MIGUEL LOPEZ

Building RESTful APIs with Akka HTTP & Scala

Codemunity

Akka HTTP RESTful APIs

Building Reactive RESTful APIs with Scala and Akka

Codemunity

This book is for sale at http://leanpub.com/akka-http-rest-apis

This version was published on 2020-03-08

Tweet This Book!

Please help Codemunity by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought the Building Akka HTTP RESTful APIs book!

Contents

Introduction

Hey there, welcome to the Akka HTTP RESTful APIs book!

This book will take a different approach, where only what's used will be explained, we don't want to get lost on every detail and we want to stick to the point. The idea is to take a real-world approach, where we will learn things as we use them, but don't worry we will specify resources for those who want to dig deeper into the different topics.

What Should I Know?

You should already be familiar with Scala, at a basic level at least, otherwise being familiar with functional programming basics and another object-oriented language should be enough. We will explain the more advanced stuff so don't worry. We also have a book[1] that introduces Scala at a beginner level.

If you have zero knowledge of Akka HTTP, we have a free course[2] where you'll build a simpler application, and the tool-specific concepts and explored in greater depth.

You definitely need to have done some programming, and you should be comfortable installing tools or at least following some tutorials, we won't cover tool installations as there are plenty of guides out there, we would have to cover at least Linux, macOS and Windows, and that would mean repeating ourselves, it's better to spend that time building something.

Everything in this book was developed using a Mac, so if you have any problems let us know[3] and we'll sort them out.

What Are We Going to Build?

In this book, we will build a REST API for a bookstore, and the whole point is to provide a base project for you to build your own APIs using Akka HTTP, we will cover project creation and setup, routing, `one-to-many` and `many-to-many` database relationships, storage with PostgreSQL and Slick, model migration using Flyway, user authentication… This might seem like a lot, but we'll cover all those topics gradually, and each chapter will build on the previous one.

Let's start with the requirements to have a clear picture of what we need to build:

- Users should be able to register.

[1] http://bit.ly/2vraixB
[2] http://link.codemunity.io/lb-akka-http-quickstart
[3] https://www.codemunity.io/contact

- Users should be able to buy books.
- Users should be able to login and see the books they have purchased.
- Users can buy more than one of the same book.
- Users should be able to find books by their category.
- Admins should be able to add, list, update and delete books categories.
- Admins should be able to add, list, update and delete books.

We will build most of our API using a TDD approach, at least after we cover the integration with our third-party libraries, which we are not supposed to test directly anyway.

What Are We Going to Use?

Let's cover the technologies we'll use to build our bookstore:

- Database: PostgreSQL
- Web Framework: Akka HTTP
- Programming Language: Scala
- Build Tool: SBT
- IDE: IntelliJ Community Edition (this is optional, you can use whatever you prefer)

The libraries will be covered when we include them in our project.

For reference, or if you want to check out the code, you can find the repository here: https://github.com/Codemunity/akkahttp-bookstore.

Project Setup

As mentioned already, we'll use the free version of IntelliJ, we won't be doing anything fancy though, if you want to follow along with another IDE feel free to do so.

