+Software Design Specification

Twoefay

A Multi-factor Authentication System for Aerospace Corporation

Chris Orcutt: 204 305 633
Jonathan Woong: 804 205 763
Rossen Chemelekov: 504 216 616
Mengfei Wu: 104 152 503
Vu Le: 004 497 690
Anthony Nguyen: 703 877 763

University of California, Los Angeles
CS130: Software Engineering, Spring 2016
Professor Paul Eggert

Table of Contents

1 Introduction

This document is intended to serve as a reference to developers and researchers interested in the design and implementation of multi-factor authentication services for websites and web applications. It provides detailed high- and low-level descriptions of the architecture and interfaces involved in the creation of such systems.

The intended audience of this document is individuals with technical background, with understanding of networking concepts and applications, architecture diagrams, interface design, and event-driven programming experience.

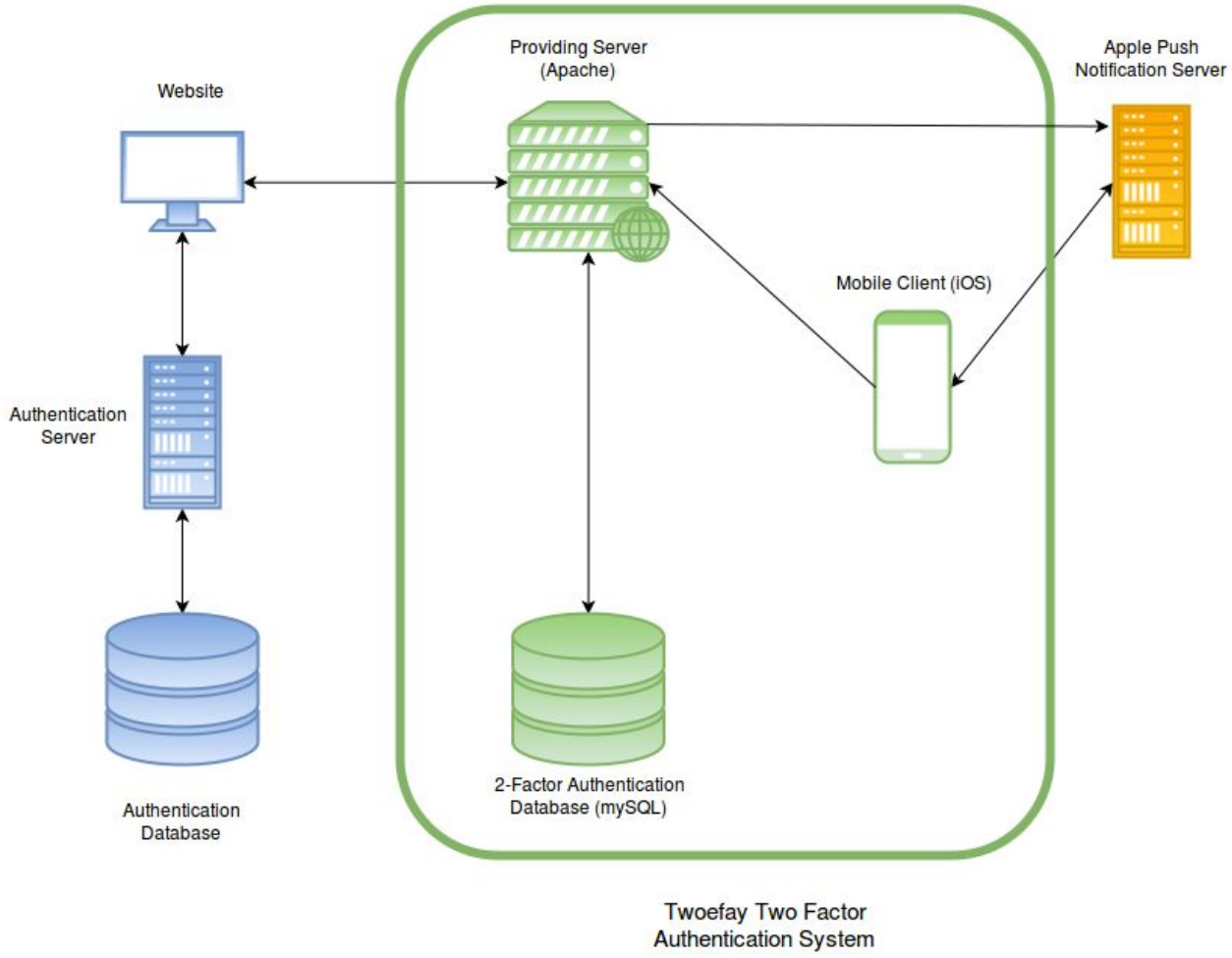
This document has an accompanying software requirements specification document (version 2) that provides insight into the requirements for the authentication system design.

