**ACCOUNT SECURITY SERVICE**

**Reference Implementation**

**Design Document**

Version 0.1

Mind2Mobile

25 June 2019

|  |  |  |  |
| --- | --- | --- | --- |
| Document Revision History | | | |
| Revision | Date | Description of Change | Originator |
| 0.1 | 06/25/19 | Created | Charles Stack |
| 0.2 |  |  |  |
| 0.3 |  |  |  |

|  |  |  |
| --- | --- | --- |
| Approvals | | |
| Role | Approval | Date |
| Author |  |  |
|  |  |  |

**Table of Contents**

[License and Copyright 4](#_Toc12531994)

[Introduction 5](#_Toc12531995)

[Document Purpose 5](#_Toc12531996)

[Audience 5](#_Toc12531997)

[Related Documents 5](#_Toc12531998)

[Conventions 5](#_Toc12531999)

[Acronyms & Definitions 5](#_Toc12532000)

[Design Considerations 6](#_Toc12532001)

[Technology Stack 6](#_Toc12532002)

[Platform 6](#_Toc12532003)

[Implementation Language 6](#_Toc12532004)

[Package Management 6](#_Toc12532005)

[Database 7](#_Toc12532006)

[Testing 7](#_Toc12532007)

[Development Environment 7](#_Toc12532008)

[Source and Version Control 7](#_Toc12532009)

[Building the Account Security Service 9](#_Toc12532010)

[Overview 9](#_Toc12532011)

[Establishing the Project Structure 10](#_Toc12532012)

[Prerequisites 10](#_Toc12532013)

[Brief Introduction to Two-Factor Authentication 11](#_Toc12532014)

[The Account Security API 12](#_Toc12532015)

[**ENDPOINT: GET /users/{userid}/tfa** 12](#_Toc12532016)

[**ENDPOINT: POST /users/{userid}/tfa/{enable}** 12](#_Toc12532017)

# License and Copyright

The content of this document is published and licensed in accordance with the MIT License. The terms and conditions are outlined below:

Copyright (c) 2019, Charles Stack. All Rights Reserved

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Introduction

## Document Purpose

This document describes the design of the User Service API Reference Implementation Service. The reference implementation exposes the User Service API and provides a set of REST based APIs intended to provide a general-purpose service for managing the users of a more complex system.

The reference implementation is not intended to be deployed “as-is”. It does, however, demonstrate how build a version of the API which is functional and provides a working version by which other implementations can test against.

## Audience

This document is intended for use by Software Development and Software Testing Teams.

## Related Documents

1. **Swagger\_UserAPI.yaml** – A Swagger 2.0 document describing the User Management Service API.
2. **OAS\_UserAPI.yaml** – An OAS 3.0 document describing the User Management Service API.
3. **UserManagement.yaml** – Contains just the User Management aspects of the Swagger\_UserAPI.yaml
4. **UserAccess.yaml** – Contains just the User Access aspects of the Swagger\_UserAPI.yaml
5. **PasswordManagement.yaml** – Contains just the Password Management aspects of the Swagger\_UserAPI.yaml
6. **AccountSecurity.yaml** – Contains just the Account Security aspects of the Swagger\_UserAPI.yaml

## Conventions

N/A

## Acronyms & Definitions

|  |  |
| --- | --- |
| **Term** | **Definition** |
| JSON | Javascript Object Notation |
| MIT License | A permissive free software license originating at the Massachusetts Institute of Technology (MIT) |
| OTP | One-Time-Password. A unique code generated either by an application or device the user holds or sent to the user via an Out-Of-Band (OOB) method (such as SMS). |
| REST | Representational State Transfer is a software architecture style that defines a set of constraints to be followed when creating web services. Web services that confirm to the REST conventions are known as RESTful web services and provide interoperability with other systems on the Internet. |
| SWAGGER/OAS | An open-source software framework that helps developers, design, build, document and consume RESTful Web Services. See <https://swagger.io/>  There are two specifications: Swagger (Version 2.0) and Swagger 3.0 (now known as the Open API Specification (OAS). |
| UUID | Universally Unique Identifier |
| YAML | Yet Another Markup Language |

# Design Considerations

## Technology Stack

The technology stack chosen is highly scalable and open-source. The Swagger and its successor, Open API Standard (OAS), provide a mechanism to clearly define the API in a manner that is both human and machine readable.

