# Architecture and Design
*r/place*

## Goal
To create a simple, elegant clone of Reddit's r/place event. This web application will allow users to collaborate by placing colored tiles, one at a time, on a shared grid-based canvas. Users will be able to view the artwork throughout its evolution as different users contribute, resulting in a beautiful, shared work of art.

## Architecture and Design Description
The overall architecture will follow the Model-View-ViewModel (MVVM) pattern. This pattern allows for the clear separation of responsibilities between application parts, and makes it easy to keep state synchronized across the layers of the application. This will also allow team members to work on various parts of the application in parallel without conflicts.

### Frontend
The frontend application will be built on the Angular framework (which is optimized for MVVM applications) and will run in the user's browser. It will consist of Views (HTML Templates) paired with ViewModels (TypeScript source files containing app logic). Angular allows the Views and ViewModels to be connected via two-way data binding, keeping them in sync as their states change. Angular provides a powerful CLI to help in generating boilerplate code and quickly serving locally for development and debugging.

### Backend Server
The server will be built on the Node.JS framework. This will allow the backend to use the same language as the frontend (TypeScript is a superset of JavaScript), allowing engineers to easily switch between working on either side. It will use Express.JS to expose an HTTP Web API, consisting of GET and POST endpoints for retrieving and updating data. Express.JS is a simple, popular, and reliable library for creating Web APIs. The server will handle API requests by querying/updating the database, constructing HTTP responses, and sending them to the client. It will communicate with the Firebase Cloud Firestore database using Firebase's JavaScript API. The server is also responsible for serving the Angular application itself to the user's browser via HTTP upon connection. It will be hosted and deployed using Google Firebase or Docker.

### Database
We will use Firebase Cloud Firestore to store and manage data. It is a NoSQL, document-based database, which allows for flexibility should the structure of the data

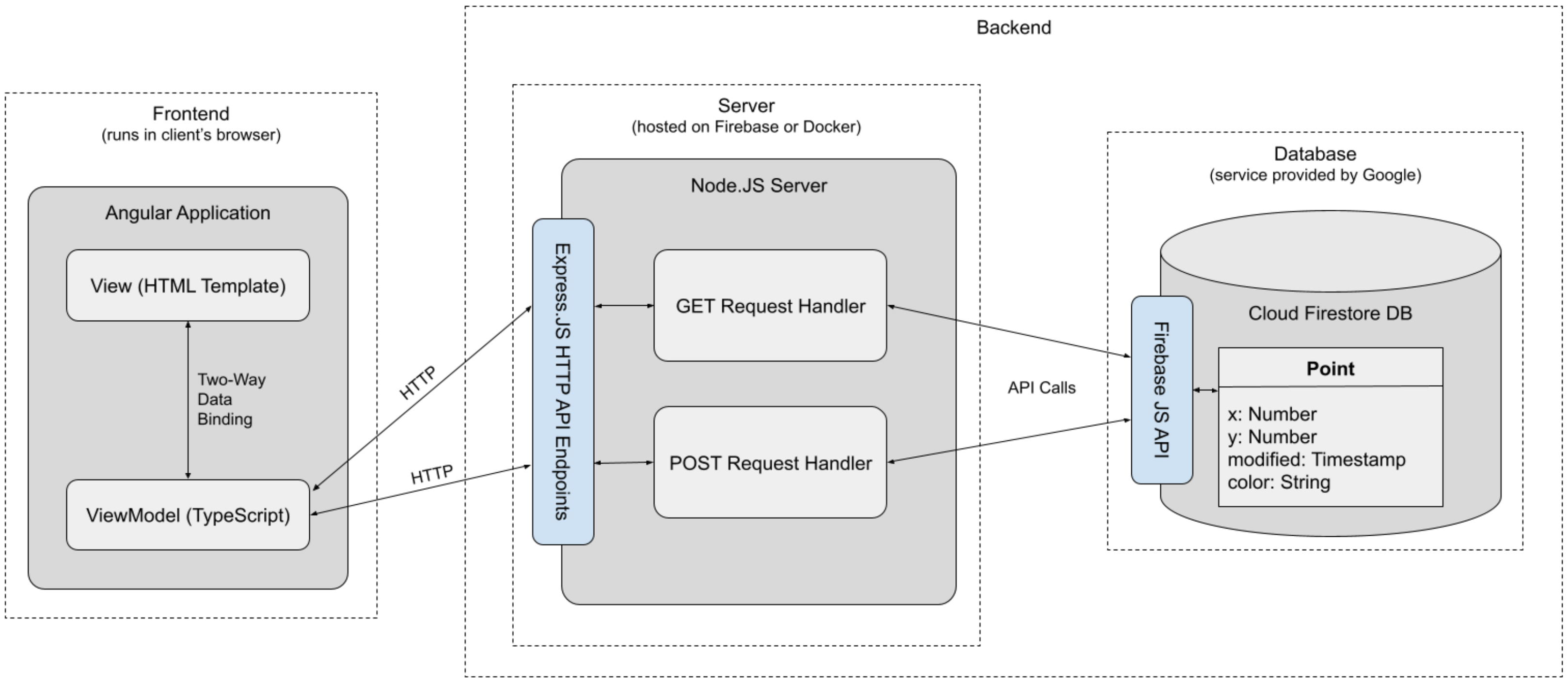
need to change in the future. It also provides a JavaScript API for rapid development and easy integration into the Node.JS server. Because Firebase is provided as a service by Google, we do not need to host or manage the DBMS ourselves.

The only collection in the database will contain “Pixels.” This collection of pixels makes up the shared canvas that is central to the application. Each Pixel will be a simple document containing the following fields:

- x: Number
- y: Number
- modified: Timestamp
- color: String (hexadecimal color value)

The Node.JS server will call the Firebase JS API to query and modify this data in response to user requests.

## Architecture Diagram


Backend
Frontend
(runs in client's browser)
Angular Application
View (HTML Template)
Two-Way
Data
Binding
ViewModel (TypeScript)
HTTP
HTTP
Server
(hosted on Firebase or Docker)
Node.JS Server
Express.JS HTTP API Endpoints
GET Request Handler
POST Request Handler
API Calls
Database
(service provided by Google)
Cloud Firestore DB
Firebase JS API
Point
x: Number
y: Number
modified: Timestamp
color: String