Flexwork Documentation

ECE590-05 Mobile Application Development
2016 Fall
Sponsor: Ric Telford
Team Veggies:
Yuchen Zhou
Jiasheng Yang
Jianyu Zhang

Overview

Flexible is a flexible RESTful backend framework which provides simple backend solution for app developers. We run a server which receives HTTP request, handles database query and sends HTTP response back to the client with the data. Developers can define the query fields they need and make request as they want with UserAPI.

User Guide

You can download Flexwork through gitlab.

User API

You can find our User API in ~/project/backend/Sources/UserAPI.swift file.

We provide several API functions for Flexwork users.

1. Test: to test connection to Flexwork

```
func test(completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())
```

2. Get: get data from Flexwork

```
func get(dbName: String, collectionName: String, completion: @escaping (_ data: Data?,
    _ response: URLResponse?, _ error: Error?) -> ()) {

func get(dbName: String, collectionName: String, operation: Comparison, field: String, value: String,
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ()) {

func get(dbName: String, collectionName: String, filters: [[String]],
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())
```

3. Put: update data in Flexwork

```
func put(dbName: String, collectionName: String, operation: Comparison, field: String, value: String,
    newItem: [String:Any], completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())
```

4. Post: create new key value pairs in Flexwork

```
func post(dbName: String, collectionName: String, item: [String:Any],
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())
```

```
func post(dbName: String, collectionName: String, items: [[String:Any]],
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())
```

5. Delete: delete data in Flexwork

```
func delete(dbName: String, collectionName: String,
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ())

func delete(dbName: String, collectionName: String, operation: Comparison, field: String, value: String,
    completion: @escaping (_ data: Data?, _ response: URLResponse?, _ error: Error?) -> ()) {
```

You can find more examples about how to use Flexwork User APIs in our demo app.
Note: Please use Dispatch Queue to do asynchronous update for your UIView.

Back-end

You need to set up the the following fields for the back-end in main.swift before running the server:

1. A dictionary. Key is database name. Value is also a dictionary with collection name as its name and collection configuration as its value. This is used for defining the database, collection and field type of the collection. CollectionConfiguration is a class encapsulating each fieldname and fieldtype of each collection. Because Flexwork is intended to be generic and allows developer to define the field they need, we support almost all the fieldtype supported by MongoDB.
2. DatabaseConfiguration. DatabaseConfiguration contains the information necessary for connecting to the MongoDB server including host address, port number, username and password.
3. Please make sure the database is existing in the MongoDB before running the server.

Following is an example code to set up flexwork. A developer wants to have a database called “veggies_info_db”, collection called “veggies_info_collection”. “veggies_info_collection” contains three fields: “name” field with type string, “team” field with type string, “age” field with type int32.

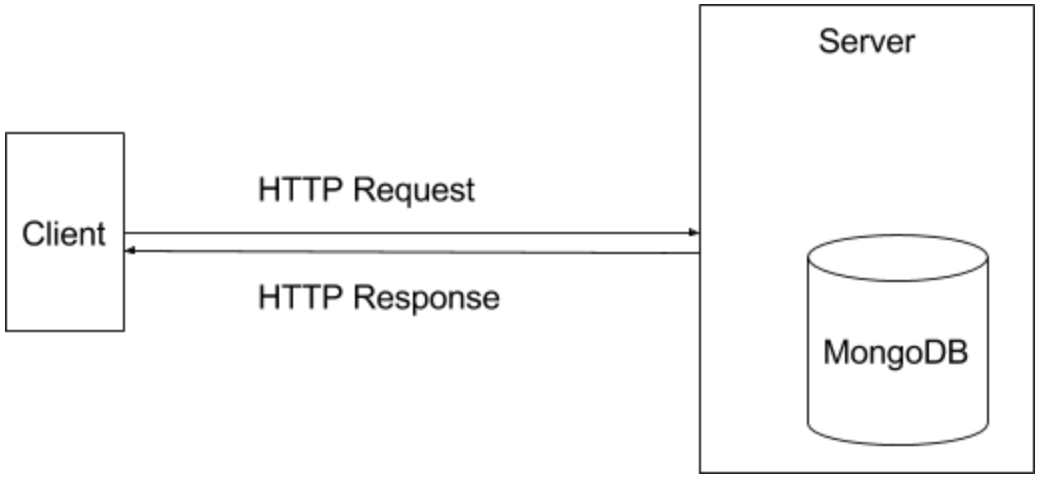
```
import Kitura
import MongoKitten

// database we want to use.
let databaseName = "veggies_info_db"

// collection in the database we want to use.
let collectionName = "veggies_info_collection"
// first fieldname is "name", fieldtype is string
let fieldName_1 = "name"
let fieldType_1 = FieldType.string
// second fieldname is "team", fieldtype is string
let fieldName_2 = "team"
let fieldType_2 = FieldType.string
// third fieldname is "age", fieldtype is int32
let fieldName_3 = "age"
let fieldType_3 = FieldType.int32
let collectionConfig = CollectionConfiguration(collectionName: collectionName)
// add three fields into the collectionConfig
collectionConfig.addNewFieldType(fieldName: fieldName_1, type: fieldType_1)
collectionConfig.addNewFieldType(fieldName: fieldName_2, type: fieldType_2)
collectionConfig.addNewFieldType(fieldName: fieldName_3, type: fieldType_3)
// information about the database.
let dictionary = [databaseName: [collectionName: collectionConfig]]
// database configuration, host: 127.0.0.1, port: 27017, no specified username and password
let dbConfig = DatabaseConfiguration(host: "127.0.0.1", port: UInt16(27017), username: nil, password: nil)
// get an instance of flexwork by passing the DatabaseConfiguration and database dictionary
let flexwork = Flexwork(dbConfig, dictionary: dictionary)
// set up controller
let controller = FlexworkController(backend: flexwork)
// set up server
Kitura.addHTTPServer(onPort: 8090, with: controller.router)
// run server
Kitura.run()
```

Architecture & Design

Back-end



The overall back-end is implemented with RESTful architecture. Communication between client and server is mainly through sending configured HTTP request and response. Developer can use UserAPI to send the corresponding HTTP request they want and get the response from the server. To be specific, developer can use UserAPI to send GET, POST, DELETE, PUT request with query fields to get the data they want.

The http server we use is Kitura. Kitura is lightweight web framework that allows you to build web services with complex routes. We define the routing rule to route the GET, POST, PUT, DELETE request sending to the server and sending the data back through http response.

Test

Unit test is implemented for Flexwork.
There is an internal bug in swift on Linux. If you want to run the unit test by “swift test”, remember to remove the main.swift in Sources folder. More information can be found at: https://bugs.swift.org/browse/SR-1503

You need to add test_db to your MongoDB and also configure the database configuration in TestFlexwork.swift before running the test.

TestFlexwork mainly mocks the interaction between the application server and MongoDB and check the get(), insert(), find(), count(), delete() and update() functions work properly.

Appendix

You can check our Demo App in ~/frontend/ as an example of how to use Flexwork in your app development.