Mastering Ionic

Working with Cloud Firestore

James Griffiths

Mastering Ionic - Working with Cloud Firestore

Saints at Play Limited

Thanks to...

The teams at Ionic and Angular for creating such phenomenal products that allow millions of developers worldwide to realise their ideas quickly and easily.

The awesome developers and communities behind PHP, MySQL, SQLite, Firebase, PouchDB, CouchDB, Docker, NodeJS, ExpressJS, MongoDB, Mongoose, ElectronJS and rxJS.

The developers, contributors and drivers behind all of the JavaScript/TypeScript packages and libraries used within the projects covered in this ebook - you guys are awesome!

Every developer who ever helped answer a question that I had or a software bug that I was trying to fix - I may have forgotten many of your names but I will always appreciate the assistance you have provided.

Those who believed in me and gave me a chance to shine when many others didn’t or just wouldn’t - you are not forgotten!

God above all others - without whom nothing would be possible.

Table of Contents

7

Introduction

Thank you for purchasing this digital copy of Mastering Ionic: Working with Cloud Firestore.

My goal with this ebook is to guide you through working with Cloud Firestore and Firebase Storage to seamlessly integrate data into an Ionic application.

We start with exploring database concepts and terminology - for both SQL and NoSQL databases before progressing onto developing/publishing Cloud Firestore powered Ionic Progressive Web Applications.

We'll cover working with the Cloud Firestore database and Firestore API to provide CRUD (Create, Read, Update and Delete) functionality for our Ionic applications.

Along the way we'll work with a range of technologies and services including:

- Cloud Firestore
- Firebase Storage
- Firebase hosting
- HTML5 APIs
- Angular Material library
- Progressive Web Apps
- rxJS library methods

We'll cover tips for best practice, discuss known limitations and/or potential challenges with the services/tools that we are using and I'll provide you with further resources to reinforce what you've learnt within the pages of this e-book.

No matter what your level of experience working with the technologies that we'll be covering, I hope you find this e-book both useful and enjoyable and I look forward to receiving your feedback.

What this book isn't
If you're looking for an in-depth technical guide into all the functionality and features available within the Firebase BaaS platform then this simply is NOT the book for you.

What is covered

This book provides detailed information on the core essentials of Cloud Firestore and Firebase Storage, performing CRUD related operations and how to integrate data from Cloud Firestore into an Ionic frontend.

I promise you there will be plenty of useful real-world information that you can take and use within your own projects/applications but a deep-dive into the Firebase BaaS platform (and all of its core features) is simply not possible (as the book would have to be at least twice as large…and even then it would only be scratching the surface).

Prerequisites

I am assuming (hopefully not wrongly!) that you already have, at the very least, a basic understanding and level of familiarity with developing projects for web, iOS & Android using the Ionic CLI.

You will also need to be familiar with understanding and being able to use HTML5, Sass/CSS and Angular/TypeScript which are core languages/frameworks used within Ionic.

I won't be covering those languages/frameworks in depth (other than demonstrating how they can be used throughout the projects we'll be developing) so, if you do require any background information/further instruction on their usage, a good place to start would be with the following resources:

- Angular
- TypeScript
- HTML5
- Sass
- CSS

You will also need to have some familiarity/experience with command line usage as we'll be creating projects, page components & Angular services, installing the required plugins and software libraries as well as deploying projects to Progressive Web Apps with Firebase Hosting through both the Ionic and Firebase Tools CLI.

Last, but not least, you'll also need to have a basic understanding (as well as some experience) with object oriented programming (otherwise referred to by its acronym of OOP) as TypeScript is a class based OOP language (actually a superset of JavaScript if you want to get technical).

If you're not all that familiar with Object Oriented Programming (in the context of TypeScript/JavaScript) then I would recommend starting with the following online resource which should help get you up to speed.

So who am I and why should you listen to me?
I'll answer the last part of that heading first....only if you feel you want to!

Joking aside my background in web/mobile development stretches back to 2002 when developing online projects, almost exclusively in the form of websites (as the iPhone was still 5 years away and mobile development was, to put it mildly, an extremely small niche due to limited possibilities with the available technologies, tools and device/browser support - anyone reading this remember WAP?), was in a relatively nascent stage.

Even though largely forgotten and, in many developer circles (for those of us who are old enough to remember) widely derided and scorned, Macromedia Flash MX was my introduction to developing websites and applications (albeit only browser and CD/DVD-ROM based at the time).

I'm probably going to invite ridicule and exasperation with the following statement but I loved working with that software and the possibilities it opened up for creative experimentation, programming and designing/developing applications.
As the first decade of the twenty-first century progressed, and browser support for language standards and features improved, it became more and more apparent that Flash had certain limitations that working with HTML/CSS (and the re-emergence of JavaScript as a scripting language - helped with frameworks like jQuery and MooTools as well as a trend towards consistent browser support) didn't.

This gradually led to more frontend focussed development as well as incorporating PHP/MySQL into my digital toolbox.

Fast forward to 2010 and I'm starting my initial journey into developing mobile applications with jQTouch and PhoneGap, subsequently followed by jQuery mobile before finally settling on Ionic in 2014.

Along the way I've delivered websites and mobile/tablet applications for a variety of clients including Halco Energy, West Midlands Police, Maplecroft, WQA, Virgin Media, EDF Energy, Evans Cycles, Shelter and the British Science Association as well as various digital agencies, marketing companies and small business clients.

I'd like to think, as a result of the past 20+ years, that I've accumulated a certain wealth of experience, knowledge and skills that can be shared with the wider development community.

I certainly don't consider myself to have reached any vast summit of knowledge and, in some respects, I feel like I'm only just starting to scratch the surface of discovering what's possible with all these incredible web based technologies that we have access to now.

Just like yourself I'm still on a journey of learning, growing and maturing as a developer...and with the rate of technological change that is continually taking place there's always more to learn!

Support

So you've purchased my e-book (or are maybe considering making a purchase) and you might find yourself with some questions regarding how often the content is kept updated with changes to Ionic and/or the associated technologies that are covered in these pages, what help/assistance is available with possible development issues you might encounter and what further resources you might be to access.

Firstly, schedule permitting (although even the best will in the world can be thwarted by external events and circumstances), I endeavour to keep the e-book content updated within 7 days of significant changes to Ionic and/or featured databases and technologies.

I routinely e-mail my customers with news of updated e-book content that has been published.

Finally, all downloadable code examples for each chapter and the featured case studies are available here.

Conventions used within this e-book

Fortunately there's only a small number of conventions employed within this e-book that you need to be aware of.

ALL of the code examples that are featured in each of the chapters and case studies are displayed within a grey rectangle, which may (or may not) contains additional comments (rendered in italics), like so:

```
// Install the required platforms
npx cap add electron
```

Important information that requires your full attention is prefixed in its own paragraph like so:

IMPORTANT

Previous code examples that have been covered/explained will, where further additions to that script are required, be rendered with a placeholder in the following format:

...

There will, due to the limitations imposed by the width of the page dimensions of this e-book, be instances where code might run onto other lines. Where this occurs a hyphen will be inserted into that line of code to indicate that the displayed code is all part of the same line.

Use of hyphens in this specific context do NOT form part of the code logic but merely demonstrate that the code is continuing from one line to the next.

Where external resources are mentioned/used within each chapter these are rendered in the form of hyperlinks along with an additional list of those hyperlinked resources displayed at the end of each chapter.

Finally, each chapter will, where necessary, conclude with a summary of the key concepts and information that has been covered.

IMPORTANT: You'll notice, as you go through the code examples covered in each chapter and case study, that I employ the following practice:

- Use of JSDoc syntax for commenting project component and service classes
- Specific naming conventions for class properties and methods (to help readily identify, or hint at, the purpose of that segment of code) • Formatting the code so it is more easily readable

You don't have to adopt the same practice (as each developer will have their own specific coding/formatting style) but it is a good idea to invest time into making your code as understandable/readable as possible (which is why I employ the above approaches in this e-book as well as my own digital projects).

After all, if you come back to a project 3 or more months later (or are working with other developers), such efforts will help make managing the project quicker and easier in the long run - and that can't be a bad thing (especially if you happen to forget why you coded something in a certain way!)

Glossary

Technical terms

Technical glossary

To wrap up the introduction to this e-book let’s quickly cover some of the keywords and terms that we’ll be encountering/using over the following chapters.

I imagine most of you will already be familiar with these so feel free to press on to the next chapter if that’s the case! If not, please take a few minutes to read through the following terms and familiarise yourself with their meaning.

ACID

Acronym for Atomicity, Consistency, Isolation & Durability - a measure used to determine how effective a database system is as at performing transactions

Angular

A front-end component-based framework for building scalable web applications that is the default choice of framework for Ionic

API

Application Programming Interface - A set of tools for a particular software library, framework or service that developers can utilise in their own projects

Authentication

The act of verifying that a supplied identity is genuine

Authorisation

The act of granting access to a system or service

Backend as a Service

Often referenced as the acronym BaaS refers to a cloud computing model which allows web/mobile application developers to connect with services such as cloud storage, push notifications and NoSQL databases through the use of dedicated API’s/SDK’s

BASE

Acronym short for **B**asically **A**vailable, **S**oft state, **E**ventual consistency - a data consistency model used by many NoSQL databases

CapacitorJS
A cross-platform runtime API similar to Apache Cordova that is focussed on performant mobile applications that adhere to Web Standards while accessing native device functionality on platforms where support is available

Content Delivery Network
A Content Delivery Network, often abbreviated as CDN, is a system of distributed servers, spanning multiple geographical locations, that allows websites and applications to benefit from high availability of content, low network latency and improved performance

Class based programming
A style of Object-Oriented Programming where objects are generated through the use of classes

CLI
Command Line Interface - A software utility that allows commands to be executed solely through text input

CRUD
Acronym for Create, Read, Update and Delete, which are common operations performed on data

Electron
An application development framework that allows users to build cross-platform desktop applications using HTML, CSS & JavaScript

Firebase
A Google owned/managed BaaS platform which provides a variety of cloud related services such as Authentication, Storage, NoSQL databases & Push notifications

Hybrid Apps
Mobile applications that are typically developed using web based languages such as HTML, CSS and JavaScript which are then able to be published within native mobile wrappers for deployment to iOS, Android, Windows Mobile devices etc

Ionic Framework
An open source application development framework for developing progressive web apps and mobile applications

JSON
JavaScript Object Notation - A subset of the JavaScript programming language that specifies/provides a standard for exchanging data

Node
An open-source, cross platform JavaScript environment for developing server-side web applications

NoSQL
A type of database where data is typically stored in the form of JSON objects

Object Oriented Programming
Often referenced by its acronym of OOP - A type of programming where code is developed based around the concept of objects and their relationship to one another

Package Manager
A tool, or collection of tools, for managing the installation, configuration, upgrading, removal and, in some cases, browsing of software modules on a user's computer

SDK
Software Development Kit - A suite of development tools that developers can use with a particular software program, library or platform

Transaction
A unit of work performed within a database system

Databases

A short summary

In its simplest definition a database is a structured container for storing data and allowing that to be acted upon (I.e. CRUD related operations, importing/exporting data and performing searches).

Databases come in a variety of models (and often implement schemas).

Models and schemas

A database model determines the logical structure of a database including the relationships and constraints of how data is able to be stored and accessed.

Common database models include:

- Hierarchical
- Relational (aka Relational Database Management System [RDBMS] or SQL database)
- Non-relational (NoSQL)
- Object-oriented
- Network

Of these the Relational and Non-relational (NoSQL) database models are the most commonly used - at least as far as most organisations/developers are concerned.

A database schema refers to the logical grouping, organisation and structure of objects that are used within the database (such as tables, views, indexes, stored procedures etc).

For example, a database model may define the overall structure of the database and how data is stored/accessed (i.e. relational or NoSQL) yet there may be one or more schemas defining the structure, organisation and relationships between certain parts of that database system (i.e. accounting schemas, auditing schemas, reporting schemas etc)

Models and schemas can often be confusing to define as they are sometimes used interchangeably or given slightly different meanings with some database systems.

