# Iota

Project Codebook

Cole Cassidy

*CS489 – Spring 2022 – Dr.Wang*

## Contents

## iotaEngine

The following source code is used to power the iota API server. It is written entirely in JavaScript and executed in the Node.JS runtime.

Utilized open-source libraries:

- Express.JS - https://expressjs.com/
    - HTTP(S) server framework
- body-parser - Express extension for handling contents of request bodies
- cors - Express extension for handling cross origin resources
- MongoDB - https://www.mongodb.com/docs/drivers/node/current/
    - Database connection driver
- jsonwebtoken - https://github.com/auth0/node-jsonwebtoken
    - Library for handling JSON Web Tokens
- bCrypt - https://github.com/kelektiv/node.bcrypt.js
    - Library for generating and handling password hashes

### package.json

This is an essential file for all node projects. It indicates the entry point for various launch functions, as well as any dependencies for the node package manager to install before launch.

```
{
  "name": "iotaengine",
  "version": "0.1.0",
  "description": "the logic behind iota",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Cole Cassidy",
  "license": "UNLICENSED",
  "dependencies": {
    "bcrypt": "^5.0.1",
    "body-parser": "^1.19.2",
    "cors": "^2.8.5",
    "express": "^4.17.3",
    "jsonwebtoken": "^8.5.1",
    "mongodb": "^4.4.1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/thecolec/iotaengine.git"
  },
  "bugs": {
    "url": "https://github.com/thecolec/iotaengine/issues"
  },
```

```json
  "homepage": "https://github.com/thecolec/iotaengine#readme"
}
```

## src/index.js

This is the main initialization file for the project. Executing this file in the Node.JS runtime sets environment variables, loads the dependencies, and starts the server.

```
// INDEX.JS
// Cole Cassidy - 2022
// Express API router.

// external libraries
var cors = require('cors');
var bodyParser = require('body-parser');

// express unique import
var express = require('express'),
    app = express(),
    port = 3000;
app.use(cors());

// internal libraries
var iota = require('./iota.js');
const iotaDB = require('./dbConn.js');
const orgs = require('./organizations');
const users = require('./users');
const minutes = require('./minutes');
const auth = require('./authentication');
const validator = require('./validator');

// express configuration
app.use(express.json());
app.use(bodyParser.urlencoded({extended: false}));
app.use(bodyParser.json());

// test endpoint to validate server is up
app.get('/', async (req, res) => {
    try{
        await iota.testConn();
        res.send("Hello World");

    } catch(e) {
        console.log(e)
    }

});

// test endpoint 2 for routing
```

```
app.get('/auth', (req, res) => {
    res.send("Future Authentication Endpoint")
    console.log("Auth Request Received");
});

// User API
// Deprecated and replaced by the contents of users.js
// Kept here to demonstrate simple express behaviors.
app.get('/user/:uid', async (req, res) => {
    try{
        console.log("User Request: "+req.params.uid+" From: "+req.ip);
        const userInfo = await users.findUser(req.params.uid);
        res.json(userInfo);
    } catch(e) {
        console.log(e);
    }
});

// Express routing table.
// Triggers route processing in imported files.
app.use('/org', orgs.router);
app.use('/usr', users.router);
app.use('/min', minutes.router);
app.use('/auth', auth.router);

// Waits to initialize the API server until DB connection is achieved.
console.log("Waiting for DB Connection");
iotaDB.connect(() => {
    app.listen(port);
    console.log('API START');
});
```

## src/iota.js

Provides software configuration. Centralizes authentication settings and error messages.

```
// IOTA.JS
// Cole Cassidy - 2022
// Application config/supporting functions.

// ERR Obj
// Predefined error messages to be used elsewhere in the codebase.
var ERR = {};
ERR.JSONNull         = {Error: true, msg: "Null JSON returned from DB"};
ERR.RequestInvalid   = {Error: true, msg:"Inproper Request"};
ERR.Unknown          = {Error: true, msg:"Unknown Error. Server may still be
initializing."};
ERR.UserExists       = {Error: true, msg:"User Already Exists"};
ERR.UserDNE          = {Error: true, msg:"User Does not exist"};
ERR.None             = {Error: false, msg:"Operation Succesful."};
ERR.Unauthorized     = {Error: true, msg:"User is not authorized for this
resource"};

// Auth config Obj
var AUTHCONFIG = {};
AUTHCONFIG.SaltRounds    = 10;
AUTHCONFIG.TokenKey      = "thisisatestkey";

// Exports
// Provides the following interfaces.
module.exports = {
    ERR,
    AUTHCONFIG
}
```

## ./src/authentication.js

Provides authentication related endpoints and their required functions.

```
// AUTHENTICATION.JS
// Cole Cassidy - 2022
// Handles server side authentication via JSON Web Tokens

// Imports
const express = require('express');
const router = express.Router();

const { ObjectId } = require('mongodb');

const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const { hash } = require('bcrypt');

const iota = require('./iota');
const dbconn = require('./dbConn');


const validator = require('./validator');



// ========================================
//        Authentication Functions
// ========================================

// ----- Core -----

// Registers a new user in the database.
// Creates unique entry in both user and auth databases
const regUser = async (user) => {

    var result;

    // create doc for user db
    newUsr = {
        "uName": user.uName.toUpperCase(),
        "fName": user.fName,
        "lName": user.lName,
        "rank": 0,
        "joindate": new Date()
    }
    // create doc for auth db
    secUsr = {
        "uName": user.uName.toUpperCase()
    }

    // perform duplicate check against user db
    var check = await checkUserExists(user.uName.toUpperCase());
    if(check) {
        return iota.ERR.UserExists;
    }

    // append hashed password to auth db doc
    secUsr.pWord = await passCrypt(user.pWord);

    // write new user doc to user db
    try {
        result = await dbconn.get().collection("users").insertOne(newUsr);
        result = newUsr;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }

    // generate a user unique token
    token = await makeToken(result);

    // append unique token to auth doc
    secUsr.uid = result._id;
    secUsr.token = [token];

    // write auth doc to db
    dbconn.get().collection("auth").insertOne(secUsr);

    // append new token to result for the registering application
    result.token = token;
    return result;
}

// Handles user authentication.
// Returns a token if user credentials are correct.
const authUser = async (doc) => {
    const userAuth = await dbconn.get().collection("auth").findOne({"uName":
doc.uName.toUpperCase()});
    console.log(userAuth);
    if(userAuth == null) return iota.ERR.UserDNE;
    // iota.AUTHCONFIG.SaltRounds;

    const user = await dbconn.get().collection("users").findOne({"_id":
userAuth.uid });
    // console.log("A");
    // console.log(user);
    // console.log(await bcrypt.compare(doc.pWord, userAuth.pWord));



    if(await bcrypt.compare(doc.pWord, userAuth.pWord)){
        token = await makeToken(user);
        console.log(token);
        user.token = token;
        return user;
    }
    return iota.ERR.Unauthorized;
}

// ----- Support -----

// Validates if user with given uName exists.
const checkUserExists = async (name) => {
    // Returns True if user exists. Otherwise returns false.
    // Also returns true if error occurs in order to protect DB from duplicates.
    try {
        const doc = await dbconn.get().collection("users").findOne({"uName":
name.toUpperCase()});
        if(doc != null) return true;
        return false;
    } catch(e) {
        console.error(e);
        return true;
    }
}

// Returns the hash for a given password.
const passCrypt = async (badPass) => {
    // Hashes cleartext password.
    return await bcrypt.hash(badPass, iota.AUTHCONFIG.SaltRounds);
}

// Generates A new signed token.
const makeToken = async (user) => {
    // Creates new token from provided user doc
    const token = jwt.sign(
        user,
        iota.AUTHCONFIG.TokenKey
    );
    return token;
}

// Returns an updated user token.
const updateToken = async (user) => {
    try {
        const newDoc = await dbconn.get().collection("users").findOne({"_id": new
ObjectId(user._id)});
        const token = jwt.sign(
            newDoc,
            iota.AUTHCONFIG.TokenKey
        );
        return token;
    } catch(e){
        console.error("Failed Token Update");
        return iota.ERR.Unknown;
    }
}

// ========================================
//              Express Routes
// ========================================

// middleware to log endpoint access
router.use((req, res, next) => {
    console.log("REQ: AUT  IP: "+req.ip);
    next();
});

// ---- GET ----

// API endpoint to return an updated token if current token is valid
// "../auth/update"
router.get('/update', validator.checkToken, async (req, res) => {
    const doc = await updateToken(req.user);
    res.json(doc);
});

// ---- POST ----
// API endpoint to allow users to authenticate
```

```javascript
// "../auth"
router.post('/', async (req, res) => {
    const doc = await authUser(req.body);
    res.json(doc);
});


// API endpoint to allow registration of new user
// "../auth/reg"
router.post('/reg', async (req, res) => {
    const doc = await regUser(req.body);
    res.json(doc);
});

// API endpoint to allow user to authenticate (alternate)
// "../auth/login"
router.post('/login', async (req, res) => {
     console.log(req.body);
     const doc = await authUser(req.body);
     res.json(doc);
});

// EXPORTS
module.exports = {
    router
}
```

