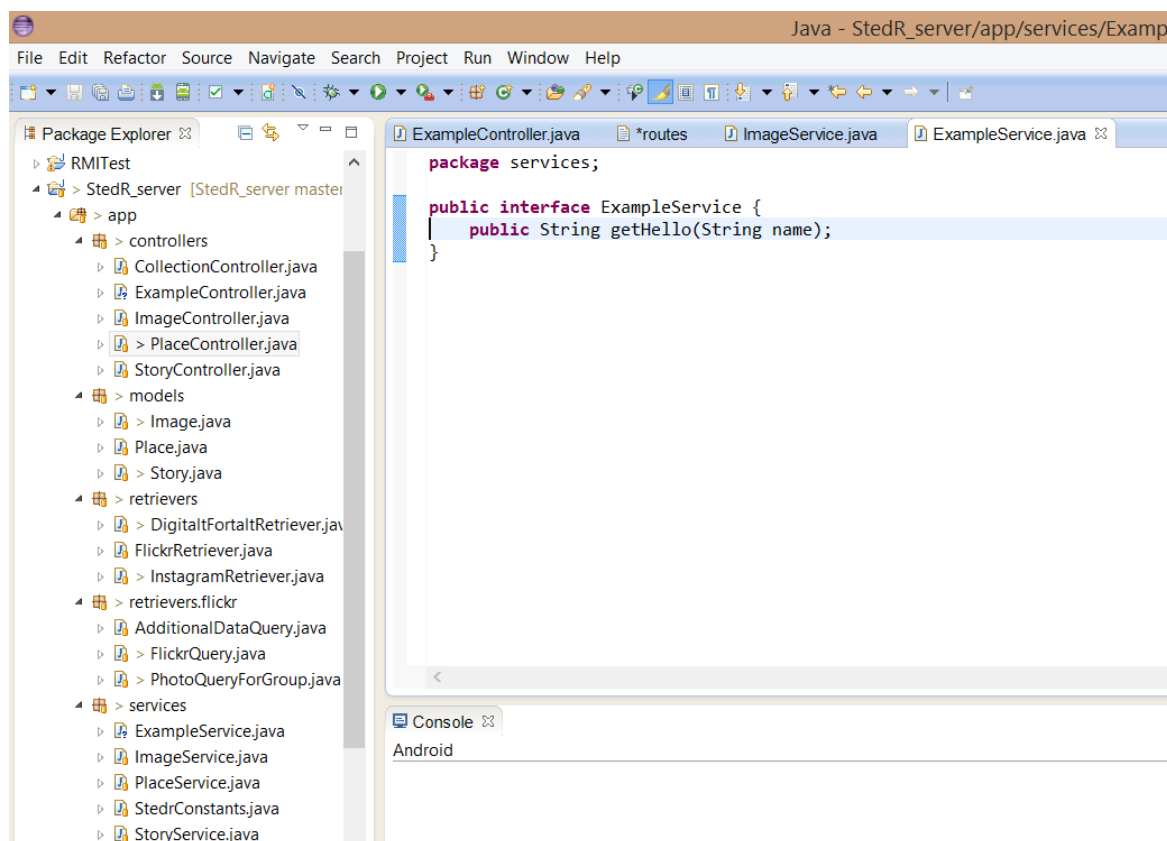
Now, let's create a controller for our example application:



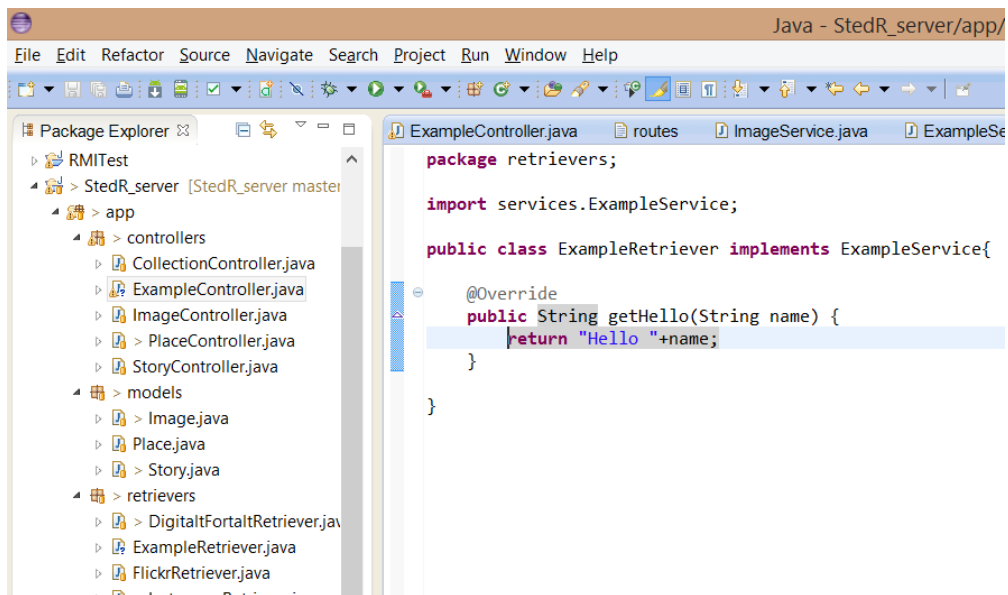
We ask for a parameter called `name` which naturally is the name you want to be displayed in the smartphone-application. To pass a parameter you have to edit the file called `routes` in the `conf`-folder, the passing is done directly from the smartphone application.



