Setup

Installation

Run the following command from root folder of the cloned project to install all dependencies.

`npm install`

Verify Setup

In order to verify that everything is setup correctly, run the following command, which should show you the failing tests. This is good! We'll be fixing these tests once we jump into the build step.

`npm run test`

Every time you want to check your work locally you can type that command, and it will report the status of every task in that module.
As you move through the modules, you can run module-specific tests with the script `npm run test:module1`, replacing the number with one that corresponds with the module you are working in.

Previewing Your Work

In order to see your changes in a browser, you can run `npm start` to start the application, and when you visit `http://localhost:3000` in a browser, you should see your components rendered.

Module 01 - Use Express to Create a Server

1.1 - Require Built-in Libraries

@app-require-built-ins In `app.js` require the built-in library `fs` and store a reference to it in a `const` called `fs`. Next, require the built-in library `path` and store a reference to it in a `const` called `path`.

1.2 - Require the Express Framework

@app-require-express-const-app In `app.js`, require the express framework and store a reference to it in a `const` called `express`. Next, call the express function and store it in a `const` called `app`.

1.3 - Configure the View Directory and Engine

@app-set-views-directory-engine Still in app.js, use the `set` function of your newly created `app` to configure the directory where our `views` can be found. Using the same `set` function, set the `view engine` to `ejs`. **Hint: `path.join()` & `__dirname`**

1.4 - Configure the Static Directory

@app-use-express-static All of our CSS/JS for the client-side is found in the `public` directory. We need to point express to `public`.

- In app.js call the `use` function of `app` with a call to the `express.static()` function as the only parameter.
- `express.static()` should be passed the full path to the `public` directory. **Hint: `path.join()` & `__dirname`**

1.5 - Create the Index View File

@index-ejs-create-view-file Create a new file called `index.ejs` in the `src/views/` directory.

1.6 - Create the Index View

@index-ejs-create-view In the newly created file `index.ejs` complete the following:

- Include `header.ejs` **Hint: <%- %>**
- Add a `div` element with a class of `container`.
- In the container div display the value of the `title` key in an `h1` element. **Hint: <%= %>**
- Add an anchor element below the `h1` that points to the `/profile` URL path, and has the text content "Profile".
- Below the container div add a line break and another anchor element that points to the `/transfer` URL path with the text content `Transfer`.
- Include `footer.ejs` **Hint: <%- %>**

1.7 - Create the Index Route

@app-get-index-route In `app.js` create a `get` route that points at the root URL path '/'. Render the `index` view (created in the next step) and pass an object with a single key value pair, `title: 'Index'`.

1.8 - Start Server

@app-listen-console-log In `app.js` using the `listen` function to create a server that listens on port `3000` and then prints the message `PS Project Running on port 3000!` to the console after the server is created.

Module 02 - File Handling and Routing

2.1 - Read Account Data

@app-read-account-data In `app.js` above the index route, use the `readFileSync` function of the built-in `fs` library to read the contents of the file located at `src/json/accounts.json`. Declare a `const` called `accountData` to store the contents of the file. `accountData` now contains JSON, use `JSON.parse` to convert it to a javascript object. Declare a `const` called `accounts` to store this javascript object.

2.2 - Read User Data

@app-read-user-data In `app.js` near the index route, use the `readFileSync` function of the built-in `fs` library to read the contents of the file located at `src/json/users.json`. Declare a `const` called `userData` to store the contents of the file. `userData` now contains JSON, use `JSON.parse` to convert it to a javascript object. Declare a `const` called `users` to store this javascript object.

2.3 - Update the Index Route

@app-update-index-route In `app.js` update the object passed to the existing index route. The `title` should be "Accounts Summary". A new key value pair should be added, `accounts: accounts`.

2.4 - Update the Index View

@index-ejs-update-view In `index.ejs` and after the container div, add the ejs markup to include the `summary` view for each account in the `accounts` variable, savings, checking, and credit. **Hint: you will have three include statements(`<%- %>`), each `include` function will be passed a different account, i.e `{ account: accounts.checking }`.**

2.5 - Create the Savings Account Route

@app-get-savings-account-route In `app.js` near the index route, create a `get` route that points at the `/savings` URL path. Render the `account` view and pass an object with the following key value pair:

- `account: accounts.savings`

2.6 - Create the Checking & Credit Routes

@app-get-other-account-routes Now that you have created the savings account route, create similar routes for the checking and credit accounts in the `app.js` file.

2.7 - Show Account Transactions

@account-ejs-show-transactions In `account.ejs` after the header markup, add the ejs markup to include the `transactions` view. Pass the include function an object with the following key value pair:

- `account: account`

2.8 - Create the Profile View File

@profile-ejs-create-view-file Create a new file called `profile.ejs` in the `src/views/` directory.

