README.md

HttpServer Game Scores

Java application that keeps track of the user scores by level.

Compiling the application

Prerequisites:

- Java SE Development Kit 8
- Apache Maven (tested with version 3.3.3)

To compile execute the following command from the root of the project:

```
mvn clean package
```

Running the application

Quick start

To run the application with its default configuration you can use the following command:

```
java -jar target/httpserver-scores.jar
```

> Note: this works if the `jar` file is in the `target` directory.

To stop the server just `Ctrl+c`.

Configuration parameters

To customize the behaviour of the application the following parameters can be passed as arguments:

Parameter	Values	Required	Description
port	An integer value. Port numbers less than 1024 require root permissions.	No	Port number on which the application will listen for requests.
executor	`fixed` or `cached`	No	This parameter determines which strategy to use for HttpServer Executor, it can either be `newCachedThreadPool` or `newFixedThreadPool`. More info here.
poolSize	An integer value.	No	This only applies if `fixed` executor is selected.

For example:

```
java -jar target/httpserver-scores.jar -port=8080 -executor=fixed -poolSize=10
```

Endpoints

Login

	Value	Description
Path	`/<userid>/login`	Requests the creation of a new session key. The session key is valid for the amount of minutes configured in the server. A new session key is created every time the endpoint is called.
Method	`GET`	
Response	`<sessionkey>`	Unique string that represent the session.

Example:

```
curl http://localhost:8080/100/login -> 1B4EB7BE47F046E98E1DC458B80B2D2C
```

Score

	Value	Description
Path	`/<levelid>/score?sessionkey=<sessionkey>`	Method can be called several times per user and level. Requests with invalid session keys are ignored.
Method	`POST`	

Request Body	`<score>`	Integer number that represents the users score for the level.
Response		Empty response.

Example:

```
curl -X "POST" "http://localhost:8080/10/score?sessionkey=1B4EB7BE47F046E98E1DC458B80B2D2C" \
-d "2500"
```

Get high score list

	Value	Description
Path	`/<levelid>/highscorelist`	Retrieves the high score list for a level. The list size is determined by the Application configuration.
Method	`GET`	
Response	CSV of `<userid>=<score>`	Comma separated list with user id and scores.

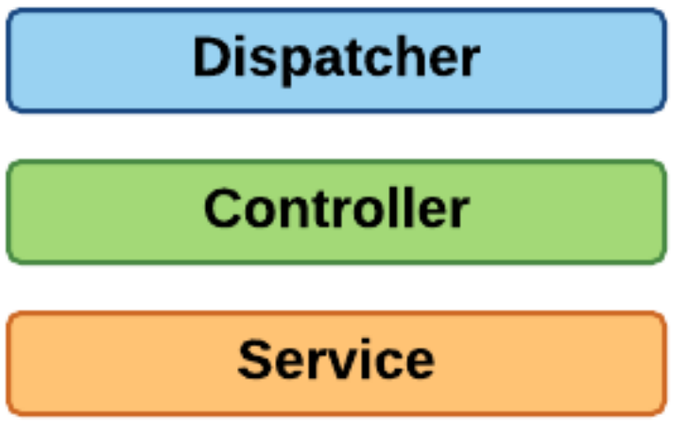
Example:

```
curl http://localhost:8080/10/highscorelist -> 100=2500
```

Technical Solution

Overview

The architecture of the application was made as simple as possible (following the KISS principle). It consists mainly of 4 layers.



- The Dispatcher is in charge of receiving the request and forwarding it to the appropriate controller.
- The Controller gathers the information that it requires for processing from the request and forwards it to the service.
- The Service applies the business logic to the received request.

Dependency Injection

The package `com.oresteluci.scores.injection` handles all the logig for automatic dependency injection. Two annotations are used for this purpose `@AutoComponent` and `@AutoInject`.

The `@AutoComponent` is a class level annotation that indicates that this class should be instantiated for later injection.