The *Acronyms and Abbreviations* section contains various shorthand notation used throughout this document to describe components and functionality of the system.

This design document covers the system overview, system components, the system architecture and design strategies, as well as policies and tactics.

2 System Overview

The Twoefay multi-factor authentication system acts as an abstract layer between a website and Apple's infrastructure for push notification and fingerprint authentication (TouchID). The Twoefay system itself contains three subsystems: the application server that offers the authentication service to the website, the database that stores username to token key-value pairs, and the iOS application that is used for fingerprint authentication.



Notice: The website (blue) and Apple server (yellow) pictured above are not included in the Twoefay system and have only been included in the system diagram for completeness.

When integrated into a production system, the Twoefay system has two primary functions: initialization and authentication. The initialization stage occurs when a new user is signing up for the website. During initialization, the primary goal is to add a username to token key-value pair into the database. The authentication stage occurs when an existing user is trying to sign into the website. During authentication, the Twoefay system communicates in the background with the APN servers and provides access to the TouchID system on the user's iOS device.

3 Design Considerations

The following section outlines all design consideration for the Twoefay multi-factor authentication system. It outlines the assumptions, dependencies, and constraints of each of our subsystems and discusses the goals, guidelines, and development methods of the project.

3.1 Assumptions & Dependencies

The Twoefay server subsystem will operate under the following assumptions:

1. The hardware system has networking capabilities and bandwidth sufficient to process requests in a timely manner.
2. The system can meet the resource demands of the web server application, such as sufficient memory size and capable CPU.
3. The server will have access to a mySQL database capable of storing list of username to APN token key-value pairs.
4. The server will communicate with APN servers over TLSv1.2 using HTTP/2.

The Twoefay iOS application subsystem will operate under the following assumptions:

1. The iOS application will run on an iPhone 5S or later, as all earlier devices do not feature the fingerprint recognition technology.
2. The user of the iOS application will have TouchID enabled.

3.2 General Constraints

Communication between the server and the APN must occur over HTTP/2.0 using TLS 1.2+ and JSON-formatted data to meet Apple's requirements. Since HTTP/2.0 is backwards-compatible with HTTP/1.1, the 2FA service uses HTTP/2.0 in all its communications with external components, such as the client web site, the APN, and the user's iPhone device. In order to meet these requirements, the server's host operating system must have Python 2.7.9 or later, as in earlier versions of Python the SSL library does not support the NPN extension required to establish a valid TLS connection with the APN.

The user must be in possession of the registered iOS device while using the Twoefay multi-factor authentication system.

3.3 Goals and Guidelines

The 2FA system has two primary goals: security and ease of use. As these goals are often mutually-exclusive, it is pertinent that the right balance between them is achieved so that the end-user does not have to sacrifice one for the sake of the other. To that end, the 2FA service aims to be as invisible to the end-user as possible, providing the minimum required contact to facilitate authentication in the form of a pop-up on the user's phone giving the user two simple options: provide authentication or reject the authentication attempt.

Additionally, performance is also very important in order to provide a usable service. The time between a website login attempt on the user's part and the iPhone notification popup should be minimal. Similarly, the time between the user authenticating themselves on their iPhone and consequently being allowed access into their website of choice should also be minimal.
The use of HTTP/2.0 in system communications aids in decreasing latency due to the protocol's design – headers are compressed using HPACK, data is transmitted in binary rather than cleartext, and, once established, the TCP connection remains open to future communication rather than tearing down and building up multiple new connections.

