Ladder: Helping Entrepreneurs Climb Towards Success

1st Anthony Fiorito 40000808
Dept. Electrical and Computer Engineering
Concordia University

2nd Fozail Ahmad 40008728
Dept. Electrical and Computer Engineering
Concordia University

***Abstract*—WRITE A COOL ABSTRACT HERE OK THANKS – Past Anthony**

***Index Terms*—rest, json, cloud, amazon web services, aws, entrepreneurs, business**

I. Introduction

Being an entrepreneur involves thinking of innovative ideas that will impact the world in a meaningful way. All it takes is one good idea to change the world. However, no matter how well thought out an idea is, struggles arise when trying to sell a business proposition with no previous client base or audience.

A. Context and Problem Statement

Marketing a product or service is not a simple task, nor is it a low-cost undertaking. In order to solve these problems, Ladder is an application we've developed to solve exposure issues for small businesses and entrepreneurs. It is common for small entrepreneurs such as freelancers to have a small or non-existent marketing budget. Many businesses will advertise their services using their social media followings which has no direct connection to the purchasing of their services.

Another downside of public forums is the lack of curation from the users of the platform. While customers can spread their satisfaction via word of mouth and the Internet, it is not always as easy to get quality recommendations for freelancers or hobbyists selling their work. Similarly, online review websites such as Yelp usually offer reviews for well established businesses as opposed to small entrepreneurs. Ladder aims to solve both these issues by leveraging direct purchases to services on the platform and a user curation system for identifying entrepreneurs offering products or services of higher quality.

B. Web Service Functionality

Ladder allows users on the platform to advertise and purchase local services. As a user offering a business proposition, they can advertise their post by providing necessary information and photos of the product if needed. Once a posting is made, other users on the platform within a local radius can contact the user offering the product for more information and directly purchase the service. When a purchase is successfully made, the customer can rate the transaction based on the quality of service they received. All ratings are accumulated and the users offering product receive an overall score based on the weighted average of all the ratings they've received. Overall, Ladder empowers entrepreneurs and customers to make transactions as seamless as possible which benefits everyone on the platform.

II. Application Architecture

The application can be divided into two sections: frontend and backend. The frontend is implemented as a Single Page Application (SPA) using React as a view framework. The backend is written in NodeJS (Node) a JavaScript Runtime environment. It is deployed on the cloud in Amazon Web Services (AWS) Lambda containers. The entire application makes calls to other AWS and external services which can be summarized by the architecture diagram (see Fig. **??**).

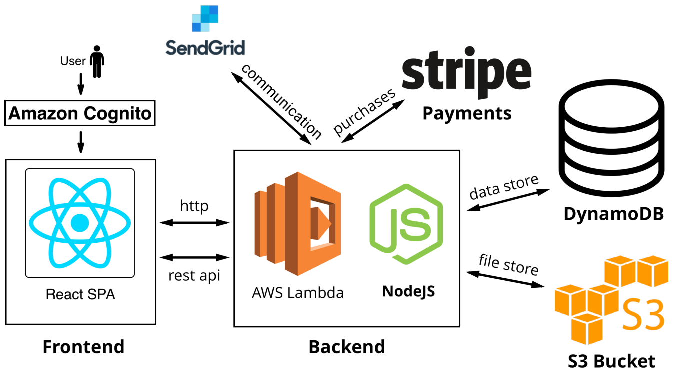

Fig. 1. Application Architecture Flow

A. Frontend Architecture and Implementation

The frontend for Ladder is a React SPA which is served from the cloud via Amazon's S3. All files are bundled and minimized for production and pushed to S3. React is a JavaScript library for building user interfaces. It allows for the creation of declarative component based descriptions of a HTML page. Using React, makes managing the application state simple since components re-render automatically whenever the state is updated. The efficient diffing algorithm used by React updates the DOM only when the data changes so the User Interface (UI) is responsive and in a consistent state with the model. The use of Bootstrap allows the site to be completely responsive so that users browsing the application on mobile devices will have a comfortable experience.

To access the core pages of the application, users must get past the sign up/sign in page. Once the user has successful logged in, they are directed to local posts page of the application. It is necessary that the user's web browser is compatible with the geolocation API implemented in most modern web browsers. From the local posts screen, users can browse all local postings by other users. In addition, they can select a post to view the posting in detail, make a new post or visit their profile.

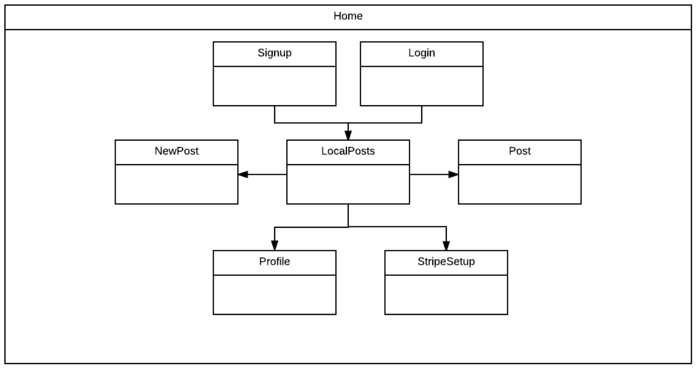

Fig. 2. Application Page Structure

A post on the website requires a title, description, price and product domain. From the posting page, potential customers can send a message for more information, see the poster's rating, view product photos and even purchase the service or product. However, to be eligible to create a posting users must first authenticate their account with Stripe. Users attempting to make a post without Stripe authentication will be redirected to the Stripe setup page. Furthermore, all the activity of a user on the website is conveniently displayed on their profile so they can see their previous purchases and sales as well as all their postings. All information is readily accessible to users so they can make the most informed decisions possible.