Platform

The reference implementation was developed on a Macintosh Pro laptop with 16 GB memory. The Operating System is, currently: macOS Mojave Version 10.14.5

## Implementation Language

The User Management Service API is, generally, programming language and platform neutral. While Swagger/OAS is used to define the API, the API can be implemented using the developer’s language and platform of choice.

This Reference Implementation has been developed using node.js. At the time of this writing, the current version of Node.js for Macintosh is 12.4.0. This version contains all the latest features. Similarly, version 10.16.0 LTS is also available.

You can download the appropriate node.js implementation from: <https://nodejs.org/en/>

## Package Management

The standard Node Package Manager, npm, is utilized to install and manage packages for this service.

npm is installed with node. At the time of this writing, the current version of npm is 6.9.0.

## Database

The design is database agnostic. However, for the reference implementation, MongoDB has been selected. Mongoose has been selected to provide object modelling.

MongoDB can be downloaded from: <https://www.mongodb.com/what-is-mongodb>

The current version of MongoDB server is: 4.0.10

The easiest way to install mongoDb is to use brew:

>brew install mongodb

At the time of this writing, brew will install: 4.0.3

Alternatively, the following link describes how to install mongoDB manually:

<https://treehouse.github.io/installation-guides/mac/mongo-mac.html>

Mongoose can be downloaded: <https://mongoosejs.com/>

MongoDB Compass provides a user friendly interface to MongoDB. The community edition can be downloaded from: <https://www.mongodb.com/download-center/compass>

To have launchd start mongodb now and restart at login:

brew services start mongodb

Or, if you don't want/need a background service you can just run:

mongod --config /usr/local/etc/mongod.conf

## Testing

In order to facilitate unit testing and promote the concept of Test-Driven Development (TDD), the reference implementation shall utilize the Mocha and Chai packages for node.js.

## Development Environment

The choice of a development environment is, usually, a personal choice. While some may prefer using a terminal and text editor, others prefer interactive development environments (IDEs).   
  
The preference for this project was Jetbrain’s WebStorm IDE. This tool has excellent support for Node.js and Swagger development.

## Source and Version Control

The API source document, source code, and related documentation is maintained on GitHub.

# Building the Account Security Service

## Overview

The initial source code was generated using the Codegen tool from within the Swagger Editor ([https://editor.swagger.io](https://editor.swagger.io/)). The Swagger code file, accountsecurity.yaml, was first imported into the Swagger editor. Then, the “Generate Server” menu, the “nodejs-server” option was selected. It is suggested that the file be renamed.

When the compressed file is unzipped, a directory called “nodejs-server-server” will exist. It contains a complete mock server using the Swagger tool suite for the account service. Rename the folder to something like “nodejs-accountsecurity-service”.

While other options exist, such as Express, for creating a REST server, this project uses the Swagger libraries. As a side-note, it is still possible to build the microservice using Express and incorporate SwaggerUI to present a way to document and test the server without a whole lot of effort. This, however, is outside the scope of this project.

The zip file was expanded in the developer directory, imported into the IDE, built and run. This performed all the necessary npm installs and create a working server that returns mock data.

## Establishing the Project Structure

The nodejs\_accountsecurity.zip file, when expanded, contains the following directories and files:

<root>

* + <.swagger-codegen>
* VERSION
  + <api>
* swagger.yaml
  + <controllers>
* AccountSecurity.js
  + <node\_modules>
    - .. lots of files
  + <service>
* AccountSecurityService.js
  + <utils>
* writer.js
  + .swagger-codegen-ignore
  + index.js
  + package.json
  + package-lock.json
  + README.md

The following should then added to produce the complete project structure

<root>

* + <models>
  + <test>
  + Dockerfile

If a Jenkins server is installed and available for use, a Jenkinsfile would also be added to implement CI/CD capability.

**Note - Implementing a Jenkins pipeline is outside the scope of this document.**

## Prerequisites

The Account Security service requires that there be a way to resolve a user’s identity. This implies that the User object and methods exists to locate a User by the userId field, read, set or unset the tfaEnabled flag, and extract the configured TFA methods from the the tfaMethods array. It is for this reason, that the User Management service should be implemented prior to implementing this service.

