MY TRAVEL

PROJECT FOR MOBILE APPLICATIONS AND CLOUD COMPUTING

Francesco D'Abbraccio 1657674
Noura Boubou 1823776

MAIN FEATURES

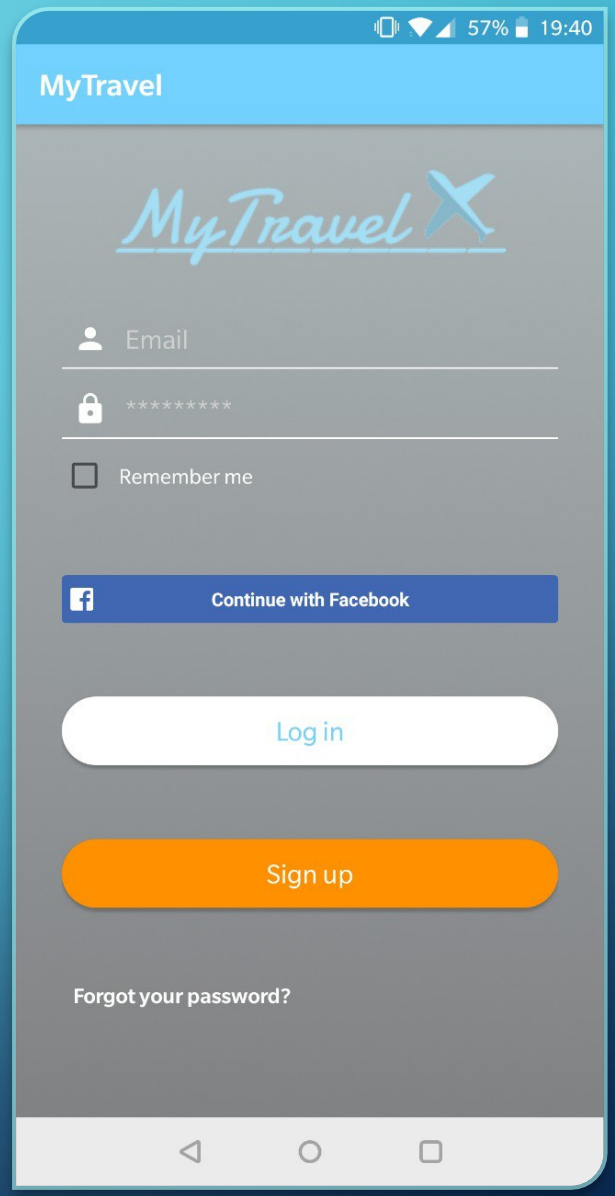
The app is designed to easily manage your trips. No matter if you travel alone or with your friends: this is the right choice for you!

Give a name to your trip and add your destinations. In each destination page you'll find useful information for you.

You can also manage your common payments. The app will dynamically calculate the balance for each participant.

HOME PAGE

- In the home page you'll find all the links you need to start interacting with the app

