Allergy Prediction Using Artificial Intelligence

Client Lead: Joseph Trembley
Team Lead: Noah Ross
Minute Taker: Ella Godfrey
Research Lead: Xerxes Tarman
Quality Assurance Lead: Alex Ong

Client: Ashraf Gaffar

Advisors: Ashraf Gaffar, Ashfaq Khokhar

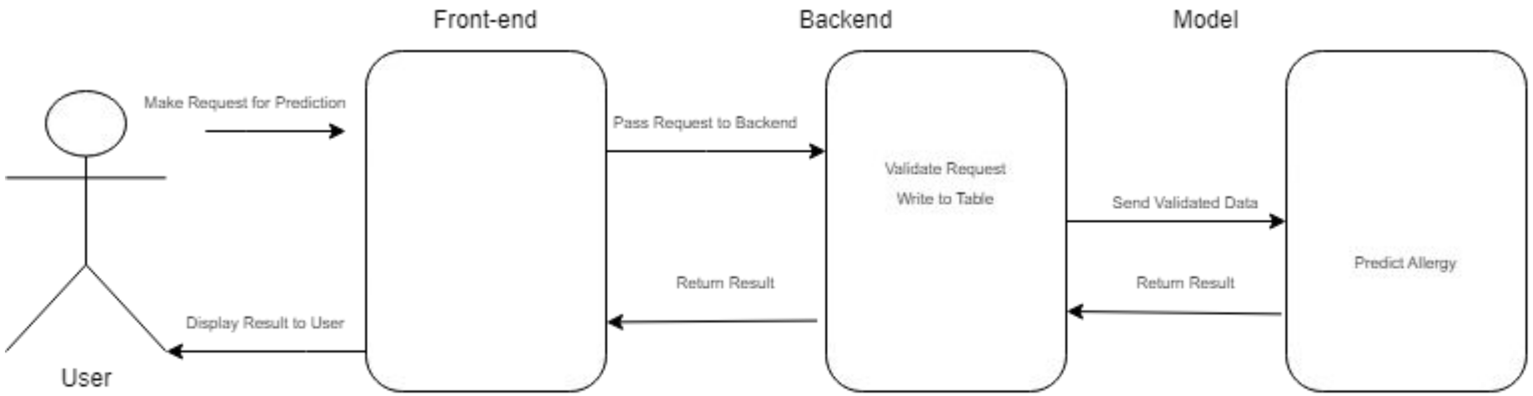
Project Vision

- The project aims to predict allergic reactions to medicines using machine learning, and optimizing testing with a rapid response time
- This benefits both healthcare providers and patients by offering faster and more efficient allergy assessments
- This will also contribute to cost reduction by eliminating the need for extensive testing procedures. The non-invasive nature reduces the need for additional patient information further improving patient experience.

Conceptual/Visual Sketch

- Intuitive and user-friendly web-based application
- Possible Users
    - Healthcare professionals
    - Medical practitioners
    - Individuals seeking rapid and efficient allergy assessments

- Integrates advanced machine learning to predict allergic reactions
    - Rapid response time
    - Minimizes invasive testing
    - Offers a cost-effective and user friendly solution



Requirements

Model Requirements:

- Accurately predicts whether a patient would have an allergic reaction to a medicine
- Able to process large number of input variables effectively

UI Requirements:

- Clear display of prediction and confidence level
- Location for user to upload information to test the model
- Visually appealing design
- Web accessibility from anywhere

Legal Requirements:

- Data collection and storage does not violate any health privacy laws

Requirements

Backend Requirements:

- Seamless data transfer from frontend to model
- Limited read/write access for database security
- Return accurate results to the frontend

Testing Requirements:

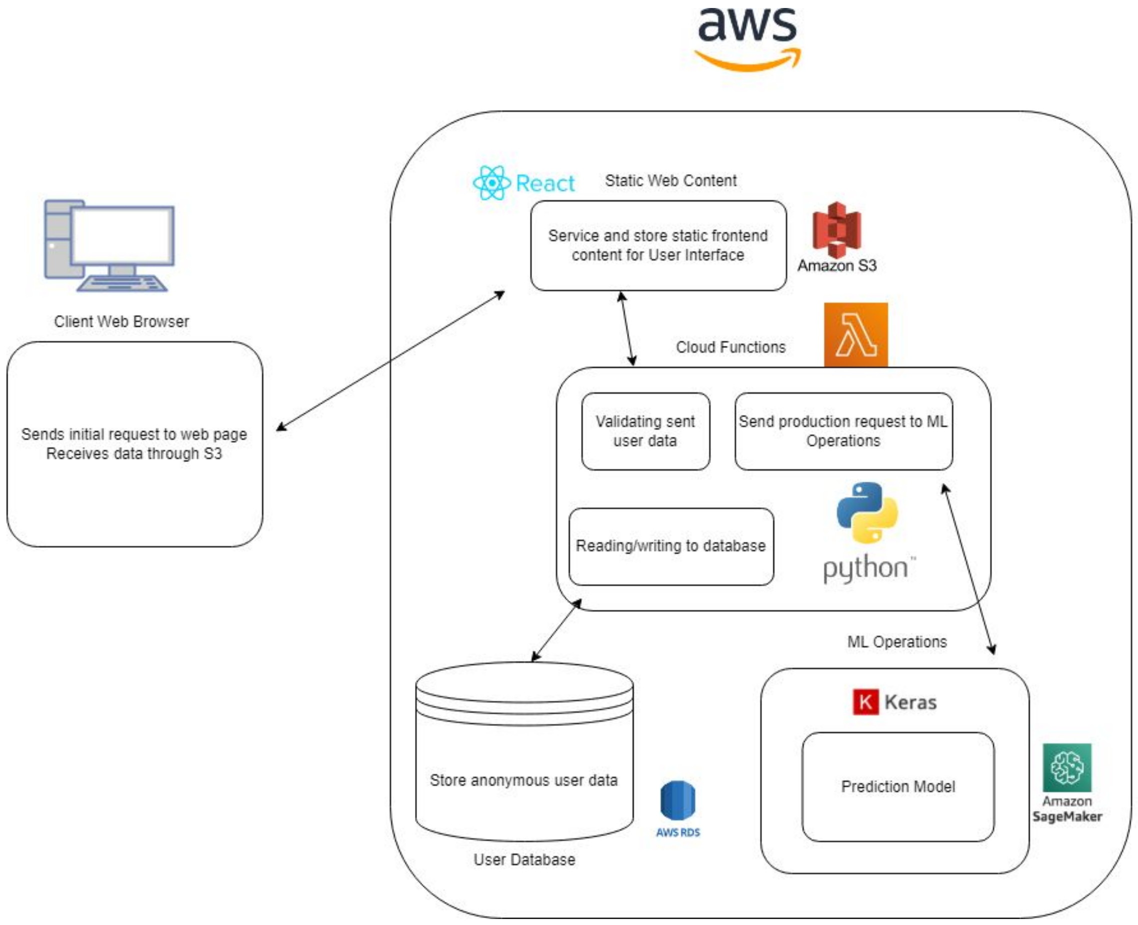
- Test model for overall accuracy percentage
- Implement logs for fault detection and error tracking

Data Requirements:

- Store data in a secure database
- Database security measures to prevent outside access

