International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161
Available at http://inpressco.com/category/ijcet

*Research Article*

# Information system using Dart and MongoDB

**Siddaharth Suman†*, Shrikar Chonkar†, Shreyas Sawant† and Shikha Moondra†**

†Department of Computer Engineering, Atharva College of Engineering, Marve Rd, Malad (West), Mumbai-95, Maharashtra, India

Accepted 01 May 2015, Available online 05 May 2015, **Vol.5, No.3 (June 2015)**

***Abstract***

*An information system is a set of information collection and processing components used for providing information and knowledge. They are seen and used by everyone at everywhere for e.g. banks, restaurants, education portals, ticket reservation centers, etc. Objectory and Mongo.dart are both helper libraries for interacting with MongoDB and manipulating it. While Mongo.dart is helpful for establishing basic connection with the database, Objectory provides a more structured approach towards interacting with the database and realizing them into data models. The information system discussed in this paper is an implementation of a restaurant system built using latest tools for developing modern web based applications. The system is reliable, robust, scalable, flexible, simple and fast.*

***Keywords:*** *Dart, MongoDB, Mongo-Dart, Objectory, Angular-Dart, Web Components, Rikulo Stream*

## 1. Introduction

Internet today has come very far from where it started first. We have come to adopt it in most parts of our lives now. Almost everything that we do today is somehow connected or acquired from the internet. Allowing people to accommodate these things into their lives would not have been possible without the evolution of the modern web.

The modern web didn't evolve in a day. It took years to reach to this place. The technologies have also developed with it over these years. What was before just a simple collection of static web pages is now fully automated and heavily featured virtual machine, where we are not just able to check email, but also play games, watch videos, listen to music and other things.

The web is still ongoing some changes and getting better. The evolution of Dart and MongoDB has opened possibilities to developing web applications better and faster than what we currently have today. Here we have an implementation of said technologies on a normal information system that is able to perform much better and faster than the similar systems based on old technologies.

## 2. Dart with MongoDB as a server

Taking a step ahead of JavaScript we find Dart. Dart is a new object oriented, end to end, structured, optionally typed language for the web. Migrating to Dart from any other web language is easy. It doesn't matter if you come from a JavaScript background (non typed) or from a Java background (strictly typed), it would be easy to develop in Dart because Dart supports optional typing, a feature which allows the developer to either give a data type to the object he/she is creating or keep it as 'var'.

Migrating to Dart also has an advantage, the current web applications are focused on the user experience (UX) the technologies do not focus much on the developer experience (DX). Dart on the other hand gives a rich developer experience by being equipped with tools required for developing any modern web application. Normally the server and client side in a modern web app are both developed using different technologies. In case of Dart, the developer doesn't have to switch languages and can continue developing in the same language whether to create a server or client.

Keeping the evolution of the web applications in mind, it is always necessary to have a back-end that is able to withstand the growing possibilities and capabilities in web applications. (Shrikar Chonkar et al, 2015)

Developing a server side on Dart is important for serving data from the disk to the client when requested and it helps in easily following the model-view-controller (MVC) architecture. The main task on server was to set up an http request listener and a connection to the MongoDB host. "MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling." (*www.docs.mongodb.org*)

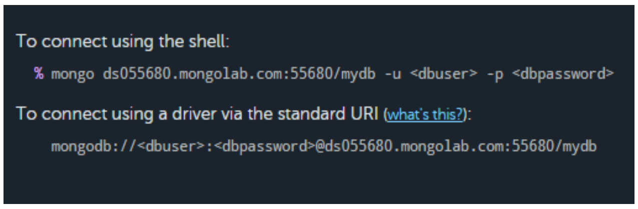
While developing the server side initially, it was discovered that the dart:io library cannot be used

*Corresponding author: **Siddaharth Suman**; **Shikha Moondra** is working as Assistant Professor when using the dart:html and vice-versa. This made sense as server should only deal with doing the disk read and write operations and the web browser part should be kept separate.

When dealing with the server side, there were two things that were being worked upon:

### *2.1 Establishing connection with MongoDB host*

Establishing communication with the database required the use of Mongo.dart, a database driver of MongoDB written in Dart. The driver requires the URL of the database which follows the syntax:



To connect using the shell:

```
% mongo ds055680.mongolab.com:55680/mydb -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds055680.mongolab.com:55680/mydb
```

**Fig.1** Syntax of database connection for MongoDB at mongolab.com (*www.mongolab.com*)

At the start, the MongoDB host was on the local machine and not on a remote server. After providing the URL, the next job was to open the database. Since opening a database here is a network operation, it will block the main thread until it opens. So, in this way, it is always better to open the database connection asynchronously, which is very well implemented in Mongo.dart with help of a concept in the Dart language called a Future.