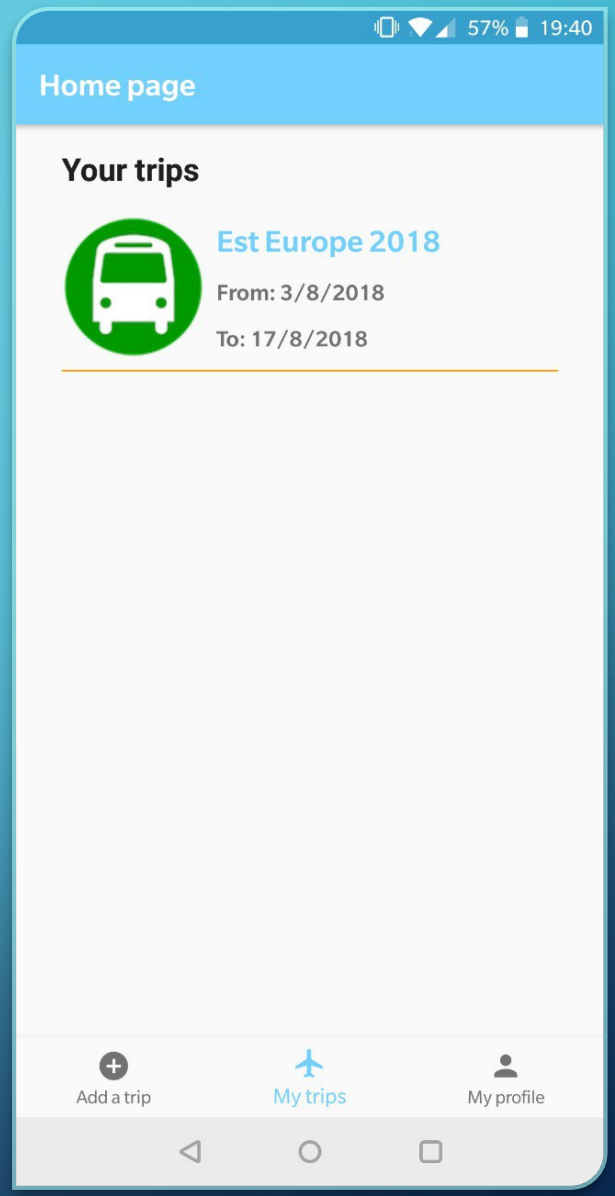


ADD A TRIP

- There are 3 pages. The default screen is where you can add a new trip to your list