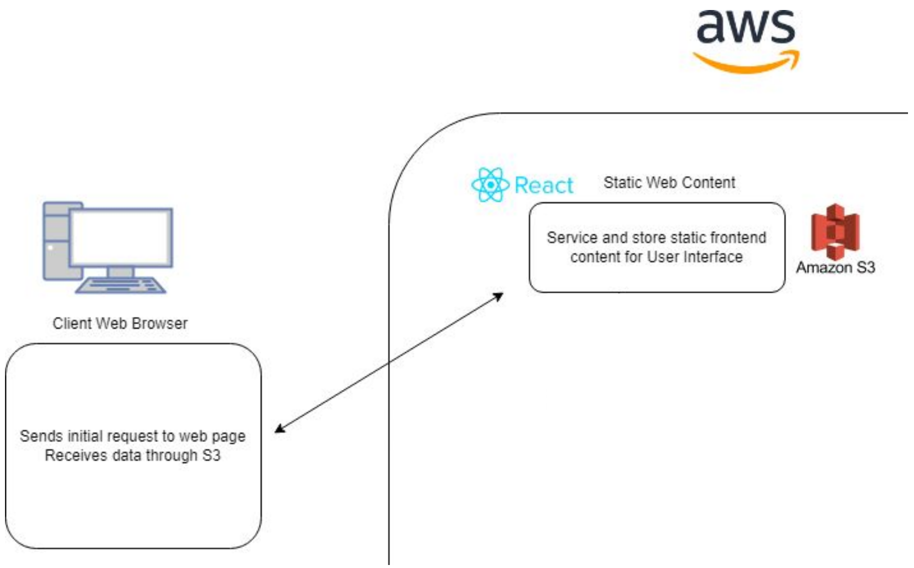
System Design Overview

- Server-Side: Data processing, validation, storing data, and running the ML model.
- Client-Side: Entering user data, calling REST API, displaying prediction results.


aws
Client Web Browser
Sends initial request to web page
Receives data through S3
React
Static Web Content
Service and store static frontend content for User Interface
Amazon S3
Cloud Functions
Validating sent user data
Send production request to ML Operations
Reading/writing to database
python
ML Operations
Keras
Prediction Model
Amazon SageMaker
Store anonymous user data
AWS RDS
User Database

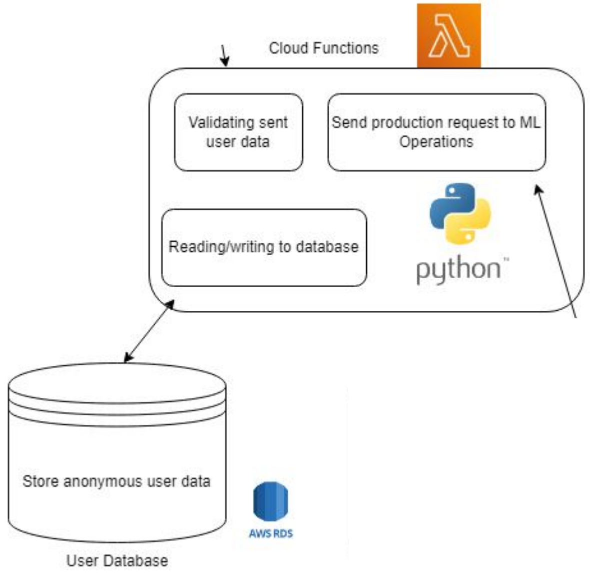
System Design UI

- Stored on S3
  - Object storage service
  - Good for static objects
- Built using React
  - Faster than HTML
  - Better responsiveness
- Only direct point of contact for user
- Displays submission form and prediction


aws
React
Static Web Content
Service and store static frontend content for User Interface
Amazon S3
Client Web Browser
Sends initial request to web page
Receives data through S3