A Future is used to represent a potential value, or error, that will be available at some time in the future. Receivers of a Future can register callbacks that handle the value or error once it is available. (*api.dartlang.org/apidocs*)

Therefore, upon opening of the database, the then() method was used on the open() function returning a Future object. In this way, the database can open up whenever it can and upon doing that, the program can perform database related tasks like querying the database and fetching required information and modifying or processing it.

### *2.2 Serving requests for access to the database*

As said previously, since the dart:io and dart:html packages cannot be used in the same program, sending result from the database is not straight forward but is made to be structured by allowing the client to send requests to the server and then getting the results back as a response. In this case, the requests would come from Objectory, it is a dart package that provides typed, checked environment to model, save and query data persisted on MongoDb. (*github.com/ vadimtsushko/objectory*)

Objectory is the dart package which allows us to read the data from MongoDB without the overhead of mapping the obtained json result into a proper data model. It also removes the overhead of type checking for various properties inside a document of a collection.

For serving database requests, the Rikulo Stream Server was used. Rikulo Stream is a Dart web server supporting request routing, filtering, template technology, file-based static resources and MVC design pattern. (*rikulo.org/projects/stream*)

## 3. Dart with MongoDB at client side

After being able to set up the server, the next step is obviously to set up a client. For this, the first step is to establish a connection with the server which is serving database requests.

Objectory provides a helper class named ObjectoryWebsocketBrowserImpl. This class has a parameterized constructor which takes three things, address of the server with port as a String, a function which registers the data models of the collections residing inside the database and a boolean value for if the collection should be dropped when the connection is established with the database. The constructor returns an Objectory object which is then used to perform queries from the client side and gives the result to be used on the browser.

Since the basic requirements of the project were met, that is, establishing connection with the database and serving requests from the client, the next step was to implement the model-view-controller architecture with help of AngularDart.

The usage of AngularDart included declaring a class which extended Module class defined in AngularDart package, binding various components used inside the application and adding the module to the application factory. After this, a component to be shown on the browser is created.

For creating a component, the class should have the @Component annotation. Then the class can define various variables and functions which it wants to provide to the corresponding html. The various elements of the class are then accessed within the {{}} double braces or 'double moustache' inside the html.

For making the view menu items component the class contains a list of all the items and other necessary variables, a function that gets result from a query service and various other functions that handle click actions performed on the page elements. For ordering the items, a running list of selected items was used and sent over to the kitchen when confirmed with the customer.

## 4. AngularDart routing and other components

AngularDart makes routing for single page apps easier than most other routing solutions. It starts with defining a function that takes two parameters: a Router object and RouteViewFactory object.

After this, the RouteViewFactory object calls the configure() function which is supplied with a map object consisting of a String and an ngRoute method call which takes certain predefined parameters like path, view, viewHtml, mount, defaultRoute, enter, etc.

In general, a route is defined like following:

```
{
    'view': ngRoute(
        path: '/view',
        view: 'view/menuController.html'
        )
}
```

**Fig.2** Example of a route named 'view' in AngularDart

This routing function is then bound with the module and assigned to the RouteInitializerFn parameter. The NgRoutingPushState is also set to false since we want to work with hash changes in the URL to route our pages instead of working with push states.

Among other components, there is the admin component, edit items component, add item component, register table component and kitchen component.

### *4.1 Admin component*

This component was made keeping the owner of the restaurant in mind. This component gives the owner permission to make changes in the menu card among other things.

This component first starts with asking the user for their login credentials. After that, it presents the user with a dashboard with a panel on left side. On the panel, the user is presented with many choices like browsing different food categories and the food items shown on right hand side for editing or deletion.

Another option provided to the user is for previewing the menu card. This allows the user to see what the menu card would look like but the user can only view the items here and not make an order. There is also an 'add new item' option which presents a form on the right side for adding the new item.

The component consists of a username and a password variable and a login function to authenticate the user. The component then has functions to get query results from query service to show the preview page. The additional features on the page are managed by other components.

### *4.2 Edit items component*

This component allows the user to make changes to existing menu items or delete them. The user clicks on the edit icon and an html modal opens up with the details of the menu item.

The component consists of a variable that keeps track of which menu item was selected, functions for opening the appropriate modal and for deleting the menu item.

### *4.3 Add item component*

The add item component presents the user with a simple form for adding a menu item to the menu card. The component consists of variables for each field inside the form. The component also consists of functions for saving the new item to the database, updating the admin panel according to the category in case a new category is introduced and updating the menu card itself.

### *4.4 Register table component*

The register table has only a single purpose, to register a specified table to the user. The component consists of a variable which stores the table number, a function which stored the table number in the user's session information so that it persists till the user closes his/her browser and a function which navigates the user back to the view items page when table has been registered.

### *4.5 Kitchen component*

The kitchen component is also an important part of the restaurant system. The component displays the information regarding the user's table number and their orders with associated quantity. The challenge with this component was to retrieve the orders in real-time without refreshing the page and neither get the performance affected by polling the server at certain intervals. A simple solution to this was the use of WebSockets.