Relational databases

A relational database model structures, groups and organises data using:

- **Tables** (structures that impose a schema on the records that they contain)
- **Rows** (the individual records that are stored within a table)
- **Columns** (the distinct fields that data is stored under for each record)

Fields come in many different data types (with potential constraints and additional flags depending on the data type being supported) which may include - for example:

- integer
- float
- text
- date
- boolean

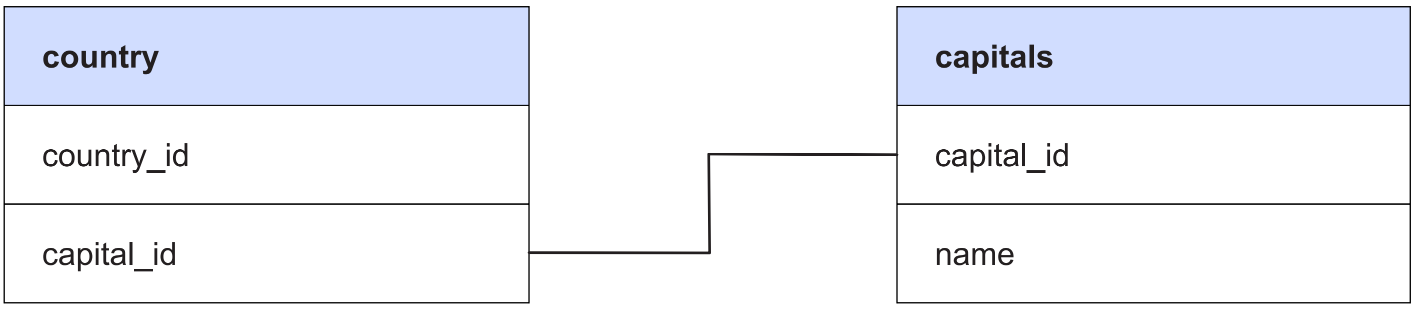
Relationships

Tables can establish relationships with one another through the use of keys:

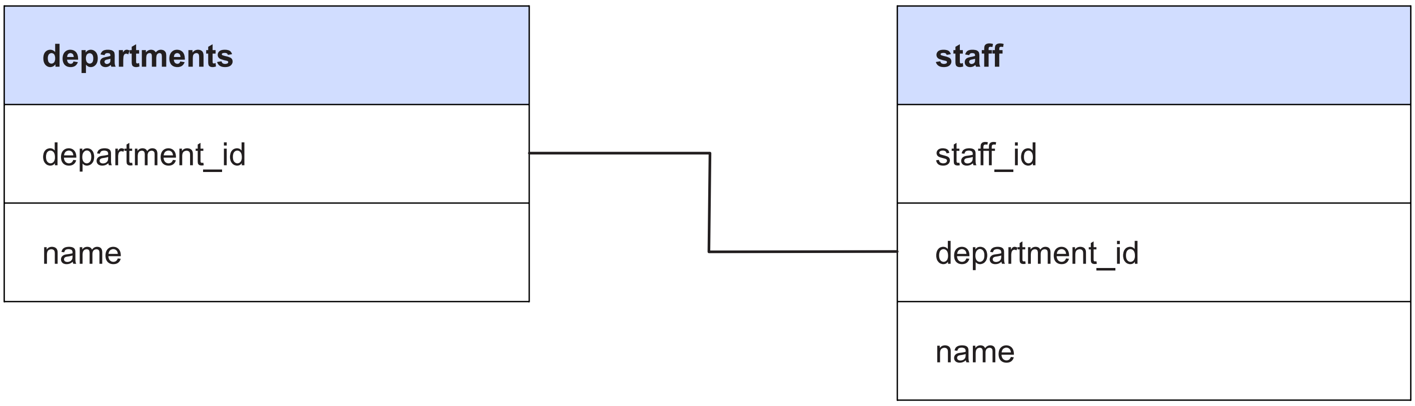
- **Primary key** - A table column where each record has a unique value
- **Foreign key** - A table column whose values references the primary key of another table

Within an RDBMS there are 3 possible types of relationships between tables.

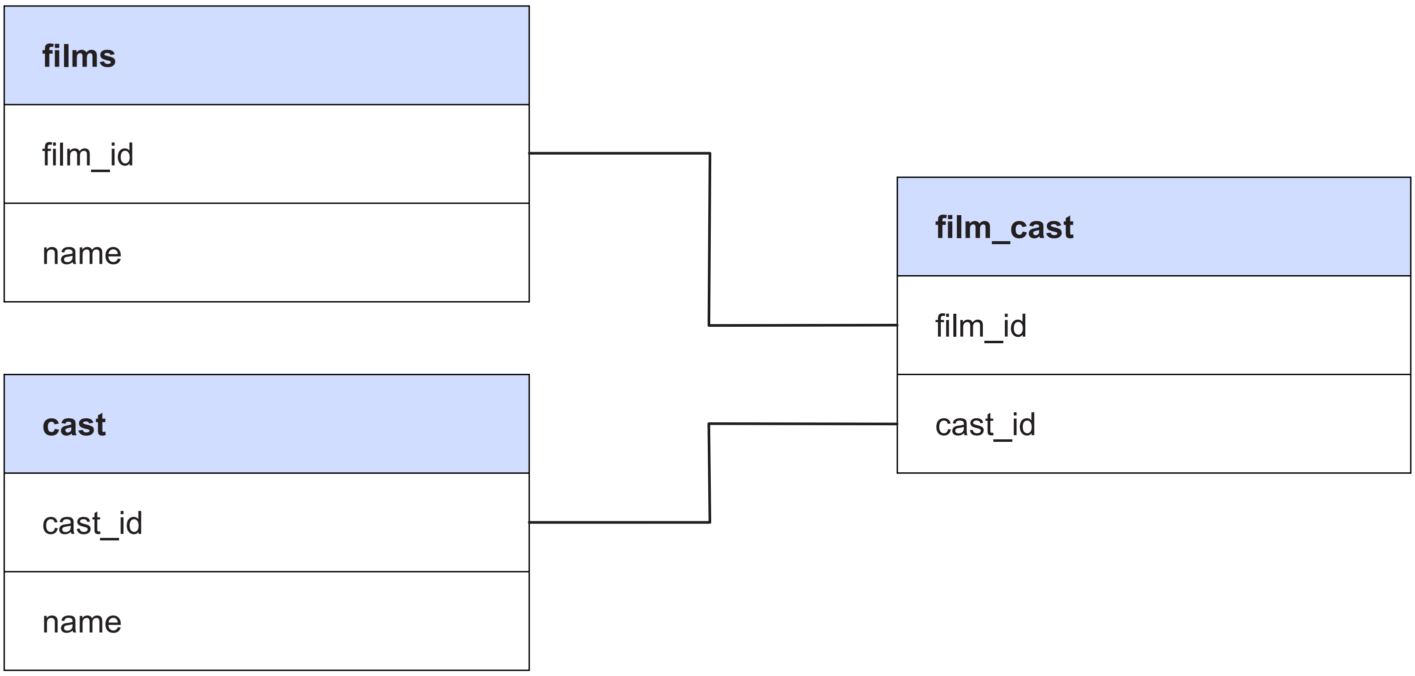
A **One-to-one** relationship consists of a single record on each side - for example, the relationship between a country and its capital city (as a country will only ever have one capital):



A **One-to-many** relationship consist of a single record on one side and many records on the other side. This could occur, for example, in a company where an employee belongs to a single department but a department can have many employees:



A **Many-to-many** relationship consists of multiple entries on both sides of the relationship. This could be represented in a movies database for example where a single movie can feature many cast members and a cast member can feature in many movies:



Normalisation

When designing the data architecture for a SQL database system (I.e. determining the quantity, types and nomenclature of fields used within the different tables and the purpose of each table) it's important to minimise data duplication to make data entry more efficient and easier.

For example if there are two or more tables that contain a product name column it would make more sense to link these together so that data only needs to be entered once and not multiple times for the same item.

This process of reducing data duplication to make a maximally efficient relational database system is known as **normalisation**.

The use of **normal forms** - a guiding set of stages for which a database achieves ever greater levels of normalisation - was introduced by computer scientist Edgar Frank Codd in the early 1970's.

These normalisation guidelines consist of six normal forms although, typically, if a database is in **Third Normal Form** (3NF) it is generally considered to be normalised.

You can read more about database normalisation here.

SQL Queries

Users can query data within a relational database management system using SQL (Structured Query Language) which can be simple in its instruction or increasingly complex depending on the requirements and granularity of the query (which is where normalisation helps by making the system more performant).

A simple query such as retrieving all book records from a single books table in descending order of entry (i.e. most recent first) might be written as follows:

```
SELECT * FROM books ORDER BY book_id DESC
```

The SQL syntax is relatively simple and intuitive so that even to a non-database specialist the purpose of the query would be relatively self-explanatory.

A more complex SQL query - say retrieving a list of all films, their cast members and the studio responsible for producing that film where ticket sales grossed over $25 million dollars might look something like the following:

```
SELECT movies.id, movies.title, cast.id, cast.name, studio.id, studio.name
FROM movies INNER JOIN cast ON cast.movieId = movies.id INNER JOIN
studio ON studio.id = movies.studioId WHERE movies.id IN (SELECT movieId
FROM movies GROUP BY sales HAVING COUNT(*) > 25000000)
```

As you can see even with more complex queries SQL is relatively intuitive and easy to grasp (although some of the more complex query logic can take time to mentally parse - especially when working with different types of JOIN statements to draw data from multiple tables).

To learn more about SQL visit this online resource.

ACID

Databases that perform transactions in a timely, reliable and efficient manner are said to be ACID complaint.

ACID is an acronym for:

- **Atomicity** (all parts of a database transaction work as expected)
- **Consistency** (database transactions are performed as expected with no deviation in behaviour)
- **Isolation** (multiple transactions can be performed concurrently without affecting one another)
- **Durability** (data is saved with successful transactions even if a system failure or power outage occurs)

Relational database systems are widely trusted due to their ACID compliance.

Real world usage

Unsurprisingly many developers and organisations make use of relational databases to deliver their products/services (and you may likely have worked with such databases in an educational and/or professional context).

Such widely used SQL databases include (but are not limited to):

- MySQL
- PostgreSQL
- MariaDB
- SQLite
- Oracle
- Microsoft SQL Server

If you've ever developed applications using PHP then MySQL (and possibly PostgreSQL and MariaDB - a fork of MySQL) will likely be somewhat familiar to you.

To summarise then:

Relational database overview	
Pros	**Cons**
ACID (Atomicity, Consistency, Isolation & Durability) compliant - ensures that a database transaction is completed accurately and in a timely fashion	More time-consuming and difficult to modify pre-existing database architecture due to constraints in the database model
Support for table joins	High volume transactions can result in decreases in performance
Ability to perform complex queries	Difficult to scale with large amounts of data
Rigid, predictable structure for data	Does not work well with unstructured data
Allows for many different data types	Data normalisation can result in performance penalties

That concludes our brief and very basic introduction/overview of relational databases (and there's a lot more to learn…but for this book I want to keep things relatively simple and focus only on what we need to know for working with Ionic) so we'll now perform a similar walkthrough with NoSQL databases.

Non-relational databases

Ironically one of the strengths of relational databases is also experienced by many organisations and developers as one of its significant weaknesses: a rigid database architecture.

This rigid structure can be time-consuming, expensive and difficult to subsequently modify should even minor changes in data architecture be required (such as, for example, the addition of further fields or a change in the data types for existing fields).

Given that modern applications consume vast amounts of data (often supplied through third-party APIs) in the form of JSON objects SQL databases are not best suited for storage of, nor scaling with, such data formats.

Non-relational, more popularly known as NoSQL, databases were developed to address and meet these particular needs (amongst others).

Types of NoSQL

Instead of using tables non-relational databases store data using different models:

- **Column-based** (data is stored by column not row)
- **Document** (data is stored, often similar to JSON objects, in documents)
- **Graph** (data sets are represented as nodes, edges and properties with relationships represented as edges - or lines - between nodes)
- **Key-Value** (items of data are stored as key-value pairs within the database)

Data is typically stored as JSON, BSON (Binary JSON) or XML depending on the type of NoSQL database and the schema(s) that are supported.

In subsequent projects we will be working with document oriented NoSQL databases and will explore their data storage model that uses collections, documents and fields.

As there are a variety of NoSQL database models there is no one uniform language (unlike SQL with relational database systems) that can be used to perform queries across different systems.

For example Neo4J uses its own SQL inspired language called Cypher Query Language which is designed to work with its graphing NoSQL database model.

Using Cypher Query Language we could (in a hypothetical Neo4J database), for example, run the following query to retrieve all movies starring Keanu Reeves:

```
MATCH (keanu:Person {name: 'Keanu Reeves'})-[r:ACTED_IN]->(movie:Movie)
RETURN keanu, r, movie
```

Looks somewhat similar to SQL doesn’t it?

If we are using NoSQL document-oriented database MongoDB however we would perform queries (in this case adding a new document to an hypothetical **users** database collection) using Mongo Query language (MQL) like so:

```
db.users.insert({
  id: "aebd4fs001",
  surname: "Bloggs",
  first_name: "Joe",
  email: "joe.bloggs@joebloggs.com",
  age: 25,
  status: "Active"
})
```

Notice how even though this is called a query language it looks nothing like traditional SQL or Cypher but shares the same dot syntax approach as working with JavaScript?

Unlike relational databases it is more difficult to port data between different NoSQL databases due to the different models that are used (document, graph, key-value and column).

Dropping ACID?

Many NoSQL databases sacrifice **atomicity** (where the integrity of the entire database transaction is guaranteed - not just a certain part of the transaction) or **consistency** (where database transactions are only successful where they meet certain database rules) in order to achieve high performance/scalability.

Although these features make SQL databases highly reliable they can present performance problems with high data/traffic usage as well as scaling issues - both of which NoSQL databases are adept in addressing and overcoming.

Typically, in lieu of ACID compliance, many NoSQL databases offer BASE properties:

- **B**asically **A**vailable - In the event of failure the system is guaranteed to be available
- **S**oft state - Data state could change without user interaction due to eventual consistency
- **E**ventual consistency - Consistency is not guaranteed at the transaction level but, after application data input, the system will be eventually consistent once data has replicated to all database nodes

Although BASE offers less assurances than ACID it effectively handles rapid data changes and scales well.

Not all NoSQL databases are non-ACID compliant but this may be an issue with organisations who require ACID compliant database in their day-to-day operations.