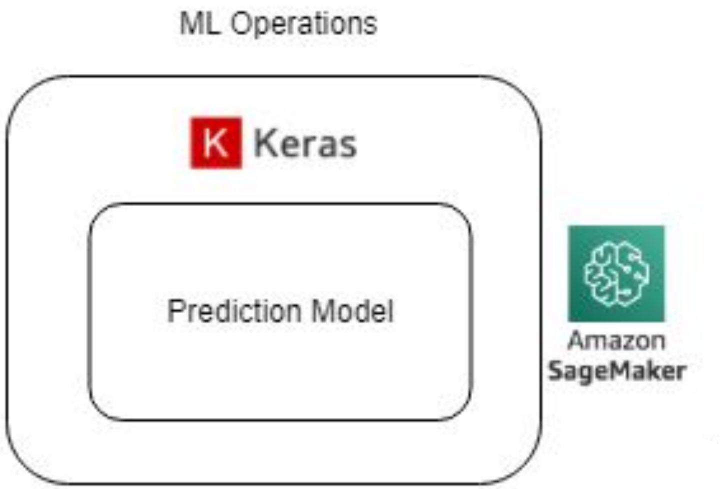
System Design Cloud Functions

- Stored in Lambda
  - Quick, on-demand functions
  - Smaller functions = more cost-effective
- Written using Python
  - Same language as model
  - Fewer lines for operations
- Call through REST API
- Validates user data
- Reads and writes to database


Cloud Functions
Validating sent user data
Send production request to ML Operations
Reading/writing to database
python
Store anonymous user data
AWS RDS
User Database

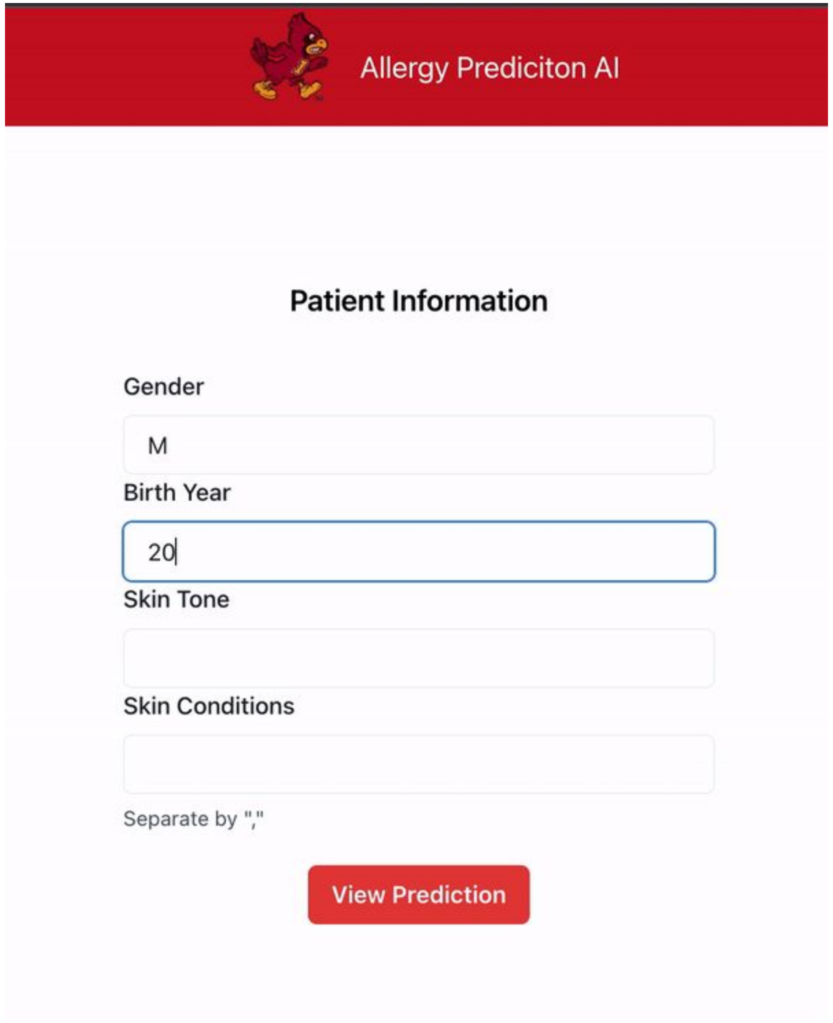
System Design Model

- SageMaker
  - Web-hosted ML training
  - Export models for future reference
- Keras API
  - Python ML library
  - Contains trainable models
- Triggered Via Lambda Function
- Predicts allergic reaction



