Assignment 2

Gaurav Singhal - gs2870
Iris Zhang - iz2140
Jonathan Sun - jys2124
Kaili Chen - kc3031

Front-end CRUD interface

http://front-end-app.s3-website-us-east-1.amazonaws.com/#/home

User Stories

1. Faith, a user, should be able to retrieve a customer's information from a valid email address.
2. Faith, a user, should be able to create a new customer with an email, first and last names, phone number, and address reference (all valid).
3. Faith, a user, should be able to update a customer's information using an email address as the key.
4. Faith, a user, should be able to delete a customer based on a valid email address.
5. Faith, a user, should be able to retrieve an address based on a unique address UUID.
6. Faith, a user, should be able to create a new address with a city, street, number, and zip code (all valid).
7. Faith, a user, should be able to update an address from an address UUID, thus generating a new address.
8. Faith, a user, should be able to delete an address based on a valid address UUID.

Architecture/Design Choices

Overview

Briefly, under API Gateway, an API called 'CustomersAddresses' is used to expose resources 'customers' and 'addresses'. Methods GET, POST, PUT, and DELETE are allowed on both resources.

For GET, UPDATE and DELETE on 'customers', the key is "email". For POST on 'customers', the body contains the data. For UPDATE on 'customers', the body is a subset of the possible fields for a Customer. Only those fields may be updated, and non-existent objects may not be updated.

For GET, UPDATE and DELETE on 'addresses', the key is "uuid", a unique identifier for an address. For POST on 'addresses', the body contains the data. For UPDATE on addresses, the body is a subset of the possible fields for an Address. Only those fields may be updated, and non-existent objects may not be updated.

For both resources, navigation works; specifically, /<resources>/{key}/<field> returns the field of that resource under 'key'.

All APIs link to triggers to AWS Lambda functions. Each method for each resources maps to its own individual lambda function. All lambda functions perform validation on input

(discussed below) and access DynamoDB databases that store all information. The two databases are "Customers" and "Addresses" with schemes detailed below. The functions attempt to perform the desired action, and return the result if successful or meaningful error messages on failure.

Database Schemas

Both tables implement lazy deletion using a boolean field 'isDeleted'.

Addresses

Key	Type
dpBarcode	String (primary)
city	String
number	String
state	String
street	String
zipCode	String
isDeleted	boolean
uuid	String (deprecated)

Customers

Key	Type
Email	String (primary)
firstName	String
lastName	String
phoneNumber	String
address_ref	String
isDeleted	boolean

Lambda Functions

Each lambda function corresponds to a customer-facing API. The usage is detailed via the swagger files.

- getCustomer
- createCustomer
- updateCustomer
- deleteCustomer

- getAddress
- getAddressForCustomer
- createAddress
- updateAddress
- deleteAdderess

Managers

- **DBManager** - a simple manager interface (database agnostic) that handles all database interactions for both address and customer.
- **ValidationManager** - a validation manager that validates individual fields of both customers and addresses as well as full datasets of either.

Development Process

The team uses git for our version control and feature-branching for git-flow. The process is generally as follows: A dev member is assigned a task. The dev creates a new branch labeled with the feature - he or she owns this feature. The dev then implements the feature and sends a request to the teammates to review his or her code/changes. When the changes are approved, a member other than the dev can merge the changes onto the mainline "master" branch. (Please see “github.png” located in the docs/ directory).

For an 'agile' workflow, the team uses a barebones version of kanban. At every meeting, each dev member briefly describes his or her past, current, or future tasks, as well as any blockers (while standing). The team uses 'trello.com' for an online electronic board. There are four columns: backlog, in progress, code review, and done. Tasks are listed in the backlog and distributed among the devs. As the devs work, the tasks are moved to 'in progress'. When the task is ready for review, it is moved to the 'code review' column. On completion, tasks are moved to 'done.' (Please see “kanban.png” located in the docs/ dicrectory).