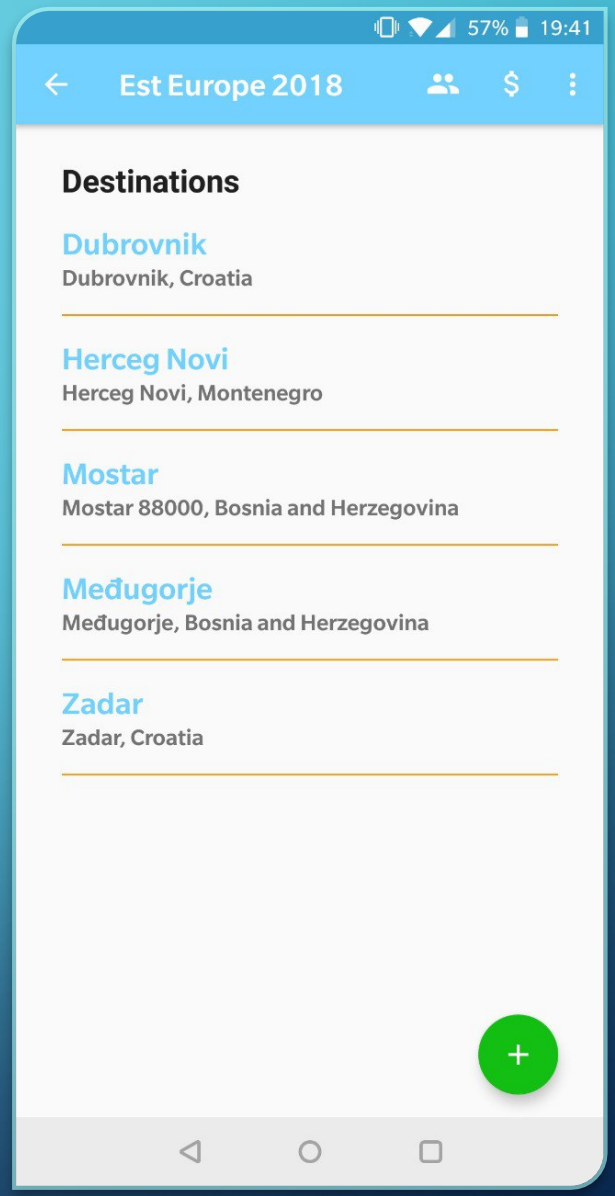
MY TRIPS

- Here you can find the list of your trips, displayed by a Recycler View



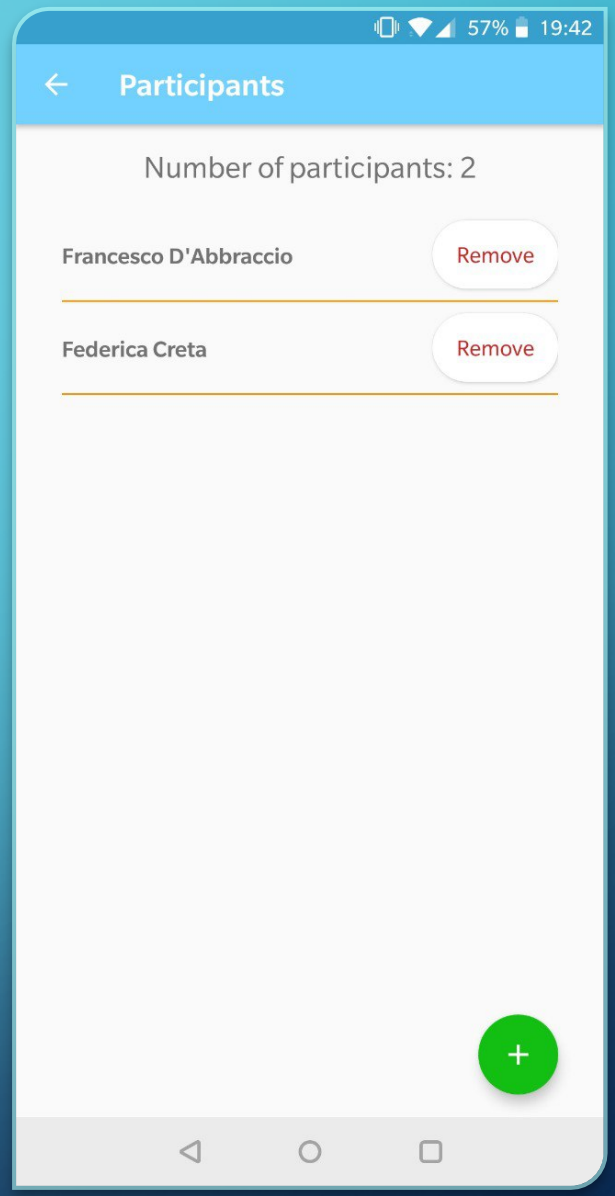
TRIP PAGE

- In the trip page you'll see the list of your destinations and three buttons on the app bar: these are for the participant page, payment page and a menu



