Security Concept - ViziWealth Finance Management

This document describes the architectural decisions, data model, technologies and standards used for the application development in the scope and for the main objective to illustrate the information security concepts as they are implemented. Here is a brief summary of the content of the document:

- Deployment architecture and separation of concerns (clients are without privileges, access control close to data, no external access to permission rules policy file)
- What backend services are used (certified Google Firebase with well known privacy policy and certifications).
- Description of data encryption model (TLS in transit, AES 256 at rest)
- Description of data access control
- Certifications of the backend services in use

General deployment architecture

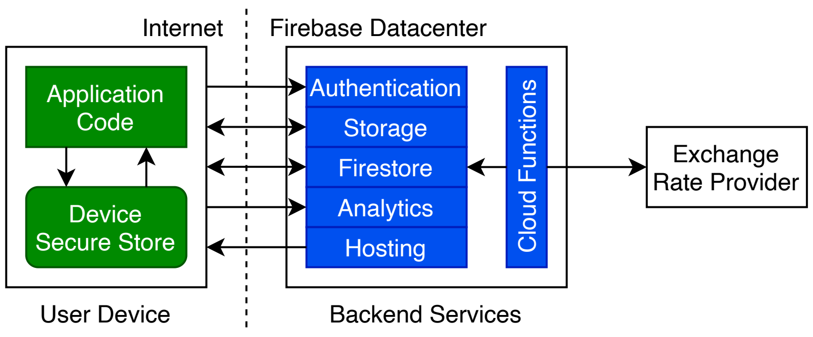
The application is separated in *client code*, *backend services* and *backend code* (in the form of cloud functions). General paradigm excludes management of virtual machines in the cloud. Most of the logic is implemented within BaaS (backend as a service) with very few exceptions implemented as cloud functions.

The client code has 3 variants:

- browser based application with 2 subvariants
  - JavaScript + html5 implementation for mobile device browsers
  - JavaScript + CanvasKit/Web Components implementation for desktop browsers
- native android application
- native iOS application.

The client code persists user context information (access token) in the secure storages of the respective devices (including browsers' secure storages). Graphical images will also be temporarily cached on the device.

Client code accesses directly the backend as a service via secure connection based on HTTPS. Protocol upgrades are possible on top of the TLS connection in order to improve performance. Such protocol upgrades are part of the BaaS specific functionality.



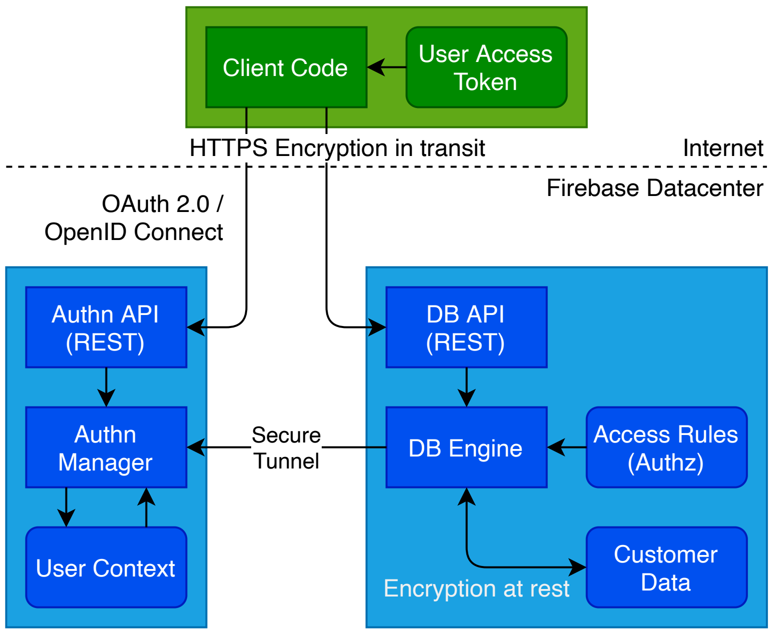
Client code identifies in front of BaaS using unique public API key. This API key allows access to a number of services. The BaaS will keep record the source IP for a period in order to prevent abuse, including DoS (denial of service) attacks and implementation of regulatory compliance for prevetion of access from embargoed countries.