## ./src/dbConn.js

Handles creating and maintaining the shared database connection.

```
// dbConn.JS
// Cole Cassidy - 2022
// Handles all database communication.

// IMPORTS
const assert = require('assert');
const res = require('express/lib/response');

const { MongoClient, ObjectId } = require('mongodb');

// MongoDB URI
// Included here for example purposes. Should be stored as environment variable.
const uri =
"mongodb+srv://iotabot:jameshalliday@cluster0.m5mj8.mongodb.net/myFirstDatabase?r
etryWrites=true&w=majority"

let iotaDB;

// Standard Database access functions
// Handle all database requests using the above query format.
// async friendly
// not structurally necessary, but makes development significantly easier.

// Creates a pooled connection for db calls.
// Includes callback option for functions to be executed after successful
connection.
function connect(callback){
    MongoClient.connect(uri, (err, db) => {
        assert.equal(null, err);
        iotaDB = db.db("iota_testing");
        console.log("Connected :D");
        callback();
    });
}

// returns the connected database as an object. Allows pooling requests over one
connection.
function get(){
    return iotaDB;
}
```

```
// closes the shared pool. Once executed no other connections are possible
without reopening.
function close(){
    iotaDB.close();
}

// EXPORTS
module.exports = {
    connect,
    get,
    close
}
```

## ./src/minutes.js

Handles the creation and retrieval of minute documents from the database. Provides API endpoints and their supporting functions.

```
// MINUTES.JS
// Cole Cassidy - 2022
// Contains core user functions and routing

// Imports
const express = require('express');
const { normalizeType } = require('express/lib/utils');
const router = express.Router();

const { ObjectId } = require('mongodb');
const dbconn = require('./dbConn');
const validator = require('./validator');

const iota = require('./iota');

// ========================================
//              DB Functions
// ========================================

// ----- Read -----

// returns a document containing all minutes.
const listAll = async () => {
    try {
        const doc = await dbconn.get().collection("minutes").find().toArray();
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.ERRUnkown;
    }
}

// returns minutes filtered by an organization ID
const minByOrg = async (id) => {
    try {
        console.log(id);
        const doc = await dbconn.get().collection("minutes").find({"oid": new
ObjectId(id)}).toArray();
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.ERRUnkown;
    }
}

// returns a minute document with the given ID
const minByID = async (id) => {
    try {
        const doc = await dbconn.get().collection("minutes").findOne({"_id": new
ObjectId(id)});
        console.log(id);
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.ERRUnkown;
    }
}


// ----- Write -----

// generates a new minutes document and stores it to the database
const newMin = async (newDoc, user) => {
    console.log(newDoc);
    console.log(user);
    var doc = {
        createDate: new Date(),
        modDate: new Date(),
        oid: ObjectId(newDoc.oid),
        title: newDoc.title,
        state: "open",
        creator: {
            uName: user.uName,
            fName: user.fName,
            lName: user.lName,
            uid: user._id,

        },
        contents: newDoc.contents
    }
    var resp = await dbconn.get().collection("minutes").insertOne(doc);
    return resp;
}
```

```
// ========================================
//              Express Routes
// ========================================

// middleware to log requests to this endpoint
router.use((req, res, next) => {
    console.log("REQ: MIN "+"IP: "+req.ip+" : "+req.url);
    next()
});

// ---- GET ----

// API endpoint that returns all minutes.
// Included for demonstration purposes. No longer used.
// "./min/"
router.get('/', async (req, res) => {
    const doc = await listAll();
    res.json(doc);
});

// API endpoint to return minutes filtered by a given organization ID
// "./min/org/<OID>"
router.get('/org/:oid', async (req, res) => {
    console.log("MINUTES BY OID: "+req.params.oid);
    const resp = await minByOrg(req.params.oid);
    res.json(resp);
});

// API endpoint to return a specific minutes document by given document ID
// "./min/s/<ID>"
router.get('/s/:id', async (req, res) => {
    const resp = await minByID(req.params.id);
    res.json(resp);
});

// ---- POST ----
// API endpoint to store new minutes document
// "./min/new"
router.post('/new', validator.checkToken, validator.authUser, async (req, res) =>
{
    const resp = await newMin(req.body, req.user);
    console.log(resp);
    res.json(resp);
})
```

```
// EXPORTS
module.exports = {
    router
}
```

## ./src/organizations.js

Provides all the endpoints for organization related API calls and their supporting functions. Also handles the generation of new registration codes and attaching them to organization documents.

```
// ORGANIZATIONS.JS
// Cole Cassidy - 2022
// Contains core organization functions and routing

// Imports
const { promise } = require('bcrypt/promises');
const express = require('express');
const router = express.Router();

const { ObjectId } = require('mongodb');
const dbconn = require('./dbConn');
const iota = require('./iota');
const validator = require('./validator');

// ========================================
//             DB Functions
// ========================================

// ----- READ -----

// returns organization document with given ID from database.
const findOrg = async (uid) => {
    try {
        const doc = await
dbconn.get().collection("organizations").findOne({"_id": new ObjectId(uid)});
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// returns verbose organization document with given ID from database.
// verbose document is the regular document + all user docs from the organization
const findOrgVerbose = async (oid) => {
    const usrDoc = listOrgUsrs(oid);
    const orgDoc = findOrg(oid);
    return Promise.all([usrDoc, orgDoc]).then((values) => {
        var doc = values[1];
        doc.members = values[0];
        return doc;
    });
}

// Returns a list of all registered organizations in database
// Deprecated. Included for demonstration purposes.
const listAllOrgs = async () => {
    try {
        const doc = await
dbconn.get().collection("organizations").find().toArray();
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// Returns a list of all users in an organization by given OID.
const listOrgUsrs = async (oid) => {
    try {
        const doc = await dbconn.get().collection("users").find({"orgs.oid": new
ObjectId(oid)}).toArray();
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}


// ----- WRITE -----

// Adds reference to an organization to a provided user.
// Uses supporting registration code functions.
const addUser = async (regCode, user) => {
    const uid = user._id;
    try {
        const org = await
dbconn.get().collection("organizations").findOne({"reg.code": regCode});
        const filter = {"_id": ObjectId(uid)};
        const change = { "$addToSet": {
            "orgs": {
                "Name": org.Name,
                "oid": org._id
            }
        }};
        const result = await
dbconn.get().collection("users").updateOne(filter,change,{});
        console.log("uid: "+uid+" linked to oid: "+org._id);
        console.log(result);
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// ========================================
//           Registration Codes
// ========================================
// The following functions handle the generation and use of registration codes.

// Validates regCode
const regCode = async (oid) => {
    const doc = await regCodeRefresh(oid);
    console.log( doc );
    return doc.reg.code;
}

// checks if regCode is stale and updates the doc if so
const regCodeRefresh = async (oid) => {
    var doc = await findOrg(oid);
    const curr = Date.now();

    // if org has no regcode generate a new one.
    if(doc.reg == null ) {
        return await updateRegCode(doc);
    }

    var regExp = doc.reg.exp;
    const diff = curr - regExp;
    const conv = 1000 * 60;

    // if older than one hour create new code and update
    if(diff/conv >= 1) {
        doc = await updateRegCode(doc);
    }

    return doc;
}

// Updates new regCodes
```

```
const updateRegCode = async (doc) => {
    const curr = Date.now();

    // mongoDB call to write new code
    const filter = {
        _id: new ObjectId(doc._id)
    }
    var newCode = await genRegCode();

    const updateCode = {
        $set: {
            'reg': {
                'code': newCode,
                'exp': curr
            }
        }
    }
    const res = await dbconn.get().collection("organizations").updateOne(filter,
updateCode);

    // update current orgDoc with new code and return
    doc.reg = {
        code: newCode,
        exp: curr
    }

    return doc;
}

// Generate Registration Code
const genRegCode = async () => {

    // Generate Code
    const charset = '0123456789ABCDEFGHIJKLMNPQRSTUVWXYZ';
    var newRegCode = "";
    for (i = 0; i<6; i++ ){
        newRegCode += charset.charAt(Math.floor(Math.random() * charset.length));
    }

    // Validate against DB
    try {
        const res = await
dbconn.get().collection("organizations").findOne({"reg.code": newRegCode});
        if(res === null) return newRegCode;
    } catch (e) {
        console.error(e);
        return "ERR";
    }
}

// ========================================
//             Express Routes
// ========================================

// Logs traffic to this endpoint.
router.use((req, res, next) => {
    console.log("REQ: ORG "+"IP: "+req.ip+" : "+req.url);
    next()
});

// Lists all organizations
//"./org/"
router.get('/', async (req, res) => {
    const doc = await listAllOrgs();
    res.json(doc);
});

// Lists all organizations
// "./org/list"
router.get('/list', async (req, res) => {
    const doc = await listAllOrgs();
    res.json(doc);
});

// Returns basic organization data.
// "./org/s/<oid>"
router.get('/s/:oid', async (req, res) => {
    const doc = await findOrg(req.params.oid);
    res.json(doc);
    //6234ec073b13b586a6b369fc
});

// Returns organization user list
// "./org/s/<oid>/users"
router.get('/s/:oid/users', async (req, res) => {
    const doc = await listOrgUsrs(req.params.oid);
    res.json(doc);
    //6234ec073b13b586a6b369fc
});
```

```
// Returns non-stale registration code
// "./org/reg/<oid>"
router.get('/reg/:oid', async (req, res) => {
    res.json(await regCode(req.params.oid));
})

// Adds user to organization by Registration Code
// "./org/reg/<regcode>"
router.post('/reg/:regcode', validator.checkToken, async(req, res) => {
    const doc = await addUser(req.params.regcode, req.user);
    res.json();
})

// Returns verbose organization data.
// "./org/s/<oid>/full"
router.get('/s/:oid/full', validator.checkToken, validator.authUser, async (req,
res) => {
    res.json( await findOrgVerbose(req.params.oid) );
})

// EXPORTS
module.exports = {
    router
}
```