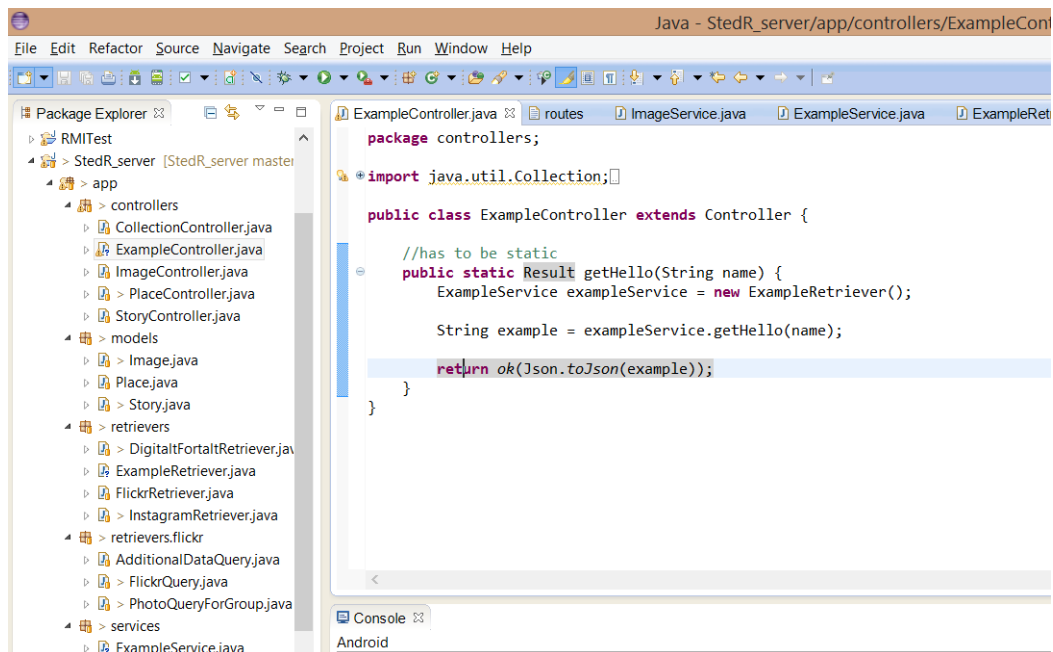
It would now have been possible to create a retriever directly, but we won't do that. In the services folder there are three files ending with `Service.java`. These files are interfaces, and the reasoning behind them is that it should be easy to add or change services. As of now stories are provided by Digitalt Fortalt, so we have a `DigitaltFortaltRetriever.java` which implements `StoryService.java`. That way we can change the content provider to Wikipedia by creating a new retriever `WikipediaRetriever.java` which implements `StoryRetriever.java`. A lot of code would then have to be written in order to get the fictional `WikipediaRetriever.java` functional. Back to the example we will therefore create an `ExampleService`.



Now we're ready to implement the `HelloRetriever`



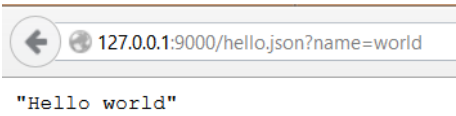
After the `HelloRetriever.java` the last thing that needs to be completed is the `ExampleController` which was created in the beginning of the example



This concludes in a server which will respond with `Hello World` if `World` is passed as a parameter. To see this, the system has to be recompiled in TypeSafe Activator. After recompiling open localhost port 9000, there you should see:

Action not found		
For request 'GET /'		
These routes have been tried, in this order:		
1	GET	/hello.json controllers.ExampleController.getHello(name:String)
2	GET	/places.json controllers.PlaceController.listAllPlaces()
3	GET	/places_in_area.json controllers.PlaceController.listPlacesInArea(startLatit
4	GET	/stories.json controllers.StoryController.listStoriesForPlace(placeId
5	GET	/stories_in_radius.json controllers.StoryController.listStoriesForPlaceInRadius
6	GET	/listCollections.json controllers.CollectionController.listCollections()
7	GET	/getCollection.json controllers.CollectionController.getCollection(tag:Stri
8	GET	/places_in_collection.json controllers.CollectionController.listPlacesWithCollecti
9	GET	/images.json controllers.ImageController.listImagesForTag(tag:String
10	GET	/assets/\$file<.+> controllers.Assets.at(path:String = "/public", file:Str

If this is correct, `http://127.0.0.1:9000/hello.json?name=world` should give:



Normally you would commit this to the git-repository on GitHub, and if the new functionality is to be a part of the running server it should also be committed or deployed to Heroku. This is done by pushing the server to `git@heroku.com:stedr-beta.git`. Note that in order to push to the server directly, you will need contributor access to `stedr-beta.herokuapp.com`. If this isn't available or provided upon request you can create a new Heroku instance, but then the request URL destinations have to be changed in the smartphone application.

2 Frontend

...