3.4 Development Methods

Each component of the 2FA system is being developed independently (at any given time) using a set of agreed-upon protocols and restrictions (HTTP/2.0, MySQL, Python). The platform for collaboration, changes, and component merging is GitHub. Components that are partially completed and pushed onto GitHub are improved upon by others. All levels of the system are being developed at the same time, as opposed to developing one layer at a time. This method of development is preferred because of the inability for developers to regularly meet (conflicting schedules, unavailability, etc.). The development flow does allow for multiple developers to work on a component at the same time, though this should not happen unless the two developers have a steady communication stream during the process (in person or through conference calling).

4 Architectural Strategies

The two sections below, Server and iOS Application, describe in detail the architectural strategies employed in the component design of the system.

4.1 Server

The server component is composed of two subcomponents: a Python server application and a MySQL database. The system is designed on a Linux host but can perform equivalently on a Windows host with minimal modification. The 2FA server can accept and process multiple user requests in an asynchronous fashion by using deferred calls. It is designed to be invisible to the user and does not require direct interaction to facilitate access. The server's main functions include:

- Accepting and processing POST requests with the user's username in a JSON string from the protected website.
- Contacting the local MySQL database with the username to obtain the user's token.
- Issuing a POST request with that token in a JSON to the APN service which generates a notification on the user's iPhone.
- Then, once the user has authenticated themselves on their phone, the server accepts a POST request from the iPhone to update the status of the user.
- And, finally, the server sends a POST request to the website to allow login access.

In addition to the main function, the server includes backup methods that do not depend on Apple so that users are still able to authenticate if communication with the APN service is not available. In high security environments, customers who chose to use our service will need to decide which if any of these backup methods they will allow their users to use.

In order of security,

[Most Secure]
*** Push Notification with Fingerprint *** Primary Method
TOTP Secret Keys
SMS Message
Email Message
[Least Secure]

Backup Method 1:

- During the initial setup for the user, the user provides our server with their phone number OR email address OR both
- During a login attempt, the server sends a 6-digit verification code to the user by either text-message OR by email
- The user receives their 6-digit code and they type it in on our website
- The Server-verifies the typed-in code matches the code that the server sent out
- The Server allows login access if the codes match. If the codes do not match, the Server cancels the login

Backup Method 2:

- During the initial setup for the user, our Server generates a secret key for the user
    - The user stores the key on our iOS app or another standard compatible authenticator app that uses the official Time-based One-time Password Algorithm (TOTP) RFC 6238 protocol.
    - This can be using either QR code or manual entry of the secret key
- During a login attempt, the user opens their app and views the current 6-digit code for the current time, and they type the code in on our website
- The server verifies the typed-in code matches the code that the server derives from the secret key given the current time-stamp
- The server sends a POST request to the website to allow login access if the codes match. If the codes do not match the server can either cancel the login entirely or send another code

4.2 iOS Application

The iPhone application is main method of user interaction in the authentication process. It is a thin client, storing and processing minimal user information, that serves to provide a simple and

intuitive user interface with the options of accepting or rejecting authentication attempts to a protected website. It has the following functions:

- Accepts push notifications from the APN service.
- Prompts the user to click the Accept button to authorize the login attempt for the listed website, whereby the user must authenticate using the iPhone's TouchID fingerprint scanner.
- The iPhone application then uses a POST request to update the server with the success or failure of the login attempt.
- Alternatively, the user is also given the option to click the Reject button to cancel the login attempt, whereby the application notifies the server of the rejected login attempt.

5 System Architecture

Our system architecture can be seen in green in Figure 1 on page 2. In this section, we will go into more detail about:

(1) The Apache Server
(2) The 2-Factor Authentication Database
(3) The iOS Mobile App

We also include workflows at the end to show how the pieces are used together.