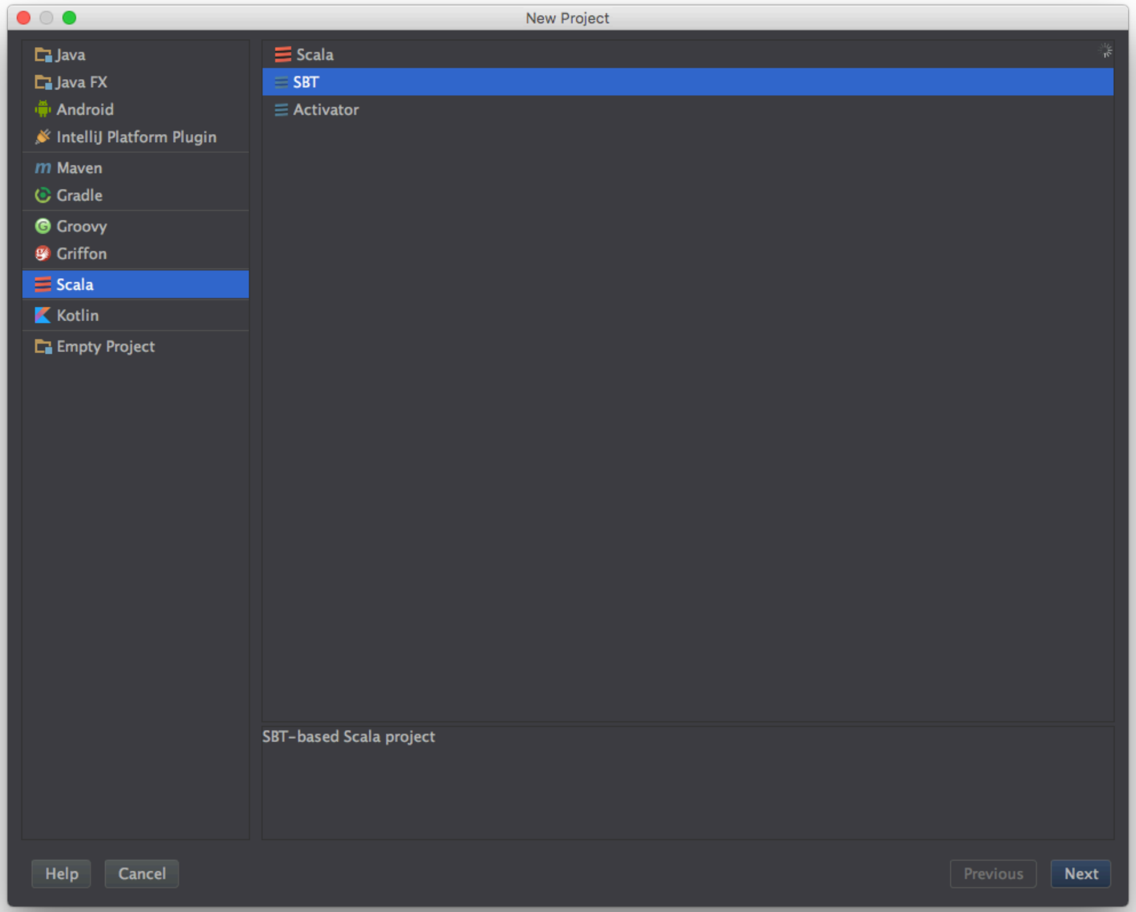
By this point you should already have installed and setup Scala, SBT, the IDE of your choice, Git. Make sure that the Scala Plugin is installed in IntelliJ.

In case you don't want to create the project and set it up manually, you can clone the repository[4] and checkout the `project-setup` branch. This way you only need to import the project in your IDE and move on to the next chapter.

Creating Our Bookstore Project

Open IntelliJ, go to `File -> New -> Project...`, choose `Scala` and create an SBT-based Scala project.

[4]https://github.com/Codemunity/akkahttp-bookstore



SBT-based Scala project

Project Creation Screen

Now, hit Next and it's time to enter some information.

Project name: AkkaHTTPBookstore

Project location: <Enter here your project destination, this folder will contain the project files, so create one if needed>

Project SDK: <Choose your Java SDK, ours is Java 1.8>

SBT version: <Same here, choose your SBT version, we'll use 0.13.16>

Scala version: <We'll use Scala 2.12.6, the latest one at the time of writing>

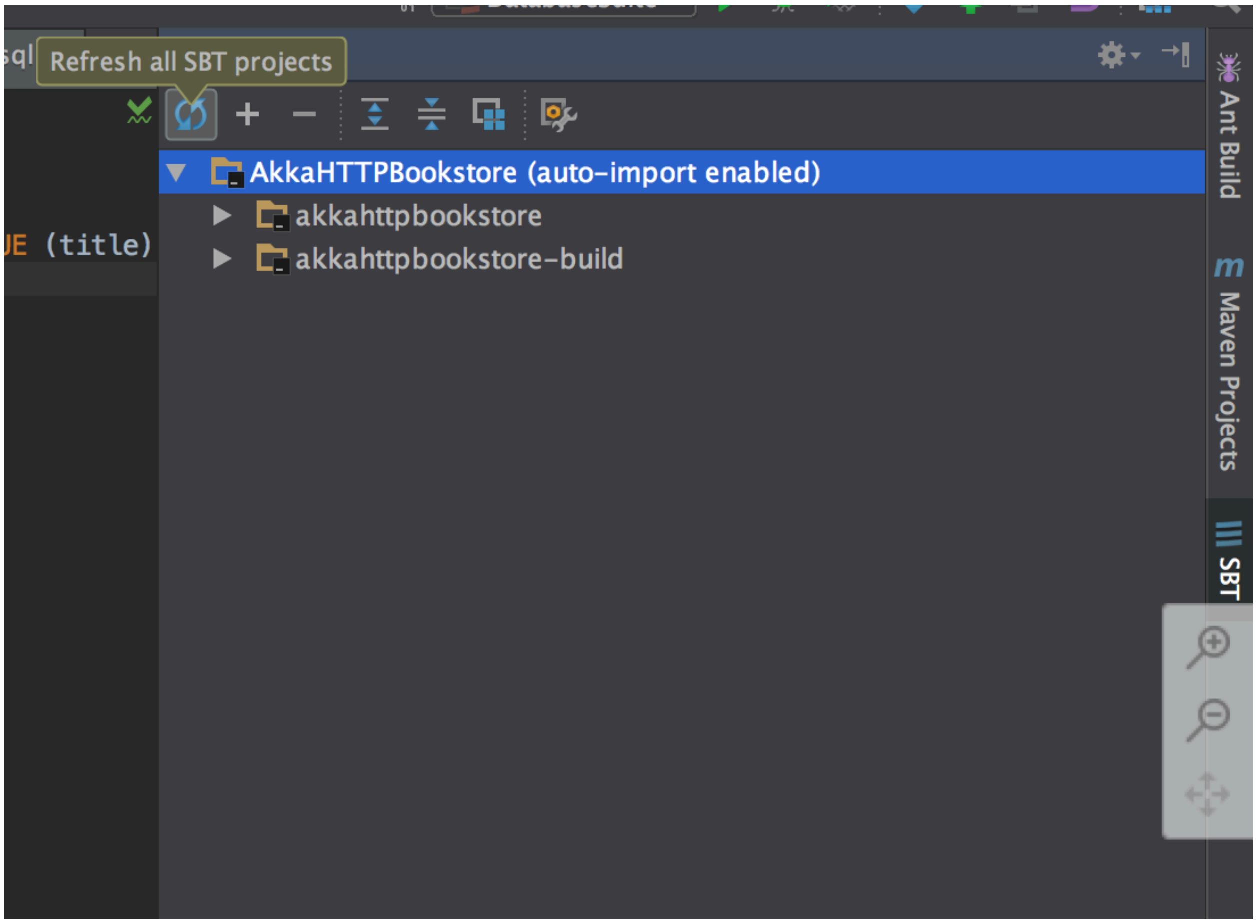
Check the `Use auto-import` option, so whenever we make changes to our `build.sbt` file it'll update our project automatically. Now click `Finish`.

