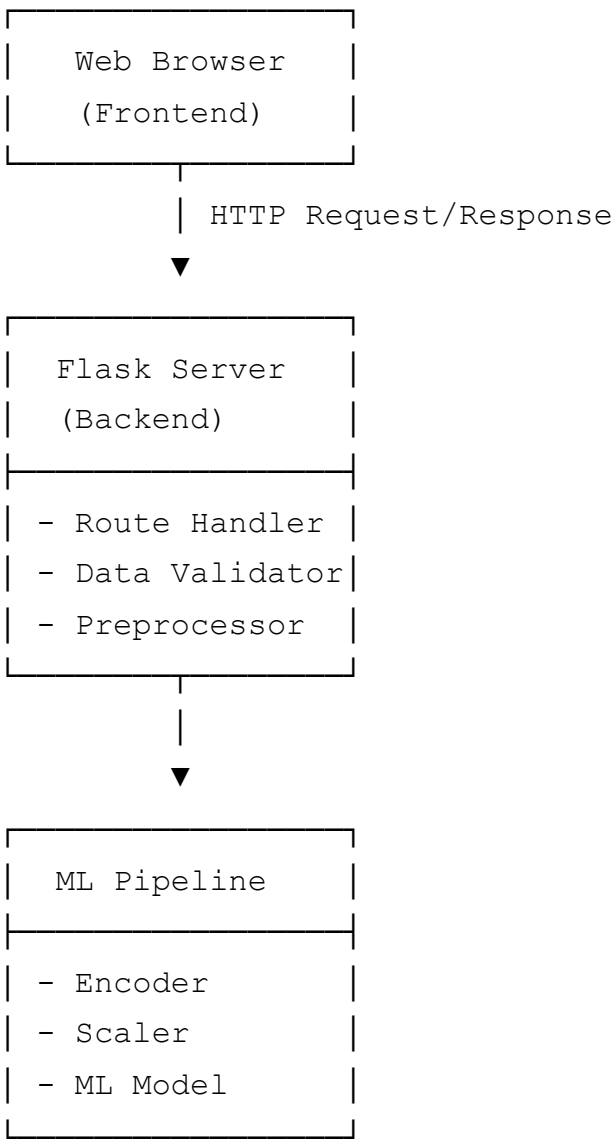
3. Project Design Phase

System Architecture

High-Level Architecture



Component Design

1. Frontend Components

1.1 Input Form (index.html)

- **Purpose**: Collect meteorological parameters from users
- **Features**:
  - Responsive grid layout (3 columns)
  - Form validation
  - Clear labels and placeholders
  - Submit button with gradient styling
- **Technologies**: HTML5, CSS3

1.2 Result Pages

chance.html (Rain Expected)

- Full-screen background image (rainy scene)
- Animated raindrops effect
- White text overlay with shadow
- "Check Again" button

noChance.html (No Rain)

- Full-screen background image (sunny beach)
- White text overlay with shadow
- "Check Again" button

2. Backend Components

2.1 Flask Application (app.py)

Routes:

- `GET /` - Render input form
- `POST /predict` - Process prediction request

Functions:

- `load_artifacts()` - Load ML model and preprocessors
- `index()` - Serve home page
- `predict()` - Handle prediction logic

2.2 Data Processing Pipeline

```
Input Data → Validation → Encoding → Scaling → Prediction → Result
```

Steps:

1. Extract form data
2. Create pandas DataFrame
3. Encode categorical features
4. Scale numerical features
5. Make prediction
6. Return appropriate template

3. Machine Learning Components

3.1 Model Training (Jupyter Notebook)

Process:

1. Data loading and exploration
2. Data cleaning and preprocessing
3. Feature engineering
4. Model training (Multiple algorithms)
5. Model evaluation and selection
6. Model serialization

Models Evaluated:

- Random Forest Classifier
- Decision Tree Classifier
- K-Nearest Neighbors
- XGBoost Classifier

3.2 Preprocessing Artifacts

- **encoder.pkl**: Label encoders for categorical features
- **scaler.pkl**: StandardScaler for numerical features
- **imputer.pkl**: Missing value imputer
- **Rainfall.pkl**: Trained ML model

Database Design

Note: Current version uses in-memory processing. No database required.

Future Enhancement:

- Store prediction history
- User profiles
- Historical weather data

UI/UX Design

Color Scheme

Rain Expected Page:

- Background: Rainy field image with 70% brightness
- Text: White with black shadow
- Button: White with transparency

No Rain Page:

- Background: Beach scene with 85% brightness
- Text: White with black shadow
- Button: White with transparency

Input Form:

- Primary: Purple gradient (#667eea to #764ba2)
- Background: White
- Borders: Light gray (#e0e0e0)
- Focus: Purple (#667eea)

Typography

- Font Family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif
- Heading Size: 2.5em - 4em
- Body Text: 1em - 1.1em
- Button Text: 1.1em - 1.2em

Responsive Design

- Mobile: Single column layout
- Tablet: 2 column grid
- Desktop: 3 column grid
- Breakpoint: 250px minimum column width

API Design

Endpoint: POST /predict

Request:

```
{
  "Location": "Sydney",
  "MinTemp": 18.5,
  "MaxTemp": 22.3,
  "Rainfall": 8.5,
  "Evaporation": 2.4,
  "Sunshine": 3.2,
  "WindGustDir": "W",
  "WindGustSpeed": 44.0,
  "WindDir9am": "SW",
  "WindDir3pm": "W",
  "WindSpeed9am": 20.0,
  "WindSpeed3pm": 28.0,
  "Humidity9am": 85.0,
  "Humidity3pm": 75.0,
  "Pressure9am": 1008.5,
  "Pressure3pm": 1006.2,
  "Cloud9am": 7,
  "Cloud3pm": 8,
  "Temp9am": 19.5,
  "Temp3pm": 21.0,
  "RainToday": "Yes"
}
```

Response:

- Success: HTML page (chance.html or noChance.html)
- Error: Error message with HTTP status code

Security Design

Input Validation

- Type checking for all inputs
- Range validation for numerical values
- Whitelist validation for categorical values

Error Handling

- Try-catch blocks for all critical operations

- Graceful degradation
- User-friendly error messages
- No sensitive information exposure

Performance Optimization

Model Loading

- Load artifacts once at startup
- Keep models in memory
- Lazy loading if needed

Response Time

- Minimize preprocessing steps
- Efficient data structures
- Caching static resources

Scalability Considerations

- Stateless application design
- Horizontal scaling capability
- Load balancing ready