Advantages of NoSQL

- Can accommodate flexible data structures
- Designed to efficiently handle larger volumes of data
- Designed with an architecture to allow scaling for greater traffic
- Can be faster to develop with as the data structure allows for changes in response to modern agile development practices (with sprints, iterations and more frequent code changes)

- NoSQL data tends to integrate more easily, due to its structure/syntax, with JavaScript in modern cross-platform applications
- Polyglot persistence - allows for use of multiple data storage models within an application (i.e. using document and graph databases for separate areas of an application that require different data models)
- Minimises impedance mismatch - a term used to describe the difference between the relational model of the database and the structure of the data that is being saved (for example graphing data being saved in a tabular format - as in relational database systems)

Disadvantages of NoSQL

- Difficulty in porting data between different types of NoSQL database
- Many NoSQL databases do not offer ACID compliance
- Schema-less architecture can be off-putting to some developers concerned with maintaining data integrity
- Data is denormalised which requires mass updating (I.e. when changing a product image)
- No standardised query language across different database models

Real world usage

Given the flexibility of architecting data structures, increased data scalability, integration with modern web applications using JavaScript and performance it’s not surprising that NoSQL databases have grown in popularity in recent years.

Some of the more widely used NoSQL databases include (but are not limited to):

- MongoDB
- Cloud Firestore
- Redis
- DynamoDB
- Apache Cassandra
- Neo4J
- PouchDB

MongoDB and Cloud Firestore are largely familiar with many frontend/mobile app developers due to their strong integration with JavaScript based technology stacks (and these, as you may remember from the contents page, along with PouchDB will be the NoSQL databases that we'll be working with in the pages of this ebook).

To summarise then:

Non-relational database overview	
Pros	Cons
Handles structured, semi-structured and unstructured data efficiently	Schema-less architecture (data integrity enforced from application not database)
Scales well with massive data storage	Not always ACID compliant
Scales well with cloud computing architecture	Difficult to port data between different NoSQL database models
Allows for multiple different types of data structures to be implemented	No standardised query language
API/data structure integrates well with JavaScript in modern cross-platform apps	
Reduced query times due to data architecture	

This then concludes our brief and very basic introduction/overview of Non-relational (or NoSQL) databases.

Unfortunately we are simply not able to cover each NoSQL database model in this book (**graph**, **key-value**, **column** and **document** - due to the volume of concepts and resources that would need to be covered).

As a result of this I have deliberately focussed on document oriented solutions as these are the most widely used NoSQL database model for many developers.

Resources

There's a lot more to learn about databases where SQL and NoSQL are concerned, and we've only scratched the surface covering the basics with this brief chapter.

Further database resources can be explored here:

- Structured Query Language
- Types of databases
- NoSQL

Firestore

Building a Kanban Board

Firebase is one of the most popular, widely used and well established Baas (*Backend as a service*) platforms used by both developers and organisations all over the world.

Firebase is often a de-facto choice for application development due to its rich and user friendly features such as:

- Feature rich API (accompanied with extensive online documentation)
- Platform specific SDKs (iOS, Android, Web, C++ & Unity)
- Framework libraries (AngularFire, VueFire, ReactFire etc)
- Command Line tools for project creation, hosting deployment etc
- Multiple services (Authentication, Storage, NoSQL database, hosting, Analytics etc)

An additional benefit for developers working with the web SDK is the smooth integration and flow between writing JavaScript/TypeScript in their applications and implementing the firebase javascript API to interact with services such as authentication, storage and reading from/writing to NoSQL databases.

There's no split responsibility between using different languages to manage the frontend and backend of an application which makes working with Firebase quite an attractive proposition for many developers in terms of ease of use, integration with existing code and not needing to switch between different languages (such as JavaScript, PHP and MySQL for example).

Having to switch between multiple languages/tools (and navigate their respective features and quirks) can quickly add layers of complexity and integration issues to a project (as well as the potential learning curves that can also be involved).

Firebase nicely eliminates all of that for us with its unified API.

The firebase admin console also allows developers/organisations to perform a wide variety of tasks such as create projects (and their applications), manage billing, monitor application performance/usage, implement security rules, handle configuration settings, perform A/B testing and engage users with targeted contextual messaging…to name but a few services/options that are available.

If you're not familiar with Firebase and not yet sold on its benefits you should be by the end of this chapter!

We'll be using a very, very small subset of the Firebase platform (in subsequent conjunction with an Ionic frontend) to perform the following tasks:

- Create a project
- Integrate the Web SDK for Firebase API usage within Ionic
- Create a Firestore database and manage that data with CRUD operations

What we'll be creating
Our project will produce a simple data-driven Kanban board To Do application with drag-and-drop capabilities.

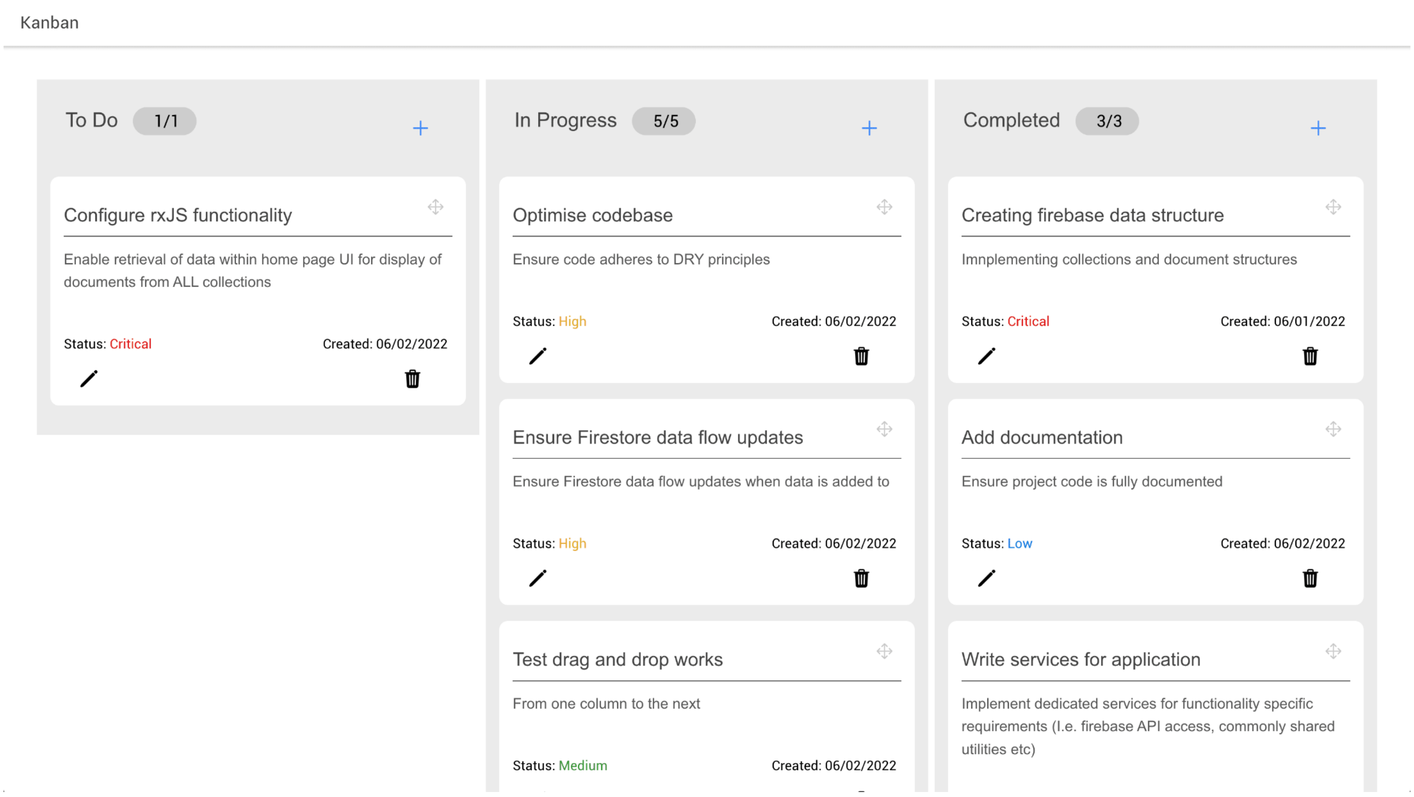
In case you didn't already know a Kanban Board is a visual tool typically used within organisations to depict work at various stages of a process using columns that represent each stage of that process. **Trello** and **Jira** being two examples of popular project management tools that implement this approach.

In prior contracts that I have undertaken Kanban boards have been used by teams to track the progress of particular tasks in a certain iteration of the project lifecycle and may, for example, consist of the following stages (columns): prototyping, development, testing, QA and deployment (with tasks assigned to each column depending on progress - which means they can move forward OR backward!).

To develop our own Kanban board we'll make use of the following tools/libraries:

- Ionic/Angular
- Angular Material/animations
- Firebase storage

By the end of this chapter you should have a fully functioning Kanban board application where you can create, update, read and delete entries as demonstrated below (your content will, of course, differ to mine!)


Kanban

To Do 1/1

Configure rxJS functionality

Enable retrieval of data within home page UI for display of documents from ALL collections

Status: Critical Created: 06/02/2022

In Progress 5/5

Optimise codebase

Ensure code adheres to DRY principles

Status: High Created: 06/02/2022

Ensure Firestore data flow updates

Ensure Firestore data flow updates when data is added to

Status: High Created: 06/02/2022

Test drag and drop works

From one column to the next

Status: Medium Created: 06/02/2022

Completed 3/3

Creating firebase data structure

Imnplementing collections and document structures

Status: Critical Created: 06/01/2022

Add documentation

Ensure project code is fully documented

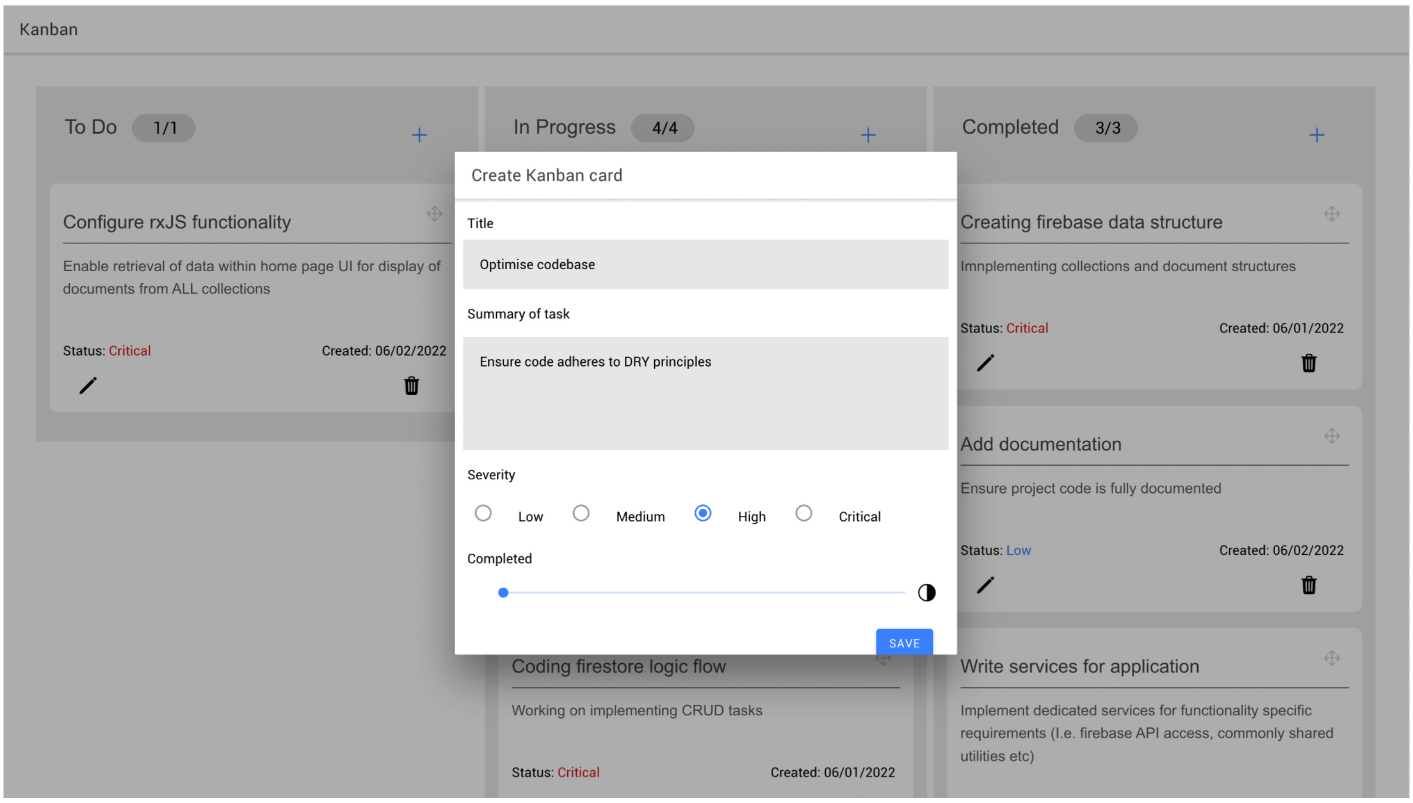
Status: Low Created: 06/02/2022

Write services for application

Implement dedicated services for functionality specific requirements (I.e. firebase API access, commonly shared utilities etc)

Our Kanban board consists of three columns: **To Do**, **In Progress** and **Completed**.

In each of these columns users can create, update and delete entries and subsequently drag existing entries between the different columns.