The controller has variables for the WebSocket address, a list of lists consisting of orders by every user, a function that accepts the orders from the server, a function that dismisses the orders when they are ready and a function that converts the server's json string messages into usable objects to be displayed on the browser page.

## 5. Kitchen server and WebSockets

WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply. (*developer.mozilla.org/en/docs/WebSockets*)

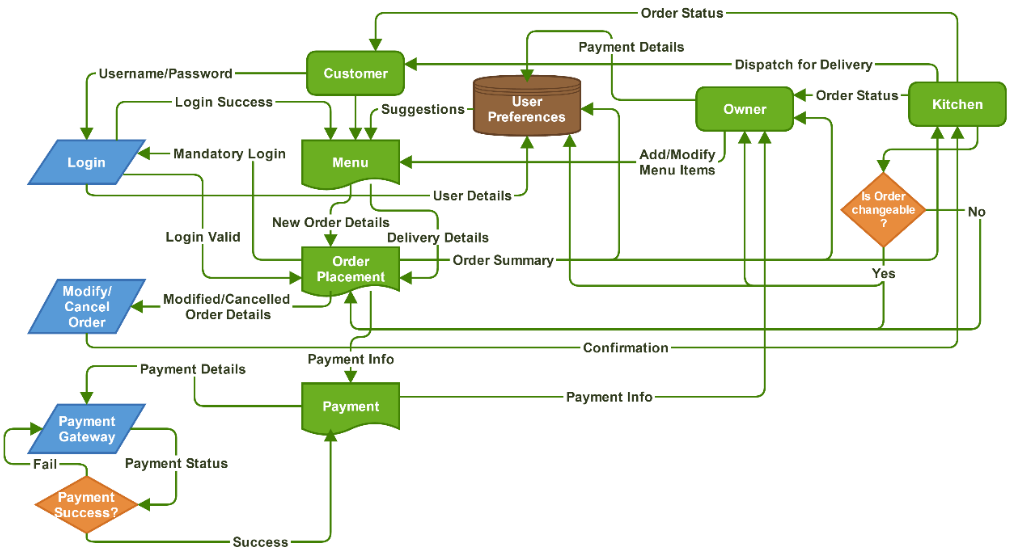


**Fig.3** Data flow diagram of the information system

The kitchen server acts as a medium to provide orders to the kitchen component in real-time. The first task was to set up an HTTP server to listen on a port. The server, whenever it encounters a request, it is checked if it is an upgrade request since the handshake of WebSockets is interpreted as an upgrade request by the HTTP servers. After this, the uri path of the request is checked to categorize the request to be for incoming orders or from kitchen.

If the kitchen connection is not open when receiving orders, they are saved inside a temporary list to be flushed as soon as the kitchen connects. The result is an efficient kitchen system which handles data in real-time without much overhead of implementation.

The resulting process of all of the above implementations is shown in Fig. 3.

## 6. Front end

Just like every ideal web application, this application needs to have a front end. Being a user centred application, it is absolutely necessary that we develop a user interface that is intuitive as well as user friendly. Since this web application targeted handheld devices as their major platform for the use, it was necessary for it to be responsive.

A responsive web application is an application that is easily adaptable to the smaller screen devices such as smartphones. To implement the responsiveness a famous CSS framework called 'Bootstrap' was used. Hence with angularjs and bootstrap an interface was built that allowed the user to place an order, increase the quantities in the placed order, see what orders he/she has placed so far and also see the total amount he/she has to pay before confirming the order.

## Future work

Considering how easy it is to set up and build a modern web application which is practically useful in our daily lives, it is right to say that building something with Dart and MongoDB is easy. The future work with Dart and MongoDB is limitless and can be used to build practically any type of web application existing in current technology and go even beyond that.

## Conclusion

From this, it can be concluded that Dart is a powerful language for building modern web applications. With its properly designed architecture and familiar syntax, it is without a doubt, prepared for what it takes to handle and properly run a modern web application and makes most current technologies fall short in front of it.

With MongoDB fuelling Dart from the back-end, it makes a complete package. MongoDB's easy scalability and flexibility in storing documents, makes it easier to employ it as the right database solution for modern web applications.

## Acknowledgement

We would like to thank our project guide Professor Shikha Moondra for helping and inspiring us throughout this project. We are also grateful to our head of the department, Professor Mahendra Patil for the support. Lastly we would not be able to conduct this research without the infrastructure and facilities provided by Atharva College of Engineering.

## References

Shrikar Chonkar, Siddaharth Suman, Shreyas Sawant and Shikha Moondra (2015), Supercharging web applications with Dart and MongoDB, *International Journal of Current Engineering and Technology*, vol. 5, no. 1, pp. 409-412.
*www.docs.mongodb.org*
*www.mongolab.com*
*api.dartlang.org/apidocs*
*www.github.com/vadimtsushko/objectory*
*www.rikulo.org/projects/stream*
*developer.mozilla.org/en/docs/WebSockets*