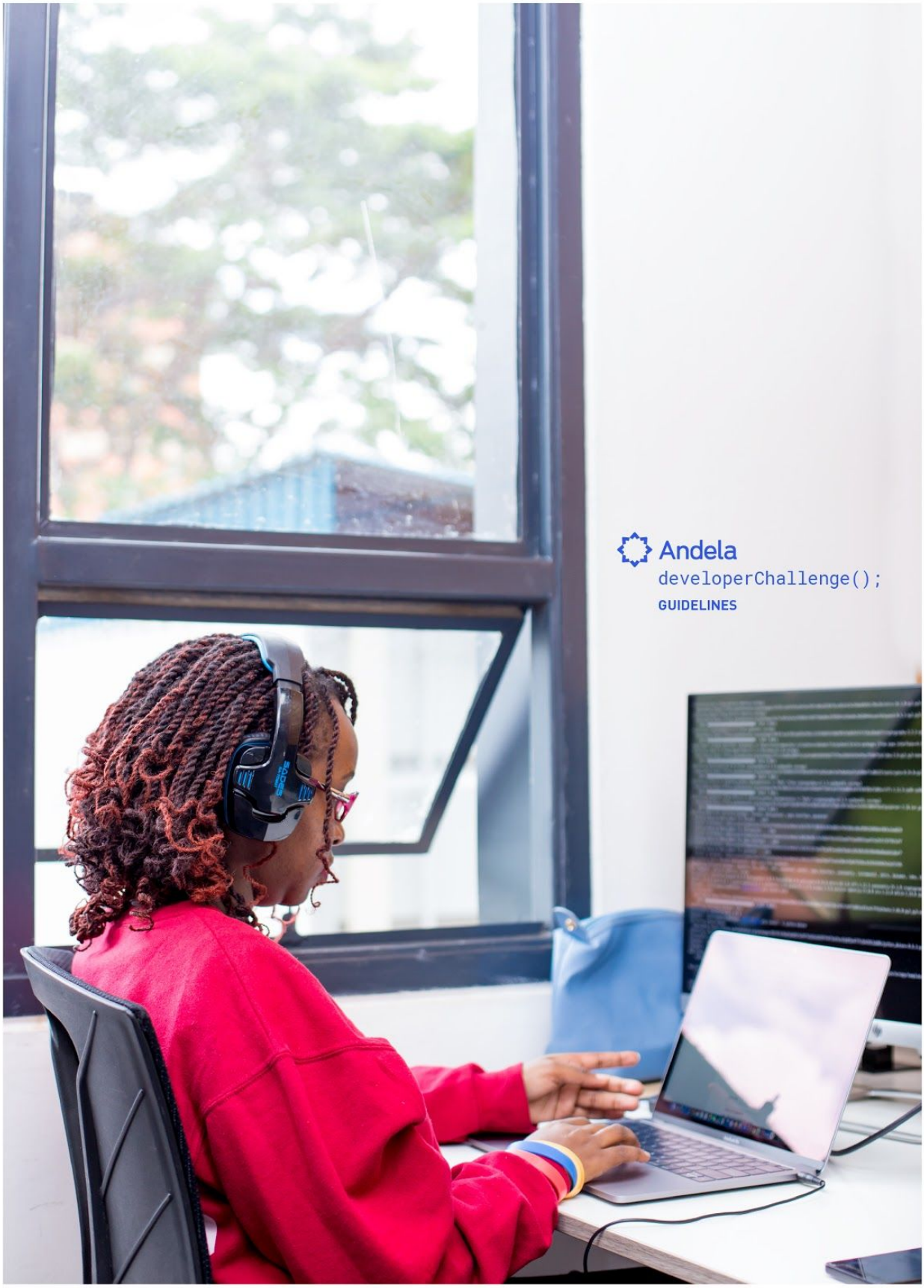
Andela



Andela

developerChallenge();

GUIDELINES

Andela

Andela Developer Challenge

Build A Product: **Banka**

Andela

BUILD A PRODUCT: Banka

Project Overview

Banka is a light-weight core banking application that powers banking operations like account creation, customer deposit and withdrawals. This app is meant to support a single bank, where users can signup and create bank accounts online, but must visit the branch to withdraw or deposit money..