5.1 Apache Server

The Twoefay server has two primary functions: networking and database management. Regarding networking, the server must listen for incoming connection requests on port 8080. By default, HTTP/2.0 uses TLS to encrypt transmitted data, so requests to the server go over HTTPS both from the webserver and iOS client. The server must also be able to send requests to the APN servers. The Twoefay system enables this functionality using a Twisted Hyper Python server to handle new connection requests, and the Hyper HTTP/2 library to send requests to the APN servers. Regarding database management, the server must be able to lookup the APN token given a username or token. The Twoefay system enables this functionality via a mySQL database.

The Twoefay system was decomposed into the minimally-necessary number of components required to accomplish authentication -- the Client as a pre-existing service, the APN as an independent service required for notification to the App, the App as the user's authentication tool, and the Server to facilitate communication between all of the above. The system is glued together by communicating over HTTP/2.0 in transmitting user data. Despite transmitting fully-encrypted data over the wire, as an extra precaution, aside from the initial registration POST request from the Client to the Server, all communication between the system components uses a randomly-generated token to identify each user.

5.2 2-Factor Authentication Database

The database will store username to token key-value pairs. The username is a unique identifier selected by the user during the signup process. The token is a unique device identifier provided by Apple that is used to request push notification to a specific device.

As a POST (or GET) request is received and processed, the functions described previously read in the JSON data transmitted with the request and, depending on the required output, call the functions listed below to obtain user data that is to be returned by the Server to the Client or user. The Server perform database access with the following methods:

Method names	Descriptions and functions
db_open()	Triggered prior to database operations, connects to the database, returns a handle "db" and a "cursor" required for database queries.
db_close()	Triggered after each database operation, closes the database.
db_validate()	Triggered by /register POST (or just a GET) request from Client to verify username exists/doesn't exist in the database.
db_validate_token()	Triggered upon each POST request to verify that the received token corresponds to a registered user.
db_get()	Triggered by a GET request from Client, pulls the token for a specified username from the database.
db_set()	Triggered by a /register POST request from Client, it adds the user's username, email, phone, and randomly-generated id_token to the database.
db_update()	Triggered by a /verify POST request from the App, updates specified user's dev_token in the database.
db_get_user_email()	Triggered by a /login request for a user who hasn't been verified by the App, uses id_token to get user's email from database.
db_get_user_phone()	Triggered by a /login request for a user who hasn't been verified by the App, uses id_token to get user's phone number from database.
db_get_user_dev_token()	Triggered by /login for a verified user, uses id_token to get user's dev_token, which is needed for APN calls.

The MySQL database stores username, email, phone, id_token, and dev_token with the following schema:
CREATE TABLE Users (

```
    username VARCHAR(40) UNIQUE NOT NULL PRIMARY KEY,
    email VARCHAR(40) UNIQUE NOT NULL,
    phone VARCHAR(15) UNIQUE NOT NULL,
    id_token VARCHAR(80) PRIMARY KEY,
    dev_token VARCHAR(80)
);
```

username	email	phone	id_token	dev_token
<user's username>	<user's email>	<user's phone #>	<random alphanumeric string>	<user's iPhone device token>

5.3 iOS Application

The iOS application consists of two primary functions: signup and authentication. The signup screen will allow the users to input a username and will request access to send push notifications. After signing up, the user will only interface with the application when authentication is requested. The authentication screen will allow the user to interface with the TouchID system and provide visual feedback indicating the status of the fingerprint scan.

