Firebase Integration
Angular

Connecting the dots

(A Quick looking-back!)

What we have done so far?

Components + **Services** + **Directives** = **Backend integration**

| Component creation Binding (Data & Event) | Dependency injection HTTP dummy-endpoint | DOM manipulation Structural, Attribute and customized | We are HERE! |

WSA | Forward looking IT finishing school
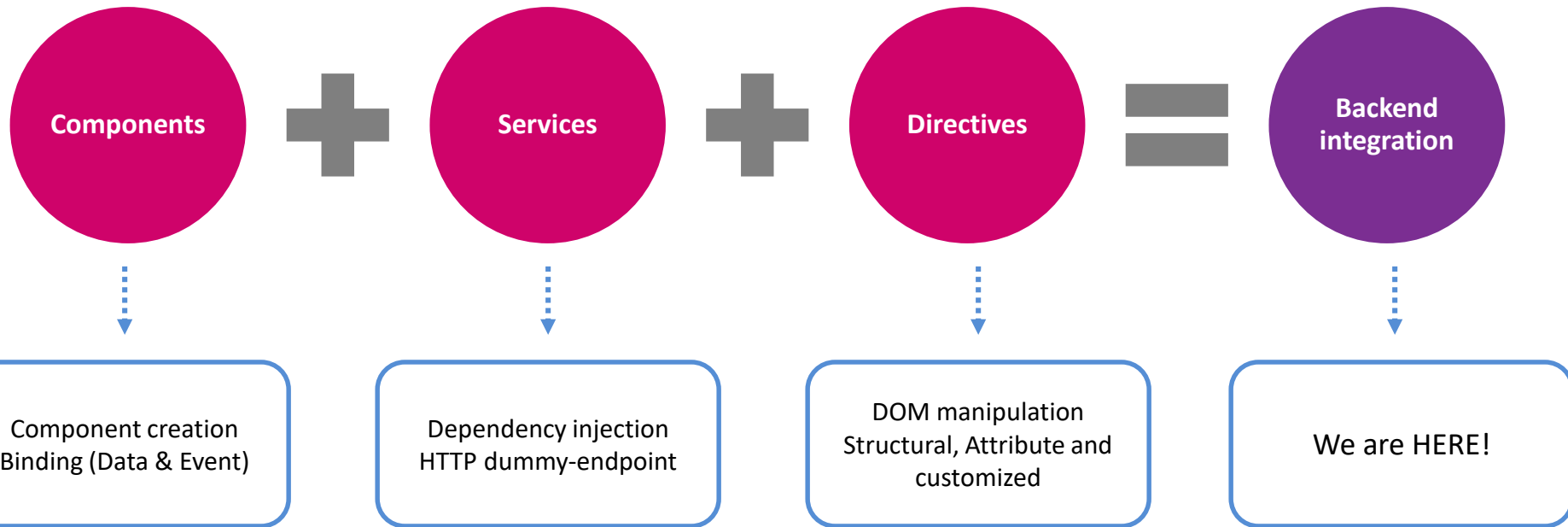
What we have done so far? – Static / Compiled data

- When we access any data in the business logic, local variables are used within the class so far (ex: **courses = ["Fullstack", "Frontend", "Backend", "Java"]**). This information is obtained in the view in form of interpolation **{{ course }}** mechanism

- We also understood how to use event (Using **() mechanism**), data (Using **[] mechanism**) and two way binding - **"Banana in a Box"** (Using **[()] mechanism**)

- Any time we wanted to demonstrate change in the data, we have edited the code changed some items and re-launched the angular application (using ng serve --open)

- However in real time application data will be fetched dynamically from an external source (ex: Firebase or your own custom backend application) which will not only ensure we get rid of using static local variables but also the data takes care of **Asynchronous updates**

- Primarily data updates include Create, Read, Update and Delete which is popularly known as CRUD operations

Introduction to Backend

(First level understanding)

Frontend and Backend

Frontend vs. Backend - Differences

Frontend	Backend
• Primarily a web client (HTTP) or a mobile app • Deals with user interface (HTML + CSS) and interactivity (JS or TS) • Obtains data (ex: Input box) or event (ex: Form submission followed by submit) • Doesn't have any storage capacity or business logic to handle data	• Primarily a web server supported by a Database (ex: MongoDB) • Deals with incoming requests from HTTP clients from various sources • Applies business logic on the input and provides appropriate response (ex: Forms) • Deals with non-functional items like security, scalability, performance. Multiple options of languages available (ex: Java, Python, JavaScript, Ruby etc..)