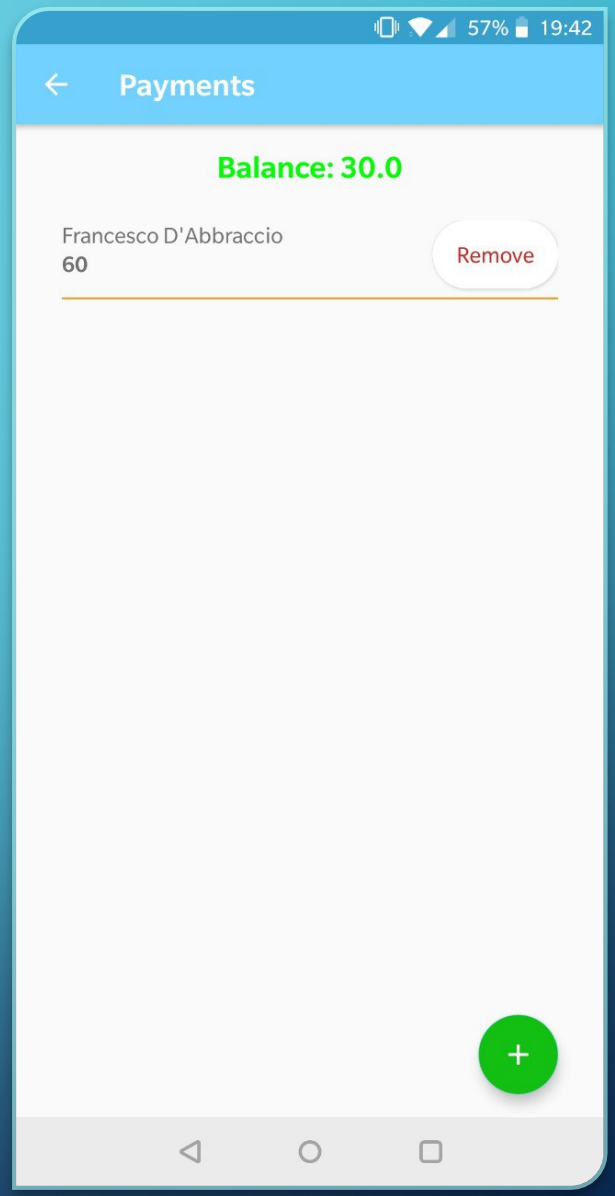
PARTICIPANT PAGE

- This is the page where you can manage the participants to the trip



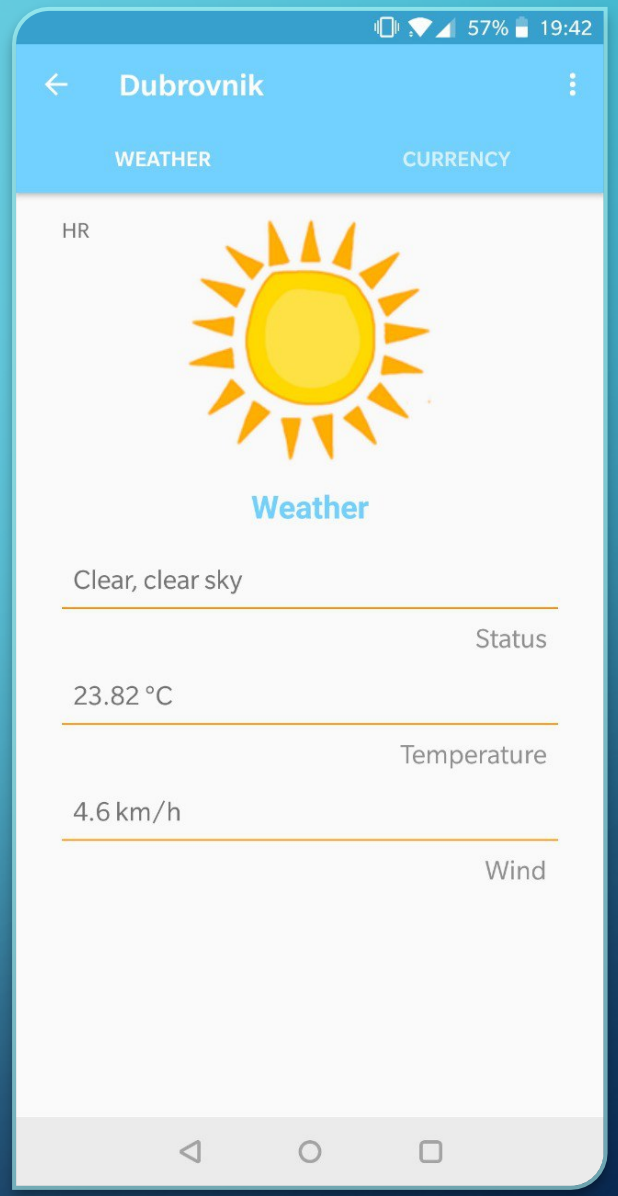
PAYMENTS PAGE

- As well as the participants, you can manage all the payments



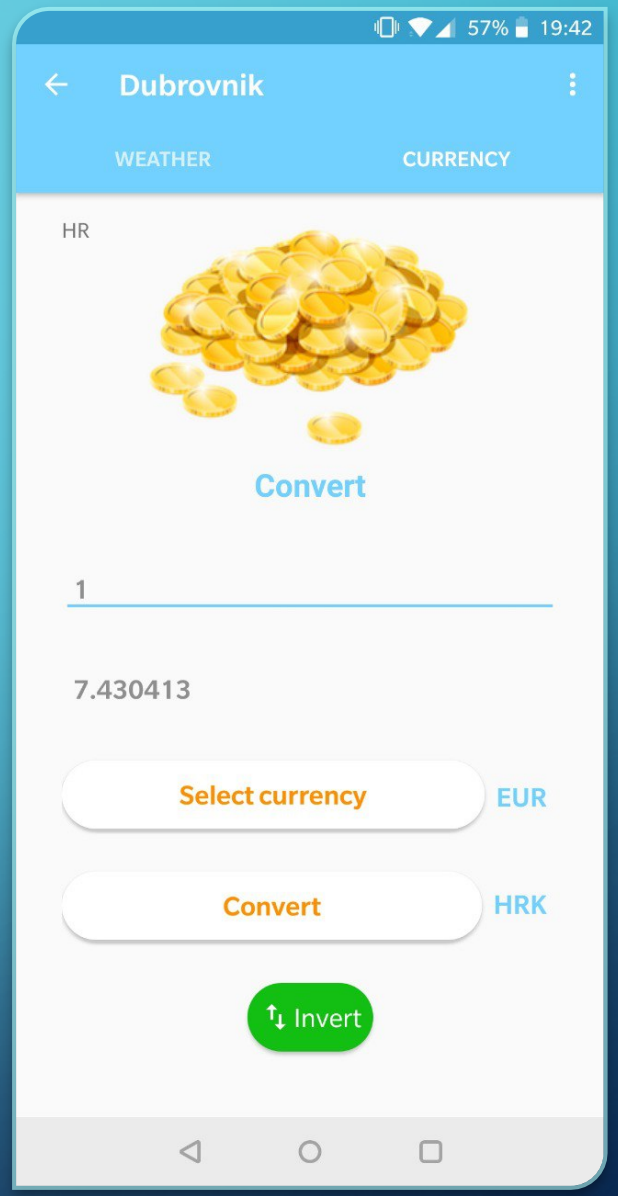
DESTINATION PAGE

- In this page you can find information about destination's weather