2.9 - Create the Profile View

@profile-ejs-create-view In the newly created file `profile.ejs` complete the following:

- Include `header.ejs` **Hint: <%- %>**
- Add an `h1` element with the text content `Profile`
- Add a `div` element below the `h1` that displays each detail of the `user` object on a new line, name, username, phone, email, and address.
- Below the `div` add a line break, then an anchor element that points to the root URL path and has the text content `Back to Account Summary`.
- Include `footer.ejs` **Hint: <%- %>**

2.10 - Create the Profile Route

@app-get-profile-route Back In `app.js` below the account get routes create a `get` route that points at the `/profile` URL path. Render the `profile` view and pass an object with the following key value pair:

- `user: users[0]`

Module 03 - Handling Form Data

3.1 - URL Encoded Middleware

@app-urlencoded-form-data In `app.js` near your other app.use statement add express middleware to handle POST data. With the `use` function add the `express.urlencoded` middleware to `app`. Make sure to set the `extended` option to `true`.

3.2 - Create the Transfer GET Route

@app-get-transfer-route Near your other routes in `app.js` create a `get` route that points to the `/transfer` URL path. It should render the `transfer` view.

3.3 - Update the Transfer View

@transfer-ejs-update-view In `transfer.ejs` in the `src/views/` directory complete the following:

- Add the necessary attributes to the `transferForm` so that it posts to a transfer route.
- Add a `name` and `id` attribute with a value of `from` to the first select.
- Add a `name` and `id` attribute with a value of `to` to the second select.

3.4 - Create the Transfer POST Route

@app-post-transfer-route Switch back to `app.js` and create a `post` route that points to the `/transfer` URL path. We will fill in the body of the function for this route in the next few steps.

3.5 - Calculate New Balances

@app-post-transfer-route-from-balance Still in `app.js` and in the function body of the post route we are going to calculate the new balances for the account we are transferring from.

We have several values that have been entered into the HTML form in `transfer.ejs`. Upon form submission the request body will contain `from`, `to`, and `amount`. Current balances are stored in the `accounts` object in this form `accounts['savings'].balance`. Using these values calculate the new balance of the account we are transferring from.

3.6 - Calculate New Balances

@app-post-transfer-route-to-balance Still in `app.js` and in the function body of the post route we are going to calculate the new balances for the account we are transferring to.

We have several values that have been entered into the HTML form in `transfer.ejs`. Upon form submission the request body will contain `from`, `to`, and `amount`. Current balances are stored in the `accounts` object in this form `accounts['checking'].balance`. Using these values calculate the new balance of the account we are transferring from. **Hint: you will need to use `parseInt()`**

3.7 - Convert Account Data to JSON

@app-post-transfer-route-write-json Still in `app.js` and in the function body of the post route, convert the `accounts` javascript object to JSON using the stringify function some this JSON in a variable called `accountsJSON`. **Hint: set the replacer argument to `null` and the space argument to 4 of the stringify function for well formatted JSON.**

3.8 - Write Account Data to JSON file

Still in `app.js` and in the function body of the post route, use the `writeFileSync` function of the built-in `fs` library to write the variable `accountsJSON` to the file located at `src/json/accounts.json`. If at any point `accounts.json` gets overwritten copy the JSON from `account_backup.json`.

3.9 - Redirect with a Message

@app-post-transfer-route-redirect Still in `app.js` and in the function body of the post route, render the `transfer` view and pass an object with the following key value pair:

- `message: "Transfer Completed"`

3.10 - Add Payment Feature

@app-payment-feature The payment feature of the application is similar to the transfer feature. Add this new feature using this general outline of the steps:

- Near your existing routes in `app.js` create post and get routed with a URL path of `/payment`. The get route should render the `payment` view.
- In the body of the post route function subtract `req.body.amount` from `accounts['credit'].balance` and add it to `accounts['credit'].available` (remember to use `parseInt()` when adding).
- Convert the javascript object to JSON and write it to a file.
- In the body of the post route function render the `payment` view and pass an object with the following key value pair, `message: "Payment Successful"`

Module 04 - Creating a Data Access Library

4.1 - Create a library file

@data-js-create-file Create a new file called `data.js` in the root of the `src/` directory.

4.2 - Require Built-in Libraries

@data-js-require-built-ins In `data.js` require the built-in library `fs` and store a reference to it in a `const` called `fs`. Next, require the built-in library `path` and store a reference to it in a `const` called `path`.

4.3 - Transition Account Data to Data Library

@data-js-transition-const-accounts In `app.js` locate the lines that are responsible for reading and parsing JSON from the `src/json/accounts.json` file. Copy and paste them to the new `data.js` file below the require statements.

