Backend: Play! - Framework

Serious Game Project at JMU University of Würzburg, by

Lena Lang

Jessica Topel

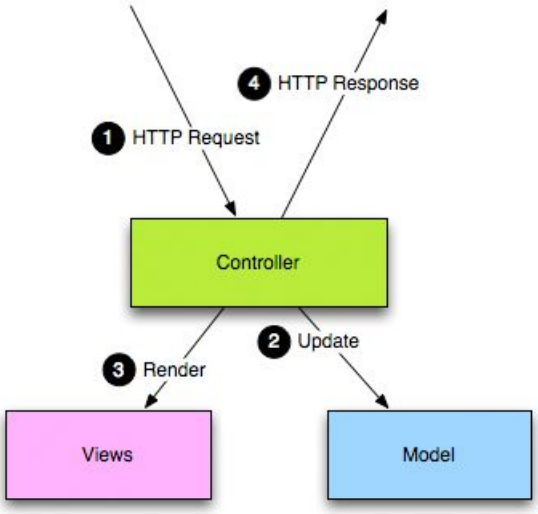
Lydia Bartels

Marlon Franz

Benedict Bihlmaier

The Model View Controller Pattern

As the Play Framework is set up to follow the MVC architectural pattern, we looked into how to efficiently divide tasks and responsibilities between those three parts.



Source: https://www.playframework.com/documentation/1.0/main

Model

The model.db package consists of java classes which represent a specific table or relationship in the database. These models allow easy caching and manipulation of the stored data, e.g. by not having to use MySQL Queries outside of the models. Creating models also enables the controller class to receive and use data without having to access the database directly, resulting in cleaner methods with less duplicate code, high flexibility and increased efficiency.

Implementation: The table “User” in our database schema consists of an user ID, a name and a password - creating a class “User” (with one integer and two Strings as attributes) enables us to construct Instances which include all data stored in the database about a user. Now with the use of SQL queries (MySQL in this case) we can either create a new “User”-Instance containing data from the database, or send a “User”-Instances (which was created in a controller for example) data to the database. Of course it allows us to delete or alter user data as well.

If the application benefits from a different arrangement of data then it is stored in the database, it is also feasible to create models which incorporate data from multiple tables, as it is the case with the “PlayedGame” class, whose Instances include a “User”-Instances’ “name” attribute which is required in one of the controllers.

View

Views enable the user to interact with the model through user interfaces. An example would be the login page which is rendered in the HTML format: The user navigates to the login page and is able to enter credentials in a form. The entered data is sent to a controller who decides what to do with it. In this case it calls methods of the modelled “User” class and checks whether the entered credentials exists in the database. Then it sends the View information to respond accordingly, e.g. redirect the user to the menu or display an error message.

Controller

Controllers respond to the actions the user takes using the interfaces provided by the views. This includes receiving and sending data, typically in the form of HTTP requests (e.g. objects or lists of objects in the JSON format which is commonly used for transferring data between views and controllers) and creating or applying changes to model objects.

Implementation: The “UserController” Class contains methods responsible for rendering a login- and a account-creation-page, as well as processing the data received from those two pages, namely login credentials and attempts to create a new account with a username and password. Upon navigating to a certain URL in a web browser, the according render method is invoked and displays a page. Now the user can interact with the page and send data back to the controller. If that data stems from the account-creation-page, the method “saveAccountToDatabase()” will be invoked. Here the controller makes sure that both the username and the password field were filled with input and that the entered username is not already taken - if that is the case, it calls a method from the “User” class, storing the new credentials in the database and re-renders the page with a success message.

Services

Services are not as precisely defined as the components of the MVC Pattern and are technically not a direct part of it. In our application the classes stored in the service package are used primarily to model objects which are not directly related to database tables. This includes classes like “AddFriendsForm” which only consists of a String, its getter and a constructor, but is necessary because of the way how the Play Framework handles Formdata sent from a view to a controller: A “FormFactory” Object handles the data it receives and stores it in an “Form” Object. To access the data from here, one needs to access the “get()” method of said Object and populate an “AddFriendsForm” Object with it. A service class which is used for a different task is “UserStats”: It combines attributes from different models needed for displaying a users and his selected friends statistics on the “Friends” page. The reason why it makes sense to create an own class for this combination is the convenient use of the Gson libraries “Gson().toJson(listname) “ method which creates a json-formatted String consisting of all elements of a list in a single line of code.