CARDEN

System Design

FEBRUARY 12, 2021

301-WEEBS

Table of Contents

1.Introduction 2

2.Three tier Architecture 2

3.Environment Interaction 2
4. CRC cards........ 2
5. Software Architecture Diagram 3
6. Presentation layer........ 3
7. Application layer 4
8. Data layer 4
9. Deploy 4

1.Introduction

Carden is a service that provides simple and innovative ways to view, share, and create greeting cards. With users being able to register, save templates and view other users' templates, our platform should be capable of handling multiple simultaneous usage of our service. We needed to take into consideration what performance, scalability, availability, and development would be like for our product. Hence during the architectural design, we came to the decision that a three-tiered architecture would be the best fit. Separating the front-end, back-end and database into three tiers would give us the benefits mentioned earlier for our platform.

2.Three tier Architecture

A Three tier architecture consists of three-layers.

1. Presentation layer – is the front-end layer and consists of the user interface. It displays content in a graphical way and information useful to an end user. It is the layer that a user sees and interacts with.

2. Application layer – is the layer where the core logic for our platform is located. It's the layers that drives the capabilities of the application. It is the middleman of the three layers and communicates with both the presentation layer and the data layer.

3. Data layer – is the layer that comprises of the database/data storage system and data access layer.

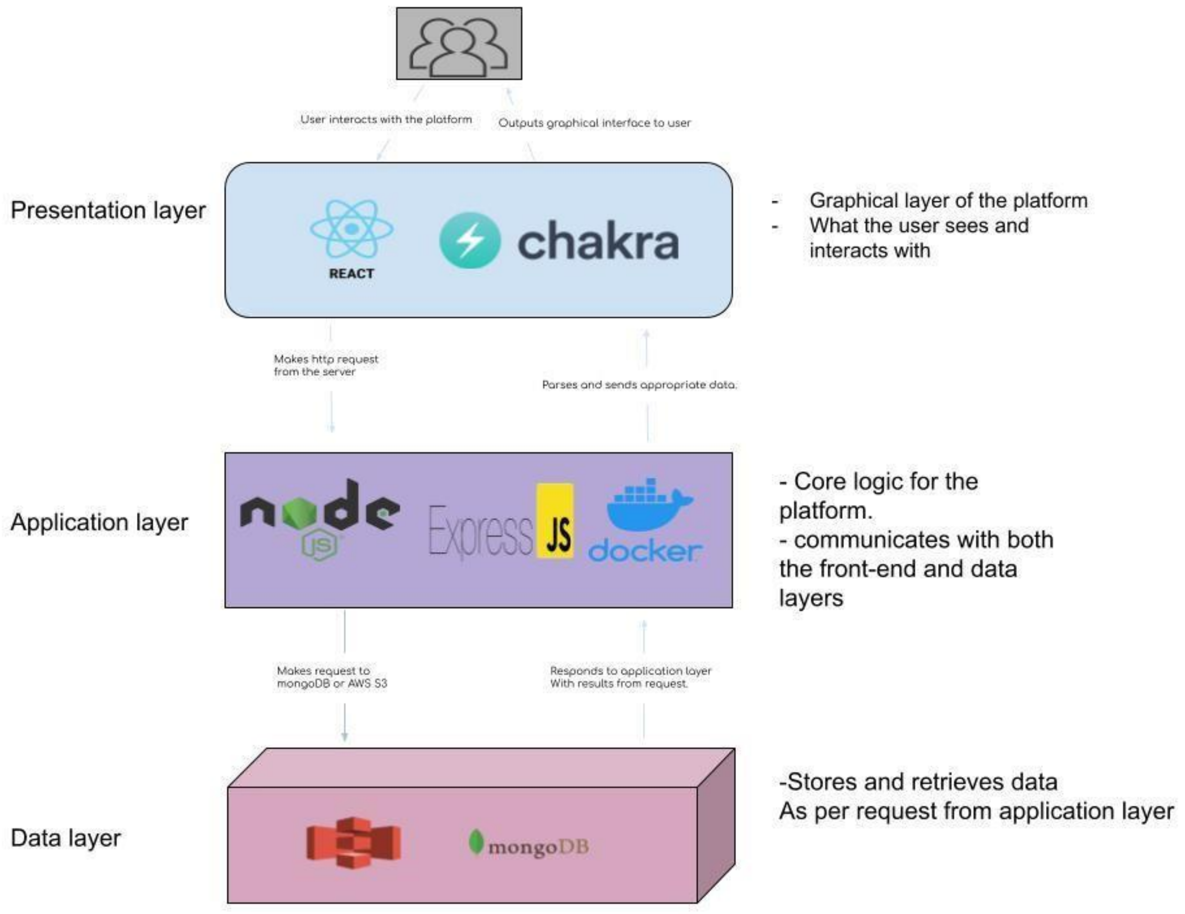
3.Environment Interaction

Carden is organized into two main directories, the client and server directories. As their names suggest, the client directory is going to be where our app's frontend logic lies, and the server directory is where the backend logic lies. We decided to use Docker to allow us to serve a local instance of our app which is compatible in any OS because of docker's containers. Since group members in our team use all 3 major OS's (Windows, MacOS, Linux), this was a natural decision to make sure all dependencies are properly met when running our app. To run our app locally, the user simply needs to install docker + dockercompose as well as Node.js (npm), then run the `npm run dev` script in either the client or server directories. The client is run on localhost port 3000, and the server is run on localhost port 5000, we use the http-proxy-middleware to setup the proxy so our client can communicate with the server. For our database, we use a cloud instance of MongoDB (Atlas) and AWS S3, our NodeJS + ExpressJS server is responsible for querying the cloud database.

4. CRC cards

Refer to CRC.pdf file in the sprint 1 folder.

5. Software Architecture Diagram


User interacts with the platform
Outputs graphical interface to user
Presentation layer
REACT
chakra
- Graphical layer of the platform
- What the user sees and interacts with
Makes http request from the server
Parses and sends appropriate data.
Application layer
node JS
Express JS
docker
- Core logic for the platform.
- communicates with both the front-end and data layers
