kahawi

November 12, 2025

Abstract

Kahawi is a web framework written in kotlin. It is designed to implement the traditional components of an MVC framework, along with some conveniences like database migrations and basic generators. Implemented in a few days, it has some maturing to do. However, along with the given components, it ships with a local server and cli to allow for quick iteration locally before deployment. If you prefer development in docker, a definition file has been provided as well.

se:ko!

The title of this section is the Mohawk word for welcome. We're excited you've dropped by and hope you will take the framework for a spin. The goal is to provide a starting point for further forays into web development. Kotlin is a wonderful language with a heritage few others can boast: it rests on top of the may years of Java progress, not the least of which is the JVM (which is one of the most well researched platforms in academia).

routing

The core of any framework is the mapping of URLs to the actions behind them. We define the actions in controllers (see below), but we map the urls in the *routes* file.

The routes file can be found alongside the controllers in src\main\kotlin\app\controllers\routes.kt, and has the following structure:

```
package app.controllers
class Config() : BaseConfig() {
    fun initRoutes() {
        try {
            // -- Define our routes here --
            get("/", "Home")
            //...
        } catch (e: Throwable) {
            println("Error in initRoutes:\n\n${e::class.simpleName}: ${e.message}")
        }
    }
}
```

[src\main\kotlin\app\controllers\routes.kt]

We can see the structure clearly enough: the get calls create mappings of the given url, pointing it to the corresponding controller prefix (in our case, this maps to the HomeController). If no action has been defined (as in our case), the method called with be “index”.

models

Controllers will define the actions to take in response to requests as outlined in the routing section, but in order to do anything interesting we will need access to a database. As in scores of other frameworks, this is done via models. In our case, we make use of the Exposed library that already does this nicely.

The code will look something like the following:

```
package app.models
import org.jetbrains.exposed.sql.*
// Define a simple table
object Users : Table() {
    val id = integer("id").autoIncrement()
    val name = varchar("name", length = 50)
    val email = varchar("email", length = 50)
    val password = varchar("password", length = 50)
    val age = integer("age")
    override val primaryKey = PrimaryKey(id)
}
[ src\main\kotlin\app\models\Users.kt ]
```

This model provides a wrapper for all columns in the database table Users. The purpose is two-fold: this gives the structure and table name that migrations use to define tables in the database, but also lets the application know what attributes are available from the objects in the database for our CRUD (Create, Read, Update & Delete) operations.

controllers

As we have alluded to, controllers are the focal point of the application. They are where the bulk of the *logic* is defined (in most cases). They look something like this:

```
package app.controllers
import app.framework.*
class HomeController() : BaseController() {
    fun index(): String  {
        return renderTemplate("hello")
    }
    // ... other methods
}
[ src\main\kotlin\app\controllers\HomeController.kt ]
```

Here we can see the HomeController with method “index” that is referenced in the routes file above. Note that in this case, we are returning a “view” named *hello*. See below for how this is defined.

views

Views define the HTML returned to the user, called in the controller methods. They will look very familiar:

```
<html>
  <body>
    Hello!
  </body>
</html>
[ src\main\kotlin\app\views\home\hello.ftl ]
```

Views will usually also contain variable values, passed by the controller.