## ./src/users.js

Provides API endpoints to access and modify user data. Also provides the supporting functions for these endpoints. Uses validator.js to protect certain endpoints.

```
// USERS.JS
// Cole Cassidy - 2022
// Contains core user functions and routing

// Imports
const express = require('express');
const router = express.Router();

const { ObjectId } = require('mongodb');
const dbconn = require('./dbConn');
const iota = require('./iota');
const validator = require('./validator');

// ========================================
//             DB Functions
// ========================================

// ----- Read -----

// Returns user document matching provided user document ID.
const findUser = async (uid) => {
    try {
        const doc = await dbconn.get().collection("users").findOne({"_id":
ObjectId(uid)});
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// Returns a single document containing all registered users.
// Depricated. Included here for demonstration.
const listAllUsers = async () => {
    try {
        const doc = await dbconn.get().collection("users").find().toArray();
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
```

```
}

// Returns a user document matching provided username.
const findUserName = async (name) => {
    try {
        const doc = await dbconn.get().collection("users").findOne({"uName":
name});
        return doc;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// ----- Write -----

// Generates a new user Document.
// This has been functionally replaced by the "regUser()" function in
authentication.js
// Included here for admin access to add demonstration users.
const addUser = async (user) => {
    newUsr = {
        "uName": user.uName.toUpperCase(),
        "fName": user.fName,
        "lName": user.lName,
        "joindate": new Date()
    }
    var check = await findUserName(user.uName.toUpperCase());
    console.log(check);
    if(check != null) return iota.ERR.Unknown;
    try {
        var result = await dbconn.get().collection("users").insertOne(newUsr);
        result.document = newUsr;
        return result;
    } catch(e) {
        console.error(e);
        return iota.ERR.Unknown;
    }
}

// Modifies a user document given the new information and user document ID.
const editUser = async (user) => {
    const filter = {
        _id: new ObjectId(user._id)
    }
    const update = {
        $set: {
            'fName': user.fName,
            'lName': user.lName,
            'bio': user.bio
        }
    }
    const res = await dbconn.get().collection("users").updateOne(filter, update);
    return res;

}

// ========================================
//             Express Routes
// ========================================

// middleware that logs access to this endpoint.
router.use((req, res, next) => {
    console.log("REQ: USR  IP: "+req.ip+" : "+req.url);
    next();
});

// API endpoint to provide a list of all registered users.
// Included to demonstrate database functions at research presentations.
// Scheduled to be removed.
// "./usr/"
router.get('/', async (req, res) => {
    const doc = await listAllUsers();
    res.json(doc);
});

// API endpoint to return a specific user document.
// "./usr/s/<uid>"
router.get('/s/:uid', async (req, res) => {
    const doc = await findUser(req.params.uid);
    res.json(doc);
});

// API endpoint to return a list of all users.
// "./usr/list"
router.get('/list', async (req, res) => {
    const doc = await listAllUsers();
    res.json(doc);
});
```

```
// API endpoint to add a new User
// Deprecated. Included for demonstration purposes.
// "./usr/add"
router.post('/add', async (req, res) => {
    const doc = await addUser(req.body);
    res.json(doc);
});

// API endpoint to edit a userbio
// "./usr/edit/<uid>"
router.post('/edit/:uid', validator.checkToken, validator.authUser, async
(req,res) => {
    const doc = await editUser(req.body);
    res.json(doc);
})

// EXPORTS
module.exports = {
    router,
    findUser
}
```

## ./src/validator.js

Middleware that can be used to validate tokens and authenticate actions whenever implemented on API endpoints.

```
// VALIDATOR.JS
// Cole Cassidy - 2022
// Validates all tokens

// Imports
const express = require('express');

const jwt = require('jsonwebtoken');
const { ObjectId } = require('mongodb');

const iota = require('./iota');

// ========================================
//            Token Validation Functions
// ========================================

// Checks token validtity.
const checkToken = (req, res, next) => {

    // Extracts token from body or headers.
    const token = req.body.token || req.headers["x-access-token"];

    // If there is no token responds with error and ends request handling.
    if(token == null) return res.status(403).json(iota.ERR.RequestInvalid);

    // Validates token and appends user data to request for use in other
functions in the callback chain.
    try {
        req.user = jwt.verify(token, iota.AUTHCONFIG.TokenKey);
    } catch(e){
        console.log(e);

        // If the extract fails the token is invalid. Responds with error and
ends request handling.
        return res.status(401).json(iota.ERR.RequestInvalid);
    }

    // Executes next callback function.
    return next();
}
```

```
// Checks authentication for user.
// Does not validate tokens.
// If this function is called without first validating a token signature, hacked
tokens can then execute privledged actions.
// ALWAYS VALIDATE FIRST.
const authUser = (req, res, next) => {

    // Extracts user and destingation data appended to request by checkToken();
    const user = req.user;
    const dest = req.body;

    // Log authentication attempt
    console.log(req.user);
    console.log(req.body);
    console.log(req.user.orgs.some(org => org.oid === req.body.oid));

    // If user has admin auth allow any action
    if(req.user.rank >= 3) return next();

    // If user is a member of affected group
    if( req.user.orgs.some(org => org.oid === req.body.oid)){
        return next();
    }

    // Same as previous statement, accounts for different methods of OID
representation
    if( req.user.orgs.some(org => org.oid === new ObjectId(req.body.oid))){
        return next();
    }

    // Same as previous statement, used for GET requests and requests with no
body.
    if(req.user.orgs.some(org => org.oid === req.params.oid)){
        console.log("hello ");
        return next();
    }

    // If user is affecting self
    if(req.user._id === ObjectId(req.body.uid)){
        return next();
    }

    // Otherwise terminate response with an unauthorized error.
    return res.status(401).json(iota.ERR.Unauthorized);
}

// EXPORTS
module.exports = {
    checkToken,
    authUser
}
```

## iotaApp

The following source code is used to build the iotaApp webapp. It is built using TypeScript/JavaScript and the Angular framework. It utilizes Bootstrap and is compiled using the Angular development environment.

Utilized open-source libraries:

- Angular - https://angular.io/
  - Framework used to build the application.
- Angular-JWT - https://github.com/auth0/angular-jwt
  - Angular extension used to handle JWTs

### Component Files

Angular components typically use four files to create a component. Each of these files is contained in a folder and sharing the name of the component.

