Final Project Report: AI-Powered Laptop Support System

Version: 0.9.5 (Beta)

Status: 95% Complete

Date: July 20, 2025

1. Executive Summary

This document provides a comprehensive technical overview of the AI-Powered Laptop Support System, a multi-service application designed to automate and streamline IT support for Windows-based laptops. The system leverages a sophisticated, multilingual AI core to interpret user requests and a high-performance backend architecture to execute system-level tasks directly on the user's machine.

The project is currently at a 95% completion stage, with all core functionalities implemented and validated. This includes natural language understanding, automated software installation via WinGet, predefined development environment setup, and real-time progress feedback. The remaining 5% of the project is focused on implementing a user-facing analytics dashboard and a full-fledged, role-based authorization system to move beyond the current development-focused security model.

This report details the project's vision, architecture, technology stack, component-level implementation, and a clear roadmap for final completion. It is intended for a technical audience, including developers, system architects, and organizational stakeholders.

2. Project Vision and Target Audience

The primary vision of this project is to democratize IT support by creating an intelligent, autonomous assistant that can resolve a wide range of common technical issues instantly. By doing so, it aims to deliver significant value to a diverse audience:

- **Organizations:** The system serves as a "first line of defense" for IT helpdesks, capable of handling a high volume of repetitive requests (e.g., software installations, environment setups). This reduces operational costs, frees up skilled IT personnel for more complex problems, and enforces standardized software configurations across the organization.
- **Foundations & Educational Institutions:** By providing a powerful, free-at-the-point-of-use tool, the system can offer high-quality technical support

to communities, students, and non-profits that may lack dedicated IT resources.
- **End-users & Power Users:** The system offers a centralized, powerful, and efficient interface for managing personal devices, far surpassing the capabilities of standard operating system tools.

3. System Architecture

The application is designed as a distributed, multi-service monorepo, a modern architectural pattern chosen to manage the complexity of its multi-language technology stack. This architecture promotes code sharing, unified tooling, and atomic commits across different parts of the system.

```
graph TD
  subgraph User's Machine
    A[Angular Frontend]
  end

  subgraph Server Infrastructure
    B[Django Backend API]
    C[AI Classifier Service]
    D[PostgreSQL Database]
  end

  subgraph User's Machine
    E[.NET gRPC Service]
  end

  A -- REST API (HTTP Requests) --> B
  A -- WebSocket --> B
  B -- Python Call --> C
  B -- SQL --> D
  B -- gRPC --> E

  style A fill:#c2185b,stroke:#333,stroke-width:2px
  style B fill:#0277bd,stroke:#333,stroke-width:2px
  style C fill:#00695c,stroke:#333,stroke-width:2px
  style D fill:#512da8,stroke:#333,stroke-width:2px
  style E fill:#689f38,stroke:#333,stroke-width:2px
```

Architectural Flow:

1. **User Interaction:** The user interacts with the **Angular Frontend**, a Single Page Application (SPA) running in their browser.
2. **Request Handling:** A user's chat message is sent via a secure **REST API (HTTPS)** call to the **Django Backend**.

3. **Intent Classification:** The Django backend passes the user's query to the **AI Classifier Service**, a Python module that uses a fine-tuned BERT model to determine the user's intent (e.g., `app_installation`) and extract entities (e.g., `vscode`).
4. **Task Delegation:**
    - For **quick queries** (e.g., "what are my specs?"), Django directly calls the .NET service.
    - For **long-running tasks** (e.g., installations), Django sends a message via the **WebSocket** connection's channel layer to the `SupportChatConsumer`.
5. **System-Level Execution:** The `SupportChatConsumer` initiates a **gRPC** call to the **.NET gRPC Service**, which runs with the necessary permissions on the user's machine to perform system-level actions like running WinGet commands.
6. **Real-time Feedback:** The .NET service streams `ProgressUpdate` messages back to the Django consumer via gRPC. The consumer then forwards these messages over the WebSocket connection to the Angular frontend, providing a live, seamless user experience.
7. **Data Persistence:** All chat messages and operational logs are stored in the **PostgreSQL Database**.

4. Technology Stack Deep Dive

The technology stack was carefully chosen to balance performance, scalability, and developer productivity across the different domains of the application.

- **Frontend (Angular 20):** Chosen for its robust framework for building complex, scalable SPAs. The use of **standalone components** and the **Signals API** represents a modern, performance-first approach to Angular development, reducing boilerplate and improving change detection efficiency.
- **Backend API (Django 4.2+):** Chosen for its "batteries-included" philosophy, providing a powerful ORM, built-in admin interface, and a mature ecosystem. **Django Channels** and the **Daphne** ASGI server extend Django's capabilities to handle asynchronous protocols like WebSockets natively.
- **AI Service (TensorFlow & HuggingFace):** The use of a pre-trained `bert-base-multilingual-cased` model provides a powerful foundation for natural language understanding in both English and Arabic. Fine-tuning this model on a custom dataset of IT support queries allows it to achieve high accuracy for its specific domain.
- **System Service (.NET 8 & gRPC):** gRPC was the definitive choice for communication between the backend and the on-machine service due to its high performance, low latency, and strongly-typed contract-first approach using Protocol Buffers. .NET is the natural choice for deep integration with the Windows

OS, providing direct access to system APIs, WMI, and PowerShell.
- **Database (PostgreSQL):** Chosen for its reliability, extensibility, and powerful support for advanced data types like `JSONB`, which is used for storing flexible metadata in the chat logs.