Prototype Implementation - Front-End

- Implemented using React.js
- Communicates with AWS endpoint
- Additional & finalized input parameters in future iterations



Allergy Prediciton AI

Patient Information

Gender

M

Birth Year

20

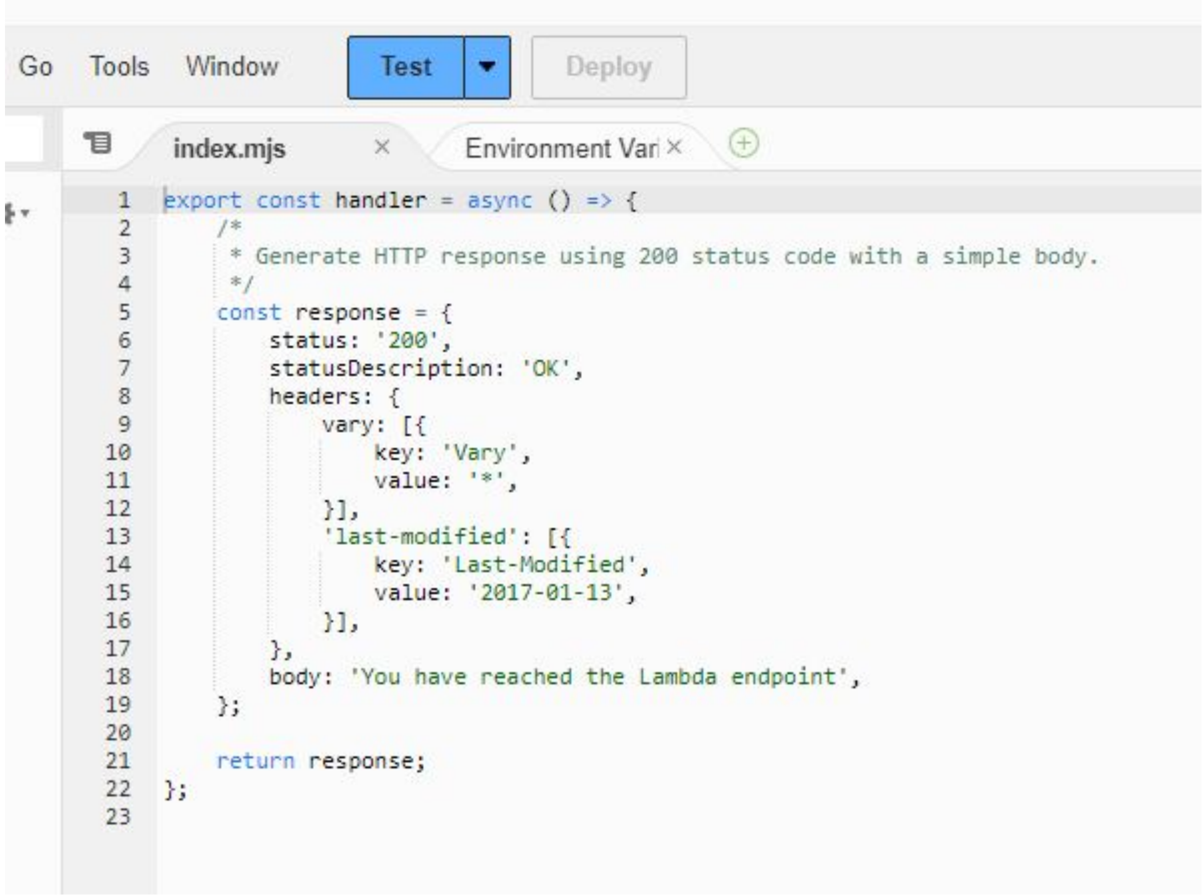
Skin Tone

Skin Conditions

Separate by ","