/<component-name>/

- <component-name>.component.html
  - Creates a template for the component if it is to be rendered.
  - Written by the developer.
- <component-name>.component.scss
  - Provides a Sassy CSS stylesheet unique to the component.
  - Usage of this file is optional and therefor is only included in this document for components that utilize this file.
- <component-name>.component.spec.ts
  - A specification file generated by the Angular development environment.
  - Will not be included in this document as these files are not unique and only serve to reference the folder and its contents to the compiler.
- <component-name>.component.ts
  - Provides the functionality of each component.
  - Written by the developer.

Due to the extensive nature of these files, the following section will contain the folders and their contents for each component. It will exclude blank or unused SCSS files. It will also exclude the ".SPEC.TS" files as they are not written by the developer and are generated by the compiler.

## add-org-page

This component generates the page used to register a user to an organization. It provides a form interface to enter and submit the code.

### *add-org-page.component.html*

The HTML template. Contains a simple single field form with a submit button. Submit button also redirects user to their dashboard.

```
<div class="container-fluid">
    <div class="row">
        <div class="col-md-4 offset-md-4">
            <div class="card">

                <div class="card-body">
                    <h3 class="card-title">
                        Let's Go!
                    </h3>
                    <p> You're on the way to joining a new organization. Enter
your 6 digit registration code below to get started!</p>
                    <form #regCodeJoin="ngForm">
                        <div class="input-group mb-3">
                            <span class="input-group-text">RegCode</span>
                            <input type="text" class="form-control"
placeholder="000000" required
                            id="docTitle"
                            [(ngModel)]="regCode" name="title" #name="ngModel">
                        </div>
                        <div>
                            <button type="submit"
                                    class="btn btn-success"
                                    [disabled]="!regCodeJoin.form.valid"
                                    (click)="joinOrg();
                                    regCodeJoin.reset();"
                                    routerLink="/dash"> Join
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>

<div class="container">
```

```
</div>
```

*add-org-page.component.ts*

Provides form functionality and submitsd to the JS regCode string for submission to the UserService

```
// Cole Cassidy
// Iota - 2022

import { Component, OnInit } from '@angular/core';
import { AuthService } from '../auth.service';
import { UserService } from '../user.service';

@Component({
  selector: 'app-add-org-page',
  templateUrl: './add-org-page.component.html',
  styleUrls: ['./add-org-page.component.scss']
})
export class AddOrgPageComponent implements OnInit {

  constructor(private iotaAPI: UserService, private iotaAuth: AuthService) { }

  regCode: string;

  ngOnInit(): void {
  }

  // submit button executes this function
  // instructs the UserService to execute the joinOrg API Call
  joinOrg(): void {
    this.iotaAPI.joinOrg(this.regCode);

  }

}
```

## authpage

Provides a user friendly wrapper for login form. Loads the “app-login-form” component.

### *authpage.component.html*

```html
<div class="container-fluid">
    <div class="row text-center">
        <h1 class="display-1"> Iota </h1>
    </div>
    <div class="row">
        <div class="col-md-4 offset-md-4">
            <app-login-form></app-login-form>
        </div>
    </div>
</div>
```

### *authpage.component.ts*

Does not provide functionality. This component is a wrapper for the functional “app-login-form” component.

```typescript
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-authpage',
  templateUrl: './authpage.component.html',
  styleUrls: ['./authpage.component.scss']
})
export class AuthpageComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

## dashboard

This is the primary view for the users. When users log in this component renders an account overview and provides links to commonly needed functions.

### *dashboard.component.html*

Provides the template for the dashboard. Populates with user information. Uses Angular Router to navigate to the add-org-page component and the userbio component

```
<div class="container-fluid">
    <div class="row">
        <div class="col mb-3">
            <h1 class="text-center">Welcome, {{whoami.fName}} </h1>
            <p class="lead text-center">You've got work to do.</p>
        </div>
    </div>
    <div class="row">
        <div class="col-md-4 mb-3">
            <app-org-panel [orgs]="whoami.orgs"></app-org-panel>
        </div>
        <div class="col-md-4 mb-3">
            <div class="d-grid gap-2">
                <button type="button" class="btn btn-success"
                routerLink="/register/join">Add Organization</button>

                <button type="button" class="btn btn-primary"
                routerLink="/usr/1/{{ whoami._id }}">Edit Account</button>

            </div>
        </div>
    </div>
</div>
```

### *dashboard.component.ts*

On initialization this component binds itself to the output of the whoami AuthService Observable. This ensures that the fields in the previous template stay populated with the most recent data.

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../auth.service';
import { Organization } from '../organization';
import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.sass']
})
export class DashboardComponent implements OnInit {

  constructor(private iotaAuth: AuthService, private iotaApi: UserService) {
  }

  whoami: User;

  ngOnInit(): void {
    this.iotaAuth.userInfo.subscribe(user => this.whoami = user);
  }

}
```

## login-form

This component provides the actual login form. It is implemented separately from the actual login page so that its wrapper can be modified separately from the form functionality.

### *login-form.component.html*

The template for the login form. Contains multiple fields including binding to the variables in the associated typescript file. Fires the newUser() function on submit and sends the user to the dashboard.

```
<div class="card">
    <div class="card-body">
        <form  #createUser="ngForm">
            <h5 class="card-title">Login</h5>
            <!-- UserName -->
            <div class="mb-3">
                <label for="new-user-uName" class="form-label">Username:</label>
                <input type="text" id="new-user-uName" class="form-control"
required
                [(ngModel)]="model.uName" name="uName"
                #name="ngModel">
            </div>

            <!-- Password -->
            <div class="mb-3">
                <label for="new-user-pWord" class="form-label">Password:</label>
                <input type="password" id="new-user-pWord" class="form-control"
required
                [(ngModel)]="model.pWord" name="pWord">
            </div>
            <!-- Submit -->
            <button type="submit" class="btn btn-primary"
            [disabled]="!createUser.form.valid"
            (click)="newUser();
            createUser.reset()"
            routerLink="/dash">Login</button>
        </form>
    </div>
</div>
```

*login-form.component.ts*

Provides the model object the form binds to, as well as the function call to submit the model object to the AuthService for authentication.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';

import { AuthUser } from '../user';

@Component({
  selector: 'app-login-form',
  templateUrl: './login-form.component.html',
  styleUrls: ['./login-form.component.sass']
})
export class LoginFormComponent implements OnInit {

  constructor(private iotaAuthAPI: AuthService) { }

  model = new AuthUser("", "");

  ngOnInit(): void {
  }

  onSubmit() {
  }

  newUser() {
    return this.iotaAuthAPI.login(this.model);
  }

}
```

## minute-editor

This component provides the minutes/reports interface. It contains the viewer and the editor, allowing for a singular user-facing minutes endpoint.

### *minute-editor.component.html*

```
<div class="container">
    <!-- If the mode is 1 render viewer div -->
    <div *ngIf="getMode() === 1 && this.document">

        <!-- Display document title, if title is empty display the ID, if ID is
empty display error msg. -->
        <h1> {{ this.document.title || this.document._id || "Invalid
Document"}}</h1>

        <!-- Call function to display rendered date -->
        <h5> Created: {{ getDate() }}</h5>

        <!-- display creator first and last name -->
        <h5> Author: {{ this.document.creator.lName }}, {{
this.document.creator.fName }}</h5>

        <!-- Display contents if available, otherwise display error. -->
        <p> {{ this.document.contents || "Missing Contents" }} </p>
    </div>

    <!-- If the mode is 0 render the editor div and form -->
    <div *ngIf="getMode() === 0 && this.document">
        <div>
            <h1>New Report</h1>
        </div>
        <!-- Minute editor form -->
        <form (ngSubmit)="onSubmit()" #writeMinutes="ngForm">
            <div class="input-group mb-3">
                <span class="input-group-text">Title</span>
                <input type="text" class="form-control" placeholder="Submission
Title" required
                id="docTitle"
                [(ngModel)]="document.title" name="title" #name="ngModel">
            </div>
            <div class="mb-3">
                <label class="form-label">Contents</label>
                <textarea type="text" class="form-control" placeholder="Write
your notes here..." required
                id="docContents" rows="5"
                [(ngModel)]="document.contents" name="contents"
#contents="ngModel"></textarea>
            </div>
            <div>
                <!-- Trigger submission function and return to parent ORG page --
>
                <button type="submit"
                        class="btn btn-success"
                        [disabled]="!writeMinutes.form.valid"
                        (click)="submitMinutes();
                        writeMinutes.reset();"
                        routerLink="/org/{{ this.id }}"> Save
                </button>
            </div>
        </form>
    </div>
</div>
```

*minute-editor.component.ts*

Handles retrieval and submission of minutes documents depending on the mode of this endpoint.