• Both client and server follow standard HTTP protocol is followed over TCP/IP network
• A brief discussion about OSI and TCP/IP

Relational vs Non-relational DB

Relational DB	Non-relational DB
• Data structure that allows you to link information from different 'tables' or different types of data buckets.	• Just stores data in forms of "documents" (JSON format) doesn't place any constraints like table. Its called "unstructured" database
• A data bucket must contain what is called a key or index (that allows to uniquely identify any atomic chunk of data within the bucket).	• Data structure that allows to store information in different forms - Trees, Graphs thereby giving more flexibility in terms of storage
• Other data buckets may refer to that key so as to create a link between their data atoms and the atom pointed to by the key.	• Other data buckets cannot refer to a particular table with the key. Data is very much local to an application.

Introduction to Firebase

(Backend integration platform)

Introduction to Firebase

- Firebase is a cloud based mobile and web app development platform provided by Google

- It offers developers with tools, databases, services etc.. to build any web or mobile application without bothering much into the backend web development

- The Firebase Database is a cloud-hosted, non-relational DB database (Mongo DB) that lets you store and sync between your users in real-time.

- It offers good amount of features for free (to develop and launch your app) and then moves onto paid model

- Advanced features (ex: Analytics) are provided to monetize your application better

Firebase – Our Goals

- Integrate a frontend application developed using Angular framework with Firebase and build end-to-end perspective

- Practically understand real-time use-cases of core angular features (components, services, directives, forms etc..) by dealing with real-time data

- Enhance "grey-box" knowledge of frontend (Angular) with "black-box" knowledge of backend (Firebase)

- Obtain introductory knowledge about backend, which will further get enhanced with backend programming topics (ex: Node.js & Express.js)

Firebase Integration – Part I

(Configuring Firebase Server)

1. Login and Access your console

Goto http://firebase.google.com and login with your Gmail / Google credentials
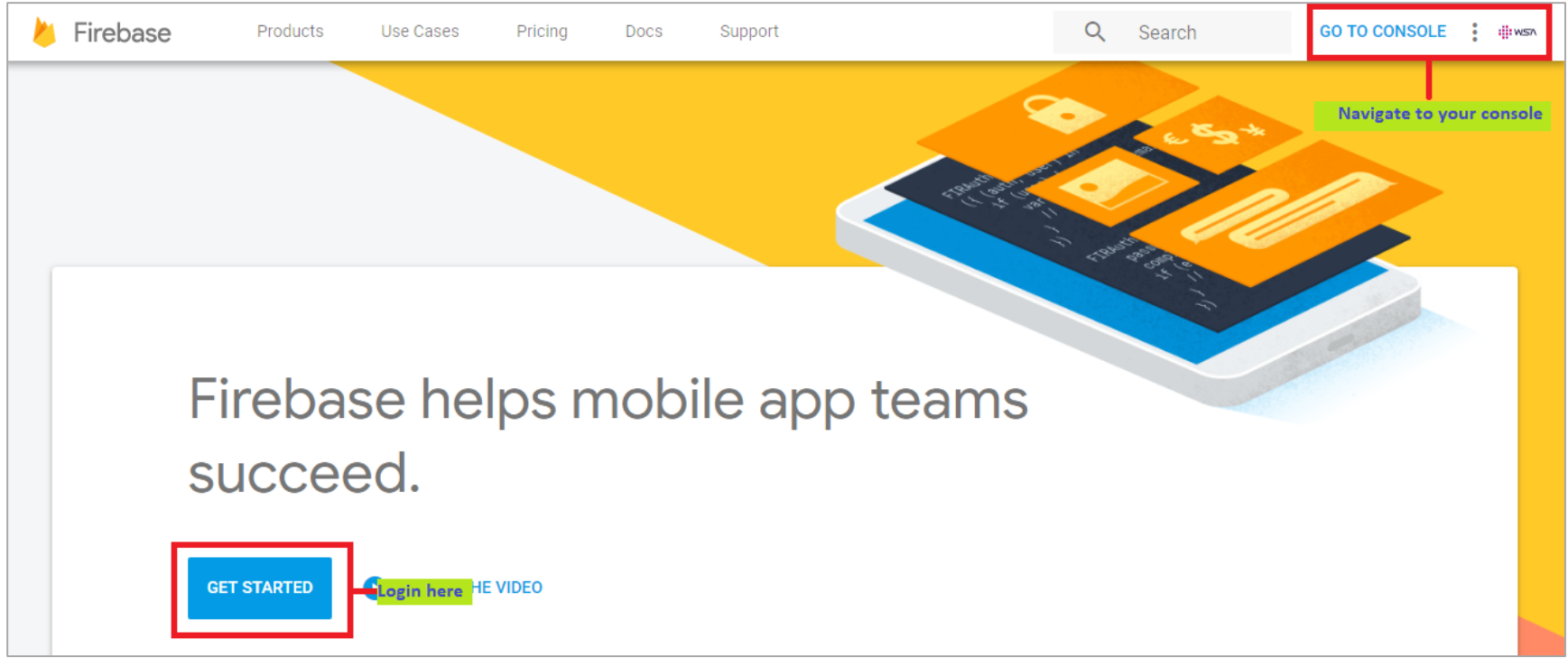
2. Create your project