Service usage

ViziWelath uses the following services:

- **Firebase Hosting.** Used for domain and certificate management and for downloading web application code.
- **Firebase Authentication.** It provides user management and user/pass based authentication for the application.
- **Firabese Storage.** File storage used for storing and protecting graphical images uploaded by end users.
- **Firebase Firestore.** A hierarchical database with finegrained access control used for storing all the data of the user.
- **Firebase Cloud Functions.** Used for updating (once a day) the exchange rates of currencies used in the application.
- **Firebase Analytics.** Used (with user consent) to store usage data, e.g. how much time users spend on different screens. Data is anonymised.

Once accessed, the BaaS API's will inspect the user token passed with the calls and check for authentication validity. Certain services imply rules based on the user identity, such as finegrained contol of access to data. This finegrained control is based on rules that are stored in the backend and can be modified only by administrators of the BaaS account (i.e. ViziWealth administrators). These rules are applied on the server side and cannot be circumvented from outside.



Authentication is based on OAuth 2.0 and OpenID Connect. Password hashes are stored instead of user passwords. This is a standard addition protection. In case of forgoten password, there is an implementation of password reset flow, coming from Firebase Authentication service.

Data encryption in transit

Communication between the application code and backend services is based on HTTPS/TLS 1.3. Http protocol upgrades are done where applicable, in order to improve performance.

Data encryption at rest

All the data in Firestore database is encrypted on the disk and only decrypted in the bounds of the Firestore database engine when data is being accessed by an authorized user. 256 bit AES encryption is used.

More information can be found here: https://cloud.google.com/firestore/docs/server-side-encryption

Data access control

Firebase authentication service is tightly integrated with other Firebase services, which allows secure operation in the backend. When an end user tries to access certain collection or object in the Firestore database, there are two checks performed on the backend:

- Whether the user is authenticated
- Whether the user is authorized to access the collection or the object, based on the defined rules.

The data model is hierarchical which allows to control the access based on elements of the hierarchy - collections and objects. Access permissions are not transitive, i.e. it is possible to configure rules such that a user can access objects in a collection, but not in its subcollections. Here is some high level information about major collections and access control for them

- Every user has a collection called *'space'* in the database and is the *'owner'* of that space.
- There are subcollections for different types of objects (assets, liabilities, tasks...).
- There is a subcollection *'clients'* where every authenticated user can add or delete their email, but cannot access other emails.
    - The *'owner'* of the *'space'* can list all emails in this collection, but cannot add other emails.
- There is a subcollection *'advisor'* which can contain shared (read/write) subcollections with advisor.
    - Advisors are other users who have the *'owner'* email added in their *'client'* subcollection.
- There is a *'public'* subcollection which can be access by every authenticated user.
    - This subcollection contains a link to the owner's avatar (profile image) if such is uploaded in Firebase storage.

Sharing

Objects and collections can be shared with other users. This is decided by the *'owner'*. The user which is granted a permission to access a collection or an object is called *'grantee'*.

Owners

- There is a subcollection called *'sharings'* where objects can be read and written by *'owners'*.
    - These objects contain coordinates of shared collection/object coupled with *'grantee'* email.
    - *'Grantees'* cannot access the '*sharings*' collection.

Grantees

- There is a *'permissions'* collection on the root level of the database.
    - It contains subcollections owned by the *'grantees'*.
    - In these subcollections, there are subcollection for collection/object *'owner'* has shared.

    - The *'grantee'* has read access and the *'owner'* has write access.
  - This allows the application to request adding object both in *'spaces/owner/sharings'* and *'permissions/grantee/owners/owner'* collection.
  - The objects store contain data used by the database backed rules definition to allow *'grantees'* to access the shared collection/object.
- We have two collections (*'sharings'* and *'permissions'*) in order to improve performance for both *'owners'* and *'grantees'*.

Certifications

Firebase services are certified based on number of standards: ISO 27001, ISO 27017, ISO 27018, SOC 1, SOC 2, SOC 3.

More infromation can be found here: https://firebase.google.com/support/privacy