5. Component Analysis

5.1 Frontend: Angular Application

- **State Management:** The application heavily utilizes **Angular Signals** for local component state, which provides a granular and highly efficient change detection mechanism. This is particularly effective in the real-time chat interface, where frequent updates could otherwise cause performance issues.
- **Real-time Layer:** The `WebsocketService` provides a resilient, auto-reconnecting connection to the Django Channels backend. It uses RxJS Subjects to broadcast incoming progress updates to any component that needs to subscribe to them.
- **UI/UX:** The UI is built with a modern, dark-themed aesthetic using SCSS and CSS custom properties for easy theming. The layout is fully responsive, with a collapsible sidebar for use on smaller screens. A key focus is on providing clear, non-verbal feedback through distinct styling for system, error, and conversational messages.

5.2 Backend: Django Application

- **Asynchronous Core:** The backend is built around an asynchronous core, using `async def` views and running on the Daphne ASGI server. This is essential for efficiently handling a large number of concurrent WebSocket connections and I/O-bound tasks like gRPC calls.
- **Task Orchestration:** The system uses a robust pattern where the `AiRequestView` acts as a dispatcher. It receives an HTTP request, classifies it, and then sends an internal message over the channel layer to the `SupportChatConsumer`. This decouples the HTTP request from the long-running task and places the responsibility of managing the task's lifecycle (including cancellation) squarely on the consumer, which is the correct architectural pattern.
- **Security:** The application uses **Django REST Framework Simple JWT** for token-based authentication. API endpoints are secured using permission classes (`IsAuthenticated`, `IsAdminUser`), and a custom middleware (`AuditLogMiddleware`) provides a comprehensive audit trail for all API interactions.

5.3 AI Service: Intent Classifier

- **Singleton Pattern:** The heavyweight `MultilingualIntentClassifier` model is loaded only once when the Django application starts, using the `AppConfig.ready()` method. This

is a critical performance optimization that prevents the model from being reloaded on every request.
- **Preprocessing Pipeline:** The service includes a robust text preprocessing pipeline that normalizes both English and Arabic text, removes irrelevant characters, and prepares the user's query for the BERT model.
- **Local Package:** The AI service is structured as an installable Python package. This resolves complex relative import issues within the monorepo and is a best practice for managing shared code in a multi-directory Python project.

5.4 System Service: .NET gRPC Application

- **Contract-First Design:** The service is defined by the `support.proto` file, which provides a strongly-typed, language-agnostic contract for all communication. This eliminates a whole class of potential integration errors.
- **Windows Integration:** The service directly uses `System.Management` to query WMI for hardware and driver information and `System.Diagnostics.Process` to run `WinGet` commands. This deep integration is what gives the system its power to perform real actions.
- **Server-Side Interceptors:** A `ServerLoggingInterceptor` is used to log every incoming gRPC call. This is a powerful and clean way to add cross-cutting concerns like logging, metrics, or authentication to the gRPC pipeline.

6. Future Work and Roadmap

The project is primed for its final 5% of development, which focuses on production-readiness and administrative features.

- **Implement Analytics Dashboard:**
  - **Backend:** The `AnalyticsDashboardView` and `SystemMetric` model are already in place. The next step is to add logic to the C# service to periodically report metrics (like CPU/memory usage) back to the Django backend.
  - **Frontend:** A new Angular component will be created at the `/dashboard` route. This component will fetch data from the analytics endpoint and use a charting library (e.g., Chart.js or D3) to visualize the time-series and summary data.
- **Implement Role-Based Access Control (RBAC):**
  - **Backend:** The temporary `AllowAny` permission on the `AiRequestView` will be replaced with a custom permission class. This new class will check if a user belongs to an organization and has the appropriate permissions to use the service. Django's built-in `Group` and `Permission` models will be used for this.
  - **Frontend:** The UI will be updated to reflect the user's permissions. For example, the "Dashboard" and "Admin" sections may be hidden for users who

do not have the required roles.

7. Conclusion

The AI-Powered Laptop Support System successfully demonstrates a modern, robust architecture for building sophisticated, multi-service applications. By combining the strengths of Angular, Django, .NET, and a powerful AI model, it provides a seamless and effective solution to a common and costly problem. The project is well-positioned for its final development phase and subsequent deployment to its target audiences.