Kanban

To Do 1/1

Configure rxJS functionality

Enable retrieval of data within home page UI for display of documents from ALL collections

Status: Critical Created: 06/02/2022

In Progress 4/4

Create Kanban card

Title

Optimise codebase

Summary of task

Ensure code adheres to DRY principles

Severity

Low Medium [✓ High] Critical

Completed

SAVE

Coding firestore logic flow

Working on implementing CRUD tasks

Status: Critical Created: 06/01/2022

Completed 3/3

Creating firebase data structure

Imnplementing collections and document structures

Status: Critical Created: 06/01/2022

Add documentation

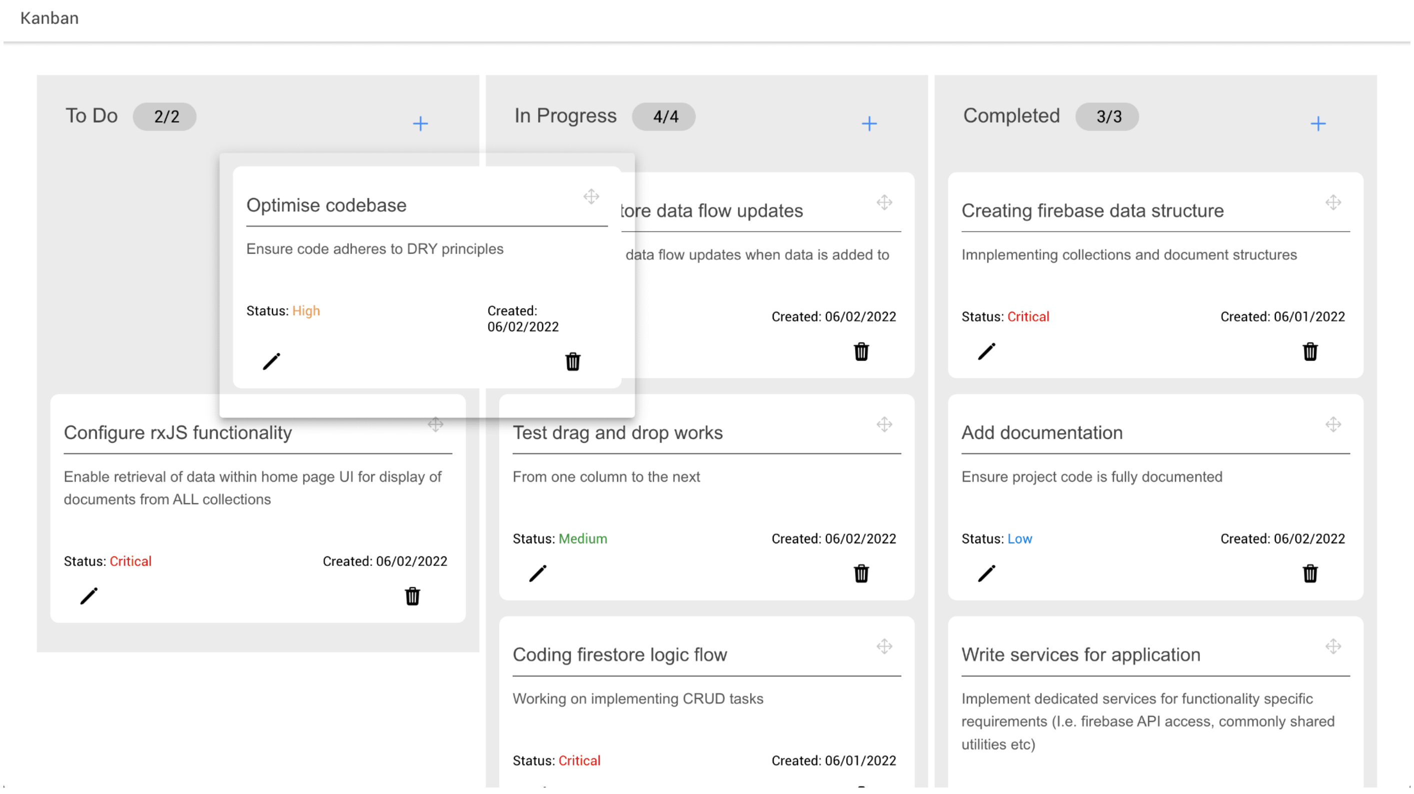
Ensure project code is fully documented

Status: Low Created: 06/02/2022

Write services for application

Implement dedicated services for functionality specific requirements (I.e. firebase API access, commonly shared utilities etc)

Dragging and dropping between columns is made possible and aided with the Angular Material and Animations libraries:



Kanban

To Do 2/2 +

Optimise codebase
Ensure code adheres to DRY principles
Status: High Created: 06/02/2022

Configure rxJS functionality
Enable retrieval of data within home page UI for display of documents from ALL collections
Status: Critical Created: 06/02/2022

In Progress 4/4 +

tore data flow updates
data flow updates when data is added to
Created: 06/02/2022

Test drag and drop works
From one column to the next
Status: Medium Created: 06/02/2022

Coding firestore logic flow
Working on implementing CRUD tasks
Status: Critical Created: 06/01/2022

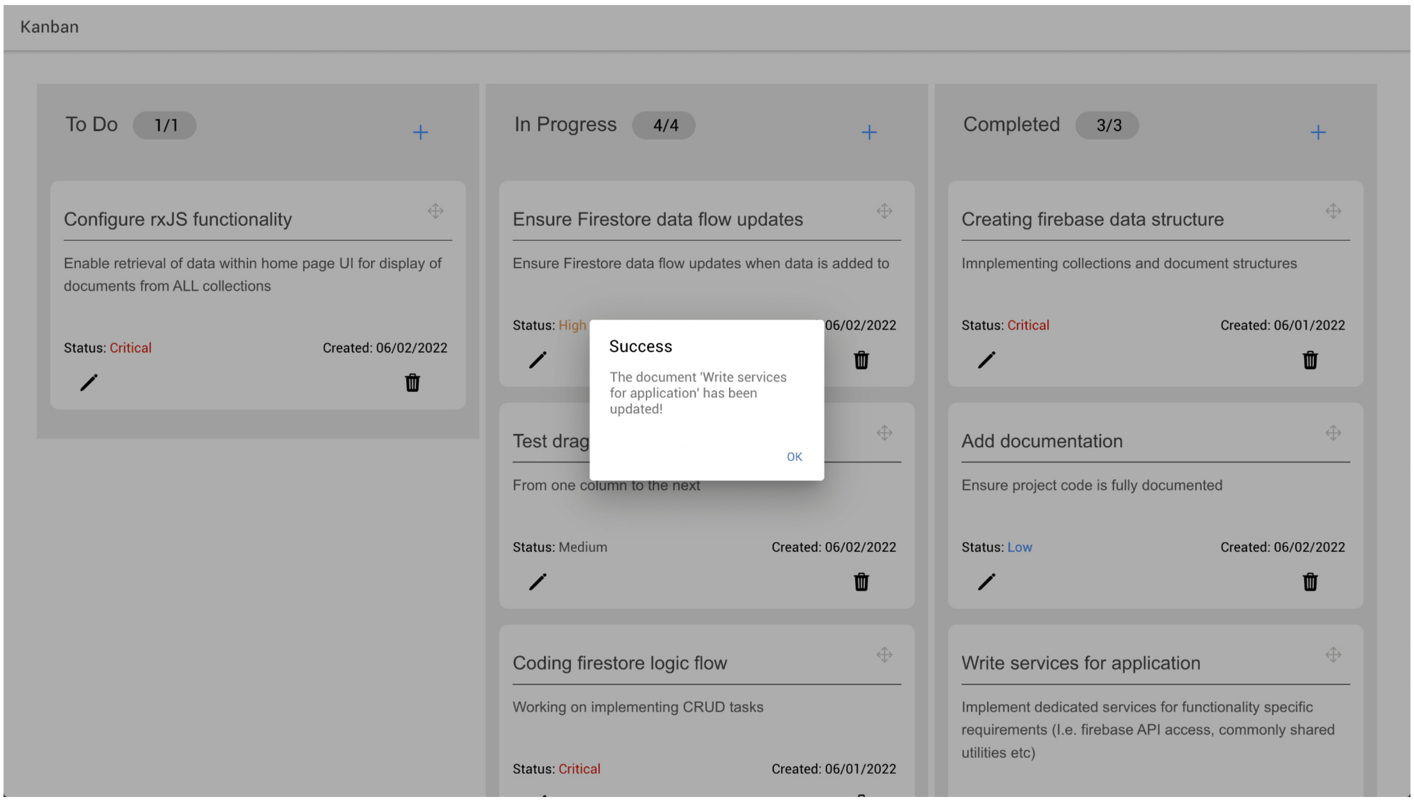
Completed 3/3 +

Creating firebase data structure
Imnplementing collections and document structures
Status: Critical Created: 06/01/2022

Add documentation
Ensure project code is fully documented
Status: Low Created: 06/02/2022

Write services for application
Implement dedicated services for functionality specific requirements (I.e. firebase API access, commonly shared utilities etc)

Users are immediately informed of the state of a card being created, updated or removed with an Ionic AlertController window, The kanban columns are then updated to display the cards after this state has been completed (i.e. totals in column headers are changed and columns shrink/grow accordingly):



Kanban

To Do 1/1 +

Configure rxJS functionality
Enable retrieval of data within home page UI for display of documents from ALL collections
Status: Critical Created: 06/02/2022

In Progress 4/4 +

Ensure Firestore data flow updates
Ensure Firestore data flow updates when data is added to
Status: High 06/02/2022

Test drag
From one column to the next
Status: Medium Created: 06/02/2022

Coding firestore logic flow
Working on implementing CRUD tasks
Status: Critical Created: 06/01/2022

Success
The document 'Write services for application' has been updated!
OK

Completed 3/3 +

Creating firebase data structure
Imnplementing collections and document structures
Status: Critical Created: 06/01/2022

Add documentation
Ensure project code is fully documented
Status: Low Created: 06/02/2022

Write services for application
Implement dedicated services for functionality specific requirements (I.e. firebase API access, commonly shared utilities etc)

Pretty simple right?

Let’s get started then!

Settings the foundations

There are two key aspects to the project that we need to put in place:

- The Ionic application
- The Firebase project

We’ll start with generating the basic structure for our Ionic application and, once this is completed, move onto setting the foundations for our Firebase project.

Open up your command line terminal of choice, navigate to a preferred location on your computer and create a new ionic project named **ionic-firestore-kanban** and, once completed, generate the necessary Angular services & custom components before finishing off with installing the required npm packages:

```
ionic start ionic-firestore-kanban blank --type=angular
// Select NgModules from the menu options prompt
cd ./ionic-firestore-kanban
ionic generate component components/kanban-column
ionic generate component components/kanban-card
ionic generate component components/kanban-editor
ionic generate module components/shared-components
ionic generate service services/data-change-listener
ionic generate service services/database
ionic generate service services/utilities
npm i firebase --save
npm i @angular/cdk @angular/material @angular/animations --save
```

Before we move onto creating our Firebase project (that we’ll subsequently integrate with our newly created **Ionic Firestore Kanban** application) let’s quickly cover the components and services we generated so that we understand their purpose.

Our custom Angular components (within the **app/components** directory) are:

- **KanbanColumnComponent** - As the name suggests this component serves as the representation for each stage displayed in the Kanban board (the columns **To Do**, **In Progress** and **Completed**)
- **KanbanCardComponent** - This provides the functionality, templating & styling for each work item displayed in the **KanbanColumnComponent** (when rendered as the columns representing the different stages of the Kanban process)
- **KanbanEditorComponent** - This provides the form functionality to create/update a Kanban board card for the selected column/stage of the process

Within the **app/components/shared-components** directory we also declare an Angular feature module to allow our custom components to be exported for use throughout different areas of the application:

- **SharedComponentsModule** - Angular module that exports the Kanban board components allowing them to be imported into other application modules where requested (and not imported across ALL of the application - as they would be if imported within the application root module - making this a more optimised approach to managing component usage)

Finally, within the **app/services** directory, we have the following services:

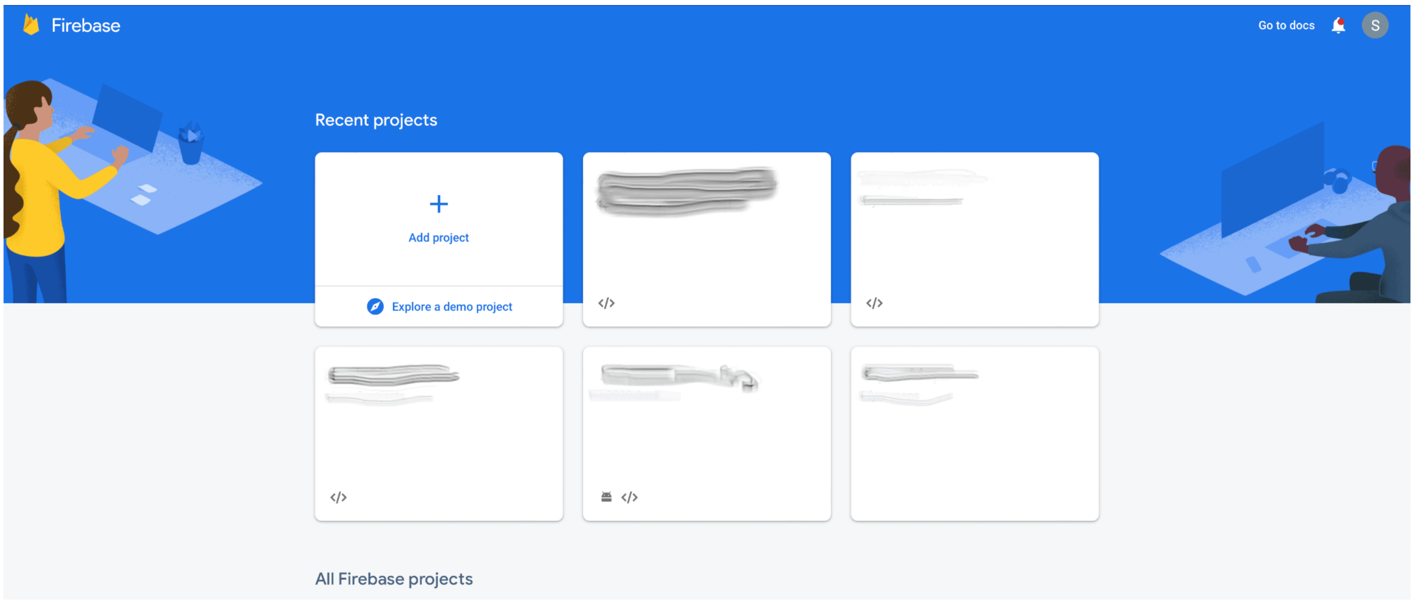
- **DataChangeListenerService** - Uses the rxJS library BehaviorSubject for cross component communication where data changes need to be subscribed (and responded) to
- **DatabaseService** - Uses Firestore API to manage CRUD operations for Kanban board
- **UtilitiesService** - Provides “helper” methods used by the application such as generating date values and determining column functionality

We’ll cover these in more depth shortly but first let’s turn our attention to creating the Firebase project that will be integrated with the **Ionic Firestore Kanban** application (otherwise our Kanban Board will have no data persistence…and that’s kind of important for our application to function effectively!)