5.4 Major Workflows

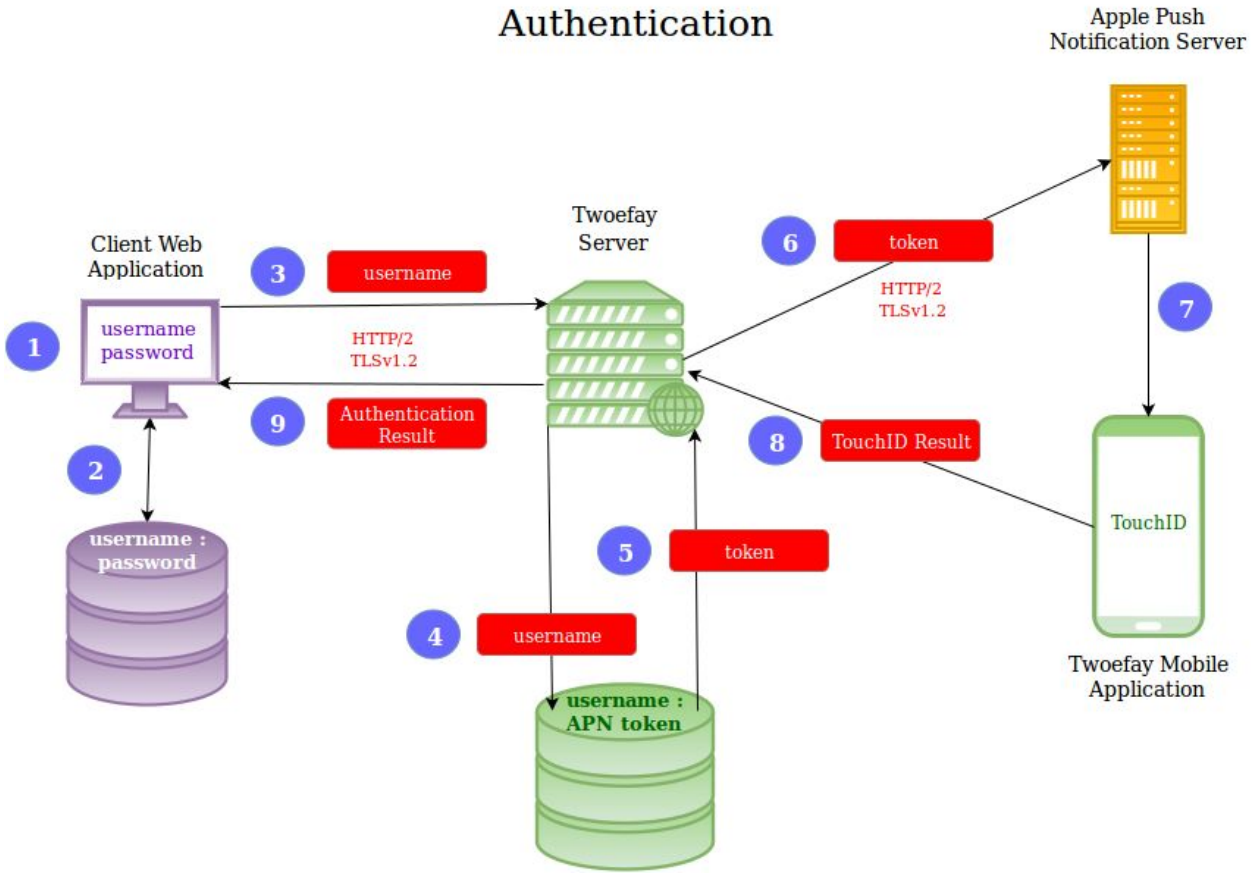
5.4.1 Initialization

When a new user signs up for a website, two things must occur to ensure proper initialization: the website must forward the username to the Twoefay server and the user must register in the iOS application with the same username used in the website. The following diagram outlines all steps necessary to properly initialize the system:

1. User signs up for website using a username and password
2. Website saves username and password in database maintained by website developer
3. Website informs Twoefay server of new username
4. Twoefay server adds username into database maintained by Twoefay
5. User signs into iOS application using the same username
6. iOS application requests APN token from APN server
7. APN server returns APN token to iOS application
8. iOS application sends username and corresponding APN token back to Twoefay server
9. Twoefay server adds APN token to corresponding username in database

5.4.2 Authentication

When an authentication event is triggered (i.e. the user tries to login to the website), the following chain of events are triggered.



1. User signs into website using a username and password
2. Website authenticates user via username and password
3. Website requests authentication from Twoefay server using user's username
4. Twoefay server indexes the database using the username
5. Corresponding token is returned
6. Twoefay server requests push notification from APN server via token

7. APN server pushes notification to users device
8. User interfaces with TouchID and result is sent to Twoefay server
9. Twoefay server responds to webserver with authentication results

6 Policies and Tactics

6.1 Apple Communication Policy

The outgoing messages from the Python server to the Apple server must satisfy three conditions: be using HTTP/2.0 protocol, SSLv1.2 certification, and HPACK header compression. Testing whether these criteria are met is done by sending a correctly formatted message to the Apple server and receiving a message back.