Project Timelines

- **Total Duration**: **6 Weeks**
- **Final Due Date**: **April 29, 2019**

Required Features

1. User (client) can sign up.
2. User (client) can login.
3. User (client) can create an account.
4. User (client) can view account transaction history.
5. User (client) can view a specific account transaction.
6. Staff (cashier) can debit user (client) account.
7. Staff (cashier) can credit user (client) account.
8. Admin/staff can view all user accounts.
9. Admin/staff can view a specific user account.
10. Admin/staff can activate or deactivate an account.
11. Admin/staff can delete a specific user account.
12. Admin can create staff and admin user accounts.

Optional Features

1. User can reset password.
2. Integrate real time email notification upon credit/debit transaction on user account.
3. User can upload a photo to their profile.

Andela

Preparation Guidelines

These are the steps you ought to take to get ready to start building the project

Steps

1. Create a **Pivotal Tracker Project**
   ***Tip**: find how to create a Pivotal Tracker Project here.*

2. Create a **Github Repository, add a README, and clone it to your computer**
   ***Tip**: find how to create a Github Repository here.*

Andela

Challenge 1 - Create UI Templates

Challenge Summary

You are required to create UI templates with **HTML**, **CSS**, and **Javascript**.

Timelines

- **Duration:** **2 Weeks**
- **Due Date:** **March 29, 2019**

NB:

- *You are not implementing the core functionality yet, you are only building the User Interface (UI) elements, pages, and views!*
- *You are to create a pull request for each feature in the challenge and then merge into your develop branch.*
- *Do not use any CSS framework e.g Bootstrap, Materialize, sass/scss.*
- *Do not use any JavaScript frontend framework e.g React, Angular, Vue.*
- *Do not download or use an already built website template.*

Guidelines

1. **On Pivotal Tracker, create user stories to setup the User Interface (UI) elements:**
    a. A page/pages where an **admin** can do the following:
        i. Activate or deactivate user account.
        ii. Create admin or staff user account.

    b. A page/pages where a **staff** (cashier) can do the following:
        i. Credit a user account.
        ii. Debit a user account.

    c. A page/pages where an **admin**/**staff** can do the following:
        i. View list of all bank accounts.
        ii. View a specific bank account record.
        iii. Delete a specific account.

    d. A page/pages where a **user** (client) can do the following:
        i. Sign up.
        ii. Sign In.
        iii. Create a bank account.
        iv. View bank account profile (dashboard).
        v. View account transaction history.

2. **On Pivotal Tracker, create stories to capture any other tasks not captured above. A task can be feature, bug or chore for this challenge.**
3. **On a feature branch, create a directory called UI in your local Git repo and build out all the necessary pages specified above and UI elements that will allow the application function into the UI directory.**
4. **Host your UI templates on GitHub Pages.**

> ***Tip:*** *It is recommended that you create a* ***gh-pages*** *branch off the branch containing your UI template. When following the GitHub Pages guide, select* ***"Project site" >> "Start from scratch"****. Remember to choose the* ***gh-pages*** *branch as the* ***source*** *when configuring Repository Settings. See a detailed guide* ***here****.*

Target skills

After completing this challenge, you should have learned and be able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	• To get started with Pivotal Tracker, use Pivotal Tracker quick start. • Here is a sample template for creating Pivotal Tracker user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended Git Workflow, Commit Message and Pull Request (PR) standards.
Front-End Development	Using HTML and CSS to create user interfaces.	• See this tutorial • See this tutorial also
UI/UX	Creating good UI interface and user experience	• See rules for good UI design here • See this article for More guide • For color palettes, see this link

Andela

Self / Peer Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it	Accurately, assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner
Version Control with Git	Does not utilize branching but commits to master branch directly instead.	Utilizes branching, pull-requests, and merges to the develop branch. Use of recommended commit messages.	Adheres to recommended GIT workflow and uses badges.
Front-End Development	Fails to develop the specified web pages using **HTML/CSS/JavaScript** or uses an already built out website template, or output fails to observe valid **HTML/CSS/Javascript** syntax or structure.	Successfully develops web pages while observing standards such as doctype declaration, proper document structure, no inline CSS in HTML elements, and HTML document has consistent markup	Writes modular CSS that can be reused through markup selectors such as class, id. Understands the concepts and can confidently rearrange divs on request.
UI/UX	The page is unresponsive, elements are not proportional, the color scheme is not complementary and uses alerts to display user feedback	The page is responsive (at least across mobile, tablet and desktops), the color scheme is complementary, and uses properly designed dialog boxes to give the user feedback	User interface is well thought out, resulting in a memorable user experience. UI is functional with captivating aesthetics.

Andela

Challenge 2: Create API endpoints

Challenge Summary

You are expected to create a set of API endpoints defined in the API Endpoints Specification section and use data structures to store data in memory (don't use a database).

Timelines

- **Duration:** **2 Weeks**
- **Due Date:** **April 12, 2019**

NB:

- *You are to create a pull request for each feature in the challenge and then merge into your develop branch*

Tools

- Server side Framework: **Node**/**Express**
- Linting Library: ***ESLint***
- Style Guide: ***Airbnb***
- Testing Framework: ***Mocha*** *or* ***Jasmine***

Guidelines

1. **Setup linting library and ensure that your work follows the specified style guide requirements.**
2. **Setup the test framework.**
3. **Write unit-tests for all API endpoints.**
4. **Version your API using URL versioning starting, with the letter "v". A simple ordinal number would be appropriate and avoid dot notation such as 2.5. An example of this will be: https://somewebapp.com/api/v1/users.**
5. **Using separate branches for each feature, create version 1 (v1) of your RESTful API to power front-end pages.**
6. **On Pivotal Tracker, create user stories to setting up and testing API endpoints that do the following using data structures:**
    - User sign up.
    - User sign in.

- Create bank account.
- Admin/Staff can activate or deactivate an account.
- Admin/Staff can delete an account.
- Staff (cashier) can credit an account.
- Staff (cashier) can debit an account.

7. **On Pivotal Tracker create stories to capture any other tasks not captured above. The tasks can be feature, bug or chore for this challenge.**
8. **Setup the server side of the application using the specified framework**
9. **Ensure to test all endpoints and see that they work using Postman.**
10. **Integrate TravisCI for Continuous Integration in your repository (with *ReadMe* badge).**
11. **Integrate test coverage reporting (e.g. Coveralls) with a badge in the *ReadMe*.**
12. **Obtain CI badges (e.g. from Code Climate and Coveralls) and add to *ReadMe*.**
13. **Ensure the app gets hosted on Heroku.**
14. **At the minimum, you should have the API endpoints listed under the API endpoints specification on page 13 working. Also refer to the entity specification on page 12 for the minimum fields that must be present in your data models.**

API Response Specification

The API endpoints should respond with a JSON object specifying the HTTP ***status*** code, and either a ***data*** property (on success) or an ***error*** property (on failure). When present, the ***data*** property is always an ***object***.

On Success

```
{
  "status" : 200,
  "data" : {...}
}
```

On Error

```
{
  "status" : 404,
  "error" : "relevant-error-message"
}
```

The status codes above are provided as samples, and by no way specify that all success reponses should have **200** or all error responses should have **400.**

Andela

Target skills

After completing this challenge, you should have learned and able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	• To get started with Pivotal Tracker, use Pivotal Tracker quick start. • Here is a sample template for creating Pivotal Tracker user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended Git Workflow, Commit Message and Pull Request (PR) standards.
HTTP & Web services	Creating API endpoints that will be consumed using Postman	• Guide to Restful API design • Best Practices for a pragmatic RESTful API
Test-driven development	Writing tests for functions or features.	• Unit Testing and TDD in Node.js
Data structures	Implement non-persistent data storage using data structures.	• JS data types and data structures
Continuous Integration	Using tools that automate build and testing when the code is committed to a version control system.	
Holistic Thinking and big-picture thinking	An understanding of the project goals and how it affects end users before starting on the project.	• Node-postgres • Node.js postgresql tutorial

Andela

Self / Peer Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it	Accurately, assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner
Version Control with Git	Does not utilize branching but commits to master branch directly instead.	Utilizes branching, pull-requests, and merges to the develop branch. Use of recommended commit messages.	Adheres to recommended GITflow workflow and uses badges.
Programming logic	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition. The code also incorporates best practices and optimizations.
Test-Driven development	Unable to write tests. The solution did not attempt to use TDD	Writes tests with 60% test coverage.	Writes tests with coverage greater than 60%.
HTTP & Web Services	Fails to develop an API that meets the requirements specified	Successfully develops an API that gives access to all the specified endpoints	Handles a wide array of HTTP error code and the error messages are specific.
Data Structures	Fails to implement CRUD or Implements CRUD with persistence	Implements CRUD without persistence	Uses the most optimal data structure for each operation
Continuous Integration • Travis CI • Coverall	Fails to integrate all required CI tools.	Successfully integrates all tools with relevant badges added to ReadMe.	

Andela

Entity Specification

User

```
{
  “id” : Integer,
  “email” : String,
  “firstName” : String,
  “lastName” : String,
  “password” : String,
  “type” : String,          // client or staff
  “isAdmin” : Boolean,      // must be a staff user account
   ...
}
```

Account

```
{
  “id” : Integer,
  “accountNumber” : Integer,
  “createdOn” : DateTime,
  “owner” : Integer,     // user id
  “type” : String,       // savings, current
  “status” : String,     // draft, active, or dormant
  “balance” : Float,
   ...
}
```

Transaction

```
{
  “id” : Integer,
  “createdOn” : DateTime,
  “type” : String,          // credit or debit
  “accountNumber” : Integer,
  “cashier” : Integer,      // cashier who consummated the transaction
  “amount” : Float,
  “oldBalance” : Float,
  “newBalance” : Float,
   ...
}
```

Andela

API Endpoint Specification

Endpoint: POST /auth/signup
Create user account

Response spec:

```
{
  “status” : Integer,
  “data” : {
    “token” :  “45erkjherht45495783”
    “id”: Integer,      // id of newly created user
    “firstName”: String,
    “lastName”: String,
    “email”: String,
    ...
  }
}
```

Endpoint: POST /auth/signin
Login a user

Response spec:

```
{
  “status” : Integer,
  “data” : {
    “token” :  “45erkjherht45495783”
    “id”: Integer,      // user id
    “firstName”: String,
    “lastName”: String,
    “email”: String,
    ...
  }
}
```

Endpoint: POST /accounts
Create a bank account

Response spec:

```
{
  “status” : Integer,
  “data” : {
    “accountNumber”: Number,
    “firstName”: String,   // account owner first name
    “lastName”: String,    // account owner last name
    “email”: String,       // account owner email
    “type”: String,        // savings, current
```

```
    "openingBalance" : Float,
    ...
  }
}
```

Endpoint: PATCH /account/<account-number>

Activate or deactivate an account. Specify the new **status** in the request body.

Response spec:

```
{
  "status" : Integer,
  "data" : {
    "accountNumber" : Integer,     // user unique id
    "status" :  String,            // active or dormant
    ...
  }
}
```

Endpoint: DELETE /accounts/<account-number>

Delete a specific bank account.

Response spec:

```
{
  "status" : Integer,
  "message" : "Account successfully deleted"
```

Endpoint: POST /transactions/<account-number>/debit

Debit a bank account.

Response spec:

```
{
  "status" : Integer,
  "data" : {
     "transactionId": Integer,
     "accountNumber" : String,
     "amount" : Float,
     "cashier" : Integer,        // cashier that consumated the transaction
     "transactionType" : String,
     "accountBalance" : String,
  }
}
```

Andela

Endpoint: POST /transactions/<account-number>/credit

Credit a bank account.

Response spec:

```
{
  “status” : Integer,
  “data” : {
     “transactionId”: Integer,
     “accountNumber” : String,
     “amount” : Float,
     “cashier” : Integer,        // cashier that consumated the transaction
     “transactionType” : String,
     “accountBalance” : String,
  }
}
```

Andela

Challenge 3: Create more API endpoints and Integrate A Database

Challenge Summary

You are expected to create all the endpoints listed under the **API Endpoints Specification** section of this challenge, in addition to those specified in Challenge 2. You are to remove the data structures used to store data in **Challenge 2** and ensure that you persist your data with a database**.**
You are to write SQL queries that will help you write to and read from your database. The endpoints are to be secured with JSON Web Token (JWT).

Timelines

- **Duration:** **1 Week**
- **Due Date:** **April 18, 2019**

Note:

- *Ensure that Challenge 2 is completed and merged to the **develop** branch before you get started.*
- *You are to create a pull request for each feature in this challenge to elicit review and feedback, before merging into your develop branch.*
- *Do not use any ORMs.*

Tools

- Database: ***PostgreSQL***

Guidelines

1. **On Pivotal Tracker, create a chore for setting up the database.**
2. **On Pivotal Tracker, create user stories for setting up and testing API endpoints that do the following using database:**
    a. User can view account transaction history.
    b. User can view a specific account transaction.
    c. User can view account details.
    d. Admin/staff can view a list of accounts owned by a specific user.
    e. Staff/Admin can view all bank accounts
    f. Staff/Admin can view all active bank accounts.

g. Staff/Admin can view all dormant bank accounts.

3. **On Pivotal Tracker, create the story(s) for the implementation of token-based authentication using JSON Web Token (JWT) and the security of all routes(including those developed in challenge 2) using JSON Web Token (JWT).**
4. **On Pivotal Tracker, create stories to capture any other tasks not captured above. The tasks could be feature, bug or chore for this challenge.**
5. **On Pivotal Tracker, create user story(s) to implement one or all of these optional features:**
    a. User get real time notification when a debit/credit transaction occurs on his/her account.

    NB: Executing one or more features from the extra credits, in addition to the required features means you have exceeded expectations.
6. **Setup a PostgreSQL database.**
7. **Write tests for all the endpoints.**
8. **Ensure to test all API endpoints and see that they work using Postman.**
9. **Use API Blueprint, Slate, Apiary or Swagger to document your API. Docs should be accessible via your application's URL.**
10. **Ensure the app gets hosted on Heroku.**
11. **At the minimum, you should have the API endpoints listed under the API endpoints specification on page 13 working. Also refer to the entity specification on page 12 for the minimum fields that must be present in your data models.**

API Response Specification

The API endpoints should respond with a JSON object specifying the HTTP ***status*** code, and either a ***data*** property (on success) or an ***error*** property (on failure). When present, the ***data*** property is always an ***array***, even if there's none, one, or several items within it.

On Success

```
{
  "status" : 200,
  "data" : [{...}]
}
```

Andela

On Error

```
{
  "status" : 404,
  "error" : "relevant-error-message"
}
```

The status codes above are provided as samples, and by no way specify that all success reponses should have **200** or all error responses should have **400.**

Target skills

After completing this challenge, you should have learned and also be able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	• To get started with Pivotal Tracker, use Pivotal Tracker quick start. • Here is a sample template for creating Pivotal Tracker user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended Git Workflow, Commit Message and Pull Request (PR) standards.
HTTP & Web services	Creating API endpoints that will be consumed using Postman	• Guide to Restful API design • Best Practices for a pragmatic RESTful API • Web services
Test-driven development	Writing tests for functions or features.	• Unit Testing and TDD in Node.js
Continuous Integration	Using tools that automate build and testing when the code is committed to a version control system.	
Databases	Using a database to store data.	• Node-postgres • Node.js postgresql tutorial

Andela

Self / Peer Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it	Accurately, assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner
Version Control with Git	Does not utilize branching but commits to master branch directly instead.	Utilizes branching, pull-requests, and merges to the develop branch. Use of recommended commit messages.	Adheres to recommended GIT workflow and uses badges.
Programming logic	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition. The code also incorporates best practices and optimizations.
Test-Driven development	Unable to write tests. The solution did not attempt to use TDD.	Writes tests with 60% test coverage..	Writes tests with test coverage greater than 60%.
HTTP & Web Services	Fails to develop an API that meets the requirements specified	Successfully develops an API that gives access to all the specified endpoints	All API endpoints provide the appropriate HTTP status codes, headers and messages
Databases	Unable to create database models for the given project	Has a database of normalized tables with relationships. Can store, update and retrieve records using SQL.	Creates table relationships
Token-Based Authentication	Does not use Token-Based authentication	Makes appropriate use of Token-Based authentication and secures all private endpoints.	

Security	Fails to implement authentication and authorization in given project	Successfully implements authentication and authorization in the project	Creates custom and descriptive error messages
Continuous Integration • Travis CI • Coveralls	Fails to integrate all required CI tools	Successfully integrates all tools with relevant badges added to ReadMe file.	

API Endpoint Specification

Endpoint: GET /accounts/<account-number>/transactions
View an account's transaction history**.**

Response spec:

```
{
  "status" : Integer,
  "data" : [{
    "transactionId" : Integer,
    "createdOn" : DateTime,
    "type" : String,            // credit or debit
    "accountNumber" : Integer,
    "amount" : Float,
    "oldBalance" : Float,
    "newBalance" : Float,
  },
  {
    "transactionId" : Integer,
    "createdOn" : DateTime,
    "type" : String,            // credit or debit
    "accountNumber" : Integer,
    "amount" : Float,
    "oldBalance" : Float,
    "newBalance" : Float,
  }]
}
```

Endpoint: GET /transactions/<transaction-id>
View a specific transaction**.**

Response spec:

```
{
  "status" : Integer,
  "data" : {
    "transactionId" : Integer,
    "createdOn" : DateTime,
    "type" : String,            // credit or debit
```

```
    "accountNumber" : Integer,
    "amount" : Float,
    "oldBalance" : Float,
    "newBalance" : Float,
  }
}
```

Endpoint: GET /user/<user-email-address>/accounts

View all accounts owned by a specific user (client)**.**

Response spec:

```
{
  "status" : Integer,
  "accounts" : [{
    "createdOn" : DateTime,
    "accountNumber" : Integer,
    "type" : String,          // savings, or current
    "status" : String,        // draft, active, or dormant
    "balance" : Float,
  },
  {
    "createdOn" : DateTime,
    "accountNumber" : Integer,
    "type" : String,          // savings, or current
    "status" : String,        // draft, active, or dormant
    "balance" : Float,
  }]
}
```

Endpoint: GET /accounts/<account-number>

View a specific account's details**.**

Response spec:

```
{
  "status" : Integer,
  "data" : {
    "createdOn" : DateTime,
    "accountNumber" : Integer,
    "ownerEmail" : String,     // user email
    "type" : String,           // savings, current or loan
    "status" : String,         // draft, active, or dormant
    "balance" : Float,
  }
}
```

Andela

Endpoint: GET /accounts

View a list of all bank accounts**.**

Response spec:

```
{
  “status” : Integer,
  “data” : [{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “ownerEmail” : String,      // user email
    “type” : String,            // savings, current or loan
    “status” : String,          // draft, active, or dormant
    “balance” : Float,
  },{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “ownerEmail” : String,      // user email
    “type” : String,            // savings, current or loan
    “status” : String,          // draft, active, or dormant
    “balance” : Float,
  }]
}
```

Endpoint: GET /accounts?status=active

View a list of all active bank accounts**.**

Response spec:

```
{
  “status” : Integer,
  “data” : [{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “createdOn” : DateTime,
    “ownerEmail” : String,      // user email
    “type” : String,            // savings, current or loan
    “status” : String,          // draft, active, or dormant
    “balance” : Float,
  },{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “createdOn” : DateTime,
    “ownerEmail” : String,      // user email
    “type” : String,           // savings, current or loan
    “status” : String,         // draft, active, or dormant
    “balance” : Float,
  }]
}
```

Andela

Endpoint: GET /accounts?status=dormant

View a list of all dormant bank accounts.

Response spec:

```
{
  “status” : Integer,
  “data” : [{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “createdOn” : DateTime,
    “ownerEmail” : String,      // user email
    “type” : String,           // savings, current or loan
    “status” : String,         // draft, active, or dormant
    “balance” : Float,
  },{
    “createdOn” : DateTime,
    “accountNumber” : Integer,
    “createdOn” : DateTime,
    “ownerEmail” : String,      // user email
    “type” : String,           // savings, current or loan
    “status” : String,         // draft, active, or dormant
    “balance” : Float,
  }]
}
```

Andela

Challenge 4: Implement Front-end App and add more API endpoints

Challenge Summary

You are expected to power your HTML templates or front-end pages from **Challenge 1** with data from the API built in **Challenge 3.**
This challenge requires that you implement the frontend logic using standard language capabilities (e.g., **ES6** for JavaScript).

Timelines

- **Duration:** **1 Week**
- **Due Date:** **April 29, 2019**

Note:

- *Ensure that Challenge 3 is completed and merged to the **develop** branch before you get started with this challenge.*
- *You are to make use of the native browser **Fetch API** for making requests to the backend built in Challenge 2 and 3.*
- *Do **NOT** to use frameworks or libraries like Angular, Vue or React.*
- *Do **NOT** use any CSS libraries/framework e.g Bootstrap or Materialize*
- *Do **NOT** use frameworks or libraries like jQuery, Angular, Vue or React*

Guidelines

1. **On Pivotal Tracker create stories to build out your frontend with standard (vanilla) Javascript.**
2. **On Pivotal Tracker create stories to capture any other tasks not captured above. The tasks can be feature, bug or chore in this challenge.**
3. **Create a new folder or repo in which you will develop your front end.**
4. **Setup linting library (ESLint) and ensure you configured the style guide properly (Airbnb)**
5. **Implement your frontend.**
6. **Deploy your front-end to Github-Pages while the backend API should be deployed to Heroku**
7. **At a minimum, you should have a working frontend application consuming all your API endpoints.**

8. **Optional: Refer to API endpoint specification on page 24 for additional API endpoints.**

API Response Specification

The API endpoints should respond with a JSON object specifying the HTTP ***status*** code, and either a ***data*** property (on success) or an ***error*** property (on failure). When present, the ***data*** property is always an ***array***, even if there's none, one, or several items within it.

On Success

```
{
  "status" : 200,
  "data" : [{...}]
}
```

On Error

```
{
  "status" : 404,
  "error" : "relevant-error-message"
}
```

The status codes above are provided as samples, and by no way specify that all success reponses should have **200** or all error responses should have **400.**

Target skills

After completing this challenge, you should have learned and also be able to demonstrate the following skills.

Skill	Description	Helpful Links
Project management	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	• To get started with Pivotal Tracker, use Pivotal Tracker quick start. • Here is a sample template for creating Pivotal Tracker user stories.
Version control with GIT	Using GIT to manage and track changes in your project.	• Use the recommended Git Workflow, Commit Message and Pull Request (PR) standards.
UI/UX	Creating good ui interface and user experience	• See rules for good UI design here • See this article for More guide • For color palettes, see this link

Andela

Self / Peer Assessment Guidelines

Use this as general guidelines to assess the quality of your work. Peers, mentors, and facilitators should use this to give **feedback** on areas that should be improved on.

Criterion	Does not Meet Expectation	Meets Expectations	Exceed Expectations
Project management	Fails to break down modules into smaller, manageable tasks. Cannot tell the difference between chores, bugs, and features	Breaks down each module into smaller tasks and classifies them. Constantly updates the tool with progress or lack of it	Accurately, assigns points to the tasks. Informs stakeholders of project progress/blockers in a timely manner
Version Control with Git	Does not utilize branching but commits to master branch directly instead.	Utilizes branching, pull-requests, and merges to the develop branch. Use of recommended commit messages.	Adheres recommended GIT workflow and uses badges.
Programming logic	The code does not work in accordance with the ideas in the problem definition.	The code meets all the requirements listed in the problem definition.	The code handles more cases than specified in the problem definition. The code also incorporates best practices and optimizations.
UI/UX	The page is unresponsive, elements are not proportional, the color scheme is not complementary and uses alerts to display user feedback	The page is responsive (at least across mobile, tablet and desktops), the color scheme is complementary, and uses properly designed dialog boxes to give the user feedback	