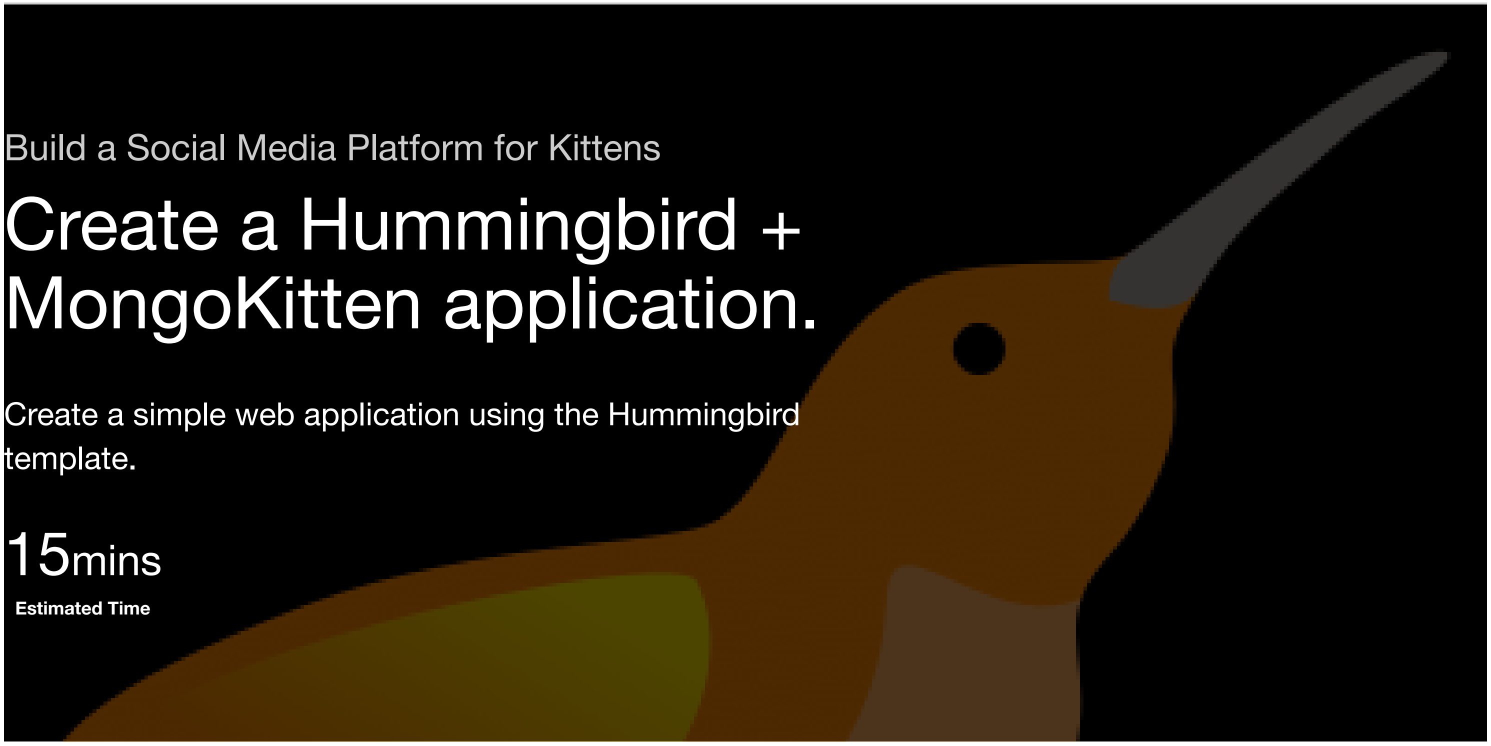

Build a Social Media Platform for Kittens

Create a Hummingbird + MongoKitten application.

Create a simple web application using the Hummingbird template.

15mins

Estimated Time

Section 1

Create your project

Clone the Hummingbird template, configure a project and review the contents of the generated project.

Step 1

Clone the Hummingbird template GitHub project

Clone template

No Preview

```
> git clone https://github.com/hummingbird-project/template
```

Step 2

Create your project, using the template configure script. Press return on each question to use the default value.

Step 3

Add the `MongoKitten` dependency.