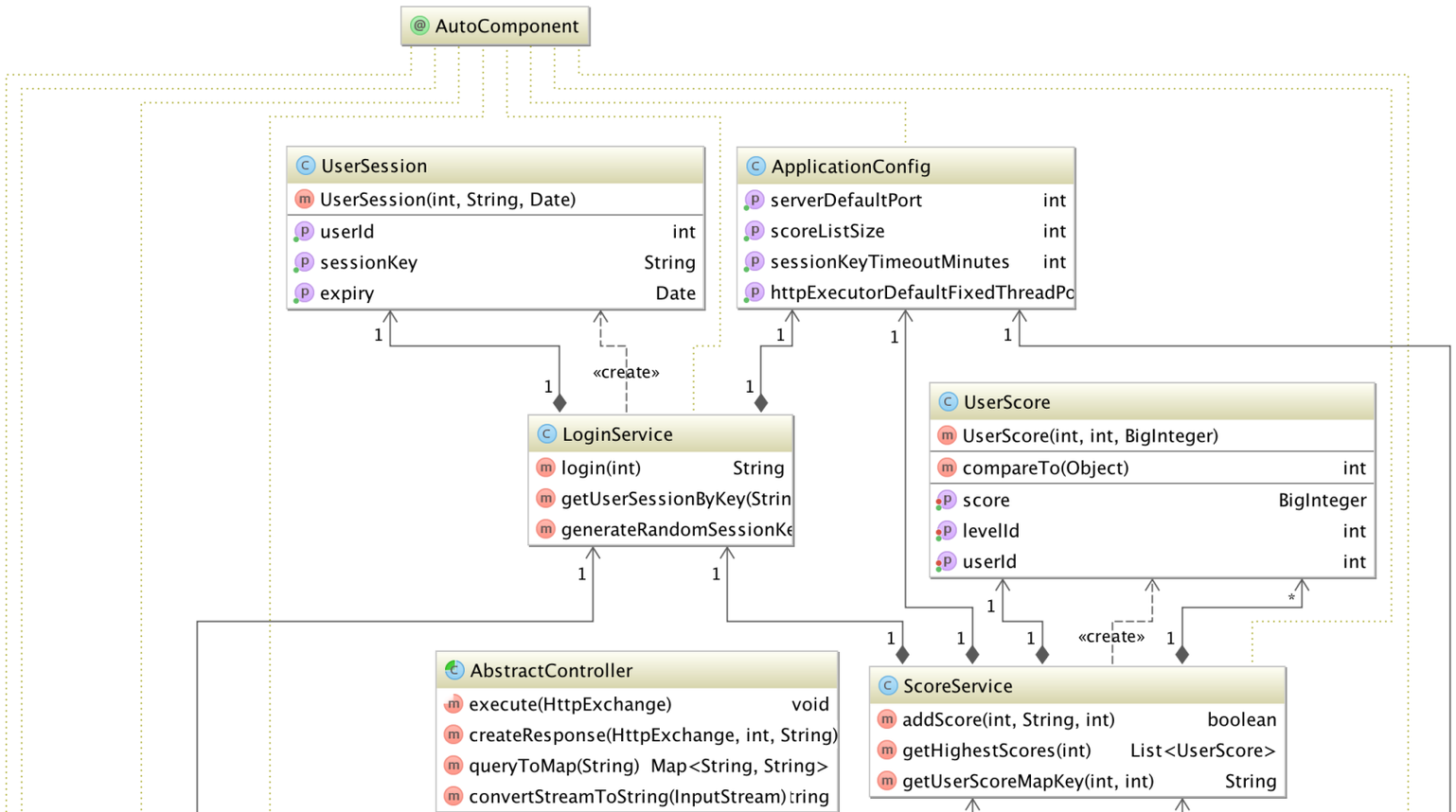
The `@AutoInject` is a field level annotation that indicates that an `@AutoComponent` should be injected in this field.