All data is retrieved via HTTP requests to the JSON REST API hosted on the cloud. Whenever a component is rendered, it requests the data from the server asynchronously and displays it to the user when it is ready. React facilitates the data retrieval process by implementing lifecycle methods for components so the data can be retrieved as the component is being rendered. To make the HTTP requests, a custom function `invokeApig` signs and makes the requests so that only users logged into the service can make requests to the api.

B. Backend Architecture and Implementation

The entire backend is built on top of a "serverless" architecture which uses stateless code containers to implement the JSON REST API. The term serverless means there is no instance of server running the backend logic. Ladder makes use of AWS Lambda to implement the serverless architecture. Lambda uses code containers called handlers (or handler functions) which receive exterior events and responds using a callback. For this application, the event which triggers the code container is a HTTP event. Therefore, every endpoint of the REST API is a separate handler function which waits for an event. The handler functions can be written in multiple languages, however, Ladder uses Node to implement the container logic.

Amazon Cognito is used to authenticate users. Not only does Cognito authenticate user credentials it provides email verification after signup which would reduce the chance of bots creating fake accounts. In addition, Cognito provides benefits for authentication to be easily intertwined with other AWS services. This means access to DynamoDB, S3 and Lambda can be limited depending on the type of authentication set up in Cognito. For the purpose of this application, only authenticated users can access the Lambda endpoints. However, Cognito requires the creation of a User Pool and Identity Pool to implement this functionality. The user pool is responsible for keeping track of users, sign-up/sign-in functionality and creates the provisioning tokens for extended authentication when using the service. The identity pool allows for the creation of unique identities to obtain temporary access to AWS services such as DyanmoDB, S3 and Lambda.

The data model of the application is stored in Amazon's NoSQL database: DynamoDB. The data model is used to store the various application data including user data, purchases, rating information and the posts made by the users (see Fig. **??**). In order to load local posts based on user location, the posts table is set up to use queries based on geolocation. The creation of the DynamoDB tables only require a primary key and optional sort key for more intricate queries. In order to query items that are not the primary key and sort key, indexes must be used. The DynamoDB tables extensively use GlobalSecondaryIndexes provided by AWS to create more complex queries on the database. The queries are made from the stateless code containers in AWS Lambda each connected to the backing store which is DynamoDB.

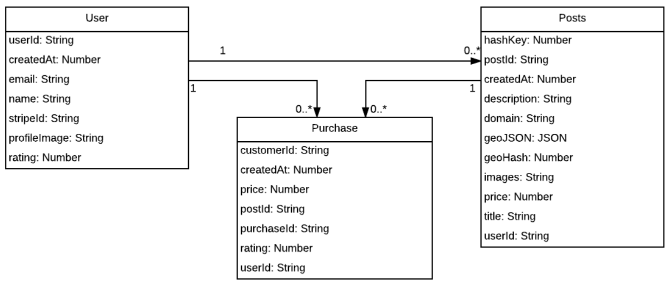

Fig. 3. Data Model UML Diagram

Most of the API endpoints perform a similar flow of execution which can be described as follows:

1) Receive a HTTP request event.
2) Retrieve information from the event using `event.body` or `event.pathParameters`.
3) Perform one or more DynamoDB operations.
4) Process the information.
5) Send a response to the client.

In the form of the handler a typical request (in pseudo-code) would be in the form:

```
 handler(event, context, callback) {
    // read request
    request = event.body
      // build DynamoDB query
    query = {...}

    // execute the query
    results = dynamoDB.query(query)

    // process the results
    results.map((result) => {...})

    // Send a response to the user
    if(success) callback(results)
    else callback(error)
}
```

C. *Other AWS and External Services*

Ladder uses Amazon's S3 extensively for serving the SPA and storing files uploaded by the users. The project uses the build tool Webpack to transpile the JavaScript code into a bundle of HTML, JavaScript and CSS files. When the users accesses the URL of the webpage, these files are served to the user. For file uploading, users can upload a custom profile picture of their choosing to make their profile more unique. In addition, users are able to upload up to three photos when creating a post. Therefore Ladder uses two S3 buckets, one for storing the static website files and one for storing user uploaded content.

The Stripe API is an essential part of the application. Stripe is used to allow users to receive payments and pay for products and services. For users wanting to make posts, they must connect their stripe account to the service. Users who have no intention of making posts but wish to buy services can simply use an accepted credit card.

