Project Assignment Programming with Domain Specific Languages (Scala)

Bob Reynders

1 ProMan

For this assignment, you are required to implement a client/server project management tool called ProMan in Scala.

The base application is a multi-user version of a simple todo-list.

1.1 Expected working time

We expect you to spend max. 40 hours on this assignment. You are allowed to turn in an incomplete assignment, it is completely normal that you do not finish all extensions.

A correct base implementation without any extensions will get you a passing grade on the project, extensions increase your maximum grade further.

1.2 Summary

The application that you're making is going to be a single-page application (SPA), a web application that loads one HTML page and interacts dynamically with the user without loading any other pages. You can use the same set-up as in the exercise sessions, one main HTML page will be served which loads the Scala.js application. The Scala.js application in turn communicates with the Scala server application to implement interactive behavior (e.g., retrieving all animals through the JSON endpoint in the exercise session).

1.3 Scenes

- **Start:** When you open the application you get a list of active projects (or an empty list) and an input box. A user can enter a new name into the

input box and hit a create button or hit enter to create a new project.

All common errors should be checked, i.e., it should not be allowed to create projects with the same name or with an empty name. Remember, this applies across all users at once!

This scene contains:

  - a title
  - a list of projects (possibly empty)
  - a textbox to enter a new project's name
  - a button
  - an area to display warnings (when someone wants to create a project that already exists)

- **Project:** When you click an existing project or start a new one you're in the project scene. Here you get to fill in a todo-list.

  New entries can be created by filling in a message in a textbox and hitting enter or the create button. Entries automatically get the date of creation added to them and are shown in the todo-list (table) sorted from new to old.

  Entries in the todo-list can be marked as done by hitting a button. 'Done' items are moved into a second list below the todo-list, they can be 'undone' by hitting a button.

  An entry's message can be modified freely, all changes supposed to be automatically saved as the user makes them, that is, every keystroke is an update.

  All common errors should be checked, i.e., it should not be allowed to create entries with an empty message.

  This scene contains:

  - a title
  - two lists of entries, an entry consists of:
    * a text message
    * a date
    * a 'mark as done or undone' button
  - an area to display warnings (adding an empty entry)

1.4 Multi-user

- All functionality has to be used by multiple users! All projects are visible to everyone who uses the application. This means that the state is likely maintained on the server.

- Users should see an updated version of the server state, the program should update every 10 seconds (that is, contact your server every 10 seconds to ask for the latest state). Updates that a user made should be visible without waiting for 10 seconds (unless the delay to the server is > 10s).

1.5 Tips

- **Look at the solution to the exercise session (I added an extra case of submitting animals to help you get started!)**.
- Ask questions on toledo, I monitor it regularly and try to answer daily. Feel free to help out and answer questions yourselves, but don't share exact solutions or code.
- Scala.js is a young project, it's a compiler for Scala to JavaScript. Most of the problems that you will encounter while solving this assignment probably do not have a Scala.js tutorial on the internet. However, JavaScript web applications are very popular, it will be very easy to find tutorials. JavaScript interop in Scala.js is very easy and you can look at the Scala.js project page to get a feel for translating JavaScript code into Scala.js.
  - https://www.scala-js.org/doc/interoperability/
- Your state should live on the server. A good structure for your application is to create a Scala.js application that communicates with JSON endpoints and updates the state that way.

Here are some links with some reading material that can help you.

- Scala collection examples
- Http4s Examples
- Hands-on ScalaJS book (direct link to interacting with user input with Scalatags)
- DOM on-event handlers

Links from the exercise session:

- Http4s
- Circe
- Scala.js DOM Ext
- XMLHttpRequest
- Scalatags JsDom
- Scala.js DOM
- DOM Query Selector

1.6 Technical Requirements

There are no technical requirements, you can use as many Scala, ScalaJS or JavaScript libraries as you want. However, all your code has to be written in Scala. If you want to use JavaScript libraries (e.g., the hotkey implementation) you should learn how to communicate with JavaScript from Scala.js.

'Official' support, that is, extensive help from myself, is only available for projects that use the default build.sbt file.

1.7 Extensions

If you have extra time, you can implement extra features for more points from easy to difficult(-ish).

- In the project scene, above the todo-list, allow for a big red stickied alert and a description box below the project name.
- Add a button to hide all 'done' items.
- Render as much as you can on the server, e.g., your initial project list should be rendered on the server.
- Instead of having only one todo-list, allow multiple lists that each have a title. This mimics a 'kanban board'. Items that are marked as 'todo' go to a *common* area below all lists.
- Add persistence to your application. Use Slick + H2 and store your server state (all projects, etc.) in a database that saves to file. The example skeleton build.sbt supports the Slick library and also contains a configuration file for Slick to start the H2 database properly so that it saves its data to a local file (application.conf). http://slick.lightbend.com/doc/3.2.1/introduction.html
- Create a search bar on the start and project scenes. The main scene search bar allows you to filter the list of projects based on the name. The project scene search bar filters a specific project's todo-list items. Looking for a matching substring is fine for this implementation.
- Come up with your own.

Make sure you fill in extensions.md. In this file you fill in which extensions you implemented (or attempted) and how you did them. I will use this file when correcting your solution!

1.8 Turning in the assignment

You have to turn in your solution for this assignment on toledo. Make sure your application runs and compiles properly when I load up SBT and run the JVM project.

The deadline is Thursday December, 21st at 23h59. If you have any questions, there is a discussion forum on Toledo that will be actively monitored.