Adding Our Project Dependencies

Let’s add and review the dependencies we’ll need for this project. Open the `build.sbt` file and update its contents to the following:

```
name := "AkkaHTTPBookstore"

version := "1.0"

scalaVersion := "2.12.6"

val akkaVersion = "2.5.14"
val akkaHttpVersion = "10.1.4"

libraryDependencies ++= Seq(
  "com.typesafe.akka" %% "akka-actor" % akkaVersion,
  "com.typesafe.akka" %% "akka-stream" % akkaVersion,
  "com.typesafe.akka" %% "akka-http" % akkaHttpVersion,
  "com.typesafe.akka" %% "akka-http-testkit" % akkaHttpVersion % Test,
  "com.typesafe.akka" %% "akka-http-spray-json" % akkaHttpVersion,
  "com.pauldijou" %% "jwt-core" % "0.17.0",
  "org.scalatest" %% "scalatest" % "3.0.5" % Test,
  "com.typesafe.slick" %% "slick" % "3.2.3",
  "com.typesafe.slick" %% "slick-hikaricp" % "3.2.3",
  "org.postgresql" % "postgresql" % "9.4-1201-jdbc41",
  "org.slf4j" % "slf4j-nop" % "1.6.4",
  "org.flywaydb" % "flyway-core" % "3.2.1",
  "com.github.t3hnar" %% "scala-bcrypt" % "3.1",
  "com.lihaoyi" %% "scalatags" % "0.6.7"
)
```

By saving the document, it should download and add them automatically, if it doesn’t (maybe you didn’t check the auto update option for SBT), you can open the `SBT` tab that’s at the right and hit the `Refresh` button as shown below:



Refresh SBT

Dependencies Overview

All the `akka` related ones are required for Akka HTTP to work, we'll use the `spray-json` library for JSON serialization/deserialization, and the `testkit` library provides some helpers that ease unit testing Akka HTTP applications.

The next library is `jwt-core`, in case you are not familiar with JWT[5] it stands for *JSON Web Tokens*, and it's the standard to authenticate clients in REST APIs.

If you are familiar with the Scala ecosystem, `scalatest` won't need an introduction, it's our preferred unit testing framework, it provides several styles to write your tests.

`slick` is a functional relational mapper for Scala, it allows you to query your database as if you were using a collection. Don't worry, we'll start using it and you'll see how great and easy to use it is. We'll provide a cool open source mini course where you can delve deeper into it in case you're interested.

Because we'll use PostgreSQL, we have to include it's connector.

[5]https://jwt.io/

The last important library is `flyway`, which allows us to use database migrations, in case you're not familiar with migrations you can think of them as versioning for your database, so you can rollback or move forward through different versions. Again, we'll look into it soon.

Setting Up Postgres for Development

We'll use Docker and Docker Compose to setup an instance of Postgres. Make sure you have Docker installed in your machine[6], it'll have Docker Compose bundled.