The Account Security service also provides a mechanism to provide authentication tokens via SMS. If this functionality is desired, an SMS gateway will be required and an appropriate node.js library. While other SMS/MMS gateway services exist, Twilio provides such a service and library for the sending of SMS message.  
  
See: <https://www.twilio.com/products> for more details on Twilio’s offerings.

Below is their sample code on how to send SMS via their service.  
<https://www.twilio.com/docs/sms/tutorials/how-to-send-sms-messages-node-js>

## Brief Introduction to Two-Factor Authentication

Two-Factor Authentication (aka TFA or 2FA) is a method by which a user provides additional proof of their identity through another authentication factor. In most cases, the user has a smartphone or device that is running an authenticator program such as Google Authenticator.  
  
Google Authenticator uses a technique called TOTP (**T**ime-based **O**ne **T**ime **P**assword) that uses a secret and a cryptographic hashing algorithm to generate a six (6) digit code that is valid for a specific time period (usually, 30 seconds).

The secret is securely stored in both the Google Authenticator and on the server-side so that both can generate a valid TFA code within the respective time window.

As only a device holding the secret can generate a valid code within the specified time window, it indicates that the user is in possession of the secret code. When combined with the user’s userid and password, it provides an extra measure of security on the user’s identity.

The following link provides a good tutorial on learning the basics of the TFA using the node library, speakeasy.

<https://www.thepolyglotdeveloper.com/2019/03/two-factor-authentication-totp-using-nodejs-speakeasy/>

When combined with the qrcode library, this library makes it easy to generate and verify industry standard TFA codes. Additionally, speakeasy can generate the URL, per <https://github.com/google/google-authenticator/wiki/Key-Uri-Format>

, that can be imported directly into an authenticator such as Google or Microsoft Authenticator.

It is the responsibility of both the authenticator AND this service to securely store the secret associated with the user’s account.

On the server side, it is recommended that the secret be encrypted in the event the database containing the secret is compromised.

## The Account Security API

The Account Security API is a subset of the methods implemented in the User Management API specification. It contains two endpoints that shall be described in detail.

### **ENDPOINT: GET /users/{userid}/tfa**

The first endpoint, GET /users/{userid}/tfa, is designed to return a TFA code to the user via an out-of-bounds (OOB) channel such as SMS. It is an optional endpoint. If not implemented, the service should return the HTTP error code 503 (Not Implemented).

Using the URL passed parameter, userid and the X-APIClientKey and X-APIClientSecret headers, this endpoint verifies the values passed via the headers and, if valid, retrieves the SMS identifier number (i.e. cell phone number, MISDN) for the user and their TFA secret from the database.

The TFA code is generated, applied to the text of the SMS message, and then the SMS message is sent to the User’s device.

Upon receipt of the SMS, the user must extract the TFA code contained in the SMS message and perform a login operation (outside of the scope of this API) to complete the login process.

### **ENDPOINT: POST /users/{userid}/tfa/{enable}**

The second endpoint, POST /users/{userid}/tfa/{enabled}, is designed configure a user account to utilize two-factor authentication.

It takes an additional Boolean query parameter, useSMS which can be used to enable (or disable) the use of SMS as a means to deliver the TFA code.

Like the GET endpoint, two header values are passed, X-APIClientKey and X-APIClientSecret to verify the caller is accessing the correct application.

This endpoint also requires that a set of user credentials be passed in the POST body. These credentials contain, at the minimum, the user’s username and password. These two values are necessary to ENABLE two-factor authentication on the User’s account.

The third element of the UserLoginRequest credentials is the twoFactorCode. This code is required ONLY to disable TFA on the account. As such, it must contain a valid TFA code for the account based on the stored secret.

If the useSMS code is provided as a parameter, the tfaMethod specifying SMS as an authentication channel can be added or removed.

When two factor authentication is successfully enabled on the account, the method returns a TFAInfoResponse. The elements of the TFAInfoResponse include:

uri – This is the text of the TOTP URI as utilized by Google and Microsoft authenticators.

image – this a MIME encoded QRCode representing the URI. This image, when decoded, can be displayed to the user and scanned, optically, by the Google and Microsoft authenticators.

imgFmt – this identifies the method used to generate the image. Examples of valid format values include: **PNG, SVG, PDF**

The node package, qrcode, only supports the PNG and SVG output options. If PDF or EPS is desired, another package supporting PDF QR code generation is required.