Click on the option and create your own project
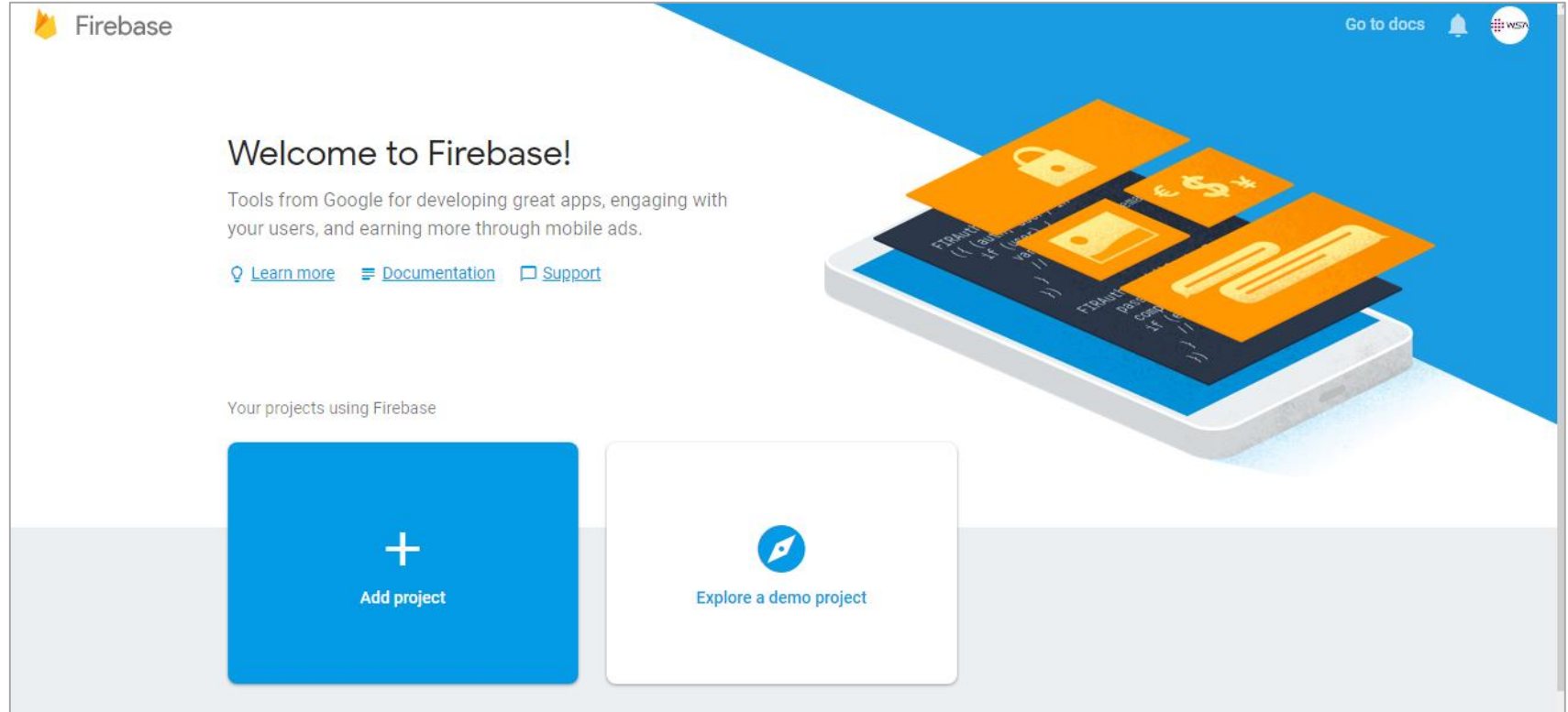