```
import { DatePipe } from '@angular/common';
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { AuthService } from '../auth.service';
import { Minutes, MinutesObj } from '../minutes';
import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-minute-editor',
  templateUrl: './minute-editor.component.html',
  styleUrls: ['./minute-editor.component.sass']
})
export class MinuteEditorComponent implements OnInit {

  constructor(private iotaAPI: UserService, private iotaAuth: AuthService,
private route: ActivatedRoute) {
  }

  private mode: number = 0;
  public id: string;
  public document: Minutes;
  // On init
  ngOnInit(): void {
    // Set Mode and ID to match the URL parameters
    this.mode = Number(this.route.snapshot.paramMap.get('mode'));
    this.id  = this.route.snapshot.paramMap.get('id') || "";
    // Load minutes
    this.loadMinuteSpec();
  }

  loadMinuteSpec(): void {
    // If in viewer mode use UserService to retrieve specific minutes
    if(this.mode === 1) {
      this.iotaAPI.getMinutesSpec(this.id).subscribe(doc => this.document = doc);
    }
    // If editor mode generate blank template to bind to form.
    if(this.mode === 0) {
      this.document = this.genTemplate();
    }
  }

  // Utility function to return private MODE
  getMode(): number {
    return this.mode;
  }

  // Generates an empty Minutes type Object containing the Organization ID and
the User Document
  genTemplate() : Minutes {
    const usrDoc: User = this.iotaAuth.readUserDoc();
    const doc = new MinutesObj("","","",this.id,usrDoc,"" ,"");
    return doc;
  }

  // Return easier-to-read date string from document date.
  getDate(): string {
    const datePipe = new DatePipe('en-US');
    const date = datePipe.transform(this.document.createDate, 'MM/dd/yy h:mm a');
    return date || "";
  }

  onSubmit() {}

  // Use UserService to submit minutes document.
  submitMinutes() {
    this.iotaAPI.putMinutes(this.document);
    console.log(this.document);
  }

}
```

## minutes-panel

This is a smaller component. It does not have a router endpoint and is instead loaded into the org-page component to list minutes by organization It displays a panel listing all minutes provided for the endpoint it is loaded on.

### *minutes-panel.component.html*

```
<ul class="minutes-panel list-group">
    <li class="list-group-item active">Reports:</li>
    <a *ngFor="let doc of minutes" class="list-group-item d-flex justify-content-
between align-items-center"
    routerLink="/min/1/{{doc._id}}">
        {{doc.title || doc._id}}
    </a>
</ul>
```

### *minutes-panel.component.ts*

```
import { Component, Input, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Minutes } from '../minutes';
import { UserService } from '../user.service';

@Component({
  selector: 'app-minutes-panel',
  templateUrl: './minutes-panel.component.html',
  styleUrls: ['./minutes-panel.component.sass']
})
export class MinutesPanelComponent implements OnInit {

  @Input() oid: string;
  minutes: Minutes[];

  constructor(private iotaApi: UserService, private route: ActivatedRoute) { }

  // On init
  ngOnInit(): void {
    // load oid from URL parameters.
    const oid = this.route.snapshot.paramMap.get('uid') || "";
    // retrieve list of minute fo the provided OID
    this.getMinutes(oid);
  }

  getMinutes(oid: string): void {
    // Use UserService to retrieve minutes list.
    this.iotaApi.getMinutes(oid).subscribe(doc => this.minutes = doc);
  }

}
```

## navbar

This component is included on the root document. It subscribes to the AuthService and the router to dynamically reflect the users current location in the application, and present options dependent on the login state.

*navbar.component.html*

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <a class="navbar-brand" routerLink="/dash">{{ title }}</a>
        <!-- bootstrap menu button. Only appears on mobile devices -->
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <!-- Dashboard - Only displayed when logged in. -->
                <li class="nav-item" *ngIf="loginState$ | async">
                    <a class="nav-link" routerLink="/dash">Dashboard</a>
                </li>
            </ul>
            <ul class="navbar-nav ms-auto">
                <!-- Logout - Only displayed when logged in. -->
                <li class="nav-item" *ngIf="loginState$ | async">
                    <a class="nav-link" (click)="logout();"
routerLink="/">Logout</a>
                </li>
                <!-- Register - Only displayed when logged out. -->
                <li class="nav-item" *ngIf="!(loginState$ | async)">
                    <a class="nav-link" routerLink="/register">Register</a>
                </li>
                <!-- Login - Only displayed when logged out. -->
                <li class="nav-item"  *ngIf="!(loginState$ | async)">
                    <button class="btn btn-primary" type="login"
routerLink="/login">Login</button>
```

```
                </li>
            </ul>
        </div>
    </div>
</nav>
```

*navbar.component.ts*

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';
import { User } from '../user';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.sass']
})
export class NavbarComponent implements OnInit {


  constructor(private iotaAuth: AuthService) {

  }

  // Create observable variables for the navbar template to subscribe to.
  loginState$ : Observable<boolean>;
  currUser$: Observable<User>;


  title = "Iota (beta)";
  Name = "";

  ngOnInit() {
    // Subscribe to the loginState provided via AuthService
    this.loginState$ = this.iotaAuth.loginStateCheck;

    // Subscribe to the user document via AuthService and assign username to Name
field.
    this.iotaAuth.userInfo.subscribe(user => this.Name = user.uName);

  }

  // Use logout method provided by AuthService when user clicks logout button.
  logout(){
    this.iotaAuth.logout();
  }

}
```

## org-page

The org-page component provides an overview interface for organizations. It Includes organization information, the user-panel component, the minutes-panel component, and it provides registration codes.

### *org-page.component.html*

```
<div class="container-fluid" *ngIf="org">
    <!-- Retrieve organization name -->
    <h1>{{ getName() }}</h1>
    <div class="row gy-2">
        <div class="col-md-4">
            <!-- Display byline. If no byline display default byline. -->
            <p>{{ org.byline || "A student organization"}}</p>
        </div>
        <div class="col-md-4">
            <!-- Display sanitized organization creation date. -->
            <p>Registered: {{ getDate() }}</p>
        </div>

    </div>
    <hr/>
    <div class="row gy-2">
        <div class="col-md-4 gy-2">
            <!-- Load user-panel component with Member object array as input. -->
            <app-user-panel [users]="org.members"></app-user-panel>
        </div>
        <div class="col-md-4 gy-2">
            <!-- Load minutes-panel component -->
            <app-minutes-panel></app-minutes-panel>
        </div>
        <!-- Function Buttons -->
        <div class="col-md-4 gy-2">
            <div>
                <!-- New Minutes -->
                <button type="button" class="btn btn-success"
                routerLink="/min/0/{{org._id}}">New minutes</button>

            </div>
        </div>
        <!-- REGISTRATION CODE -->
        <div class="col-md-2 gy-2">
            <div class="card regCode-card">
                <div class="card-header regCode-head">
                    Join Code:
                </div>
                <div class="card-body regCode-body ">
                    <!-- Display received registration code. -->
                    <h1>{{ regCode }}</h1>
                </div>
            </div>
        </div>
    </div>
    <hr/>
</div>
```

*org-page.component.ts*

```
import { DatePipe } from '@angular/common';
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { min } from 'rxjs';
import { AuthService } from '../auth.service';
import { Minutes, MinutesObj } from '../minutes';
import { Organization, OrganizationVerbose } from '../organization';
import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-org-page',
  templateUrl: './org-page.component.html',
  styleUrls: ['./org-page.component.scss']
})
export class OrgPageComponent implements OnInit {

  constructor(private iotaAPI: UserService, private route: ActivatedRoute,
private iotaAuth: AuthService) {

   }

  org: OrganizationVerbose;
  regCode: string;


  ngOnInit(): void {
    // Set OID from route parameter
    const oid = this.route.snapshot.paramMap.get('uid');

    // retrieve regcode and verbose org doc if oid is provided.
    this.getOrgDetails(oid || "");
    this.getRegCode(oid || "");

  }

  // Subscribes to verbose organization document using UserService
  getOrgDetails(oid: string): void {
    this.iotaAPI.getOrgSpec(oid).subscribe(doc => this.org = doc);
  }