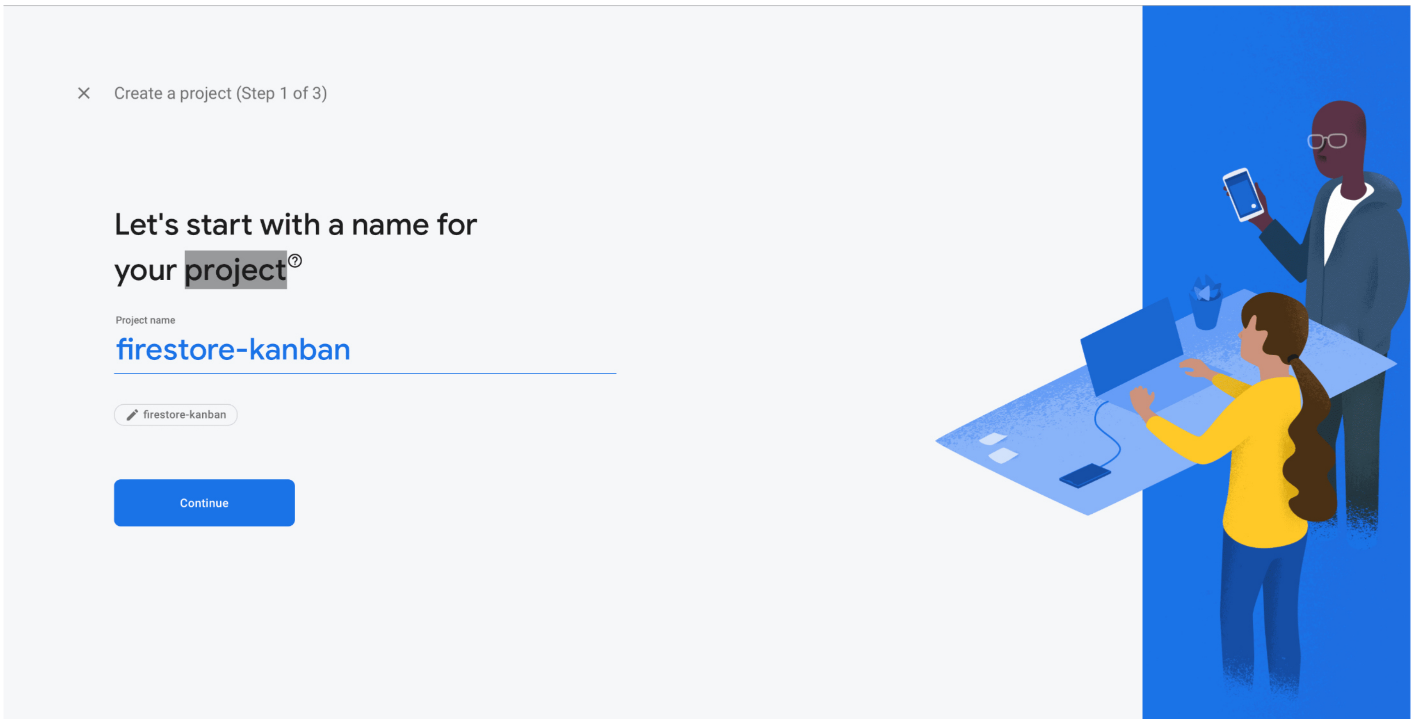
Implementing our Firebase project

If you don't already have a Firebase account head over to firebase.com and sign in using your Google account - assuming you have one…and if you don't, you'll need to create one for this project! :)

If you already have a Firebase account login at firebase.com, head to your console and select the **Add project** button featured at the start of your **Recent projects** list:



On the **Create a project** screen that we transition to the first step requires entering a name for the project (simply enter **firestore-kanban** as shown in the screen capture below):


Create a project (Step 1 of 3)

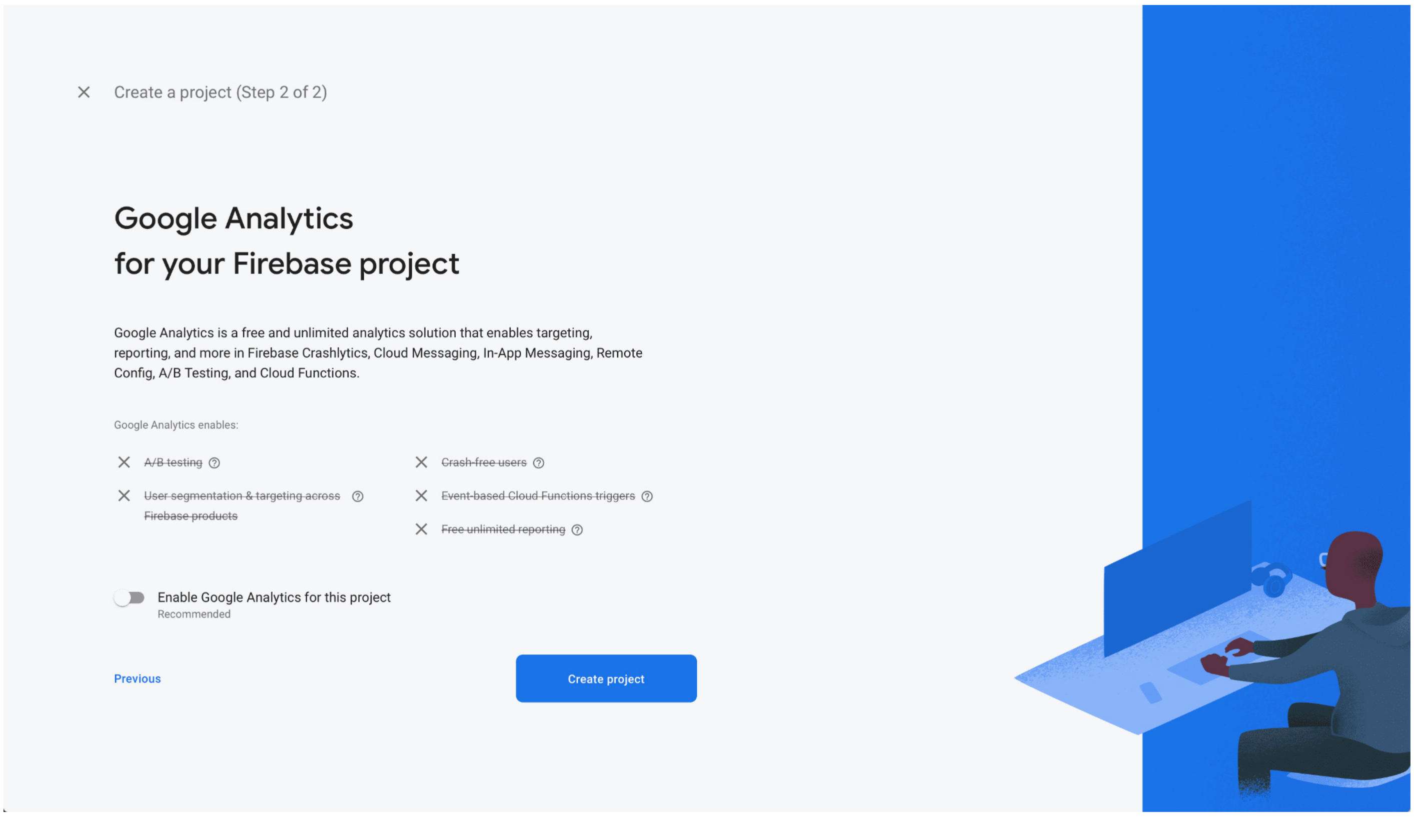
Let's start with a name for your project

Project name

firestore-kanban

firestore-kanban

The second step for this process allows us to enable Google Analytics for our project (this won't be needed for the Ionic application so ensure this is deselected):



Create a project (Step 2 of 2)

Google Analytics
for your Firebase project

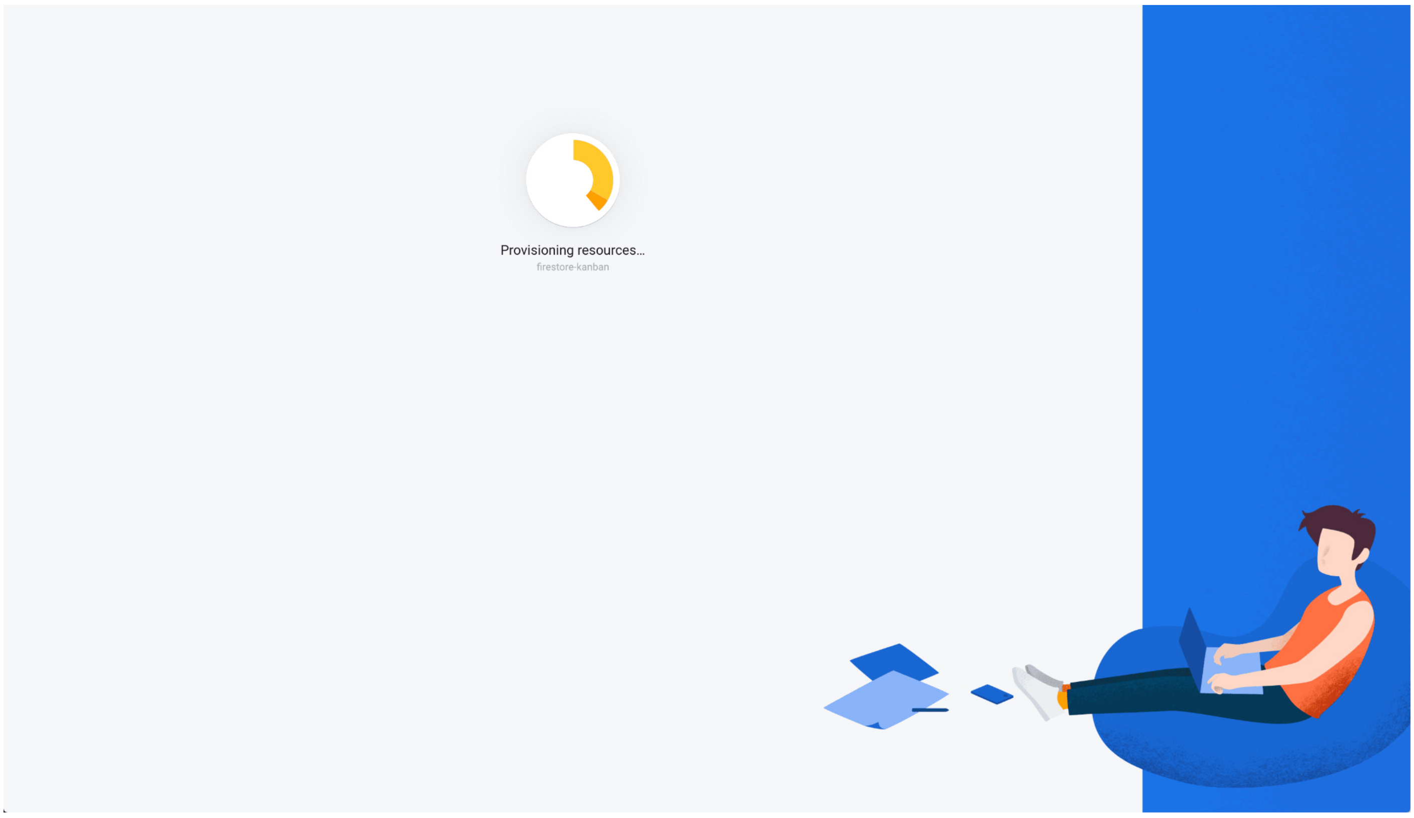
Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, and Cloud Functions.