3. Enter Basic information for your project
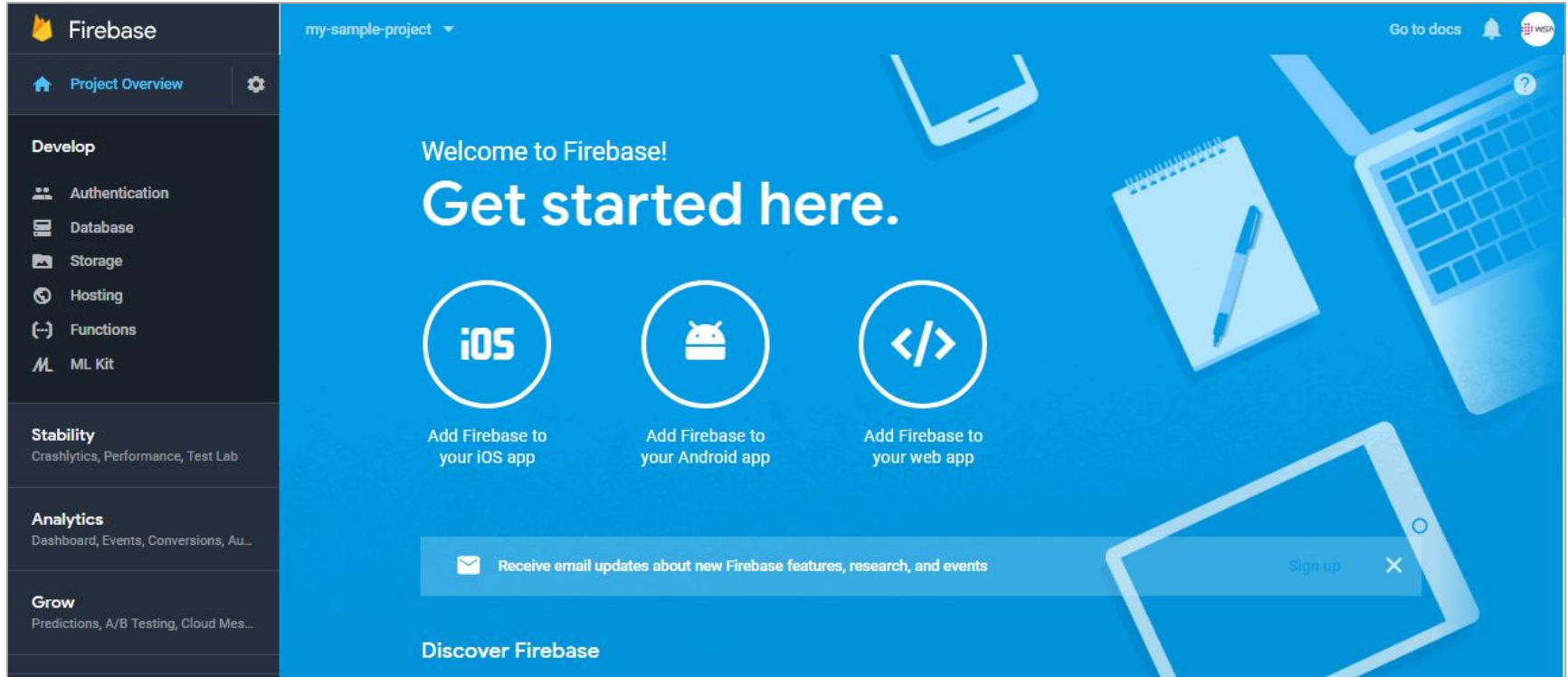