Prototype Implementation - Backend

- Simple request/response
- Does concept work?
- JavaScript instead of Python



```
export const handler = async () => {
    /*
     * Generate HTTP response using 200 status code with a simple body.
     */
    const response = {
        status: '200',
        statusDescription: 'OK',
        headers: {
            vary: [{
                key: 'Vary',
                value: '*',
            }],
            'last-modified': [{
                key: 'Last-Modified',
                value: '2017-01-13',
            }],
        },
        body: 'You have reached the Lambda endpoint',
    };

    return response;
};
```

Design Complexity

- Frontend
  - React-based user interface
  - Ensuring a seamless user-friendly experience
  - Implementing an effective mechanism for data upload
  - Designing a visually intuitive user interface
- Backend
  - Complex cloud function
  - Implementing secure data transfer
  - Managing the complexity of forwarding model results to the frontend
  - Ensuring robustness in cloud functions
- Machine Learning Model
  - Neural networking for allergy prediction
  - Training a model with neural networking to establish connections between various allergy-related factors
  - Overcoming challenges associated with accuracy in machine learning models
  - Building a robust testing suite to prevent overfitting

Project Plan - Tasks

- Predict allergic reactions
  - Develop a machine learning model using Keras to predict allergic reactions to medicines
  - Train and test the model with the given data to ensure accuracy
- Support large input sets
  - Ensure the system can efficiently process a variety of input variables
  - We will use AWS to handle large input sets
- Data training and testing
  - Iterate on training to achieve desired accuracy
- Application interface
  - Develop frontend in React and backend with Lambda cloud functions
- Logging Implementation
  - Implement logging mechanisms for error, info, and success at all levels as well as facilitate troubleshooting
- Continuous Testing
  - Test the application at every stage of development and address potential issues early in the development process

Project Plan - Mitigation Strategy

Risks:

- Inaccurate model:
  - Could arise from lack of common links between variables or insufficient training
- Connectivity issues:
  - Difficulty in communication between frontend, backend, and model
- Data uploading issues:
  - Tricky formatting for user-provided data
- Error handling/logs:
  - Lack of logging and descriptive error messages
- Code structure/format:
  - Unclear file structure impacting model functionality

Mitigations:

- Compare AI models to others for benchmarking
- Test frontend, backend, and model individually and together
- Use common formats (.csv) to be accessible
- Handle logs in each separate component
- List conventions in README and enforce simple and clear coding structures with consistent formatting

Project Plan - Milestones

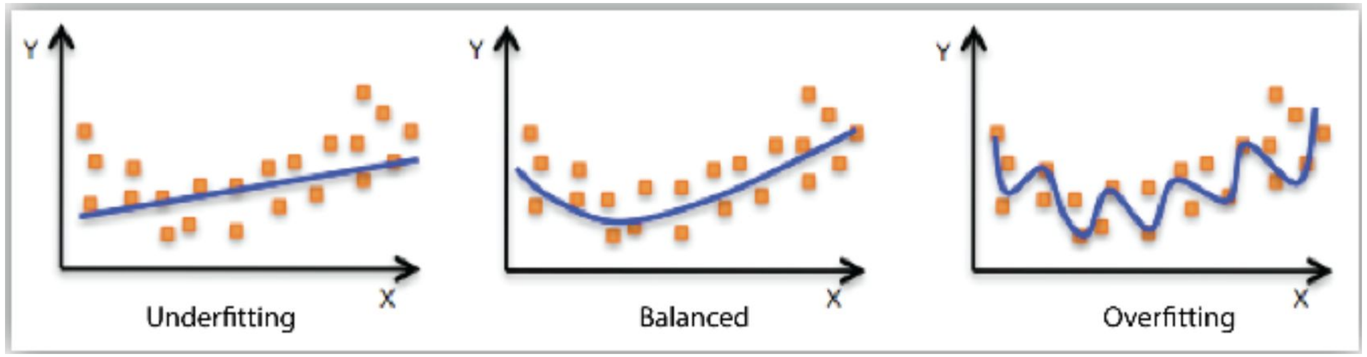
- Data Milestones:
    - Determine appropriate prediction for dataset (Classification, Clustering, Regression, Ranking)
    - Establish data collection mechanisms (mysql and python)
    - Quality of data is established (null/invalid values < 5% of data)
    - Data has large number of input variables (number of features >= 20)
- Model milestones:
    - Model is created (accuracy > 50%), refined (> 80%), and finished (> 90%)
    - Rapid Response (total request time < 5 seconds)
    - UI easy to use (< 5 clicks to submit)