CURRENCY PAGE

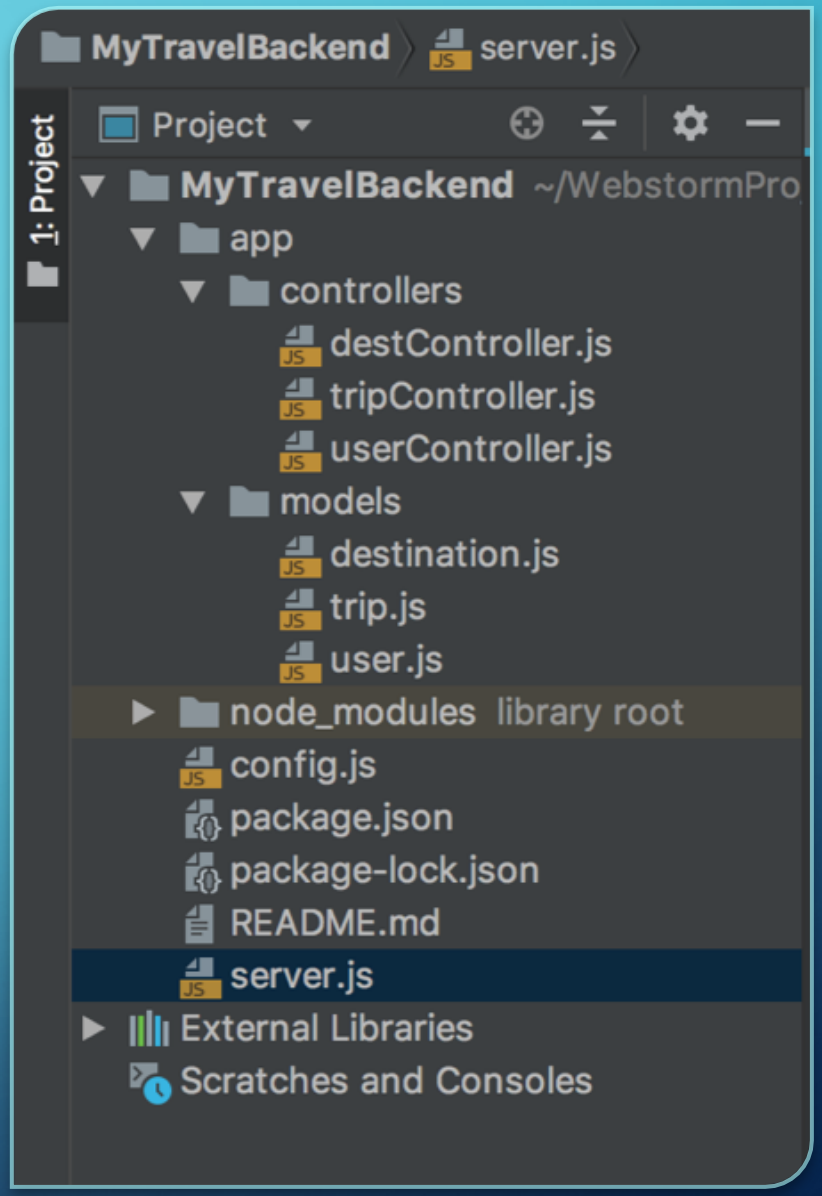
- Here you can find information about the currency used in the city and also a conversion tool



BACKEND

MAIN STRUCTURE

- This is the structure of the backend. Following the MVC pattern, here we have Models and Controllers, while the View is fully implemented on Android.