Although users can gather more information from the description of the posting and the product photos, sometimes more information is required. Thus, users can contact other users via email by clicking the message button on a post. The emails initial email to connect the users is sent via the SendGrid API. The API empowers entrepreneurs and customers by allowing for direct communication between users so they further discuss the business proposition if needed. The Lambda Node containers make calls to these APIs via additional endpoints available through the REST API.

III. Quality Attributes

The architectural design of Ladder allows it to meet four of the major quality attributes necessary for maximum user satisfaction and retention. The four quality attributes are performance, scalability, availability and security.

A. *Performance*

Although JavaScript runs in a single threaded environment, it leverages an asynchronous event-driven model so the main thread of execution is never blocked. JavaScript offers two main methods for dealing with code which will only return a value at some time in the future: Promises and Callbacks. Promises will return a promise to the future data and return the main execution thread to promise block once the data is resolved. Similarly, asynchronous functions can leverage a callback which is called when the data is retrieved in the future. The performance benefits of this programming model allows for multiple simultaneous calls over the network, including external api calls and database operations. In addition, since all the API endpoints are running in a self-contained stateless code container, each endpoint only uses a small memory footprint which can't be slowed down by other endpoints which have a slower execution or larger memory footprint.

B. *Scalability*

AWS Lambda provides a cost-effective and low maintenance solution for scaling a REST API. Lambda has continuous scaling which scale an "application by running code in response to each trigger. [The] code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload". [ADD REF HERE AWS LAMBDA SITE] Lambda requires no servers to manage, therefore there's no need for server configuration to decide the amount of memory and processing power the REST API needs. Therefore no resources are wasted by Lambda. When the REST API has a larger workload, Lambda will automatically allocate the resources necessary to meet demands. During times when no one is using the service (during development), lambda will scale down to minimal or zero resources used.

C. *Availability*

The availability of Ladder depends almost entirely on the availability of AWS. There is a rare chance that the availability of AWS would suffer due to the amount of computing resources at their disposable. Another benefit of Lambda is no server process is continually running meaning that the main server process can never crash. Every individual code container runs until completion until it receives another event. Therefore, a code container with a bug will never affect the other endpoints of the API. Other issues of availability are handled by using industry leading external API providers such as SendGrid and Stripe. Although it is possible for these services to become unavailable, it is not likely to happen.

D. *Security*

Ladder uses Amazon Cognito to authenticate users on the platform. Using Cognito makes it easy to generate email verification on sign-up, provisioning tokens and restricting access to AWS services such as DynamoDB, S3 and Lambda. Beginning with signup, users must verify their email, otherwise they are not able to access Ladder. This will ensure that the majority of the users on the platform are people with real emails which will build trust and confidence from the user base. Another aspect of security is DynamoDB, S3 and Lambda access are restricted only to API calls which a signed with a valid token. These restrictions protect users on the platform from other individuals who may try to use the REST API maliciously. Overall, Ladder aims to instil confidence in the user base by making security a priority.

IV. Acquired Technical Knowledge

The knowledge acquired during the development of the project is mainly based around the use of cloud technologies and external APIs. We were familiar with React and web development technologies beforehand. The frontend only required making calls to the backend which is pretty common for web applications.

All the cloud technologies used in the project are S3, DynamoDB, Lambda and Cognito. The documentation amazon provides to learn how to use these services is lacking. Due to the many different language bindings they offer, it is sometimes difficult to learn how to do something in the chosen programming environment, in this case JavaScript. Firstly, S3 is fairly powerful and simple to use. It can be very useful for future projects involving file storage on the cloud. Since the knowledge and use of NoSQL databases before the project was minimally, using DynamoDB required a decent amount of learning. NoSQL provides different challenges and methodologies for doing similar relational database tasks. However, DynamoDB integrated nicely with other Amazon services and provided great exposure to using NoSQL databases. Lambda is simple to setup when using the serverless node module which makes automated deployments of the code handlers simple. Developing with Lambda was a great opportunity to have experience outside of running an instance of computing environment to run the server. In addition, the scaling up and down of Lambda when necessary allowed for minimal development costs. Lastly, Cognito was confusing to learn and contains many concepts which aren't described so clearly. Without much practice with authentication beforehand, Cognito required a lot of learning and tutorials. Overall, the combination of all the cloud technologies provides a greater understand of cloud technologies especially Amazon Web Services.

Learning to use the other external APIs was fairly straightforward. SendGrid provides a powerful API to send emails in just a few lines of code. In addition, they provide a web based administration console for checking the emails being sent. The Stripe API was more difficult to use due to the fact that credit card transactions need to be made carefully. Similarly, Stripe provides an administration console to view purchases, track payments, create refunds, etc. Using these external APIs enriches the feature set of Ladder and makes entrepreneurs and customers lives easier.

V. Accessing The Application

// add this here eventually

References

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**]. Refer simply to the reference number, as in [**?**]—do not use "Ref. [**?**]" or "reference [**?**]" except at the beginning of a sentence: "Reference [**?**] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [**?**]. Papers that have been accepted for publication should be cited as "in press" [**?**]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [**?**].

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.