6.2 Testing

To test our Twoefay system, we create a sample web application that simulates how a Client's website would integrate with our system. This sample website is built using Python Flask because of the developers' previous familiarity with Python and the ease with which we can make HTTP requests. The sample Client website includes a signup form on which a Customer can create a new account. The website then prompts the user to download the mobile iOS application. In future login attempts, the Customer will initially log in using the original login form but is then redirected to the "Authenticating" page which will also include the backup code input method as an alternative in case the APN services are offline. In this way, we test Twoefay's ease of integration into existing Client websites using a simple barebones sample website.

In order to ensure secure data transfer, simulation of Man-In-The-Middle attacks can test the reliability and security of the system. Data transfers between the Python server, Apple server, and mobile device should be encrypted, and MITM can check for successful encryption. We observe the security flaws of our sample website and take note of the security upgrades with Twoefay integration.

The Twoefay server will be be maintained with a monthly low cost of $5 per month on Digital Ocean. Once the iPhone application is launched to the App Store, there may be low cost maintenance needed to update the server and application if bugs arise or technology changes. Since the code is open source, we expect that the community will customise and adapt the code as well as initialise tickets for any future issues.

7 Detailed System Design

7.1 Communication Interfaces

7.1.1 Apple

The APN servers require the following for all network communication:

1. HTTP/2 protocol
2. TLSv1.2 or greater
3. Two-way certificate exchange during TLS handshake

Regarding the two-way certificate exchange, the Twoefay server must obtain a certificate signed by the Apple, Inc Certificate Authority. Additionally, the APN servers requires JSON encoded data streams, where the data has a strict format. The Apple documentation on the specifics of the format can be found here: https://developer.apple.com/library/ios/documentation/ NetworkingInternet/ Conceptual/ RemoteNotificationsPG/Chapters/ApplePushService.html

7.1.2 Twoefay Server

The Twoefay system will require the following for all network communication:

1. HTTP/2 protocol
2. TLSv1.2 or greater

When transmitting tokens or usernames to the Twoefay system, all clients should use the following custom header fields:

1. token
2. username

7.2 Server

Under the hood, the main functions used in implementing the Server functionality are as follows:

- **validate_params():** Reads in the request parameters and ensure the it complies with the expected format, such as:
    - type of method - only GET and POST are allowed
    - number of parameters - GET can have exactly 2, while POST exactly 1
    - type of parameters
        - GET must have */user/<username>*
        - POST must be either */register*, */verify*, */login*, */success*, or */failure*
    - requests that do not comply raise an error flag that is used to reject the request and return a 404 status.
    - JSON contents - each request must be accompanied by a JSON data string containing specific elements:
        - **/register** JSON: {'username':'<username>','email':'<email>','phone':'<phone #>'}
        - **/verify** JSON: {'id_token':'<id_token>','dev_token':'<dev_token>'}
        - **/login**, **/success**, and **/failure** JSON: {'token':'<token>'}
- **handle_GET():** Reads in the GET request, verifies the username requested exists in the database and returns the token for the specified user.

- **handle_POST():** Reads in the POST request and processes it, depending on its type, by calling the appropriate functions named after each request type:
    - **/register** will attempt to register a user, if it's not already in the database,
    - **/verify** will store the provided dev_token in the database that will be used to contact the APN,
    - **/login** will attempt to contact the APN to request user authentication via TouchID
    - **/success** will return a request to the Client to permit user access to the website,
    - **/failure** will return a request to the Client to prevent user access to the website.
- **send_POST():** This is the main way the Server interacts with the other system components, it will pack up the required return data in a JSON and POST it to the Client or APN.

The Server's fallback for unverified users (i.e. users who have not yet registered on their iPhone App) is to send a One-Time-Pass (OTP) code via email or text message to the user's phone. This is done via the following methods:

- **get_email():** Uses the user's id_token to obtain the user's email from the database.
- **send_email():** Sends the OTP to the user's email which will allow the user to then access the website.
- **get_phone():** Uses the user's id_token to obtain the user's phone number from the database.
- **send_sms():** Sends the OTP to the user's phone via an SMS, which the user can then use to access the website.

