Release engineering Case study.

Tech Design Template.

Link to architectural Diagram: GitHub Diagrams with screenshots

PROJECT:

Creating a Favicon that illustrates financial independence for Cellulant Organization. The vision of the organization is to enable seamless payment for business and banks across the African Content, with this favicon design it promotes the various payment methods while considering the client's financial status.

BACKGROUND AND REQUIREMENTS:

To be able to create a favicon that portrays the company value while considering financial independence, we need certain requirements:

Divided into two aspects the technical and non-technical, For the technical part we will need people familiar with the design and also those familiar with using VMS that can easily auto scale.

opt for AWS console for the deployment bit as you can easily auto scale the application automatically according to one's needs, monitor with Cloud Watch.

Knowledge of programming language for implementing the API.

PROBLEM STATEMENT:

The intent is to create a design that allows financial independence for the end-users while trying to accomplish the Cellulant mission of creating opportunities that accelerate economic growth in Africa.

A user is supposed to view the design as a symbol of their growth when it comes to managing their finances while making payments across different platforms. Example a user trying to access mobile Banking through Cellulant platform is able to manage their savings and earnings while carrying out transactions.

Favicon Design:

Favicon, is an image of a hand uplifting a person to reach greater heights in terms of finances, this is a symbol trying to picture financial independence gained by the user during their usage of the Cellulant Applications.

It will help build brand awareness and build credibility; icon communicates willing to go an extra mile to meet customer's needs.

API part:

We can create a basic flask api to render requests and return the different designed icon.

APPROACHES:

Since it is customer centric type of application, approaches that can be applied are Iterative approach, system prototyping method and agile approach.

Iterative is suitable as we can design a small portion give it for testing and use the feedback provided to improve on the design. Weakness takes a lot of time the end-user will have to be patient it to be fully functional. Cannot afford that rem it's a payment platform for customers across the continent.

System Prototyping Approach, can be applied too. Not so effective as development is done piece by piece and only critical parts are made available to user. This can be time consuming yet we need a design that can take effect immediately.

I would recommend the agile methodology, where the end-goal is customer satisfaction while there is always room for improvement and changes. It also takes short time to deliver the results. Example in our design we focus on financial independence while making transactions in Cellulant, customer needs may change to managing the cash flow while using Cellulant platform. This creates room for further designs that suit the customer.

(Diagrams provided)

TESTING:

As there are different teams collaborating on the project using the agile methodology, GitHub is used to track changes that could be made. We also need to test the code to ensure it meets requirements. Circleci is a CICD tool which can be connected to a GitHub account to test the build stages in AWS. With Circleci a config.yml file can be created with basic stages like checking dependency environment and the api code.

ADDITIONAL CONSIDERATIONS:

Monitoring Consideration:

Using AWS instances comes with costs and the applications might fail. Monitoring can help predict infrastructure costs, predict spikes and track bugs.

Data Aggregator – CloudWatch built in AWS or Prometheus

Recommend Prometheus for monitoring inside the launched instance easy approach

While creating the instance, add port 9090 which allows traffic to Prometheus, ssh with DNS hostname from AWS console. Can also add alert channel on the Prometheus service in case of downtime.

Operational Consideration:

Being a customer centric service, AWS autoscaling instances can be used to monitor the application and automatically adjusts capacity to maintain a predictable performance. Autoscaling can be implemented in the instance which is our VM.

Explore application, Discover what we can scale according to needs, Optimize either performance or cost depending to number of users at a particular time and auto scale application