MongoKitten doesn't need any special treatment. It works seamlessly with Hummingbird.

Section 2

Setup MongoDB

MongoDB is a serivce that needs to run alongside your application. It’s a prerequisite for running MongoKitten.

Step 1

Install and Run Docker if it’s not running already.

Then, run this command to download and run MongoDB.

Terminal.app

No Preview

```
docker run -d --publish 27017:27017 mongo
```

Section 3

Add MongoKitten

With your Package.swift and database set up, lets add MongoKitten to your project.

Step 1

Open `Sources/App/Application+build.swift`.

Add the MongoKitten dependency, and modify the `AppArguments` to contain two new variables.

Step 2

Open `Sources/App/App.swift`

This contains an App type conforming to `AsyncParsableCommand` with three options,

Sources/App/Application+build.swift

```swift
import Hummingbird
import Logging
import MongoKitten

/// Application arguments protocol. We use a protocol so we can call
/// `buildApplication` inside Tests as well as in the App executable.
/// Any variables added here also have to be added to `App` in App.swift
/// `TestArguments` in AppTest.swift
public protocol AppArguments {
    var connectionString: String { get }
    var hostname: String { get }
    var port: Int { get }
    var logLevel: Logger.Level? { get }
}

// Request context used by application
typealias AppRequestContext = BasicRequestContext

///  Build application
/// - Parameter arguments: application arguments
public func buildApplication(_ arguments: some AppArguments) async throw
    let environment = Environment()
    let logger = {
        var logger = Logger(label: "MeowSocial")
```

the hostname and port are used to define the server bind address, logLevel sets the level of logging required. Finally the run() function which calls buildApplication(_:) to create an Application and then runs it using runService(). You can find out more about the argument parser library here.

Step 3

Add the new app arguments with default values.

Step 4

Open Sources/App/Application+build.swift again. We can now start the MongoKitten driver.

This will connect and login to the server immediately. If any network hiccups occur, MongoKitten reconnects automatically.

```swift
        logger.logLevel =
        arguments.logLevel ??
        environment.get("LOG_LEVEL").flatMap { Logger.Level(rawValue: $0
            .info
        return logger
    }()
    let router = buildRouter()
    let app = Application(
        router: router,
        configuration: .init(
            address: .hostname(arguments.hostname, port: arguments.port)
            serverName: "MeowSocial"
        ),
        logger: logger
    )
    return app
}

/// Build router
func buildRouter() -> Router<AppRequestContext> {
    let router = Router(context: AppRequestContext.self)
    // Add middleware
    router.addMiddleware {
        // logging middleware
        LogRequestsMiddleware(.info)
    }
    // Add default endpoint
    router.get("/") { _,_ in
        return "Hello!"
    }
```

Section 4

Add Kittens API

Add your database models and routes to edit them.

Step 1

Sources/App/Kitten.swift

No Preview

```swift
import MongoKitten
import Hummingbird

struct Kitten: ResponseCodable {
    static let collection = "kittens"

    let _id: ObjectId
```

Create a file named `Kitten.swift`, and add the following data model.

MongoKitten always requires a stored property named `_id`. This is used by MongoDB as the unique ID.

```swift
    var name: String
}
```

Step 2

If we look further down the file we can find the `buildRouter()` function.

Here we create the `Router`. We add a logging middleware to it (this logs all requests to the router). The function uses a result builder to create a stack of middleware, but you can also use `Router.add(middleware:)` to add individual middleware. Finally we add a single endpoint GET / which returns “Hello!”

Step 3

We’ll add a single route `GET /kittens`, which lists all registered kittens. This requires passing in the database handle to your routes.

Because the database is empty now, we’ll add a route `PUT /kittens` to add your own kittens.

Section 5

Test your Backend

Now that your MongoDB backend is complete, it’s time to validate the results!

Step 1

We can run this application and use curl to test it works.

First, add your own kitten!

Step 2

Then, query the list of kittens.

Step 3

Now we have a running server, lets add some functionality to it.

Test Application

No Preview

```
> curl -i -X PUT -H "Content-Type: application/json" -d '{"name":"Milo"}
HTTP/1.1 201 Created
Content-Length: 0
Date: Sat, 23 Nov 2024 09:22:26 GMT
Server: MeowSocial
```