- Enter project name
- Enter country
- Accept terms & conditions
- Create project

4. Goto your project dashboard

Navigate to your project dashboard. Ensure you are choosing the right project.

5. Copy your project integration settings


```
Add Firebase to your web app                                    ✕

Copy and paste the snippet below at the bottom of your HTML, before other script tags.

<script src="https://www.gstatic.com/firebasejs/5.1.0/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "AIzaSyDiQNsd1DiyM5DxqQkm_s3gReyk2bgun_A",
    authDomain: "angular-firebase-integra-d7ba5.firebaseapp.com",
    databaseURL: "https://angular-firebase-integra-d7ba5.firebaseio.com",
    projectId: "angular-firebase-integra-d7ba5",
    storageBucket: "angular-firebase-integra-d7ba5.appspot.com",
    messagingSenderId: "716909083593"
  };
  firebase.initializeApp(config);                              Copy
</script>

Check these resources to    Get Started with Firebase for Web Apps ⬀
learn more about Firebase for  Firebase Web SDK API Reference ⬀
web apps:                       Firebase Web Samples ⬀
```
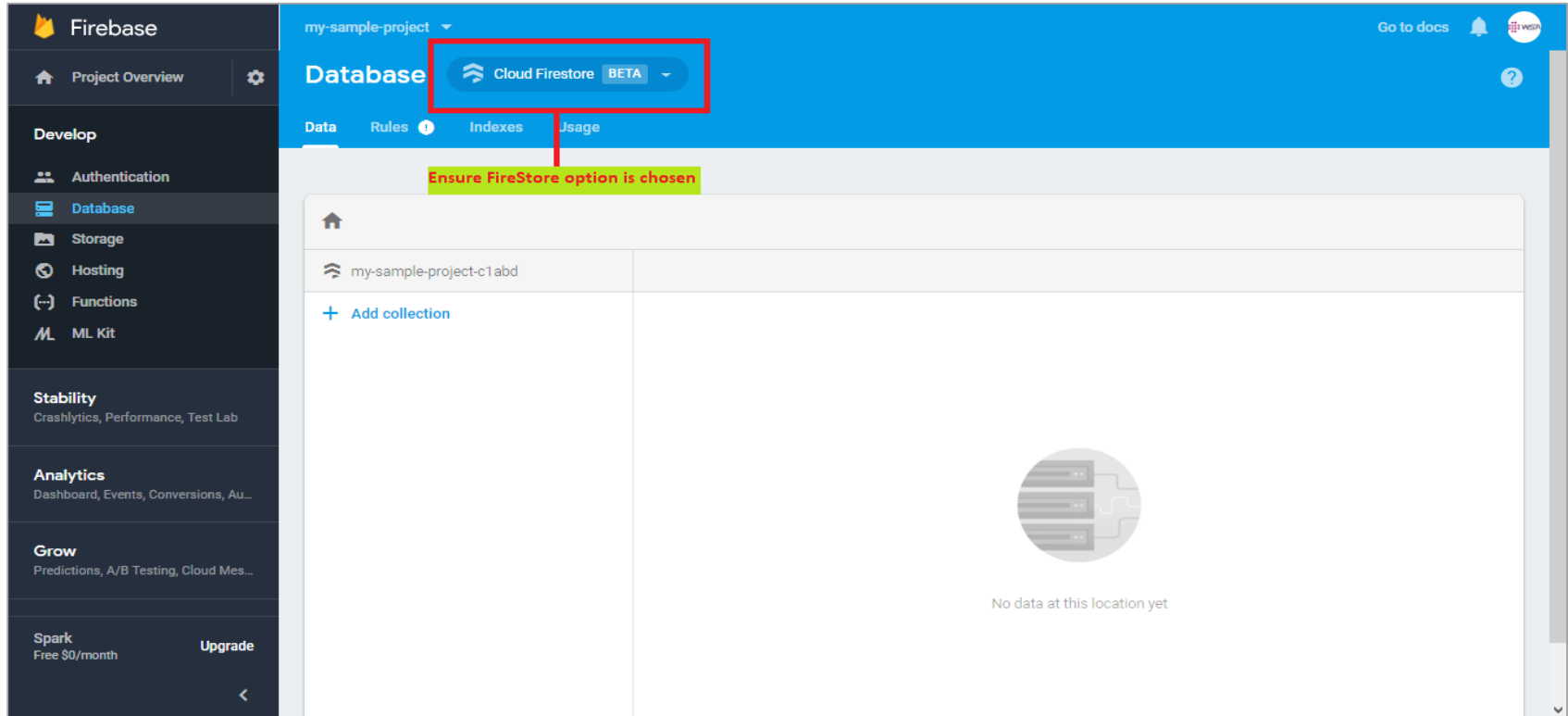
Click "Add Firebase to your Web App" it will open a project integration settings as follows. Copy the items inside "config" section and keep them handy.

