Tech Stack Template: HematoVision

Introduction

This document outlines the proposed technology stack for the HematoVision project, an AIpowered system for advanced blood cell classification. The selection of these technologies is based on the project's requirements for machine learning capabilities, web application development, scalability, and maintainability. This template provides a structured overview of the key components and their roles within the system.

1. Core Technologies

1.1 Programming Languages

- **Python:** The primary programming language for both the backend logic and the machine learning components. Python's extensive libraries and frameworks make it ideal for data science, AI, and web development.

1.2 Web Frameworks

- **Flask:** A lightweight and flexible Python web framework used for building the backend of the HematoVision web application. Flask is chosen for its simplicity, extensibility, and suitability for developing RESTful APIs.

1.3 Machine Learning Frameworks

- **TensorFlow/Keras:** A powerful open-source machine learning framework used for developing, training, and deploying the deep learning model for blood cell classification. Keras, as a high-level API, simplifies model definition and training workflows.

2. Frontend Technologies

2.1 Markup Language

- **HTML5:** The standard markup language for creating web pages and web applications. Used for structuring the content of the HematoVision user interface.

2.2 Styling

- **CSS3:** Used for styling the web pages, controlling the layout, colors, fonts, and overall visual presentation of the HematoVision application. This ensures a modern and userfriendly interface.

2.3 Interactivity

- **JavaScript:** A high-level, often just-in-time compiled language that conforms to the ECMAScript standard. Used for adding dynamic and interactive elements to the frontend, such as handling image uploads and displaying results asynchronously.

3. Data Management

3.1 Data Storage (Temporary)

- **Filesystem:** For temporary storage of uploaded images during the processing phase. Images are processed and then typically deleted to maintain privacy and optimize storage.

3.2 Database (Future Consideration)

- **SQLite/PostgreSQL:** While not explicitly used in the current iteration for core functionality, a relational database like SQLite (for local development/small scale) or PostgreSQL (for production/larger scale) would be considered for future enhancements, such as user management, storing historical classification results, or user preferences.

4. Development and Deployment Tools

4.1 Version Control

- **Git:** A distributed version control system used for tracking changes in source code during development. This facilitates collaboration among team members and maintains a history of all project modifications.

4.2 Development Environment

- **Jupyter Notebook:** Used for the development and experimentation of the machine learning model (`model.ipynb`). It provides an interactive environment for data exploration, model training, and evaluation.

- **Integrated Development Environment (IDE):** (e.g., VS Code, PyCharm) for writing and debugging Python code for the Flask backend and other scripts.

4.3 Package Management

- **pip:** The package installer for Python, used to manage project dependencies as listed in `requirements.txt` . This ensures that all necessary libraries are installed and consistent across different environments.

4.4 Deployment

- **Cloud Platform (e.g., AWS, Google Cloud, Azure):** For deploying the web application and machine learning model in a scalable and reliable production environment. Specific services like AWS EC2, Google App Engine, or Azure App Service could be utilized.

- **Gunicorn/uWSGI:** Production-ready WSGI HTTP servers to run the Flask application.

- **Nginx/Apache:** Web servers to act as a reverse proxy for the Flask application, handling static files and load balancing.

5. System Architecture Overview

The HematoVision system follows a typical client-server architecture:

- **Client-side (Frontend):** Users interact with the HTML/CSS/JavaScript interface through their web browsers.
- **Server-side (Backend):** The Flask application, powered by Python and TensorFlow/Keras, handles image uploads, processes them using the ML model, and serves the classification results.
- **Data Flow:** Images are uploaded from the client to the Flask backend, processed by the ML model, and results are sent back to the client for display.

6. Justification for Technology Choices

- **Python & Flask:** Chosen for rapid development, strong community support, and excellent compatibility with machine learning libraries.
- **TensorFlow/Keras:** Industry-standard for deep learning, offering robust tools for model development and deployment.
- **HTML/CSS/JavaScript:** Standard web technologies ensuring broad compatibility and a rich user experience.
- **Git:** Essential for collaborative development and version control.

- **Cloud Deployment:** Provides scalability, reliability, and global accessibility necessary for a production-grade application.

S.No.	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Flask (Jinja2)
2.	Application Logic-1	Core logic for image processing and classification.	Python / Flask
3.	Application Logic-2	Machine Learning Model Inference	TensorFlow / Keras / MobileNetV2
4.	Application Logic-3	Data handling and storage logic.	Python
5.	Database	Data Type, Configurations etc.	SQLite (for development) / PostgreSQL (for production)
6.	Cloud Database	Database Service on Cloud	(Future Consideration)
7.	File Storage	File storage requirements	Local Filesystem (temporary)
8.	External API-1	Purpose of External API used in the application	(Not explicitly defined, but could include cloud APIs for deployment)
9.	External API-2	Purpose of External API used in the application	(Not explicitly defined)
10.	Machine Learning Model	Purpose of Machine Learning Model	Deep Learning Model (MobileNetV2)
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud	Local (Jupyter, IDE) / Cloud Platform (AWS, Google Cloud, Azure)

6.1 Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Python, Flask, TensorFlow/Keras
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	(To be defined: e.g., HTTPS, input validation, secure coding practices)
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Cloud-based deployment, modular components
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Cloud load balancing, redundant deployments
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Optimized ML model, potential caching for frequently accessed data

6.2 References

1. **Python Official Documentation:** https://docs.python.org/
2. **Flask Documentation:** https://flask.palletsprojects.com/
3. **TensorFlow Documentation:** https://www.tensorflow.org/
4. **Keras Documentation:** https://keras.io/
5. **MobileNetV2 Research Paper:** https://arxiv.org/abs/1801.04381
6. **Git Documentation:** https://git-scm.com/doc
7. **Cloud Computing Platforms (AWS, Google Cloud, Azure):** Official documentation for respective services.

Conclusion

This tech stack template provides a clear roadmap for the technologies to be utilized in the HematoVision project. The chosen stack is robust, scalable, and well-suited for developing an advanced blood cell classification system. Adherence to this stack will ensure consistency, facilitate development, and support the long-term maintainability of the project.