  // Returns human readable date.
  getDate(): string {
    const datePipe = new DatePipe('en-US');
    const date = datePipe.transform(this.org.created, 'MM/dd/yy');
    return date || "";
  }

  // Returns organization ID, and "" if there is an error.
  getOID(): string {
    return this.route.snapshot.paramMap.get('uid') || "";
  }

  // Returns organization Name, and "Unnamed Org" if there is an error.
  getName(): string {
    return this.org.Name || "Unnamed Org";
  }

  // User UserService to retrieve most recent Registration Code
  getRegCode(oid: string): void {
    this.iotaAPI.getRegCode(oid).subscribe(res => this.regCode = res);
  }
}
```

### org-panel

This panel retrieves a list of organizations a user is subscribed to. It is used in the dashboard component. It receives an array of Organization type objects and displays them according to its template.

*org-panel.component.html*

```
<ul class="org-panel list-group">
```

```
    <li class="list-group-item active">Registered Organizations:</li>
    <!-- for each org in the org array display the following <a> element -->
    <a *ngFor="let org of orgs" class="list-group-item d-flex justify-content-
between align-items-center"
    routerLink="/org/{{org.oid || org._id}}">
        {{ org.Name }}
        <!-- Disabled until implemented for Admin level users only. -->
        <!-- <span class="badge bg-primary rounded-pill">{{ org._id }}</span> -->
        <!-- <span class="badge bg-primary rounded-pill">{{ org.oid }}</span> -->
    </a>
</ul>
```

*org-panel.component.ts*

```
import { Component, Input, OnInit } from '@angular/core';
import { Organization } from '../organization';

import { UserService } from '../user.service';

@Component({
  selector: 'app-org-panel',
  templateUrl: './org-panel.component.html',
  styleUrls: ['./org-panel.component.sass']
})
export class OrgPanelComponent implements OnInit {

  // Provide input field for organizations array
  @Input() orgs: Organization[];

  constructor(private iotaApi: UserService) { }

  ngOnInit(): void {

  }
}
```

### register-form

The register-form component is used to register new users. It is used in the *regpage* component. This component is not accessible on its own as *regpage* is its wrapper.

*register-form.component.html*

```
<div class="card">
    <div class="card-body">
        <form (ngSubmit)="onSubmit()" #createUser="ngForm">
```

```
            <h5 class="card-title">Register</h5>
            <!-- UserName - Is bound to the uName parameter of the model-->
            <div class="mb-3">
                <label for="new-user-uName" class="form-label">Username:</label>
                <input type="text" id="new-user-uName" class="form-control"
required
                [(ngModel)]="model.uName" name="uName"
                #name="ngModel">
                <div id="userNameHelp" class="form-text">Must be unique. Will be
used to sign-in.</div>
            </div>

            <!-- First Name - Bound to the fName parameter of the model-->
            <div class="mb-3">
                <label for="new-user-fName" class="form-label">First
Name:</label>
                <input type="text" id="new-user-fName" class="form-control"
required
                [(ngModel)]="model.fName" name="fName">
            </div>

            <!-- Last Name - Bound to the lName parameter of the model-->
            <div class="mb-3">
                <label for="new-user-lName" class="form-label">Last Name:</label>
                <input type="text" id="new-user-lName" class="form-control"
required
                [(ngModel)]="model.lName" name="lName">
            </div>

            <!-- Password - Bound to the pWord parameter of the model-->
            <div class="mb-3">
                <label for="new-user-pWord" class="form-label">Password:</label>
                <input type="text" id="new-user-pWord" class="form-control"
required
                [(ngModel)]="model.pWord" name="pWord">
            </div>

            <!-- Submit - Uses newUser() to submit using AuthService. Routes new
user to their dashboard.-->
            <button type="submit" class="btn btn-primary"
            [disabled]="!createUser.form.valid"
            (click)="newUser();
            createUser.reset()"
            routerLink='/dash/'>Register</button>
        </form>
```

```
    </div>
</div>
```

*register-form.component.ts*

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from '../auth.service';

import { NewUser } from '../user';

@Component({
  selector: 'app-register-form',
  templateUrl: './register-form.component.html',
  styleUrls: ['./register-form.component.sass']
})
export class RegisterFormComponent implements OnInit {

  constructor(private iotaAuthAPI: AuthService) {}

  // Create new model to bind to form
  model = new NewUser("0", "", "", "", "")

  ngOnInit(): void {
  }

  onSubmit() {
  }

  // submit filled model to AuthService for API post.
  newUser() {
    this.iotaAuthAPI.register(this.model);
  }

}
```

### regpage

The regpage component is a wrapper for the register-form component.

*regpage.component.html*

```
<div class="container-fluid">
    <div class="row text-center">
        <h1 class="display-1"> Iota </h1>
    </div>
    <div class="row">
```

```
        <div class="col-md-4 offset-md-4">
            <!-- Load register-form component -->
            <app-register-form></app-register-form>
        </div>
    </div>
</div>
```

*regpage.component.ts*

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-regpage',
  templateUrl: './regpage.component.html',
  styleUrls: ['./regpage.component.sass']
})
export class RegpageComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

### user-panel

Displays a panel that lists all users in a provided list. It indicates which user on the list is the current user.

*user-panel.component.html*

```
<ul class="user-panel list-group">
    <li class="list-group-item active">Members:</li>
    <!-- for each user in users[] display the template element <a> -->
    <a *ngFor="let user of users" class="list-group-item d-flex justify-content-
between align-items-center"
    routerLink="/usr/0/{{user._id}}">
        {{user.lName}}, {{user.fName}}
        <!-- use isMe to determine if uid of list item matches current user. If
so, display "Me" indicator-->
        <span class="badge bg-warning rounded-pill" *ngIf="isMe(user)"> Me
</span>
    </a>
</ul>
```

*user-panel.component.ts*

```
import { Component, Input, OnInit } from '@angular/core';
import { AuthService } from '../auth.service';
import { USERS } from "../mock-users";
import { User } from '../user';

import { UserService } from '../user.service';

@Component({
  selector: 'app-user-panel',
  templateUrl: './user-panel.component.html',
  styleUrls: ['./user-panel.component.sass']
})
export class UserPanelComponent implements OnInit {

  // Use input array of User type objects.
  @Input() users: User[];

  constructor(private userService: UserService, private iotaAuth: AuthService) {
  }

  ngOnInit(): void {
    // this.getUsers();
  }

  // use UserService to determine if a listed user is the current user.
  isMe(user: User): boolean {
    return this.userService.isMe(user);
  }

}
```

## userbio

This component is the viewer and editor for user bios. Depending on the mode it displays either the viewer or the editor. In editor mode it provides a from and preloads the values to match the current user.

*userbio.component.html*

```
<div class="container">
    <!-- View Mode - Displays when Mode = 0 -->
    <div *ngIf="getMode() === 0">
        <h1 class="card-title">{{ user.fName }} {{ user.lName }}</h1>
        <h3> Iota user since: {{ getDate() }}</h3>
        <h5>About:</h5>
        <p> {{ user.bio || "No bio yet" }} </p>
    </div>

    <!-- Edit Mode - Displays when Mode = 1 -->
    <div *ngIf="getMode() === 1">
        <form #bioUpdate="ngForm">
            <h1>User Info</h1>
            <div class="row g-2">
                <!-- First Name - Bound to user.fName -->
                <div class="col-md-4">
                    <label for="input-fName" class="form-label">First 
Name</label>
                    <input type="text" class="form-control" id="input-fName"
                    placeholder="John" value="{{ user.fName }}"
                    [(ngModel)]="user.fName" name="fName" #fName="ngModel">
                </div>
                <!-- Last Name - Bound to user.lName -->
                <div class="col-md-4">
                    <label for="input-lName" class="form-label">Last Name</label>
                    <input type="text" class="form-control" id="input-lName"
                    placeholder="Cena" value="{{ user.lName }}"
                    [(ngModel)]="user.lName" name="lName" #lName="ngModel">
                </div>
                <!-- Bio - Bound to user.Bio -->
                <div class="col-md-8">
                    <label class="form-label">About Me</label>
                    <textarea type="text" class="form-control" placeholder="Talk 
about yourself a little..." required
                    id="docContents" rows="5"
                    [(ngModel)]="user.bio" name="bio" #bio="ngModel"
                    ></textarea>
                </div>
                <!-- Submit - uses setUserInfo() to submit via UserService. 
Returns user to Dash -->
                <div class="col-12">
                    <button type="submit"
                            class="btn btn-success"
                            [disabled]="!bioUpdate.form.valid"
                            (click)="setUserInfo();"
                            routerLink="/dash"> Update
                    </button>
                </div>
            </div>
        </form>
    </div>
```

```
</div>
```