6. Choose your Database

Goto project "Database" and select FireStore (Beta) option

7. Enter your data

Add meaningful data in collection, document and field columns and create your DB

Firebase Integration – Part II

(Configuring Firebase Client - Angular)
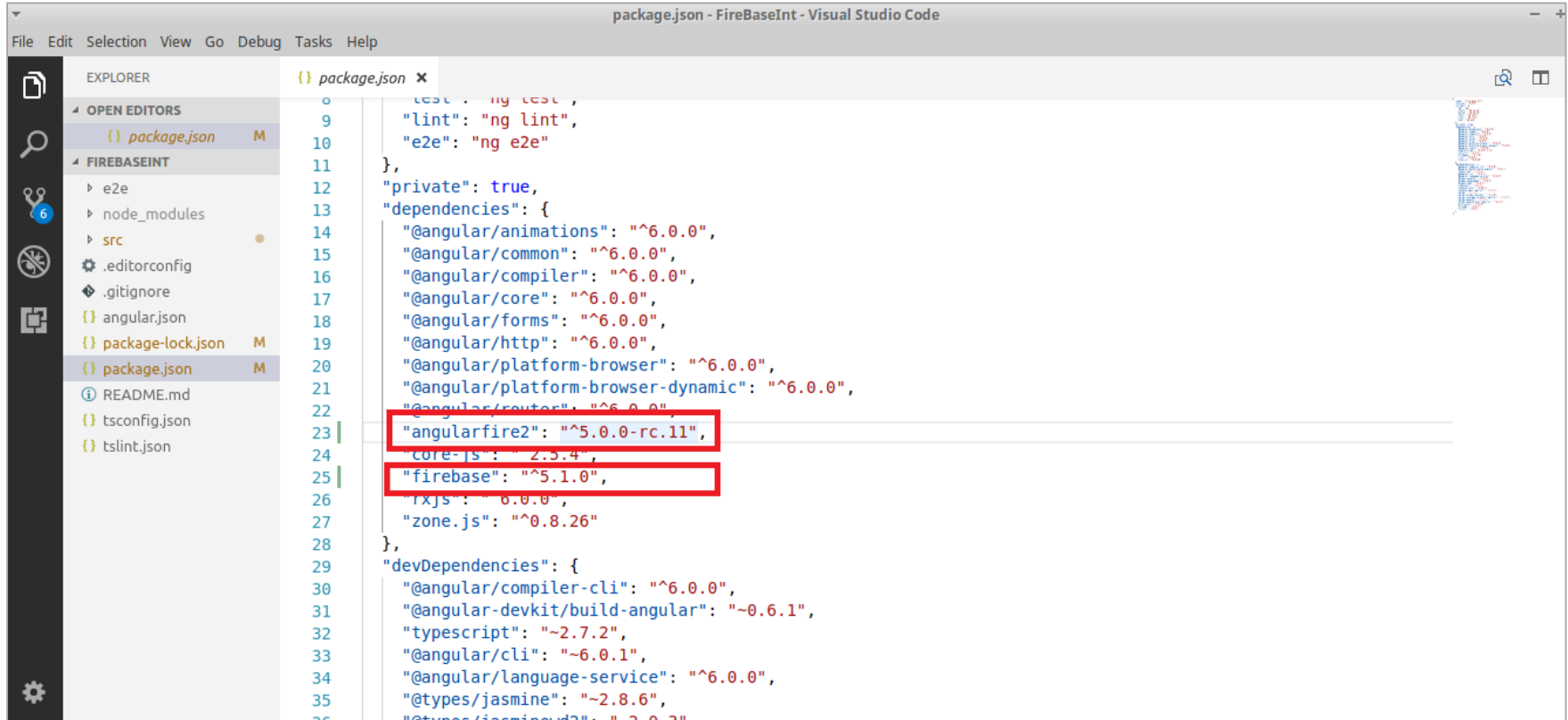
1. Installing Firebase packages

- In order to integrate Firebase with your Angular application two packages need to be installed.