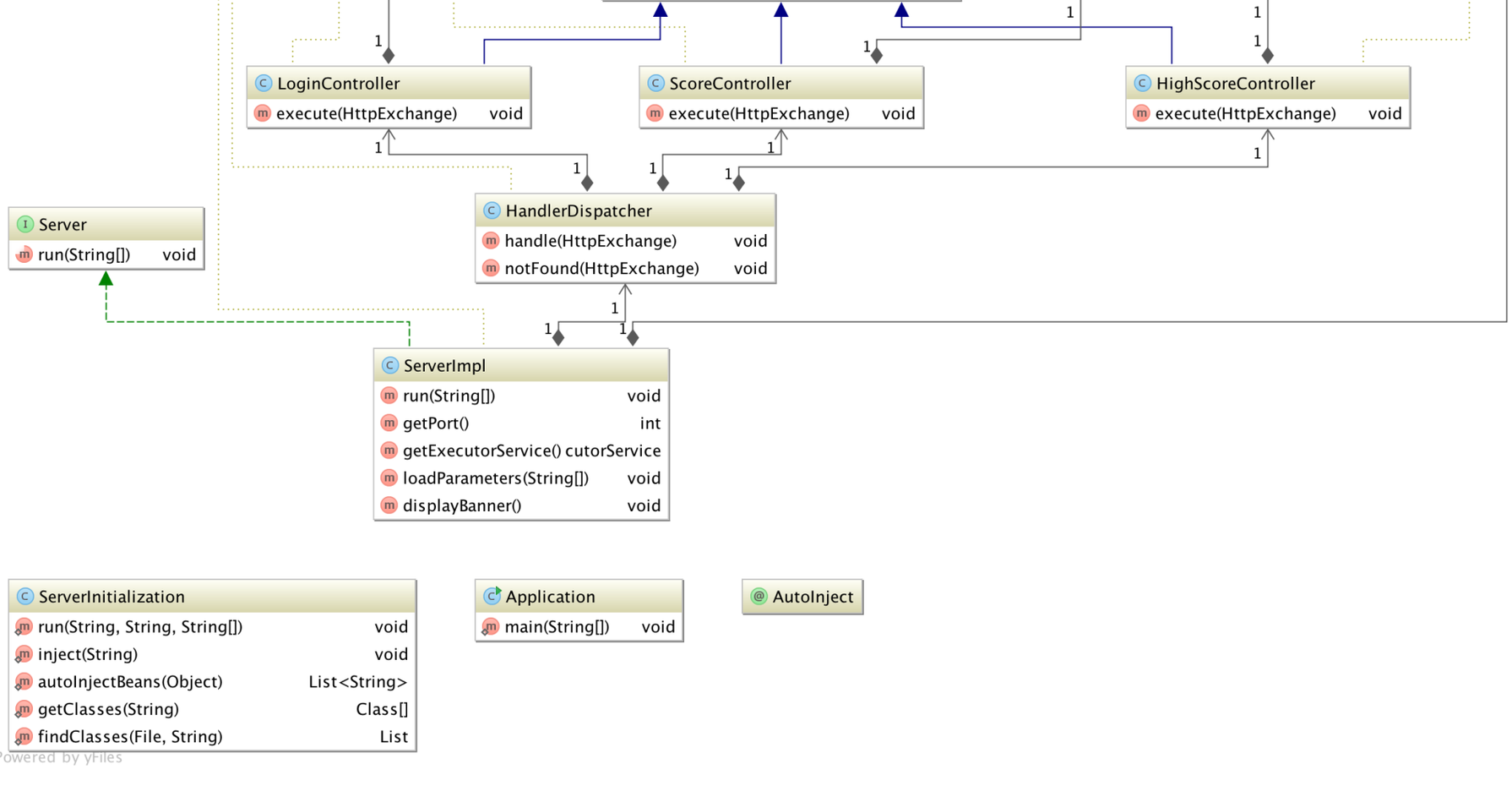
Data Structures & Concurrency

Data storage and concurrency is handled by the Service Layer. Simple denormalized data structures were used to storage and fast retrieval. Maps are used to store key/value pairs, this structure was chosen due to the high performance in getting values given a key.

In the cases where data needs to be found by value, additional maps were added with these values as keys. In this cases when updating a value, two maps need to be updated and concurrency is handled by locks.

The following diagram shows the relationship of the classes:





Improvements

The following improvements can be made to the solution:

- Removed expired session tokens with scheduler to reduce memory consumption. In the implemented solution tokens are created every time a login request is received and they are only deleted when a score post is made with an expired session key.
- More denormalization for better read performance.
- A distributed storage solution could be used to increase performace in response time and storage capacity, for example Cassandra.
- More Unit tests for better coverage.

Source Code

The source code can be found in https://github.com/Oreste-Luci/httpserver-gamescores.