Unit Tests

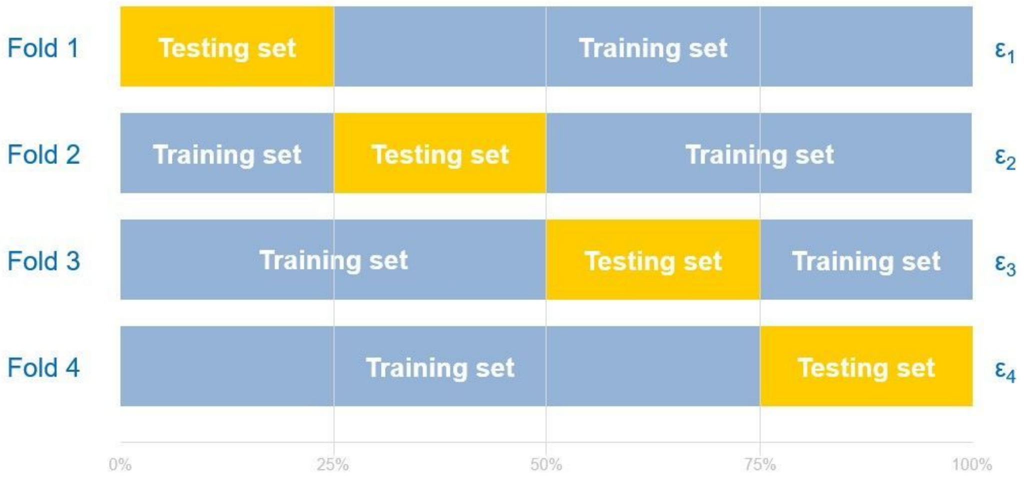
- AI Model:
  - Test the model for accuracy
  - Split Data Set, one part for training, one part for testing
  - CI/CD pipeline ensure tests run automatically
  - Baseline Model for reference point
- Frontend Component
  - Test buttons and text input fields using React Testing Library
  - Jest to run the tests and confirm if they fail or succeed

Model Testing

- Our planned method for model testing and regression will be k-fold cross-validation.
- Jupyter Notebook (Hosted on AWS)
- Useful for picking an initial ML method



4-fold validation (k=4)


Fold 1: Testing set | Training set | ε_1

Fold 2: Training set | Testing set | Training set | ε_2

Fold 3: Training set | Testing set | Training set | ε_3

Fold 4: Training set | Testing set | ε_4

Interface Tests

Frontend Tools:

- React Testing Library - virtual DOM for the tests to run in
- Jest - test individual components

Backend Tools:

- Python Unit Test - Python testing library

Acceptance Tests

Traceability

- Alignment between design requirements and testing phases
- Map each requirement to a set of tests

Client Involvement

- Have the client participate in testing the project
- Evaluate if the software performs as expected from the client
- Feedback from client such as deviations from requirements

Conclusion and current progress:

End of Planning:

- We settled on AWS for our first design
- Tested individual AWS services to understand project format

Moving Forwards:

- Create model using real data
- Expand and alter testing components