  - **Firebase:** Firebase library for web and Node.js (Server)
  - **Angularfire2:** Firebase library for Angular (Client)

- To install them execute the following commands (from your project folder)

```
$ npm install firebase angularfire2 --save
```

- By using the --save option enters the packages into your environment.

- Check your package.json file to ensure these packages are listed in the dependency

WSA | Forward looking IT finishing school

1. Installing Firebase packages

Check package.json settings to ensure packages are properly imported into the project

2. Copy your Firebase project integration settings

Goto environment.ts file and copy Firebase settings (which you copied earlier)

```
export const environment = {
production: false,
firebase : {
        apiKey: "AIzaSyCbCrluicS3ls1ER3NWcmi0CJJjVQ5aeBQ",
        authDomain: "webstack-academy.firebaseapp.com",
        databaseURL: "https://webstack-academy.firebaseio.com",
        projectId: "webstack-academy",
        storageBucket: "webstack-academy.appspot.com",
        messagingSenderId: "247936282085"
    }
};
```

3. Changes in Modules file

Make the Angular and related import paths

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AngularFireModule } from 'angularfire2';
import { AngularFirestoreModule } from 'angularfire2/firestore';

import { AppComponent } from './app.component';
import { environment } from '../environments/environment';
```

WSA | Forward looking IT finishing school

3. Changes in Modules file

Add them into import section as well

```
@NgModule({
declarations: [
    AppComponent,
],
imports: [
    BrowserModule,
AngularFireModule.initializeApp(environment.firebase,'angularfs'),
AngularFirestoreModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

WSA | Forward looking IT finishing school

4. Changes in Component files

Component Imports and constructor

```
import { AngularFirestore } from 'angularfire2/firestore';
import { Observable } from 'rxjs';

// Observable and other function details in next section
items: Observable<Item[]>;

constructor(public db: AngularFirestore) {
    this.items = this.db.collection('courses').valueChanges();
    this.items.subscribe(items => {
    console.log(items);
  });
}
```

5. Changes in view file

Using Directives display the items fetched from Firebase in the View file. When you change values in the Firebase it should reflect in the view dynamically

```html
<h1> Angular integration with Firebase </h1>

<ul>
    <li *ngFor="let item of items | async">
        {{ item.coursename }} --> {{item.duration}}
    </li>
</ul>
```

Performing CRUD operations

(CRUD Foundational Concepts)

CRUD Operations

- In real-time use cases front end will interact with a database to perform **C**reate, **R**ead, **U**pdate and **D**elete (popularly referred as CRUD) operations.

- For example, you have used Firebase to retrieve course list from the DB, which is nothing but a Read operation. However if you want to add new courses (ex: Data Science Course) into the list, modify existing course properties (ex: Duration) or delete existing course you need to know other operations.

- Also while dealing with remote databases (like Firebase) CRUD operations will not happen in real-time.

- There would be some delays (ex: Due to Network traffic), so the client applications like Angular need to handle them in an **Asynchronous manner**

- Angular provides a mechanism called **Observables**, using which Asynchronous handling is done

CRUD Operations – Foundational Concepts

CREATE READ UPDATE DELETE

CRUD
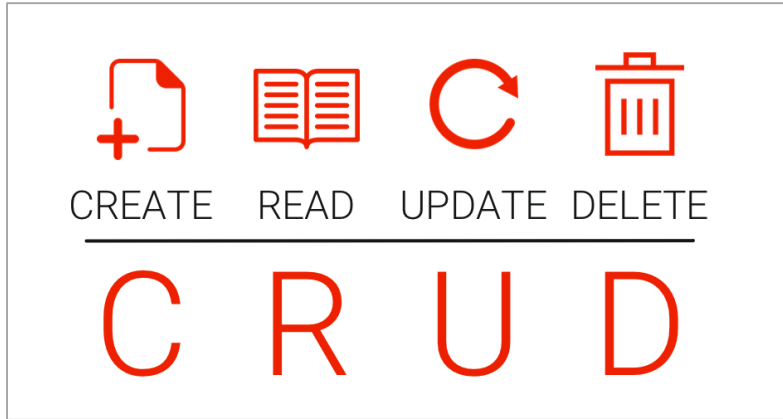
- Synchronous vs Asynchronous handling