The following utility functions aid in providing security and backup methods of authenticating the user:

- **gen_token():** This function is called upon a /register POST request to generate a random alphanumeric string to be used as the user's id_token in communications with other system components.
- **generate_TOTP_code():** This function is called for unverified user, uses the PyOTP library to generate a time-sensitive 6-digit code that is sent to the user's email, phone, and is utilized as backup in user authentication as Method 2. The function creates an OTP object that it uses to obtain the one-time-pass code from the OTP module.
- **generate_HOTP_code():** This function is called for unverified user, uses the PyOTP library to generate a counter-based 6-digit code that is sent to the user's email, phone, and utilized in user authentication as Method 1. The function uses a counter that is incremented with each use to ensure a unique OTP code is generated and passes its value to the OTP module handling the HOTP code generation.

7.2.3 API

As the server must have a mechanism for handling incoming requests, querying the database, and sending push notification to the APN servers, the Twoefay server API has been broken down into three distinct partitions: network incoming, network outgoing, and querying.

The *network incoming* API will contain the following public methods:

> ***void listen(integer port, function callback)***
> The *listen* method listens for incoming connections from a specific port number and passes them to the callback method.
>
> ***request_type parse_request(string request)***
> The *parse_request* method takes as input a request from the network, parses the request, and returns the type of the request.
> This request type returned from this method will be passed in as the parameter to the handle_request method.
>
> ***void handle_request(request_type type)***
> The *handle_request* method takes as input a request type, takes the appropriate action depending on the request type, and returns nothing.
> For requests to authenticate the user using the iPhone app, the request will be handled by retrieving necessary information such as user token and then calls the push_notification method.

The *network outgoing* API will contain the following public methods:

> **bool push_notification(byte_string token)**
> The *push_notification* method will take as input an APN token, sends a push notification request to the APN servers, and returns an indication of whether the request was successful.
> This method will wait for user authentication on the iPhone application to complete, or to return a failure.

The *query* API will contain the following public methods:

> **byte_string retrieve_token(string username)**
> The *retrieve_token* method takes as input a username string, queries the database for the corresponding token, and returns the token if found.
> The token returned from this method will be passed in as the parameter for the push_notification method.

7.3 iOS Application Components

The iOS application performs the following tasks:

1. Accept incoming push notifications from the server
    a. Afterwards, the app notifies the server of a successful or failed authentication from the user
2. Regularly send updates to our Server with any changes to the iPhone's dev_token
3. Continually generate an OTP based on a secret key which is accepted via QR code scanning or manual string entry.

7.3.1 3rd Party Libraries

We make good use of several 3rd party libraries to speed up development of our iOS app.

1. HTTP communications are handled using the 3rd party library Alamofire which we discuss in more detail in section **7.4**
2. OTP generation is handled using the OneTimePassword library
3. QR Code scanning is handled by the QRCodeReader library

7.3.2 OTP Handling

OTP core functions are maintained in the OTP class

OTP:

class func storeToken(name: String, issuer: String, secretString: String) -> NSData?

- Name: Typically username or email address
- Issuer: Company who generated the secret key
- secretString: Secret Key used to derive 6-digit passwords on a recurring basis
- Return value: NSData with the identifier for the token which was stored

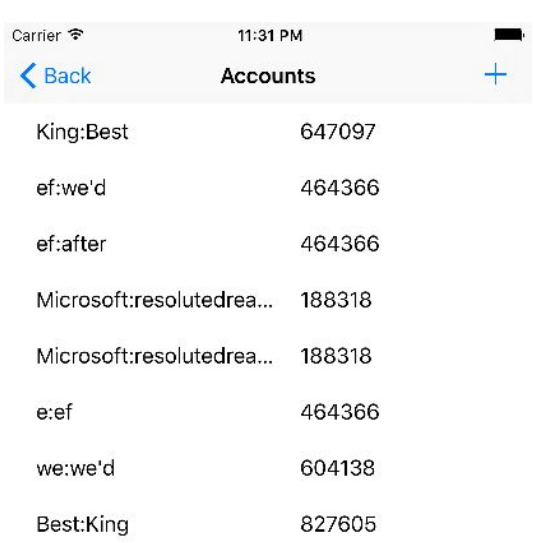
class func getAllTokens() -> Set<PersistentToken>?