*userbio.component.ts*

```
import { DatePipe } from '@angular/common';
import { Component, Input, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { VerboseUser } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-userbio',
  templateUrl: './userbio.component.html',
  styleUrls: ['./userbio.component.sass']
})

export class UserbioComponent implements OnInit {

  // Allow loading component to provide user template
  @Input() user: VerboseUser;

  private mode: number = 0;

  constructor(
    private iotaUser: UserService,
    private route: ActivatedRoute) {
  }

  ngOnInit(): void {
    // set mode and UID to match the route parameters
    this.mode = Number(this.route.snapshot.paramMap.get('mode'));
    const uid = this.route.snapshot.paramMap.get('uid');

    // retrieve userinfo for given UID
    this.getUserInfo(uid || "");
  }

  // Use UserService to retrieve User information.
  getUserInfo(uid: string): void {
    this.iotaUser.getUserVerb(uid).subscribe(doc => this.user = doc);
  }

  // Return human-readable date string.
  getDate(): string {
    const datePipe = new DatePipe('en-US');
    const date = datePipe.transform(this.user.joindate, 'MM/dd/yy');
    return date || "";
  }

  // Returns mode
  getMode(): number {
    return this.mode;
  }

  // Use UserService to submit changes to API
  setUserInfo(): void {
    this.iotaUser.setUserInfo(this.user);
  }

}
```

### welcome-page

This component contains the primary landing page for the site. When navigating to the root of the server, the router loads this component.

*welcome-page.component.html*

```
<div class="container-fluid">
    <div class="row text-center">
        <div class="col-12">
            <h1 class="display-1">Iota</h1>
            <h5 class="display-5">Simplifying Student Organizations</h5>
        </div>
        <div class="col-12">
            <h5>Why Iota?</h5>
            <p>The disorganization of student organizations is the result of the 
lack of tools.<br>
            This project was an opportunity for me to explore the standards of 
modern web development, <br>
            as well as develop new tools for the organizations I love.<br>
            </p>
            <h5>What is Iota?</h5>
            <p> Iota is an asynchronous, stateless, Progressive Web App (PWA).
                <br>
                It is built with Angular for the Front-End, and a custom API 
Server running in Node.JS on the Back-End.
                <br>
                It employs modern development standards seen in environments such 
as Facebook, Amazon, Netflix, etc.
```

```
            </p>
            <h5>What next?</h5>
            <p> The goal is to grow Iota over the next few years after
graduation. <br>
            I firmly believe that continued development will make this an
essential tool for all students/faculty.<br>
            Not just those involved in greek life.</p>
        </div>
        <div class="col-12">
            <h5>~</h5>
            <p>Created By Cole Cassidy - 2022
                <br>
                <br>
                Special Thanks to:
                <br>
                Dr. Zizhong John Wang - Virginia Wesleyan University - Stage
Right Lighting
                <br>
                <br>
                Technology Credits:
                <br>
                Angular - Bootstrap - Express.js - Node.js - MongoDB
            </p>
        </div>
    </div>
</div>
```

*welcome-page.component.ts*

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-welcome-page',
  templateUrl: './welcome-page.component.html',
  styleUrls: ['./welcome-page.component.scss']
})
export class WelcomePageComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

## Supporting Files

Other than components. Angular requires several other supporting files. The following files exist in the root directory of the project and are not loaded/compiled in the same way as components.

### app-routing.module.ts

This Angular module defines the URL routes used by the application. It specifies which paths lead to which components, as well as which components should be used for redirects.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserPanelComponent } from './user-panel/user-panel.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { AuthpageComponent } from './authpage/authpage.component';
import { RegpageComponent } from './regpage/regpage.component';
import { OrgPageComponent } from './org-page/org-page.component';
import { UserbioComponent } from './userbio/userbio.component';
import { MinuteEditorComponent } from './minute-editor/minute-editor.component';
import { AddOrgPageComponent } from './add-org-page/add-org-page.component';
import { WelcomePageComponent } from './welcome-page/welcome-page.component';

// Define client-facing routes.
// Example iotaengine.org/login
const routes: Routes = [
  { path: 'users', component: UserPanelComponent},
  { path: 'org/:uid', component: OrgPageComponent},
  { path: 'dash', component: DashboardComponent},
  { path: 'login', component: AuthpageComponent},
  { path: 'register', component: RegpageComponent},
  { path: 'usr/:mode/:uid', component: UserbioComponent},
  { path: 'min/:mode/:id', component: MinuteEditorComponent},
  { path: 'register/join', component: AddOrgPageComponent},
  { path: '', component: WelcomePageComponent},
  { path: '**', redirectTo: ''}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

## app.component.html

This serves as the root html document. When a page is loaded, this page is loaded first, and the router defines which components should be appended to this document using the URL.

```
<!-- load navbar  -->
<app-navbar></app-navbar>
<!-- load component defined by router -->
<router-outlet></router-outlet>
```

## app.component.ts

This is the root typescript file for the application. Any application wide logic would go here. However it is best practice to leave the logic in either services or their respective components.

```
import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.sass']
})
export class AppComponent {

  constructor() {
  }
}
```

## auth.service.ts

This handles tokens on the client side, and any authentication related communication with the API server. It maintains login state, validates and reads the local token, and can transmit data via HTTP(s).

```
import { EventEmitter, Injectable } from '@angular/core';

import { BehaviorSubject, Observable } from 'rxjs';

import { HttpClient } from '@angular/common/http';

import { JwtHelperService } from '@auth0/angular-jwt';

import { AuthUser, NewUser, User } from './user';

import { environment } from 'src/environments/environment';
import { Router } from '@angular/router';


@Injectable({
  providedIn: 'root'
})
export class AuthService {

  // tracks user login state
  isLoggedIn: boolean;

  // swap variable for current user document
  private currentUser = {};

  // imports the Auth0/JWTHelper library
  private jwtHelper = new JwtHelperService();

  // Async observable field exports
  public loginStateObs = new BehaviorSubject<boolean>(this.checkLoginState());
  public userDoc = new BehaviorSubject<User>(this.readUserDoc());



  constructor(private http: HttpClient,) {
  }

  // Handles authentication given an AuthUser type object.
  login(user: AuthUser) {

    // Make post request to /auth/login with authentication data.
    // returns a success state if the post request and its callback function are
successful.
    return this.http.post(environment.apiURL+'/auth/login', user).subscribe((res:
any) => {

      // if there is a token error run logout to ensure the application state is
clear.
      if(res.token.Error){this.logout(); return;}

      // log the usertoken for debug usage.
      console.log(this.jwtHelper.decodeToken(res.token));

      // store the login state to the browsers local storage
      localStorage.setItem('logged-in', "1");
```

```
   // store the received token to the browsers local storage
   localStorage.setItem('token',res.token);

   // advance the userDoc and loginState observables.
   this.loginStateObs.next(true);
   this.userDoc.next(this.readUserDoc());
  });
 }

 // Resets login state and clears token from local storage.
 logout(){
  // Set login state to 0
  localStorage.setItem('logged-in', "0");

  // remove token from localstorage if it exists.
  if(localStorage.getItem('token') !== null) {localStorage.removeItem('token')}

  // advance userDoc and loginState observables.
  this.loginStateObs.next(false);
  this.userDoc.next(this.readUserDoc());
 }

 // Submits a new user registration and logs in
 register(user: NewUser){

  // Make POST request containing new user data.
  // returns success state if post request and callback are successful.
  return this.http.post(environment.apiURL+'/auth/reg',user).subscribe((res:
any) => {

   // set login state to 1 and add received token to local storage.
   localStorage.setItem('logged-in', "1");
   localStorage.setItem('token',res.token);

   // advance loginState and userDoc observables
   this.loginStateObs.next(true);
   this.userDoc.next(this.readUserDoc());
  });
 }


 // Retrieves new token from server.
 // Necessary when there are updates to the users groups or bio.
 refresh() {
    //Make get request for new token.
    return this.http.get(environment.apiURL+'/auth/update').subscribe((res: any)
=> {

      // ensure the login state and token are up to date.
      localStorage.setItem('logged-in', "1");
      localStorage.setItem('token',res);

      // advance loginState and userDoc observables.
      this.loginStateObs.next(true);
      this.userDoc.next(this.readUserDoc());
    });
  }

  // converts local storage integer representation of login state to a boolean
type.
  checkLoginState(): boolean{

    //TODO: Set login state using user doc presence
    var chk = localStorage.getItem('logged-in');
    if(chk == "1"){
      return true;
    } else {
      return false;
    }

  }

  // decodes token from localstorage and returns a User type object.
  readUserDoc(): User{
    try {
      // If no token return empty User object.
      if(localStorage.getItem('token') === null ){return new
User("","","","",0,[]);}

      // Otherwise return decoded User object.
      return this.jwtHelper.decodeToken<User>(localStorage.getItem('token') ||
'{}');
    } catch (error) {

      // If there is a token error log error and return an empty User type
object.
      console.log(error);
      console.log(localStorage.getItem('token'));
      return new User("","","","",0,[]);
    }
  }

  // For future use. returns boolean reflecting users admin status.
  checkAdmin(): boolean{
    var num;
    this.userInfo.subscribe(doc => num = doc.rank);
    return num === 3;
  }

  // GET's
  // in typescript, these are functions that easily return variables in a
protected way.
  // in angular they are needed to make observables more reliable.
  get loginStateCheck(){
    return this.loginStateObs.asObservable();
  }

  get userInfo(){
    return this.userDoc.asObservable();
  }


}
```

