



 Andela  
developerChallenge();  
GUIDELINES

# Andela Developer Challenge

Build A Product: **Banka**

## 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:** **4 Weeks**
- **Final Due Date:** **April 12, 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.

## 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](#).*

## 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 page/pages where an **admin** can do the following:
  - Activate or deactivate user account.
  - Create admin or staff user account.
- A page/pages where a **staff** (cashier) can do the following:
  - Credit a user account.
  - Debit a user account.
- A page/pages where an **admin/staff** can do the following:
  - View list of all bank accounts.
  - View a specific bank account record.
  - Delete a specific account.
- A page/pages where a **user** (client) can do the following:
  - Sign up.
  - Sign In.
  - Create a bank account.
  - View bank account profile (dashboard).
  - 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
<b>Project management</b>	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	<ul style="list-style-type: none"><li>• To get started with Pivotal Tracker, use <a href="#">Pivotal Tracker quick start</a>.</li><li>• <a href="#">Here</a> is a sample template for creating Pivotal Tracker user stories.</li></ul>
<b>Version control with GIT</b>	Using GIT to manage and track changes in your project.	<ul style="list-style-type: none"><li>• Use the recommended <a href="#">Git Workflow</a>, <a href="#">Commit Message</a> and <a href="#">Pull Request (PR)</a> standards.</li></ul>
<b>Front-End Development</b>	Using HTML and CSS to create user interfaces.	<ul style="list-style-type: none"><li>• <a href="#">See this tutorial</a></li><li>• <a href="#">See this tutorial also</a></li></ul>
<b>UI/UX</b>	Creating good UI interface and user experience	<ul style="list-style-type: none"><li>• See rules for good UI design <a href="#">here</a></li><li>• See this article for <a href="#">More guide</a></li><li>• For color palettes, see this <a href="#">link</a></li></ul>

## 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
<b>Project management</b>	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
<b>Version Control with Git</b>	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.
<b>Front-End Development</b>	Fails to develop the specified web pages using <b>HTML/CSS/JavaScript</b> or uses an already built out website template, or output fails to observe valid <b>HTML/CSS/Javascript</b> 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.
<b>UI/UX</b>	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.

## 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 responses should have **200** or all error responses should have **400**.

## Target skills

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

Skill	Description	Helpful Links
<b>Project management</b>	Using a project management tool (Pivotal Tracker) to manage your progress while working on tasks.	<ul style="list-style-type: none"><li>To get started with Pivotal Tracker, use <a href="#">Pivotal Tracker quick start</a>.</li><li><a href="#">Here</a> is a sample template for creating Pivotal Tracker user stories.</li></ul>
<b>Version control with GIT</b>	Using GIT to manage and track changes in your project.	<ul style="list-style-type: none"><li>Use the recommended <a href="#">Git Workflow</a>, <a href="#">Commit Message</a> and <a href="#">Pull Request (PR)</a> standards.</li></ul>
<b>HTTP &amp; Web services</b>	Creating API endpoints that will be consumed using Postman	<ul style="list-style-type: none"><li><a href="#">Guide to Restful API design</a></li><li><a href="#">Best Practices for a pragmatic RESTful API</a></li></ul>
<b>Test-driven development</b>	Writing tests for functions or features.	<ul style="list-style-type: none"><li><a href="#">Unit Testing and TDD in Node.js</a></li></ul>
<b>Data structures</b>	Implement non-persistent data storage using data structures.	<ul style="list-style-type: none"><li><a href="#">JS data types and data structures</a></li></ul>
<b>Continuous Integration</b>	Using tools that automate build and testing when the code is committed to a version control system.	
<b>Holistic Thinking and big-picture thinking</b>	An understanding of the project goals and how it affects end users before starting on the project.	<ul style="list-style-type: none"><li><a href="#">Node-postgres</a></li><li><a href="#">Node.js postgresql tutorial</a></li></ul>

## 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
<b>Project management</b>	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
<b>Version Control with Git</b>	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.
<b>Programming logic</b>	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.
<b>Test-Driven development</b>	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%.
<b>HTTP &amp; Web Services</b>	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.
<b>Data Structures</b>	Fails to implement CRUD or Implements CRUD with persistence	Implements CRUD without persistence	Uses the most optimal data structure for each operation
<b>Continuous Integration</b> <ul style="list-style-type: none"> <li>Travis CI</li> <li>Coverall</li> </ul>	Fails to integrate all required CI tools.	Successfully integrates all tools with relevant badges added to ReadMe.	

## 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,
  ...
}
```

## 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,  
  }  
}
```

**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,
  }
}
```