Google Analytics enables:

- ~~A/B testing~~
- ~~User segmentation & targeting across Firebase products~~
- ~~Crash-free users~~
- ~~Event-based Cloud Functions triggers~~
- ~~Free unlimited reporting~~

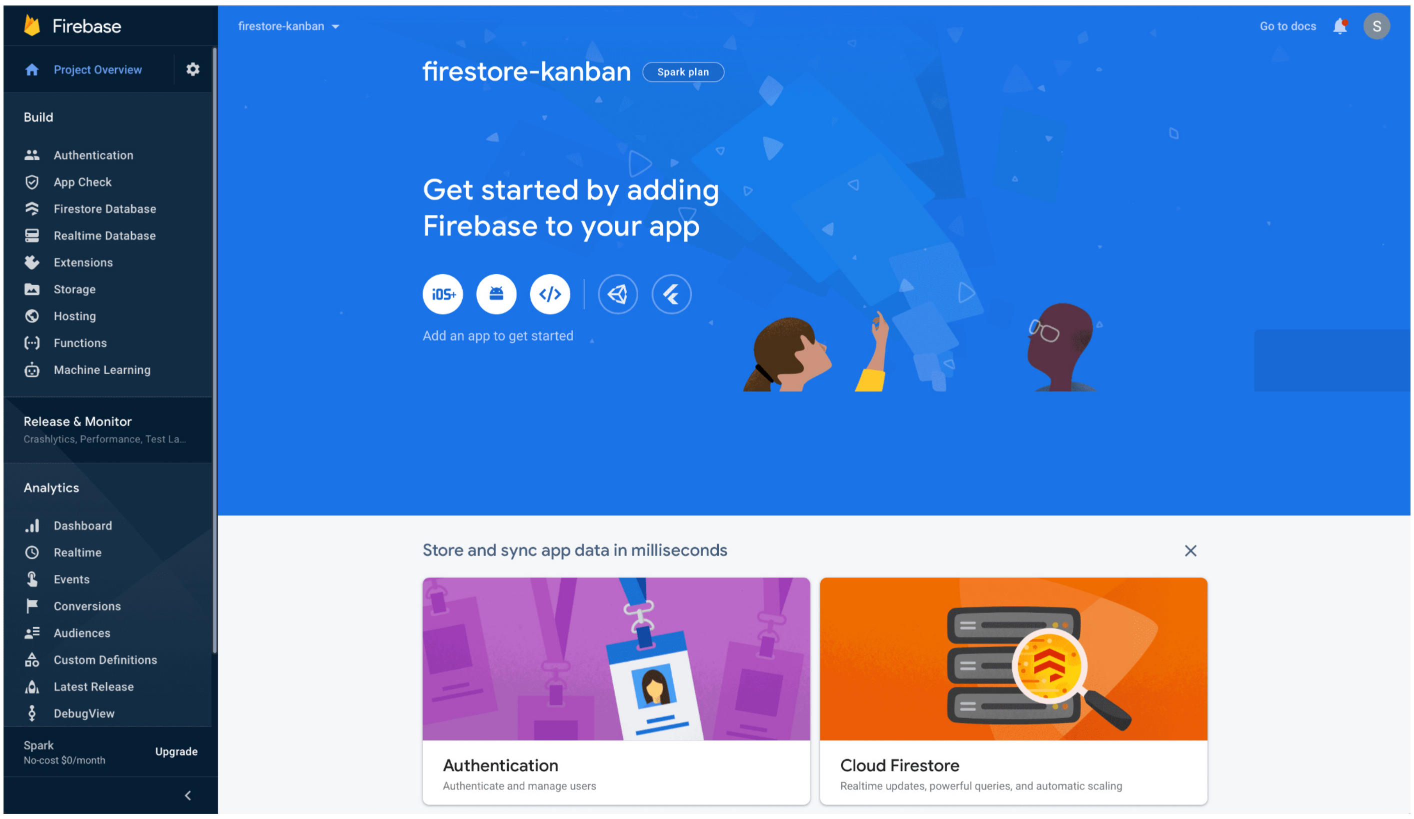
Enable Google Analytics for this project
Recommended

Click on the **Create project** button and wait for the project to be generated:



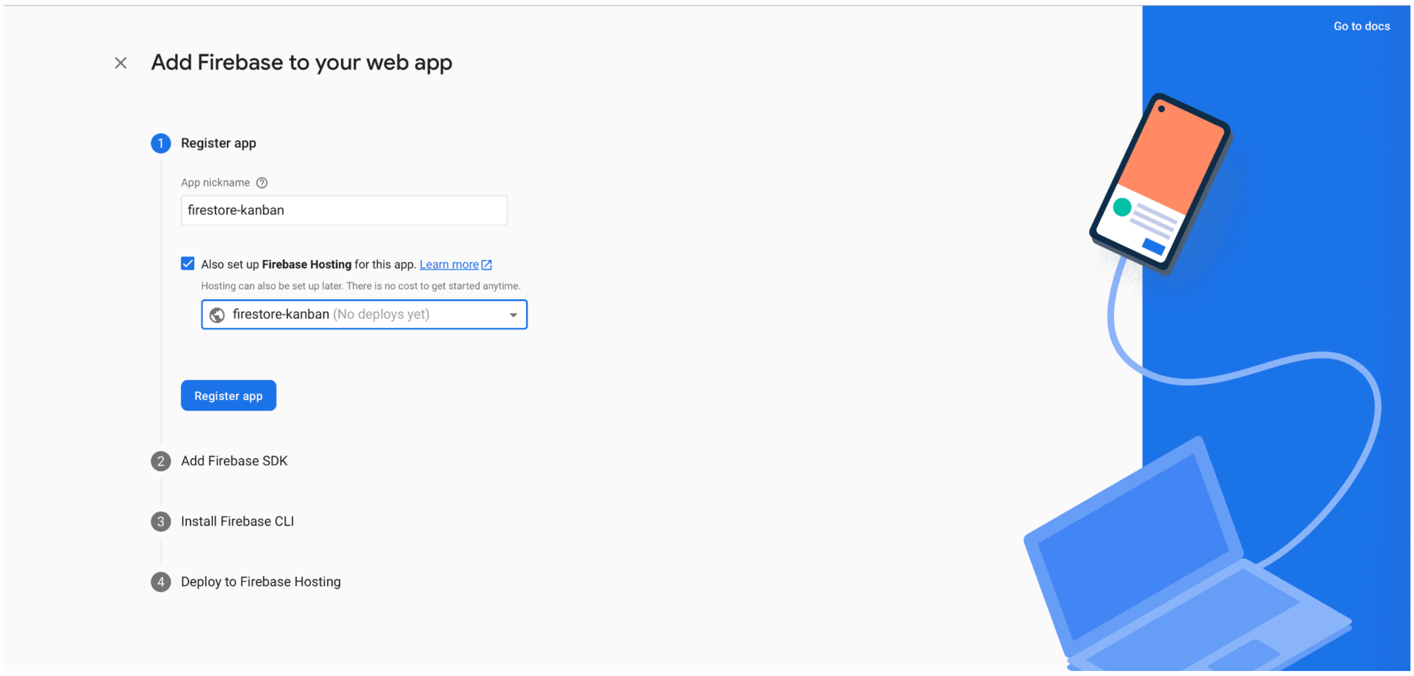
Provisioning resources...
firestore-kanban

Once created click on the Continue button that is displayed and you will be directed to a screen displaying a **Get started by adding Firebase to your app** heading.



A firebase project (the **firestore-kanban** project that we just created) is simply a container within which iOS, Android and Web applications can be assigned to.

Here you will need to select the Web Icon (the HTML tag displaying a forward slash) after which you are presented with an *Add Firebase to your web app* wizard whose first step invites you to *Register app* (enter the name **firestore-kanban** and select Firebase Hosting (which automatically assigns **firestore-kanban** as the app value:



App nickname: firestore-kanban

☑ Also set up **Firebase Hosting** for this app. Learn more

Hosting can also be set up later. There is no cost to get started anytime.

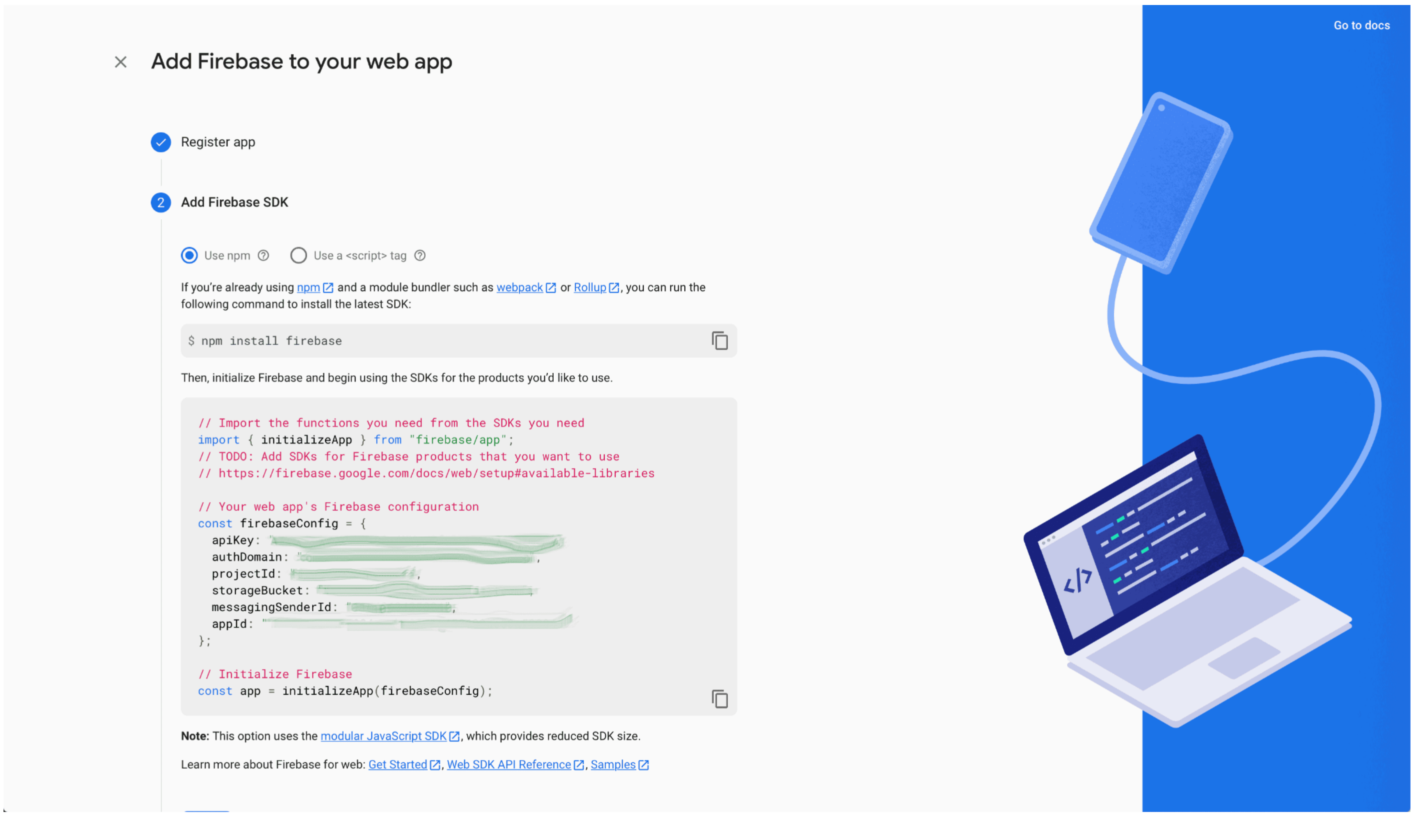
firestore-kanban (No deploys yet)

Click on the Register app button once completed.

The second step of the wizard prompts you to *Add Firebase SDK* (which we can safely skip as this was previously added during the creation of our Ionic application earlier in this chapter).

We are also presented with a block of initialisation code which needs to be added to our Ionic application. This initialisation code allows our app to “speak” with Firebase and use its API across various services such as Authentication, Cloud Firestore and Cloud Storage (to name but a few that are available for the Firebase Web app).

Copy this initialisation code and paste into a temporary text file - we’ll come back to this later when we return to working on the Ionic application codebase - and click the Next button to advance to the third stage of the wizard.



Add Firebase to your web app

✓ Register app

2 Add Firebase SDK

[✓ Use npm] Use a <script> tag

If you’re already using npm and a module bundler such as webpack or Rollup, you can run the following command to install the latest SDK:

```
$ npm install firebase
```

Then, initialize Firebase and begin using the SDKs for the products you’d like to use.