Makes request to mongoDB or AWS S3
Responds to application layer With results from request.
Data layer
mongoDB
-Stores and retrieves data As per request from application layer

6. Presentation layer

This the topmost layer of the application, it is the only layer that the user views and interacts with. It displays output to the user and takes input from the user.

- In our system, this client is represented using react.js and chakra UI.
  Chakra UI is a simple modular and accessible component library that gives us building blocks needed for our react application. React.js accepts user input and displays appropriate output based on the response from the application layer.
- Since the presentation layer does not interact with the database, its only functions is to: take input from the user, display output to the user, and make requests to the application layer.
- The benefits of having a presentation layer, is that it can be scaled independently, improves security since the front end will not communicate with the backend preventing SQL injections and other malicious exploits, and allows faster development since front-end and back-end can be worked on separately.

7. Application layer

This is middle layer of the application where all the main logic for how our application should behave. It receives request and responds to the presentation layer and makes request and receives response from the data layer.

- We make use of a docker, node.js and express.js, aws Lambda in our platform. Docker allows for our developers to easily deploy and run the server and client on any OS and test it. Express.js is for routing api requests, it is what communicates with the data base. Node.js is for setting up the webserver and connects all components together with different APIs.

- the most recent addition to the application layer is aws Lambda. Aws lambda allows us to create a serverless endpoint between our application and the AWS S3 database. This allows our application to have an API gateway that runs a POST and GET method without the need of having a server running for our web application.

8. Data layer

This is the bottom layer of the application where all the user information and data are stored. The only thing to interact with this layer is the application layer. It takes request queries from the application layer and sends back the queried results.

- We make use of 2 primary databases. The first being MongoDB where we store information of users, finished cards, card templates and more. The second being AWS S3 which will store large image/audio/video files.
- Having the data layer independent and only accepts communication from the application layer allows us to enforce security of our data. It also prevents loss of data since the databases run in their own instances and are not affected by the other layers in the case of a client or server crash.

9. Deploy

Our project uses AWS Elastic Beanstalk to deploy our app. This allows us to easily scale our web applications. Elastic Beanstalk automatically launches an environment. It creates and configures the AWS resources needed to run our application. After the environment is launched, we can then manage the environment and deploy new application versions.