API Design

Firebase API provides Secure Login through Facebook/Google accounts:

- **Create an account**

    The first thing you need to do to get started with Firebase is sign up for a free account. A brand new Firebase app will automatically be created for you with its own unique database URL ending in firebaseio.com. We'll use this database URL to store and sync data.

- **Install Firebase**

    To include the Firebase client library in your website, add a script tag to the <head> section of your HTML file.

    ```
    <script src="https://cdn.firebase.com/js/client/2.4.2/firebase.js"></script>
    ```

- **Read & Write Data**

    To read and write data to and from a Firebase database, we need to first create a reference to it. We do this by passing the URL of our Firebase app into the Firebase constructor:

    ```
    var myFirebaseRef = new Firebase("https://<YOUR-FIREBASE-APP>.firebaseio.com/");
    ```

    Writing Data:

    Once we have a reference to our database, we can write any valid JSON object to it using set().

    Example:

    ```
    myFirebaseRef.set({
            title: "Hello World!",
            author: "Firebase",
            location: {
            city: "San Francisco",
            state: "California",
            zip: 94103
           }
    });
    ```

    Reading Data:

    Reading data from database is accomplished by attaching callbacks and handling the resulting events. Assuming we already wrote to myFirebaseRef above, we can retrieve the city value by using the on() method:

Example:

```
myFirebaseRef.child("location/city").on("value", function(snapshot) {
  alert(snapshot.val());  // Alerts "San Francisco"
});
```

In the example above, the value event will fire once for the initial state of the data, and then again every time the value of that data changes.

- **Deploy to Firebase Hosting:**

Firebase Hosting lets us deploy our application's static files (HTML, CSS, JavaScript, etc.) to the web with a single command. To get started, download firebase-tools via npm:

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

Firebase Hosting is a production-grade service, with security, reliability, and scalability baked-in.

- **Authenticate Users**

Firebase provides full support for authenticating users with Email & Password, Facebook, Twitter, GitHub, Google, or your existing authentication system.

To get started with Email & Password auth, enable the Email & Password provider in the App Dashboard:

1.Choose the Login & Auth tab.
2.Select the Email & Password tab and enable authentication.

Example: creating a user

```
myFirebaseRef.createUser({
  email : "bobtony@firebase.com",
  password : "correcthorsebatterystaple"
}, function(error, userData) {
  if (error) {
    console.log("Error creating user:", error);
  } else {
    console.log("Successfully created user account with uid:", userData.uid);
  }
});
```

Once you've created your first user, you can log them in using the authWithPassword method.

- **Secure Your Data:**

Powerful expression-based Security and Firebase Rules provide fine-grained control over who has access to the data and allow to validate writes to the database data:

Example:

```
{
 "rules": {
   ".read": true,
   ".write": "auth.uid === 'admin'",
   ".validate": "newData.isString() && newData.val().length < 500"
 }
}
```

Security and Firebase Rules are enforced consistently whenever data is accessed. The rules language is designed to be both powerful and flexible, so that to maintain fine-grained control over application's data.

Google Places API Web Service:

Get data from the same database used by Google Maps and Google+ Local. Places features more than 100 million businesses and points of interest that are updated frequently through owner-verified listings and user-moderated contributions.

- **Place Search:**

  The Google Places API Web Service allows you to query for place information on a variety of categories, such as: establishments, prominent points of interest, geographic locations, and more. You can search for places either by proximity or a text string. A Place Search returns a list of places along with summary information about each place.

  Nearby Search Requests:

  A Nearby Search lets you search for places within a specified area. You can refine your search request by supplying keywords or specifying the type of place you are searching for.

  A Nearby Search request is an HTTP URL of the following form:

  https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters

  where output may be either of the following values:

  •json (recommended) indicates output in JavaScript Object Notation (JSON)
  •xml indicates output as XML

  Required parameters:

- key — Your application's API key. This key identifies your application for purposes of quota management and so that places added from your application are made immediately available to your app. See Get a key for more information.

- location — The latitude/longitude around which to retrieve place information. This must be specified as latitude,longitude.

- radius — Defines the distance (in meters) within which to return place results. The maximum allowed radius is 50 000 meters. Note that radius must not be included if rankby=distance is specified.
- If rankby=distance is specified, then one or more of keyword, name, or type is required.
- **Place Details:**

Place Details Requests:

A Place Details request is an HTTP URL of the following form:

https://maps.googleapis.com/maps/api/place/details/output?parameters

where output may be either of the following values:

•json (recommended) indicates output in JavaScript Object Notation (JSON)

•xml indicates output as XML

Certain parameters are required to initiate a search request. As is standard in URLs, all parameters are separated using the ampersand (&) character. Below is a list of the parameters and their possible values.

•key (required) — Your application's API key. This key identifies your application for purposes of quota management and so that places added from your application are made immediately available to your app. See Get a key for more information.

• Either placeid or reference (you must supply one of these, but not both):

•placeid — A textual identifier that uniquely identifies a place, returned from a Place Search. For more information about place IDs, see the place ID overview.