ADDITIONAL PLUGINS

- In order to build a complete application, we need some external libraries, like Express and Mongoose.

server.js

```
// ============================
// get the packages we need ====
// ============================
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const morgan = require('morgan');

const jwt = require('jsonwebtoken'); // used to create, sign, and verify tokens
const config = require('./config'); // get our config file
const mongoose = require('mongoose');
const request = require('request');

// Require controller modules.
const user_controller = require('./app/controllers/userController');
const trip_controller = require('./app/controllers/tripController');
const dest_controller = require('./app/controllers/destController');

// =====================
// configuration =========
// =====================
const port = process.env.PORT || 8080; // used to create, sign, and verify tokens

mongoose.connect(config.databasecloud, {useNewUrlParser: true}); // connect to database
```

MODEL STRUCTURE

- The database used for the project is MongoDB, a NoSQL database. We don't interact directly with it but through mongoose, a package for Node JS

```
// get an instance of mongoose and mongoose.Schema
const mongoose = require('mongoose');
const uniqueValidator = require('mongoose-unique-validator');

const Schema = mongoose.Schema;

mongoose.set('useCreateIndex', true);

const tripSchema = new Schema({
    name: {type: String, required: true},
    startDate: {type: String, required: true},
    endDate: {type: String, required: true},
    creator: {type: String, required: true},
    participants: [String],
    payments: [{username: {type: String}, amount: {type: String}}]
});

tripSchema.plugin(uniqueValidator);

// set up a mongoose model and pass it using module.exports
module.exports = mongoose.model('Trip', tripSchema);
```

TOKEN AUTHENTICATION

- When we login in the app, we receive as response a token that expires in 24 hours. Every request must come with this token in order to check if we are able to access the content we want.

```
// route to authenticate a user (POST http://localhost:8080/app/authenticate)
exports.authenticate = function (req, res) {

    // find the user
    User.findOne({
        email: req.query.email
    }, function (err, user) {

        if (err) throw err;

        if (!user) {
            res.json({success: false, message: 'Authentication failed. User not found.'});
        } else if (user) {

            // check if password matches
            if (user.password !== req.query.password) {
                res.json({success: false, message: 'Authentication failed. Wrong password.'});
            } else {

                // if user is found and password is right
                // create a token with only our given payload
                // we don't want to pass in the entire user since that has the password
                const payload = {
                    id: user._id,
                    username: user.username
                    //admin: user.admin
                };

                const token = jwt.sign(payload, app.get('superSecret'), {
                    expiresIn: 1440 // expires in 24 hours
                });

                // return the information including token as JSON
                res.json({
                    success: true,
                    message: 'Enjoy your token!',
                    token: token
                });
            }
```

MIDDLEWARE FUNCTION

- In order to authenticate the token, we define this function. Each route written after this middleware will execute this code first, for authentication

```
// middleware for token
apiRoutes.use(function (req, res, next) {

    // check header or url parameters or post parameters for token
    const token = req.body.token || req.query.token || req.headers['x-access-token'];

    // decode token
    if (token) {

        // verifies secret and checks exp
        jwt.verify(token, app.get('superSecret'), function (err, decoded) {
            if (err) {
                return res.json({success: false, message: 'Failed to authenticate token.'});
            } else {
                // if everything is good, save to request for use in other routes
                req.decoded = decoded;
                next();
            }
        });

    } else {

        // if there is no token
        // return an error
        return res.status(403).send({
            success: false,
            message: 'No token provided.'
        });

    }
});
```

FACEBOOK AUTHENTICATION

- Using the Facebook authentication token we get from Android, we can access the Facebook API and get user's name and email to create an account

```
// route for facebook login (POST http://localhost:8080/app/facebook_login)
exports.facebook_authenticate = function (req, res) {
    const facebook_token = req.query.facebook_token;

    const options = {
        timeout: 3000,
        pool: {maxSockets: Infinity},
        headers: {connection: "keep-alive"}
    };

    graph
        .setAccessToken(facebook_token)
        .setOptions(options)
        .get('me?fields=name, email', function (err, fbres) {
            if (err) {
                fbres.json({success: false, message: err.message});
            }

            const name = fbres['name'];
            const email = fbres['email'];
```

DEFINING ROUTES

- With Express, we are able to define the routes for our RESTful interface, using HTTP verbs

```
/// USER CONTROLLER


// route to create an account (POST http://localhost:8080/app/register)
apiRoutes.post('/register', user_controller.register);

// route to authenticate a user (POST http://localhost:8080/app/authenticate)
apiRoutes.post('/authenticate', user_controller.authenticate);

// route for facebook login (POST http://localhost:8080/app/facebook_login)
apiRoutes.post('/facebook_login', user_controller.facebook_authenticate);

// route to retrieve forgotten password (GET http://localhost:8080/app/forgot_password)
apiRoutes.get('/forgot_password', user_controller.forgot_password);
```