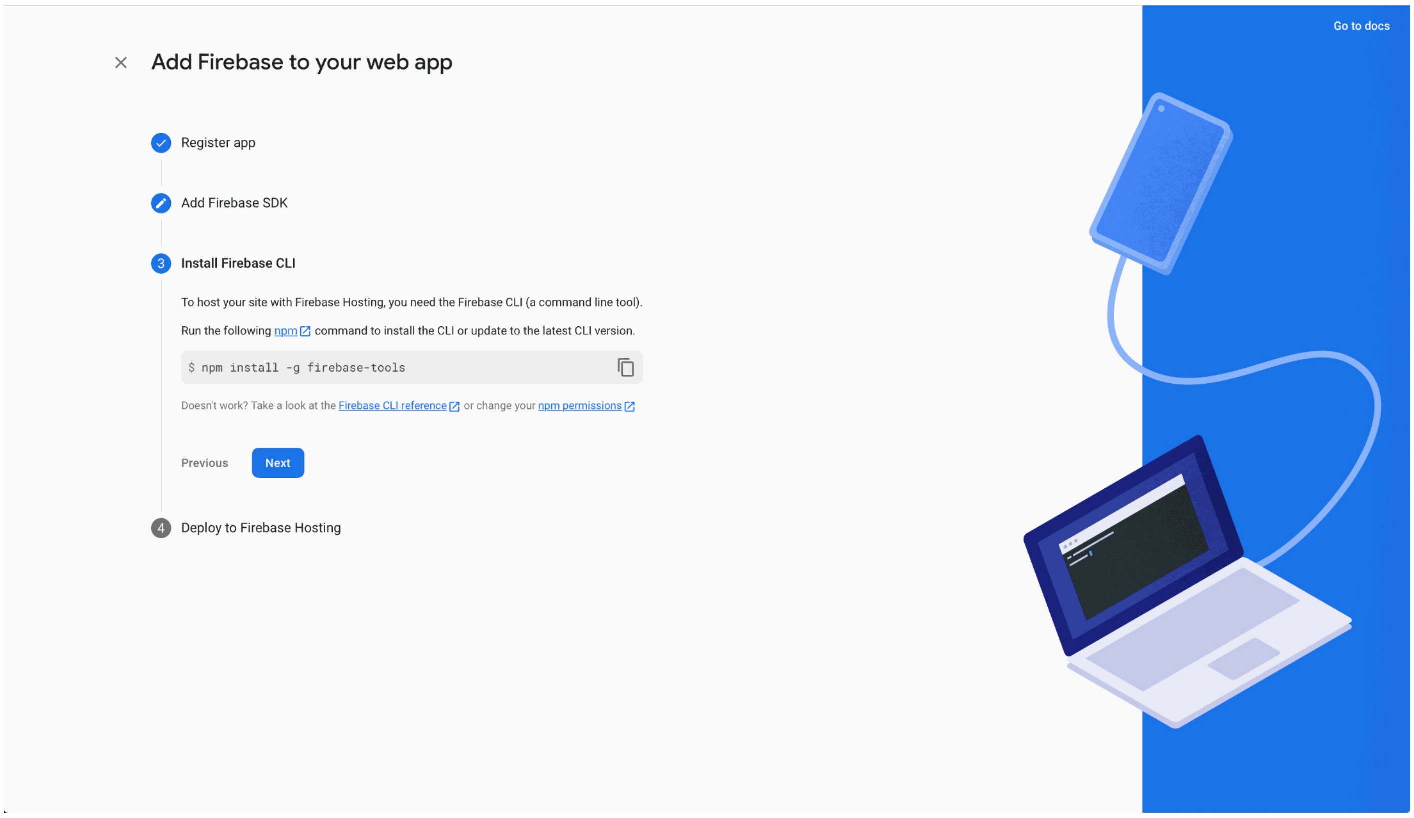
```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: [redacted]
  authDomain: [redacted],
  projectId: [redacted],
  storageBucket: [redacted]
  messagingSenderId: [redacted],
  appId: [redacted]
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Note: This option uses the modular JavaScript SDK, which provides reduced SDK size.

Learn more about Firebase for web: Get Started, Web SDK API Reference, Samples

Here we are prompted to install the Firebase CLI which allows developers to deploy their projects to Firebase Hosting from the command line.

Copy the displayed npm command and run this in your system software terminal before proceeding with the remainder of the wizard.



Add Firebase to your web app

Register app

Add Firebase SDK

3 Install Firebase CLI

To host your site with Firebase Hosting, you need the Firebase CLI (a command line tool).

Run the following npm command to install the CLI or update to the latest CLI version.

```
$ npm install -g firebase-tools
```

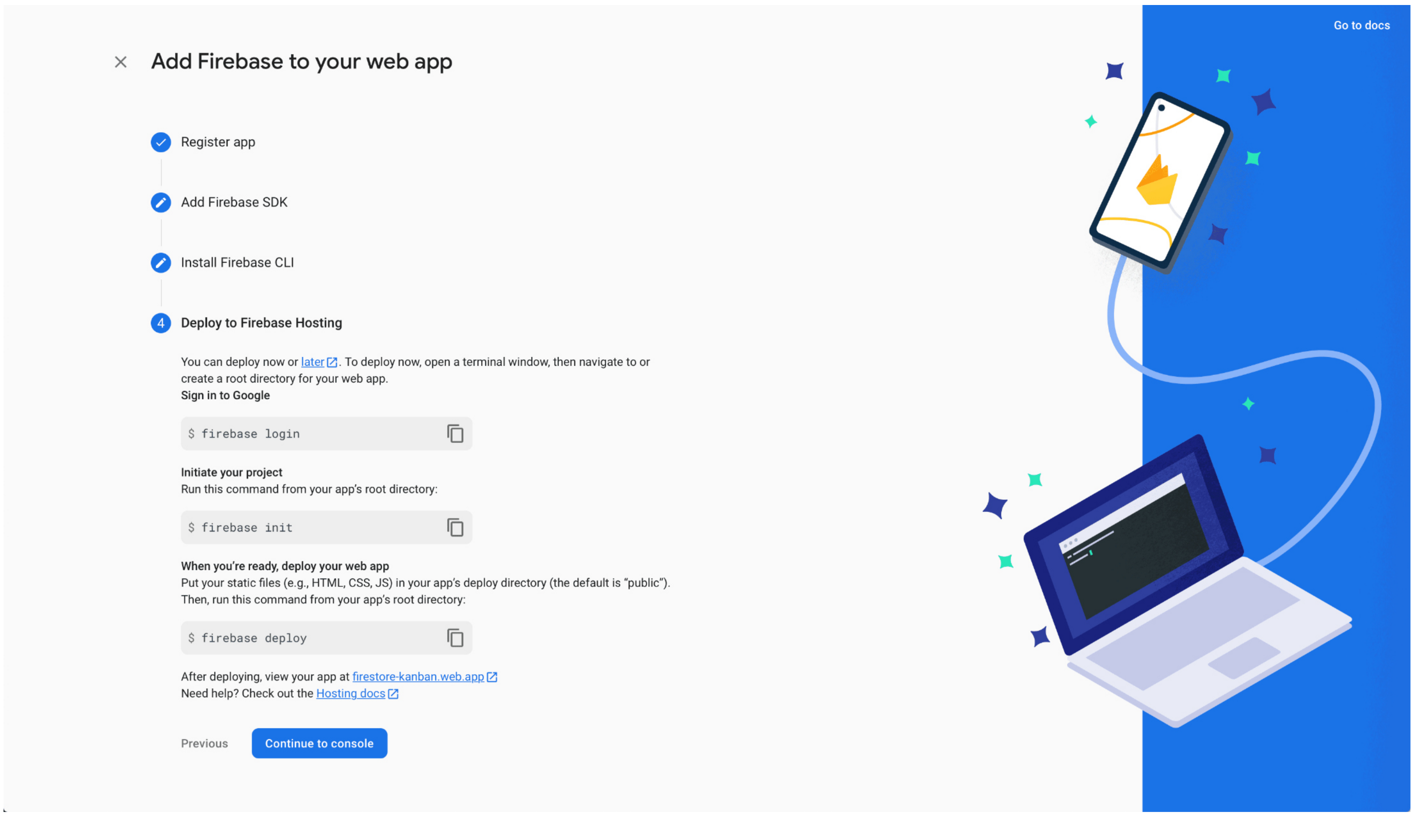
Doesn't work? Take a look at the Firebase CLI reference or change your npm permissions

Previous Next

4 Deploy to Firebase Hosting

The fourth and final stage of the wizard provides us with the necessary command line instructions and information required to deploy our project to Firebase Hosting.

We won’t do this right now (as there’s nothing to currently deploy) but we will revisit this later once development for our ionic application has been completed.



Add Firebase to your web app

Register app

Add Firebase SDK

Install Firebase CLI

4 Deploy to Firebase Hosting

You can deploy now or later. To deploy now, open a terminal window, then navigate to or create a root directory for your web app.

Sign in to Google

```
$ firebase login
```

Initiate your project

Run this command from your app’s root directory:

```
$ firebase init
```

When you’re ready, deploy your web app

Put your static files (e.g., HTML, CSS, JS) in your app’s deploy directory (the default is “public”). Then, run this command from your app’s root directory:

```
$ firebase deploy
```

After deploying, view your app at firestore-kanban.web.app

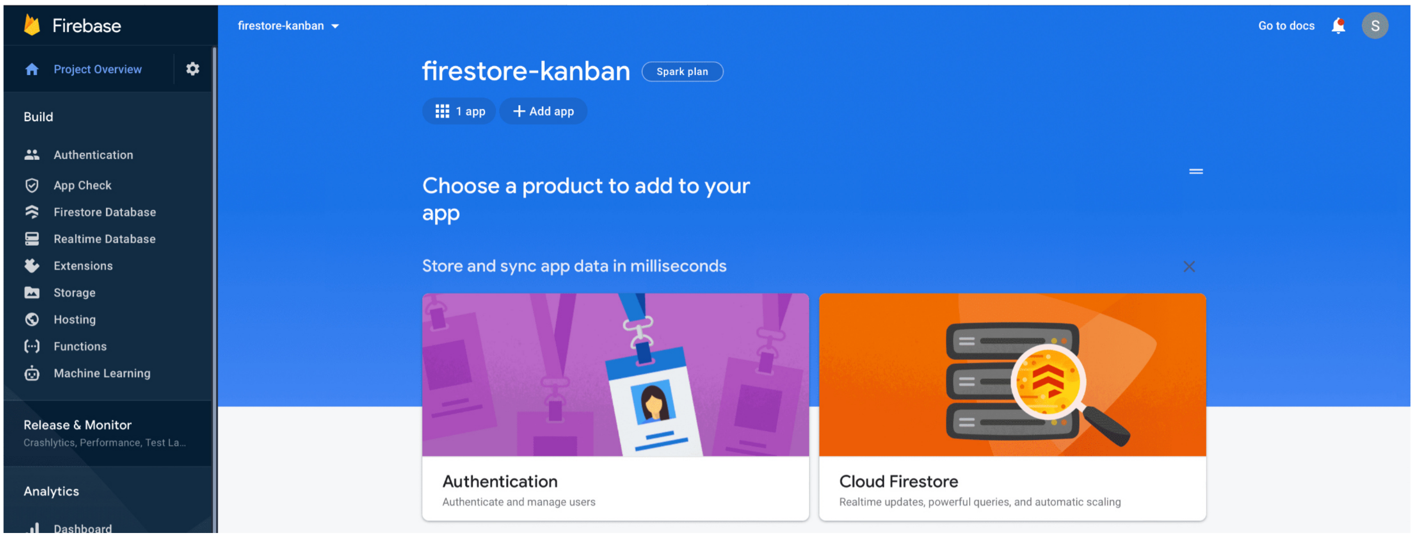
Need help? Check out the Hosting docs

Previous Continue to console

Once completed you are then redirected to the console for the **firestore-kanban** project. Notice the button underneath the project title informing you of one application associated with the project? This is the Web application that we just created.

The **Add app** button sat to the right of this allows you to add further application platforms to the firebase project including: *iOS*, *Android*, *Unity* & *Flutter*.

For our purposes we only need to concentrate on the Web application that we recently created for our Firebase project.



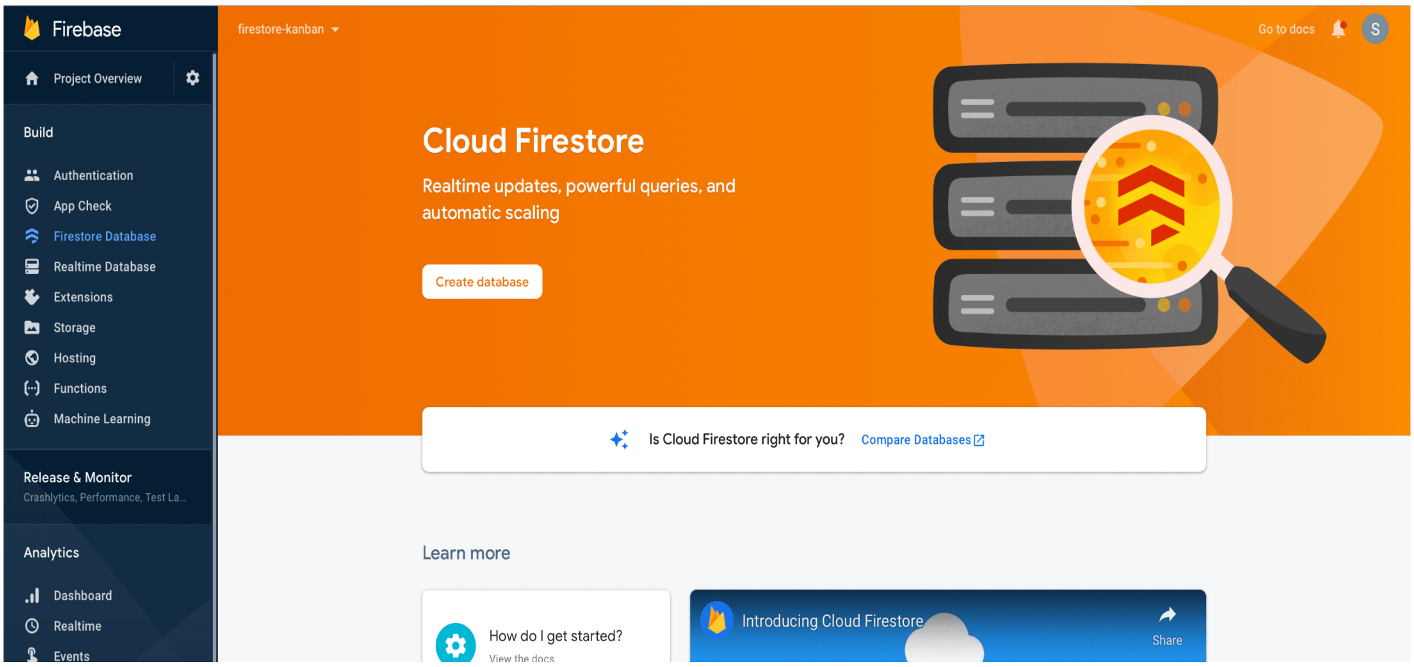
From the product listings displayed on the console home page click onto the Cloud Firestore link - this is where we'll start to configure the project database as well as associated security rules for data access.

Cloud Firestore is a NoSQL database that stores data using **collections** (data containers that store & organise documents) and **documents** (which represent individual records) in a JSON-like format.

A document contains key/value mappings which can correspond to any of the following data types: boolean, number, string, geo point, BLOB (Binary Large Object), timestamp, arrays & nested objects (aka maps).

As you might guess this wide range of data types gives us quite a lot of flexibility in terms of what data we can store when creating documents within a Cloud Firestore database.

From the Cloud Firestore home screen click the **Create database** button to begin.



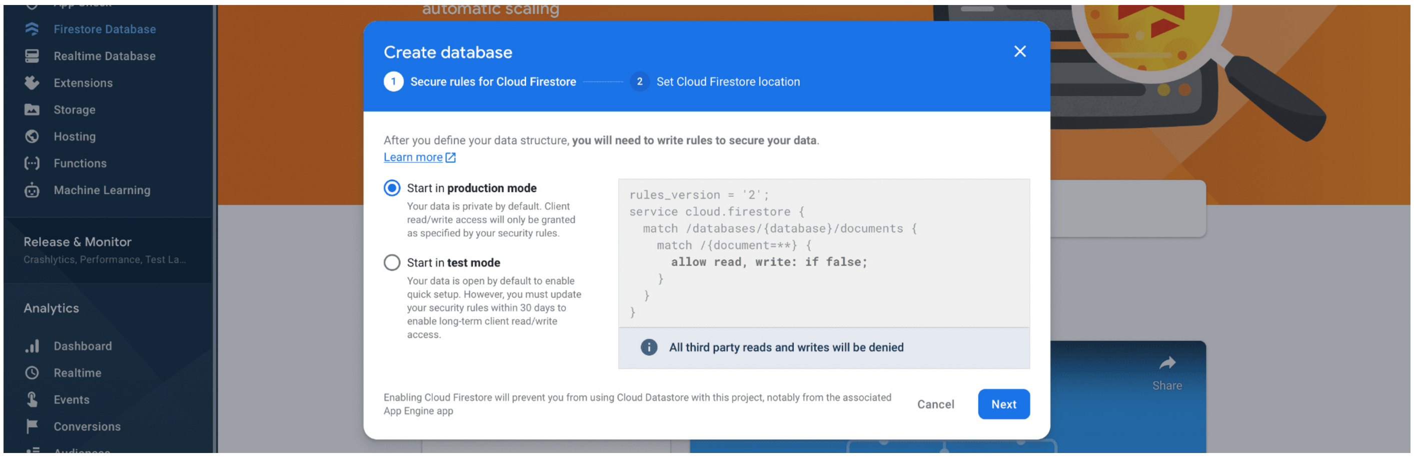
The Create database popup window that appears presents 2 steps to be completed:

- Secure rules for Cloud Firestore
- Set Cloud Firestore location

Starting with the secure rules for the database step simply select the Start in test mode option which will grant default public access to the Cloud Firestore database.

For the purpose of this chapter this will suffice but be aware this is a security risk and must not be used in a production context. Realistically if you're using Cloud Firestore even in a development context you should lock down security but this does require that Firebase authentication is enabled to allow access.

As we are not enabling/using Firebase authentication we deliberately set the rules for the database to start in test mode.


Create database

1 Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

After you define your data structure, **you will need to write rules to secure your data**.
Learn more

[✓ Start in **production mode**]
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

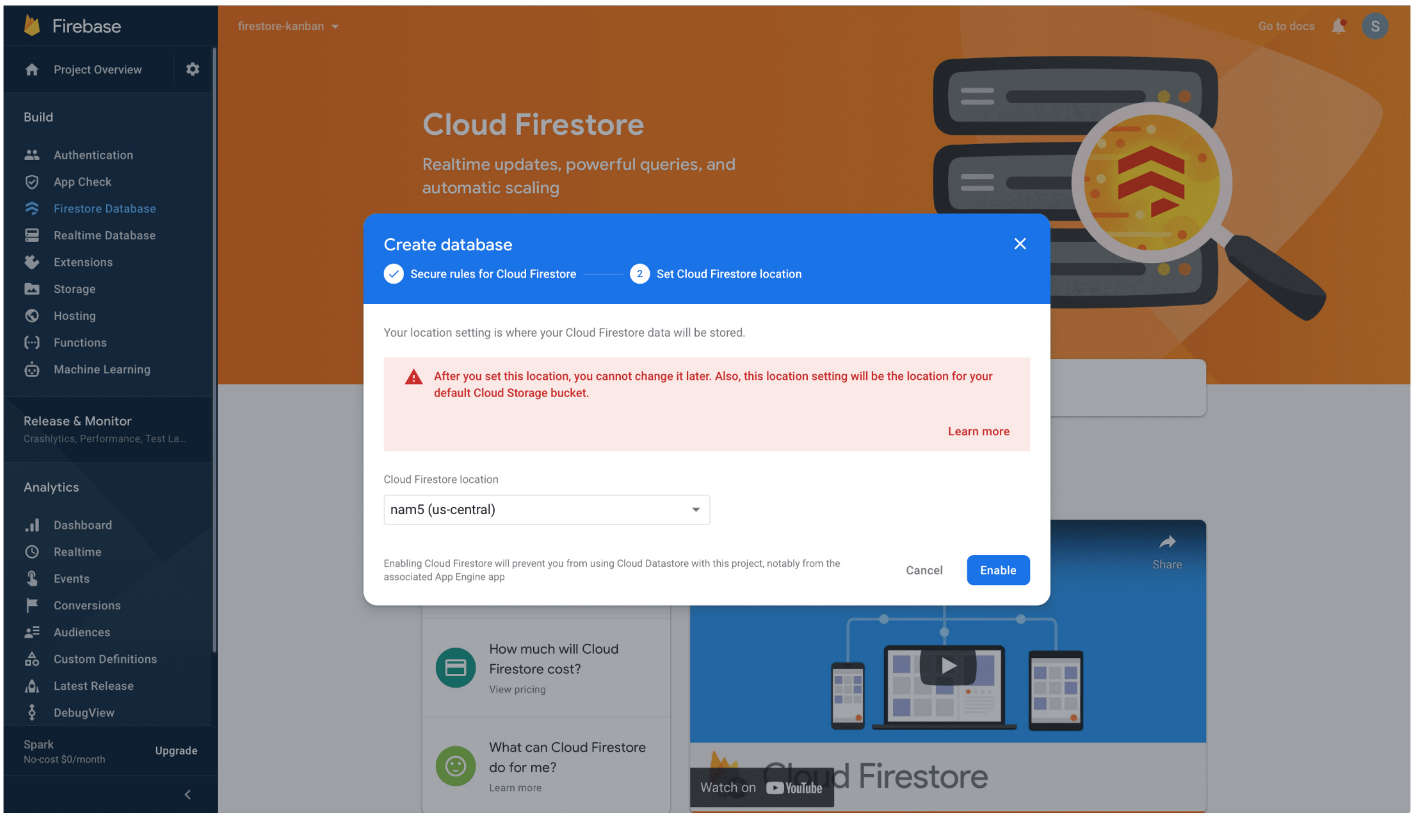
Start in **test mode**
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

All third party reads and writes will be denied

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Cancel Next



Create database

Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

Your location setting is where your Cloud Firestore data will be stored.

After you set this location, you cannot change it later. Also, this location setting will be the location for your default Cloud Storage bucket.

Learn more

Cloud Firestore location: nam5 (us-central)

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Cancel | Enable

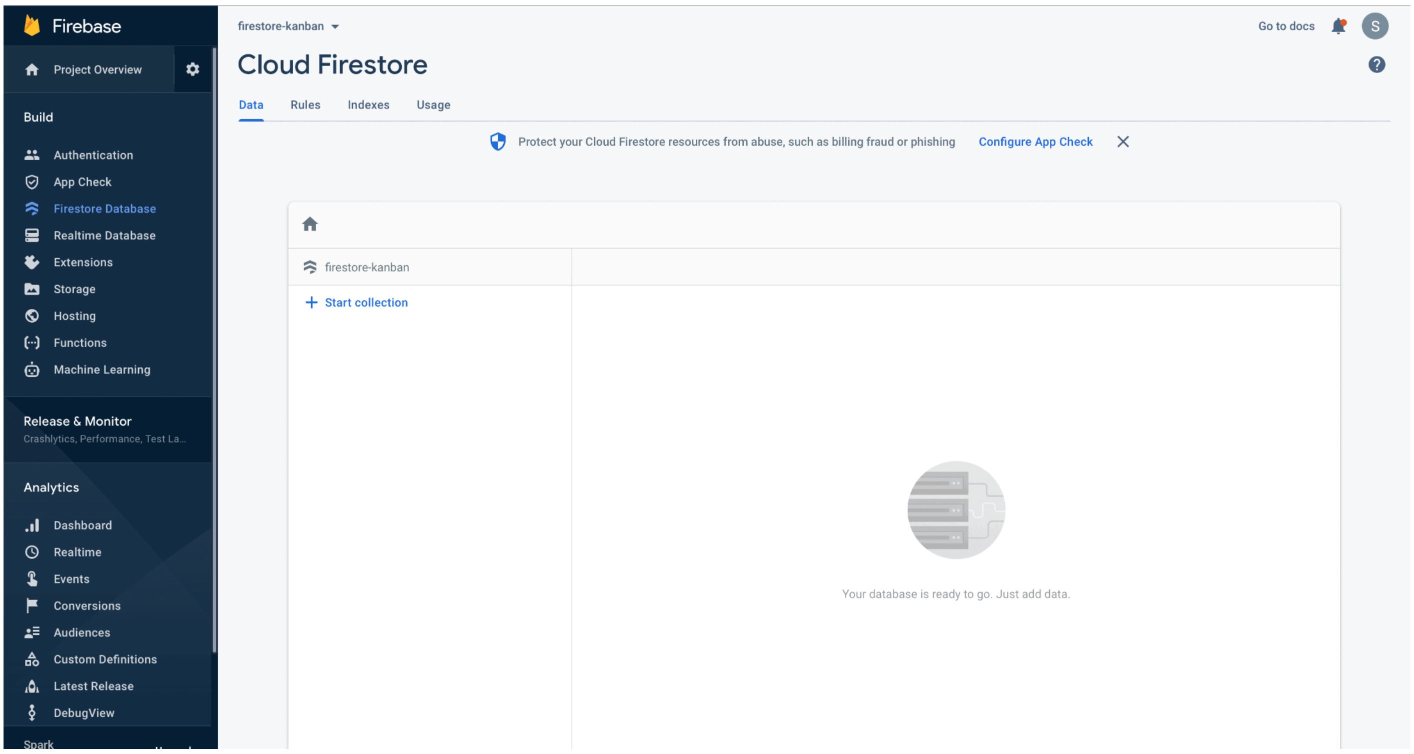
How much will Cloud Firestore cost?
View pricing

What can Cloud Firestore do for me?
Learn more

In the final step of the Create database popup window we are asked to set the location of the Cloud Firestore database.

The default is nam5 (us-central) which we can safely leave as our selected location.

Click the **Enable** button and wait for the database creation script to complete before our newly generated Cloud Firestore database is loaded and displayed:



Cloud Firestore

Protect your Cloud Firestore resources from abuse, such as billing fraud or phishing — Configure App Check

firestore-kanban

+ Start collection

Your database is ready to go. Just add data.

With this final step we've now concluded our firebase configuration for the **Ionic Firebase Kanban** project so let's now return to the Ionic project codebase and start building this out.

Integrating Firebase

Remember that Firebase initialisation code I told you to copy and paste into a temporary text file earlier in this chapter?

Let's start with adding that code to our Ionic application so we can communicate with the Firebase service and make API calls.

Open both of the following files:

- **ionic-firestore-kanban/src/environments/environment.ts**
- **ionic-firestore-kanban/src/environments/environment.prod.ts**

Within each of these files add the following block of code (using the values from the Firebase initialisation code that you pasted into that temporary text file) within the body of the declared **environment** object:

```
firebase : {
    apiKey: 'YOUR-COPIED-API-KEY-VALUE-PASTED-HERE',
    authDomain: 'YOUR-COPIED-AUTHDOMAIN-VALUE-PASTED-HERE',
    projectId: 'YOUR-COPIED-PROJECT-ID-VALUE-PASTED-HERE',
    storageBucket: 'YOUR-COPIED-BUCKET-VALUE-PASTED-HERE',
    messagingSenderId: 'YOUR-COPIED-SENDER-VALUE-PASTED-HERE',
    appId: 'YOUR-COPIED-APP-ID-VALUE-PASTED-HERE'
  },
  collections: {
    toDos: 'toDos',
    inProgress: 'inProgress',
    completed: 'completed'
  }
```

Here we simply declare the Firebase initialisation code within its own self-contained **firebase** object and include an additional **collections** object which provides names for the database collections that we will be using for our Cloud Firestore database.

End of preview