Architecture Document: Placement Predictor Web Application

1. Introduction

The Placement Predictor is a web application designed to predict placement outcomes for students based on their academic and personal details. This document provides an overview of the architecture of the Placement Predictor, detailing its components, data flow, and deployment setup.

2. System Overview

The Placement Predictor application provides a user-friendly interface for submitting student data and receiving placement predictions from multiple machine learning models. The application is built using Flask for the backend and utilizes pre-trained models for prediction. The frontend consists of a modern HTML/CSS interface.

3. Architecture Diagram

Below is a high-level architecture diagram of the Placement Predictor web application:

```
+--------------------+      +--------------------+      +--------------------+
|                    |      |                    |      |                    |
|   User Interface   | ---> |    Flask Backend   | ---> |   Machine Learning|
|    (HTML/CSS/JS)   |      |     (app.py)       |      |       Models       |
|                    |      |                    |      | (random_forest_model.pkl) |
+--------------------+      +--------------------+      |  (svm_model.pkl)   |
                    |                    |      | (logistic_regression_model.pkl)|
                    +--------------------+      +--------------------+
                              |
                              |
                              v
                    +--------------------+
                    |    Data Storage    |
                    |   (Optional)       |
                    +--------------------+
```

4. Components

4.1. User Interface

- **Description:** The frontend of the application where users input their data and receive predictions.
- **Technologies:** HTML, CSS, JavaScript
- **Responsibilities:**
  - Collect user inputs through forms.
  - Display prediction results.
  - Handle form submission and interaction with the backend using JavaScript.

4.2. Flask Backend

- **Description:** The backend server that processes user requests and interacts with machine learning models.
- **Technologies:** Python, Flask

- **Responsibilities:**
    - Serve the frontend HTML/CSS/JS files.
    - Handle form submissions via HTTP POST requests.
    - Load and utilize machine learning models for predictions.
    - Return prediction results to the frontend.

4.3. Machine Learning Models

- **Description:** Pre-trained models used for generating placement predictions based on user input.
- **Technologies:** Python, scikit-learn, joblib
- **Responsibilities:**
    - **Random Forest Model:** Predicts placement outcomes based on input features.
    - **SVM Model:** Provides an alternative prediction.
    - **Logistic Regression Model:** Offers another perspective on the placement prediction.

4.4. Data Storage

- **Description:** Optional component for storing user data or prediction logs.
- **Technologies:** SQLite, PostgreSQL, or other databases (optional)
- **Responsibilities:**
    - Store historical prediction data.
    - Manage user data (if applicable).

5. Data Flow

1. **User Interaction:**
    - Users enter their data into the HTML form and submit it.
2. **Form Submission:**
    - JavaScript captures the form data and sends it as a JSON object to the Flask backend.
3. **Backend Processing:**
    - Flask receives the data, processes it, and formats it as needed.
    - Flask loads the appropriate machine learning models.
    - Models generate predictions based on the input data.
4. **Prediction Results:**
    - Flask compiles the results from all models.
    - Results are sent back to the frontend as a JSON response.
5. **Display Results:**
    - JavaScript receives the response and updates the HTML page to show the predictions.

6. Deployment Architecture

6.1. Deployment Platform

- **Platform:** Render.com (or any preferred cloud provider)
- **Components:**
    - **Web Server:** Hosts the Flask application.

- **Static Assets:** Served from a CDN or directly from the web server.

6.2. Deployment Steps

1. **Prepare Application:**
    - Ensure all dependencies are listed in `requirements.txt`.
    - Create a `Procfile` to specify the web server startup command.
2. **Push to Repository:**
    - Push the code to a Git repository (e.g., GitHub).
3. **Configure Deployment:**
    - Connect the repository to Render.com.
    - Set up build and run commands as per Render.com requirements.
4. **Monitor and Scale:**
    - Monitor application performance and adjust scaling settings as needed.

7. Security Considerations

- **Input Validation:** Ensure all user inputs are validated and sanitized to prevent malicious data from compromising the system.
- **HTTPS:** Use HTTPS to secure data in transit between users and the server.
- **Model Security:** Ensure that machine learning models are not exposed to unauthorized access or tampering.

8. Performance Considerations

- **Caching:** Implement caching for static assets to improve load times.
- **Load Balancing:** Consider load balancing if the application experiences high traffic.
- **Optimizations:** Optimize the machine learning model loading and prediction processes to reduce latency.

9. Future Enhancements

- **User Authentication:** Implement user authentication and authorization if storing user-specific data.
- **Extended Analytics:** Integrate with analytics tools to track user interactions and application performance.
- **Additional Models:** Incorporate additional machine learning models or features based on user feedback and requirements.