GitGrade Analysis Report

Repository: Kartikey1405/Medibot

Language: TypeScript

Stars: 0 | Forks: 0

Final Grade: 90/100

Executive Summary:

This is a competently structured full-stack monorepo for a promising AI health assistant. The README clearly outlines a modern tech stack and features, and the separation of concerns between frontend and backend is well-executed. While the base logic for the AI prediction and web services appears solid, the overall project engineering lacks critical operational components for deployment, scalability, and maintainability.

Improvement Roadmap:

1. Implement Robust Containerization [Architecture/DevOps]

Currently, services operate as bare applications. Create individual `Dockerfile`s for both the `backend` and `frontend` services. Subsequently, implement a `docker-compose.yml` to define, link, and orchestrate these services, standardizing the development environment, simplifying dependency management, and paving the way for consistent deployments across environments.

2. Establish MLOps Practices for Model Lifecycle [MLOps]

The presence of Jupyter notebooks (e.g., `99Training1Testing.ipynb`) directly in the `backend` folder is an operational hazard. Refactor training logic out of notebooks into dedicated scripts, implement model versioning (e.g., DVC, MLflow), and establish an automated pipeline for model retraining, evaluation, and deployment to ensure reproducibility and reliable updates.

3. Formalize Data Management and Versioning [MLOps/Data Management]

Training and testing data (e.g., `Training.csv`, `Testing.csv`) are directly under `backend/data`. This basic approach is insufficient for a production-grade ML system. Introduce a data versioning tool (e.g., DVC) to track changes in datasets, ensuring reproducibility of training runs and establishing a clear data lineage. Consider moving data to a root `data/` directory or within a dedicated `ml_data/` service for clearer separation.

4. Implement Automated CI/CD Pipeline [DevOps]

There is no visible continuous integration or deployment setup. Configure a CI/CD pipeline (e.g., using GitHub Actions) to automate linting, run unit/integration tests for both frontend and backend, build Docker images, and potentially deploy to staging environments on every push to the main branch. This is non-negotiable for project reliability.

5. Centralize Configuration Management [DevOps/Security]

Environmental variables and secrets appear unmanaged. Implement a robust configuration management strategy by leveraging environment variables (e.g., `APP_ENV`, `API_KEY`) for sensitive data and varying settings. Utilize `.env` files for local development (properly `.gitignore`d) and rely on orchestrator/cloud-native mechanisms for production secrets.

6. Standardize Code Quality Enforcement [Code Quality]

While the frontend has `eslint.config.js`, there's no clear equivalent for Python. Integrate linters (e.g., Black, Flake8) and formatters (e.g., isort) into the backend development workflow. Enforce these via pre-commit hooks (e.g., `pre-commit.com`) and CI steps to maintain a consistent code style and catch common errors early across the entire monorepo.