- Call-back functions

- Observables

- Reactive programming

Synchronous vs Asynchronous Programming

- The definition of Synchronous and Asynchronous is very broad, in our context it can be defined as follows:

  - Synchronous means that the **caller waits** for the response for completion
  - Asynchronous that the **caller continues** and a **response comes later** (if applicable)

- Synchronous is easier to handle but it consumes higher amount of resources

- Asynchronous requires sophisticated mechanism (ex: Call-backs) but it consumes lesser amount of resource

- In modern day development, Asynchronous mechanisms are highly preferred

- Observables in Angular is practically implementing Asynchronous programming

WSA | Forward looking IT finishing school

Discount Purchase - Example – Asynchronous way

REGISTER MOBILE NUMBER FOR DISCOUNT

NO WAIT

SMS NOTIFICATION SENT

DISCOUNT PURCHASE

25% discount

WSA | Forward looking IT finishing school

Observables in Angular

- Observables provide support for passing messages between **publishers** and **subscribers** in your application (also known as "pub-sub" model).

- Interested Angular component can subscribe for a particular event (in our case it is Firebase DB handling) and Observarable will notify once the event occurs (ex: Database read) via call-back

- Observables offer significant benefits over other techniques for event handling, asynchronous programming, and handling multiple values. An observable can deliver multiple values of any type—literals, messages, or events, depending on the context.

- Because setup and teardown logic are both handled by the observable, your application code **only needs to worry about subscribing to consume values**, and when done, unsubscribing.

- Observables are used in variety of use-cases (HTTP response / Interval timer) apart from Firebase. Because of these advantages, observables are used extensively within Angular, and are recommended for app development as well.

Reactive Programming

- Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change.

- **RxJS (Reactive Extensions for JavaScript)** is a library for reactive programming using observables that makes it easier to compose asynchronous or call-back based code (RxJS Docs).

- RxJS provides an implementation of the Observable type, which is needed until the type becomes part of the language and until browsers support it.

- The library also provides utility functions for creating and working with observables. These utility functions can be used for:
    - Converting existing code for async operations into observables
    - Iterating through the values in a stream
    - Mapping values to different types
    - Filtering streams
    - Composing multiple streams

The subscribe() Method – A closer look!

- A handler for receiving observable notifications implements the Observer interface. It is an object that defines call-back methods to handle the three types of notifications that an observable can send:

  - **Next:**
    - ✓ Required. A handler for each delivered value.
    - ✓ Called zero or more times after execution starts.
  - **Error:**
    - ✓ Optional. A handler for an error notification.
    - ✓ An error halts execution of the observable instance.
  - **Complete:**
    - ✓ Optional. A handler for the execution-complete notification.

- The above mentioned functionality is implemented via the subscribe() function

- The type of data returned and handling mechanism is purely based on the use-case

WSA | Forward looking IT finishing school

The subscribe() Method – A closer look!

```
myObservable = ..... // Async method

myObservable.subscribe(
    x   => console.log('Observer got a next value: ' + x),
    err => console.log('Observer got an error: ' + err),
    ()  => console.log('Observer got a complete notification')
);
```

And Finally….to back to our same source code….

```
import { AngularFirestore } from 'angularfire2/firestore';
import { Observable } from 'rxjs';

// Observable and other function details in next section
items: Observable<Item[]>;

constructor(public db: AngularFirestore) {
    this.items = this.db.collection('courses').valueChanges();
    this.items.subscribe(items => {
    console.log(items);
  });
}
```

CRUD operations in Angular

(CRUD APIs in Angular)

Data organization in Firebase

- In Firebase DB, data is organized at three levels:
  - Collection:
  - Document:
  - Field:

https://github.com/angular/angularfire2/blob/HEAD/docs/install-and-setup.md

Exercise

- Implement the "to-do" list for the user with the following functionality:

    - Enter to-do item, category and priority (ex: "Learn TypeScript" , "Skill", "High")
    - Integrate the user entered data with the Firebase backend
    - Provide buttons to perform the following operations:
        - ✓ Add
        - ✓ Modify
        - ✓ Delete
    - Implement a service to handle the Firebase endpoint functionality
    - Use Interfaces and export them for common usage
    - Display task items based on category

![WSA Logo] **WSA** | Forward looking IT finishing school

Thank You

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media: