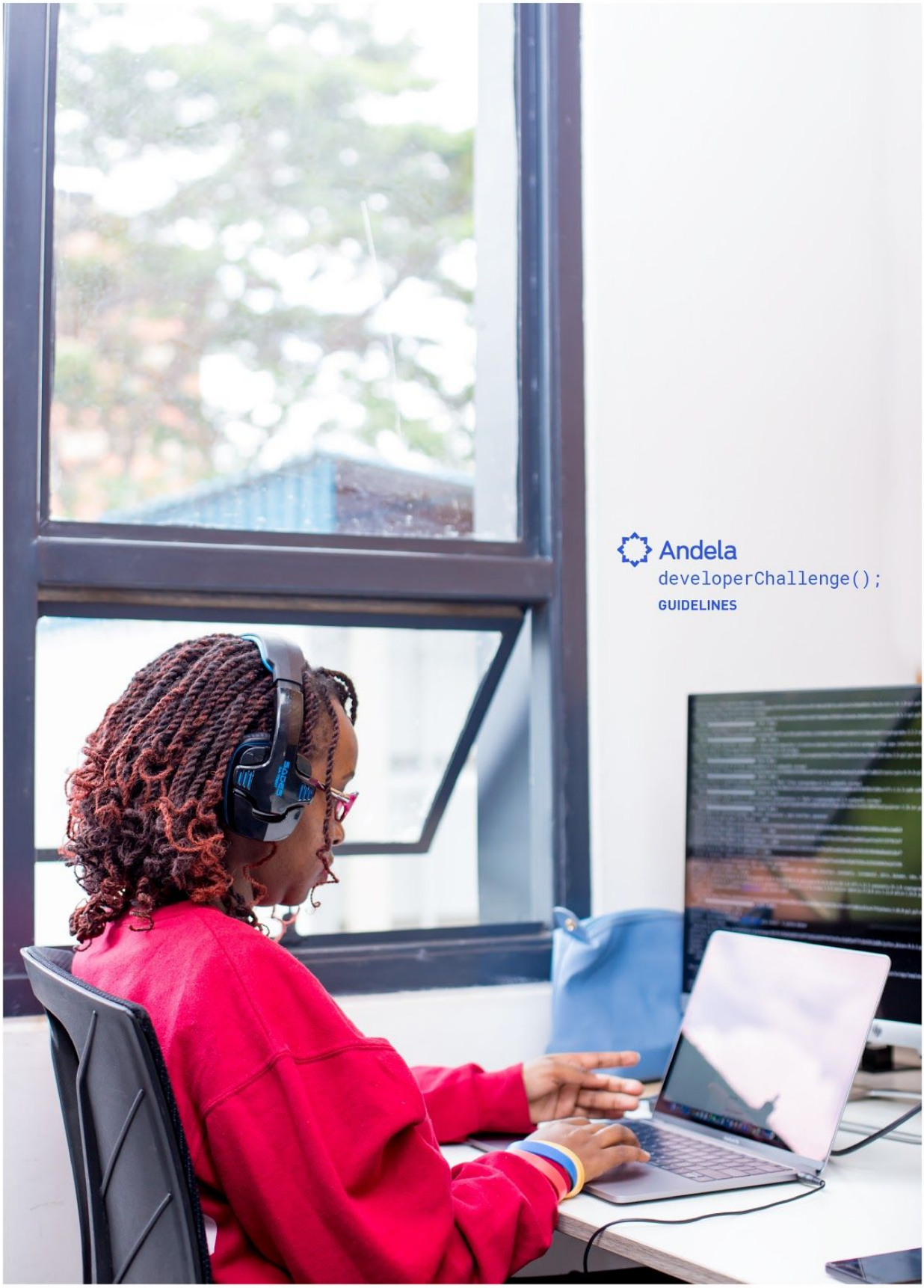
Andela



Andela

developerChallenge();

GUIDELINES

Andela

Andela Developer Challenge

Build A Product: **WayFarer API**

Andela

BUILD A PRODUCT: WayFarer API

Project Overview

WayFarer is a public bus transportation booking server. You are required to develop the back-end API.

Project Timeline

- **Final Due Date**: **July 16, 2019**

Required Features

1. User can sign up.
2. User can sign in.
3. Admin can create a trip.
4. Admin can cancel a trip.
5. Both Admin and Users can see all trips.
6. Users can book a seat on a trip.
7. View all bookings. An Admin can see all bookings, while user can see all of his/her bookings.
8. Users can delete their booking.

Optional Features

- Users can get a list of filtered trips based on origin.
- Users can get a list of filtered trips based on destination.
- Users can specify their seat numbers when making a booking.

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 API endpoints

Challenge Summary

You are expected to create a set of API endpoints listed under the **API Endpoints Specification** section. 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).

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 1.0. An example of this will be: https://somewebapp.com/api/v1/.**
5. **Using separate branches for each feature, create version 1 (v1) of your RESTful API.**
6. **On Pivotal Tracker, create user stories to setting up and testing API endpoints that do the following using databases:**
    - User can sign up.
    - User can sign in.
    - Admin can create a trip.
    - Both Admin and Users can see all trips.

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

On Success

```
{
  "status" : 'success',
  "data" : {...}
}
```

On Error

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

Target skills

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

Andela

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 **branch**, **pull request** and **commit message** naming convention. • 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.	• Getting started with TDD in Node Node.Js
Data Structures	Implement non-persistent data storage using data structures.	• JavaScript data types and data structures.
Databases	Using a database to store data.	• Node-postgres • Node.js postgresql tutorial
Continuous Integration	Using tools that automate build and testing when the code is committed to a version control system.	• Setup Continuous Integration with Travis CI in Your Nodejs App
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.
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
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,
  “first_name” : String,
  “last_name” : String,
  “password” : String,
  “is_admin” : Boolean,
   ...
}
```

Bus

```
{
  “id” : Integer,
  “number_plate” : String,
  “manufacturer” : String,
  “model” : String,
  “year” : String,
  “manufacturer” : String,
  “model” : String,
  “capacity” : Integer,
   ...
}
```

Trip

```
{
  “id” : Integer,
  “bus_id” : Integer,
  “origin” : String,                // starting location
  “destination” : String,
  “trip_date” : Date,
  “fare” : Float,
  “status” : Float,                 // active, cancelled - default is active
   ...
}
```

Booking - use a composite primary key consisting of (trip_id and user_id)

```
{
  “id” : Integer,
  “trip_id” : Integer,
  “user_id” : Integer,
  “created_on” : Date,
   ...
}
```

Andela

API Endpoint Specification

Endpoint: POST /auth/signup
Create user account

Request spec: (body)

```
{
  “email”: Integer,
  “password”: String,
  “first_name”: String,
  “last_name”: String,
    ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “user_id”: Integer,
   “is_admin”: Boolean,
   “token” :  String,
    ...
  }
}
```

Endpoint: POST /auth/signin
Login a user

Request spec: (body)

```
{
  “email”: Integer,
  “password”: String,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “user_id”: Integer,
```

```
    "is_admin": Boolean,
    "token" :  String,
     ...
  }
}
```

Endpoint: POST /trips
Create a trip.

Request spec: (body)

```
{
  "token": String,
  "user_id": Integer,
  "is_admin": Boolean,
   ...
}
```

Response spec: (body)

```
{
  "status" : "success",
  "data" : {
    "trip_id" : Integer,
    "bus_id" : Integer,
    "origin" : String,
    "destination": String,
    "trip_date" : Date,
    "fare" : Float,
    ...
  }
}
```

Andela

Endpoint: GET /trips

Get all trips.

Request spec: (body)

```
{
  “token”: String,
  “user_id”: Integer,                    // User id
  “is_admin”: Boolean,
   ...
}
```

Response spec: (body)

```
{
  “status” : “success”,
  “data” : [
    {
      “trip_id” : Integer,
      “bus_id” : Integer,
      “origin” : String,
      ”destination”: String,
      “trip_date” : Date,
      “fare” : Float,
      ...
    }, {
      “trip_id” : Integer,
      “bus_id” : Integer,
      “origin” : String,
      ”destination”: String,
      “trip_date” : Date,
      “fare” : Float,
      ...
    }, {
      “trip_id” : Integer,
      “bus_id” : Integer,
      “origin” : String,
      ”destination”: String,
      “trip_date” : Date,
      “fare” : Float,
      ...
    },
  ]
}
```

Andela

Challenge 2: Integrate A Database and Create more API Endpoints

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 1. 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).

Note:

- *Ensure that Challenge 1 is completed and merged to the **develop** branch before you get started.*
- *You are to create a pull request for each feature in the challenge and then merge 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. Users can book a seat on a trip.
    b. View all bookings. An Admin can see all bookings, while user can see all of his/her bookings.
    c. Users can delete their booking.
    d. Admin can cancel a trip.
3. **On Pivotal Tracker, create stories to capture any other tasks not captured above. The tasks could be feature, bug or chore for this challenge.**
4. **On Pivotal Tracker, create user story(s) to implement one or all of these optional features:**

a. User can get and filter trips using trip destination.
b. User can get and filter trips using trip origin.
c. User can change seats after booking.

NB: Executing one or more features from the extra credits (optional features), in addition to the required features means you have exceeded expectations.

5. **Write tests for all the endpoints.**
6. **Ensure to test all API endpoints and see that they work using Postman.**
7. **Use API Blueprint, Slate, Apiary or Swagger to document your API. Docs should be accessible via your application's URL.**
8. **Ensure the app gets hosted on Heroku.**
9. **At the minimum, you should have the API endpoints listed under the API endpoints specification on page 17 working. Also refer to the entity specification on page 10 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*** *or an* ***array***.

On Success

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

On Error

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

Target skills

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

Andela

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 **branch**, **pull request** and **commit message** naming convention. • 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
Continuous Integration	Using tools that automate build and testing when the code is committed to a version control system.	• Setup Continuous Integration with Travis CI in Your Nodejs App
Databases	Using a database to store data.	• Node-postgres • Node.js postgresql tutorial

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

Andela

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.	

Andela

API Endpoint Specification

Endpoint: POST /bookings
Book a seat on a trip

Request spec: (body)

```
{
  “token”: String,
  “user_id”: Integer,
  “is_admin”: Boolean,
  “trip_id”: Integer,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “booking_id” :  Integer,
   “user_id”: Integer,
   “trip_id” :  Integer,
   “bus_id” :  Integer,
   “trip_date”: Date,
   “seat_number” :  Integer,
   “first_name”: String,
   “last_name”: String,
   “email”: String,
    ...
  }
}
```

Endpoint: GET /bookings

View all bookings. An Admin can see all bookings, while user can see all of his/her bookings.

Request spec: (body)

```
{
  “token”: String,
  “user_id”: Integer,
  “is_admin”: String,
   ...
}
```

Andela

Response spec:

```
{
  “status” : “success”,
  “data” : [
    {
      “booking_id” :  Integer,
      “user_id”: Integer,
      “trip_id” :  Integer,
      “bus_id” :  Integer,
      “trip_date”: Date,
      “seat_number” :  Integer,
      “first_name”: String,
      “last_name”: String,
      “email”: String,
       ...
    }, {
      “booking_id” :  Integer,
      “user_id”: Integer,
      “trip_id” :  Integer,
      “bus_id” :  Integer,
      “trip_date”: Date,
      “seat_number” :  Integer,
      “first_name”: String,
      “last_name”: String,
      “email”: String,
       ...
    },{
      “booking_id” :  Integer,
      “user_id”: Integer,
      “trip_id” :  Integer,
      “bus_id” :  Integer,
      “trip_date”: Date,
      “seat_number” :  Integer,
      “first_name”: String,
      “last_name”: String,
      “email”: String,
       ...
    },
  ]
}
```

Andela

Endpoint: DELETE /bookings/:bookingId
Delete a booking.

Request spec: (body)

```
{
  “token”: String,
  “user_id”: Integer,
  “is_admin”: Boolean,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “message” :  “Booking deleted successfully,
    ...
  }
}
```

Endpoint: PATCH /trips/:tripId
Cancel a trip.

Request spec: (body)

```
{
  “token”: String,
  “user_id”: Integer,
  “is_admin”: Boolean,
   ...
}
```

Response spec:

```
{
  “status” : “success”,
  “data” : {
   “message” :  “Trip cancelled successfully”,
    ...
  }
}
```