## token-interceptor.interceptor.ts

Interceptors are another unique component in Angular. A defined interceptor will execute its "intercept()" function whenever the defined functions get executed. Iota uses the interceptor object to ensure that the token is defined in the request header on any API calls.

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';

@Injectable()
export class TokenInterceptorInterceptor implements HttpInterceptor {

  constructor(private iotaAuth: AuthService) {}
  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    const authToken = localStorage.getItem('token');
    // Check if token is present
    if (authToken) {
      // Append the token to the outgoing request.
      request = request.clone({
        setHeaders: {
          'x-access-token': authToken
        }
      });
    }
    return next.handle(request);
  }
}
```

### user.service.ts

The UserService handles all communication with the API server outside of authentication. It submits and requests user information, minutes/reports documents, and organization information. It also provides several utility functions used throughout the codebase.

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable, of } from 'rxjs';
import { User, VerboseUser} from './user';
import { Organization, OrganizationVerbose } from './organization';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { environment } from 'src/environments/environment';
import { AuthService } from './auth.service';
import { Minutes } from './minutes';



@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(
    private http: HttpClient,
    private iotaAuth: AuthService
  ) { }
  // Compile time feature. Sets url string to match a definied environment
variable.
  // localhost for local development
  // iotaEngine AWS IP for production.
  private url = environment.apiURL;

  // ===== User Functions =====

  // makes API call to retrieve user list
  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(this.url+'/usr/list');
  }

  // makes API call to add a new user.
  // Depricated, used during testing. Included here for demonstration.
  addUser(user: User): Observable<User> {
    return this.http.post<User>(this.url+'/usr/add', user);
  }

  // Requests verbose user information from API server
  getUserVerb(uid: string): Observable<VerboseUser> {
    return this.http.get<VerboseUser>(this.url+'/usr/s/'+uid);
  }

  // Sets verbose user infromation on API server
  setUserInfo(user: VerboseUser): void {
    this.http.post(this.url+'/usr/edit/'+user._id, user).subscribe((res: any) =>
{
      console.log(res);
    })
  }

  // ===== Organization Functions =====

  // retrieves array of Organizations from API server
  getOrgs(): Observable<Organization[]> {
    return this.http.get<Organization[]>(this.url+'/org/list');
  }

  // retrieves verbose information for a specific organization from API server
  getOrgSpec(oid: string): Observable<OrganizationVerbose> {
    return this.http.get<OrganizationVerbose>(this.url+'/org/s/'+oid+'/full');
  }

  // retrieves most recent registration code from API server
  getRegCode(oid: string): Observable<string> {
    return this.http.get<string>(this.url+'/org/reg/'+oid);
  }

  // POSTs registration code to API server to join organization
  joinOrg(regcode: string): void {
    this.http.post(this.url+'/org/reg/'+regcode, {}).subscribe((res: any) => {
      console.log(res);
      this.iotaAuth.refresh();
    })
  }

  // ===== Minutes Functions =====

  // retrieves list of minutes by organization ID from API
  getMinutes(oid: string): Observable<Minutes[]> {
    console.log(oid);
    return this.http.get<Minutes[]>(this.url+'/min/org/'+oid);
  }

  // retrieves specific minutes from API using minutes document ID
  getMinutesSpec(id: string): Observable<Minutes> {
    console.log(this.url+'/min/s/'+id);
    return this.http.get<Minutes>(this.url+'/min/s/'+id);
  }

  // POSTs new minutes to API server
  putMinutes(doc: Minutes): void {
    this.http.post(this.url+'/min/new',doc).subscribe((res: any) => {
      console.log(res);
    })
  }

  // ===== Internal Functions =====
  // validates that current user matches provided User type object.
  isMe(user: User): boolean {
    if(user._id === this.iotaAuth.readUserDoc()._id) return true;
    return false;
  }

}
```

## index.html

This is the true root of the application. When compiled. Everything generated by the Angular compiler is executed from the <app-root> section of this html file. This is where the imports for Bootstrap are placed, and where any metadata for the site is defined.

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>IotaAngApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <!-- Load bootstrap stylesheet from CDN -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
</head>
<body>
  <!-- Load IotaAPP -->
  <app-root></app-root>
<!-- Load bootstrap JS from CDN -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js
" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>
</body>
</html>
```

## /environments

This is another Angular specific folder. It contains the definitions for the development and compilation environment. When running the test server environment.ts is used. When building the application for release environment.prod.ts is used. These do not execute, or get bundled in the final compilation. They simply define global variables that should be set depending on the compile setting.

### *environment.ts*

```typescript
export const environment = {
  production: false,
  apiURL: 'http://localhost:3000'
};
```

*environment.prod.ts*

```
export const environment = {
  production: true,
  apiURL: 'http://54.172.35.178:3000'
};
```

## Type Definitions

A core element of Angular is typescript. While typescript is similar to JavaScript, it requires type definitions for any new objects generated by the developer. This ensures that data is carried over accurately between components .

### user.ts

```
import { Organization } from "./organization";
// Defines various user type formats used throughout the application
export interface User {
    _id: string;
    uName: string;
    fName: string;
    lName: string;
    rank: number;
    orgs: Organization[];
}

export interface NewUser {
    _id: string,
    uName: string,
    fName: string,
    lName: string,
    pWord: string
}

export interface AuthUser {
    uName: string,
    pWord: string
}

export interface VerboseUser {
    _id: string;
    uName: string;
    fName: string;
    lName: string;
    rank: number;
    orgs: Organization[];
    bio: string;
    joindate: string;
}

// defines user clases for easily generating new users of specific types.
export class User {
    constructor(
        public _id: string,
        public uName: string,
        public fName: string,
        public lName: string,
        public rank: number,
        public orgs: Organization[]
    ) { }
}

export class NewUser {
    constructor(
        public _id: string,
        public uName: string,
        public fName: string,
        public lName: string,
        public pWord: string
    ) { }
}

export class AuthUser {
    constructor(
        public uName: string,
        public pWord: string
    ) { }
}
```

minutes.ts

```
import { User } from "./user";

// Defintes minutes object type interface and class constructor.
export interface Minutes {
    _id: string;
    createDate: string;
    modDate: string;
    oid: string;
    creator: User;
    title: string;
    contents: string;
}

export class MinutesObj {
    constructor(
        public _id: string,
        public createDate: string,
        public modDate: string,
        public oid: string,
        public creator: User,
        public title: string,
        public contents: string
    ){ }
}
```

organizations.ts

```
import { User } from "./user";

// Defines organization and organization verbose interfaces.
export interface Organization{
    _id: number;
    Name: string;
    oid: string;
}

export interface OrganizationVerbose extends Organization {
    _id: number;
    Name: string;
    byline: string;
    created: string;
    members: User[];
}
```

## Summary

The code provided in this document was written by the Author unless otherwise stated. Files generated by the development environment, or by any open source libraries were not included in this codebook. Both for reasons of brevity and academic integrity. The entire codebase for each application is downloadable from the student website:

https://zeus.vwu.edu/~cwcassidy/Thesis/thesis.php