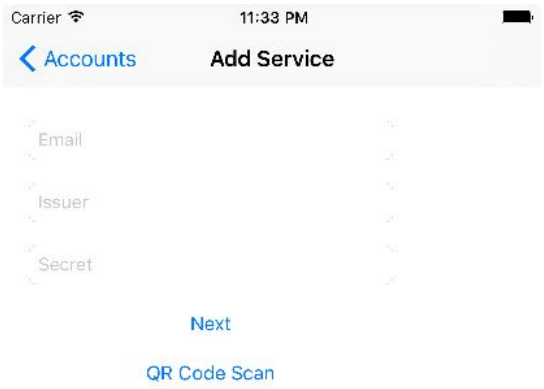
- Return value: A Set of Persistent Token objects, which are objects that will provide a 6-digit key based on the current time upon request

class func getPassword(identifier: NSData) -> String?

- Identifier: A token identifier in the format provided by the storeToken function
- Return value: A 6-digit code

These functions are used on the two View Controllers that govern storing and retreiving 6-digit codes. The function **getAllTokens()** is called every time the Accounts page is loaded to get all the Token objects for every account the user has stored in the app. Every 30 seconds, **getPassword()** is called and all the cells are updated in order to keep the 6-digit codes in sync. The function **storeToken()** is called on the Add Service page after the user enters the Email, Issuer, and Secret into the text box OR scans an appropriate QR code.





- **func submitSecret(name, issuer, secret):** Allows for 6-digit OTP token generation via input strings instead of QR code.
    - Name: Typically a username or email
    - Issuer: The site which provides the 2FA service
    - Secret: A unique string provided by the site
- **func scan():** Activates the camera and scans the QR code
- **func read():** Parses the data read from the QR code (name, issuer, secret) and calls QRhandler to create an OTP token with the read data as input
- **func QRhandler():** Converts the username, issuer, and secret into a 6-digit OTP token

7.4 iOS Application Communications

The iOS application has essentially two types of communication: Incoming Communication and Outgoing Communication.

7.4.1 Incoming Communications

Apple Push Notifications to the iOS application are controlled fully by Apple, and implementation details of this process are out of the scope for this system. Incoming notifications contain:

1. The name of the app that is requesting the authentication
2. The IP address the login request is coming from
3. The location of the IP address

This way, if an unexpected push notification comes in from a foreign country, the push notification will notify the user of the login attempt immediately.

7.4.2 Outgoing Communications

Our iOS application will communicate with our Server via the Alamofire library. The methods that send POST requests to the server are implemented using Alamofire's request method:

Alamofire.request(HTTPmethod, path, parameters): Makes a GET/POST request to a domain

- HTTP Method: GET or POST
- Path: A domain (server) URL ending in a requested action
    - /verify: If provided an id_token (username or email), the server updates the corresponding dev_token (device token)
    - /success: Notifies the server of successful authentication for a particular id_token
    - /failure: Notifies the server of failed authentication for a particular id_token
- Parameters:
    - **/verify**: ["id_token": "<id_token>", "dev_token": "<dev_token>"]
    - **/success**, and **/failure**: ["token": "<token>"]

8 Acronyms and Abbreviations

1. *2FA***:** Two-factor authentication
2. *APN*: Apple Push Notification
3. *APNS*: Apple Push Notification Service
4. *Server*: Twoefay server running the multi-factor authentication system
5. *Client*: The customer who wants to use our service to provide two-factor authentication for their website or service
6. *User(s)*: The individuals who are using the website or service provided by the Client
7. *iOS App*: Twoefay iOS application
8. *Website*: Website providing multi-factor authentication via the Twoefay system
9. *TouchID*: The fingerprint recognition system provided by the iPhone.
10. *OTP*: One-time-password

The End