4.4 - Transition User Data to Data Library

@data-js-transition-const-users In `app.js` locate the lines that are responsible for reading and parsing JSON from the `src/json/users.json` file. Copy and paste them to the new `data.js` file below the `accounts` const.

4.5 - Write JSON Function

@data-js-write-json-function In `data.js` below the account and user data lines create a function called `writeJSON`. **Hint: It is best to use ES6 arrow style function (=>).**

4.6 - Write JSON Function Body

@data-js-write-json-function-body In `app.js` locate the lines in the `transfer` post route function body that are responsible for writing JSON data to a file. **Hint: there are two lines.** Copy these lines to the body of the `writeJSON` function in the `data.js` file.

4.7 - Export Data and Function

@data-js-export-data In `data.js` use `module.exports` to export an object containing the constants `accounts`, `users`, and the `writeJSON` function.

4.8 - Require Data Library

@app-js-require-data-js Back In `app.js` require `data.js` and at the same time use object destructing to create three constants for `accounts`, `users`, and `writeJSON`. Remove the lines in `app.js` that create the `accountData`, `accounts`, `userData`, and `users` consts. `accounts`, `users`, and the `writeJSON` function are now brought in by the require statement.

4.9 - Function Call Transfer

@app-js-call-write-json-transfer In `app.js` locate the lines in the `transfer` post route function body that are responsible for writing JSON data to a file and replace them with a call to the `writeJSON()` function.

4.10 - Function Call Payments

@app-js-call-write-json-payment In `app.js` locate the lines in the `payment` post route function body that are responsible for writing JSON data to a file and replace them with a call to the `writeJSON()` function.

Module 05 - Using the Express Router

5.1 - Create a Account Routes File

@routes-accounts-js-create-file Create a new file called `accounts.js` in the directory `src/routes/` (you will need to create this directory).

5.2 - Require Express

@routes-accounts-js-require-express In the new `accounts.js` require the express framework and store a reference to it in a `const` called `express`. Next, call the `express.Router()` function and store it in a `const` called `router`.

5.3 - Require Data Library

@routes-accounts-js-require-data In `accounts.js` require `data.js` and at the same time use object destructing to create one constant called `accounts`.

5.4 - Move Account Routes

@routes-accounts-js-move-routes In `app.js` locate the savings, checking, and credit get routes, cut and paste these routes in `accounts.js` below the require statements. Now in `accounts.js` update the routes to be part of the router by replacing `app.get` with `router.get`.

5.5 - Export the Router

@routes-accounts-js-export-router In `accounts.js` export the `router` using the `module.exports` syntax.

5.6 - Use the Routes

@app-use-account-routes In `app.js` where your account routes used to be, call the `use` function on `app` with two arguments. The first argument should be `/account` and the second is the `accountRoutes` const.

5.7 - Create a Services Routes File

@routes-services-js-create-file Create a new file called `services.js` in the directory `src/routes/`.

5.8 - Require Express

@routes-services-js-require-express In the new `services.js` require the express framework and store a reference to it in a `const` called `express`. Next, call the `express.Router()` function and store it in a `const` called `router`.

5.9 - Require Data Library

@routes-services-js-require-data In `services.js` require `data.js` and at the same time use object destructing to create two constants called `accounts` and `writeJSON`.

5.10 - Move Services Routes

@routes-services-js-move-routes In `app.js` locate the transfer and payment post and get routes, cut and paste these routes to `services.js` below the require statements. Now in `services.js` update the routes to be part of the router by replacing `app.get` with `router.get`.

5.11 - Export the Router

@routes-services-js-export-router In `services.js` export the `router` using the `module.exports` syntax.

5.12 - Require account routes

@app-require-account-routes Switch to `app.js` and require the `routes/accounts.js` file and store a reference to it in a `const` called `accountRoutes`.

5.13 - Require services routes

@app-require-services-routes Switch to `app.js` and require the `routes/services.js` file and store a reference to it in a `const` called `servicesRoutes`.

5.14 - Use the Routes

@app-use-services-routes In `app.js` where your account routes used to be, call the `use` function on `app` with two arguments. The first argument should be `/services` and the second is the `servicesRoutes` const.

5.15 - Update Views

@views-update-for-routes Since all URL paths have changed for accounts and services we need to change the following views:

- In `src/views/index.ejs` change `href="transfer"` to `href="/services/transfer"`
- In `src/views/summary.ejs` change `href="<%= account.unique_name %>"` to `href="/account/<%= account.unique_name %>"`
- In `src/views/transfer.ejs` change `action="/transfer"` to `action="/services/transfer"`
- In `src/views/payment.ejs` change `action="/payment"` to `action="/services/payment"`