Create a new directory called `docker` inside the project's root directory, and create another directory called `postgres` under the new `docker` one.

We will need a simple script to create the two databases we will use, one for development and another one for testing. Create a new file called `create_db.sql` under `docker/postgres`:

```sql
CREATE DATABASE bookstoredb;
CREATE DATABASE bookstoretestdb;
```

Next, we will create a `Dockerfile` in the same folder:

```dockerfile
FROM postgres:9.4

ENV POSTGRES_PASSWORD bookstore
ENV POSTGRES_USER bookstore

ADD create_db.sql /docker-entrypoint-initdb.d/
```

We're using the official Docker image for `postgres`. We set some environment variables so we can access the database, and we add the script we created earlier under the `/docker-entrypoint-initdb.d` directory, which the `postgres` image reads to execute all the `sql` scripts that in there.

Last, we will create a `docker-compose.yml` file under the `docker` folder:

```yaml
version: '3'
services:
  bookstore-postgres:
    build: ./postgres
    ports:
     - "5432:5432"
```

This file allows us to simplify our commands to run and stop `postgres`. It will build the `Dockerfile` and expose the necessary port for us to access it.

If we run `tree docker` in the project's root directory, the directory structure should look like:

[6] https://docs.docker.com/install/

```
$ tree docker
docker
├── docker-compose.yml
└── postgres
    ├── Dockerfile
    └── create_db.sql
1 directory, 3 files
```

Now let’s test what we’ve done. In a terminal, make sure you are in the `docker` directory and run `docker-compose up`, you should see the following output:

```
...
postgresbookstoretestdb_1  | LOG:  MultiXact member wraparound protections are now e\
nabled
postgresbookstoretestdb_1  | LOG:  database system is ready to accept connections
postgresbookstoretestdb_1  | LOG:  autovacuum launcher started
```

The important line is `database system is ready to accept connections`, which means we are ready to move on!

Conclusion

Up to this point, we can confidently say that we're now familiar with Akka HTTP, we've done several things you can find in a real-world application, the only thing our application needs is time and usage, with those elements it'll definitely grow into a "real" system.

We hope that you've enjoyed the book so far, this does not mean that the book is finished, it might never be finished, we will including minor updates for non-breaking versions of Akka, otherwise we would have to basically rewrite the whole book.

If there's something you think is missing that is vital for your learning let us know, we're always willing to provide extra resources or update an explanation if that improves your learning experience.

Additional Resources

Before we say goodbye, we wanted to share with you additional resources that might be of your interest:

Books

We will only add books that we've read, we have no affiliation or direct relationship with the publishers.

- **Akka in Action**[7] **book**: a great introduction into the whole Akka framework, you'll get to see a bit of Akka HTTP, Akka actors in general, Akka Streams and Clustering as well.
- **Functional and Reactive Domain Modeling**[8] **book**: want to learn how to architect functional applications? This is the book we recommend!

Documentation

- **Akka Official Documentation**[9]: it's the official documentation, it has to be included… No but really, it is useful and we often find ourselves checking it.

[7]https://www.manning.com/books/akka-in-action
[8]https://www.manning.com/books/functional-and-reactive-domain-modeling
[9]http://akka.io/docs/

Frameworks

- **Lagom Framework**[10]: an opinionated framework, if you're interested in microservices, Lagom is definitely worth a look.
- **Squbs**[11]: a framework based on Akka and Akka HTTP, it ships different tools needed for large scale applications, their repo reflect an active project and nice amount of stars.
- **Colossus**[12]: is a lightweight I/O framework for building Scala services made by Tumblr, their repo is quite active and have a nice amount of stars.

We will be updating the list above as we find useful resources, if you know of something worth adding, let us know!

See you around!

We want to make something clear, even though the book is technically finished, we want it to be continually updated as long as there's still interest from you, let us know if there's something missing or if something can be improved.

If you liked the book, please share it with your peers, the more people we have, the more we can invest time into improving the book and answer to requests.

We also want to congratulate you on coming this far down the road, we know it's not easy but it sure is rewarding

Last but not least, we want to thank you for joining us in this learning path of Akka HTTP, and trusting us to deliver that knowledge to you, we hope that you liked the content of our book and that you also want to keep in touch with us, we are always working in new content, the end of this book means the begin of another one. We also offer tutorials[13] and we are starting with our courses[14] as well.

We really hope to hear from you, what you are interested in, what your current goals are or just stop by and say hi!

We'll definitely see you around, and happy coding!

[10]https://www.lagomframework.com/documentation/latest/scala/Home.html
[11]http://paypal.github.io/squbs/
[12]http://tumblr.github.io/colossus/0.9.1/
[13]https://www.codemunity.io/tutorials
[14]https://www.codemunity.io/courses