•reference — A textual identifier that uniquely identifies a place, returned from a Place Search. Note: The reference is now deprecated in favor of placeid. See the deprecation notice on this page.

Optional Parameters :

•extensions (optional) — Indicates if the Place Details response should include additional fields. Additional fields may include premium data, requiring an additional license, or values that are not commonly requested. Extensions are currently experimental. Supported values for the extensions parameter are: •review_summary includes a rich and concise review curated by Google's editorial staff.

•language (optional) — The language code, indicating in which language the results should be returned, if possible. Note that some fields may not be available in the requested language. See the list of supported languages and their codes. Note that we often update supported languages so this list may not be exhaustive.

The following example requests the details of a place by placeid:

https://maps.googleapis.com/maps/api/place/details/json?placeid=ChIJN1t_tDeuEmsRUsoyG83frY4&key=YOUR_API_KEY

- **Place Detail Results:**

  When the Places service returns results from a details request, it places them within a single result. Each result may contain the following fields:
  •address_components[] is an array of separate address components used to compose a given address. For example, the address "111 8th Avenue, New York, NY" contains separate address components for "111" (the street number, "8th Avenue" (the route), "New York" (the city) and "NY" (the US state). Each address_component typically contains:

  •types[] is an array indicating the type of the address component.

  •long_name is the full text description or name of the address component.

  •short_name is an abbreviated textual name for the address component, if available. For example, an address component for the state of Alaska may have a long_name of "Alaska" and a short_name of "AK" using the 2-letter postal abbreviation.

  •formatted_address is a string containing the human-readable address of this place. Often this address is equivalent to the "postal address," which sometimes differs from country to country. This address is generally composed of one or more address_component fields.

  •formatted_phone_number contains the place's phone number in its local format. For example,
  the formatted_phone_number for Google's Sydney, Australia office is (02) 9374 4000.

  •geometry contains the following information:

  •location contains the geocoded latitude,longitude value for this place.

  •viewport contains the preferred viewport when displaying this place on a map as a LatLngBounds if it is known.

  •icon contains the URL of a suggested icon which may be displayed to the user when indicating this result on a map.

  •id contains a unique stable identifier denoting this place. This identifier may not be used to retrieve information about this place, but can be used to consolidate data about this place, and to verify the identity of a place across separate searches. As IDs can occasionally change, it's recommended that the stored ID for a place be compared with the ID returned in later Details requests for the same place, and updated if necessary. Note: The id is now deprecated in favor of place_id. See the deprecation notice on this page.

  •international_phone_number contains the place's phone number in international format. International format includes the country code, and is prefixed with the plus (+) sign. For example, the international_phone_number for Google's Sydney, Australia office is +61 2 9374 4000.

  •name contains the human-readable name for the returned result. For establishment results, this is usually the canonicalized business name.

  •opening_hours contains the following information:

•open_now is a boolean value indicating if the place is open at the current time.

•periods[] is an array of opening periods covering seven days, starting from Sunday, in chronological order.
Each period contains:
•open contains a pair of day and time objects describing when the place opens: •day a number from 0–6, corresponding to the days of the week, starting on Sunday.
For example, 2 means Tuesday.

•time may contain a time of day in 24-hour hhmm format. Values are in the range 0000–2359. The time will be reported in the place’s time zone.

•close may contain a pair of day and time objects describing when the place closes. Note: If a place is always open, the close section will be missing from the response. Clients can rely on always-open being represented as an open period containing day with value 0 and time with value 0000, and no close.

•weekday_text is an array of seven strings representing the formatted opening hours for each day of the week. If a language parameter was specified in the Place Details request, the Places Service will format and localize the opening hours appropriately for that language. The ordering of the elements in this array depends on the language parameter. Some languages start the week on Monday while others start on Sunday.

•permanently_closed is a boolean flag indicating whether the place has permanently shut down (value true). If the place is not permanently closed, the flag is absent from the response.

•photos[] — an array of photo objects, each containing a reference to an image. A Place Details request may return up to ten photos. More information about place photos and how you can use the images in your application can be found in the Place Photos documentation.

A photo object is described as:
•photo_reference — a string used to identify the photo when you perform a Photo request.
•height — the maximum height of the image.
•width — the maximum width of the image.
•html_attributions[] — contains any required attributions. This field will always be present, but may be empty.

•place_id: A textual identifier that uniquely identifies a place. To retrieve information about the place, pass this identifier in the placeId field of a Places API request. For more information about place IDs, see the place ID overview.

•scope: Indicates the scope of the place_id. The possible values are: •APP: The place ID is recognised by your application only. This is because your application added the place, and the place has not yet passed the moderation process.

•GOOGLE: The place ID is available to other applications and on Google Maps.

•alt_ids — An array of zero, one or more alternative place IDs for the place, with a scope related to each alternative ID. Note: This array may be empty or not present.
If present, it contains the following fields:

•place_id — The most likely reason for a place to have an alternative place ID is if your application adds a place and receives an application-scoped place ID, then later receives a Google-scoped place ID after passing the moderation process.
•scope — The scope of an alternative place ID will always be APP, indicating that the alternative place ID is recognised by your application only.