Project Plan

DevForge

Project Objectives

The project aims to develop an AI-powered Command Line Interface (CLI) agent that automates the process of setting up a development environment on Linux or macOS systems. Traditionally, developers must manually install tools, configure settings, and verify versions, a repetitive and error-prone process. This project proposes replacing those steps with a natural language interface where users can issue plain English commands (e.g., “Upgrade Java to version 17”), and the agent will interpret and execute the appropriate system operations.

At its core, the agent combines system-level automation with Natural Language Processing (NLP), ideally leveraging Large Language Models (LLMs) like GPT-4 or open-source alternatives. The agent should support tasks across at least three categories, such as software installation, system configuration checks, and development tool setup (e.g., installing VS Code, setting environment variables, or configuring Git). It must handle OS-specific variations (like using apt vs. brew) and execute safely prompting for confirmation when necessary and gracefully managing errors.

The project serves both a practical and educational purpose. Practically, it simplifies onboarding and environment setup for developers. Educationally, it offers experience in integrating AI with system operations, designing reliable software, and making thoughtful technology choices. Deliverables include a working CLI agent, documentation, a demonstration video, and a GitHub repository with code and usage instructions.

Ultimately, the project showcases how AI can make complex technical workflows more intuitive, efficient, and accessible through natural language interaction.

Project Timeline

Week 1(June 4-10): Foundation & Research

- Setup: Create GitHub repo, setup Jira, define team roles, review requirements
- Tech Selection: Research about LLM (OpenAI/Hugging Face), MCP frameworks
- Architecture: Finalize a potential architecture for the project

Week 2(June 11-17): Core Infrastructure

- MCP Server: Start implementing server structure
- LLM Setup: Look into chosen LLM’s prompt engineering for task interpretation
- Command Framework: Basic OS detection

Week 3(June 18-24): Tool Handlers Development

- Software Installation: Package manager detection (apt/brew), version checking

- System Configuration: Environment variables, service status, port checking
- Development Tools: Git config, VS Code installation, SSH key management

Week 4(June 24 - July 3): Advanced Features & Integration

- Container Tools: Docker installation, service management, user permissions
- Multi-step Tasks: Complex workflow handling with confirmation prompts
- Fault Tolerance: Comprehensive error handling, retry mechanisms, input validation

Week 5 (July 4-10): Testing & Refinement

- Unit Testing: Test suites for each component and OS variations
- Integration Testing: Check cross-platform compatibility, workflows etc
- Optimization: Performance tuning, bug fixes, security enhancements

Week 6(July 11-17): Documentation & UX

- Documentation: README, architecture docs, API documentation
- Examples: Usage tutorials, troubleshooting guides
- UI Polish: Enhanced CLI interface, better error messages, progress indicators

Week 7(July 18-24): Demo Preparation

- Demo Scenarios: Design compelling demonstrations covering all tool types
- Final Testing: Fresh system testing (Linux/macOS), documentation validation
- Recording: Create backup demo video and presentation materials

Week 8(July 25 – Aug 1): Final Preparation & Presentation

- Polish: Code cleanup, final documentation review, repository finalization
- Rehearsal: Practice demonstrations, prepare Q&A responses
- Presentation: Deliver live demo showcasing AI-powered automation

Roles

Project Manager – Yash Mathur: Oversees project timeline, coordinates team activities, and ensures we stick to deadlines

User Experience & Evaluation Head – Gokul Sajeevan: Ensures smooth user-agent interactions and evaluates system performance metrics

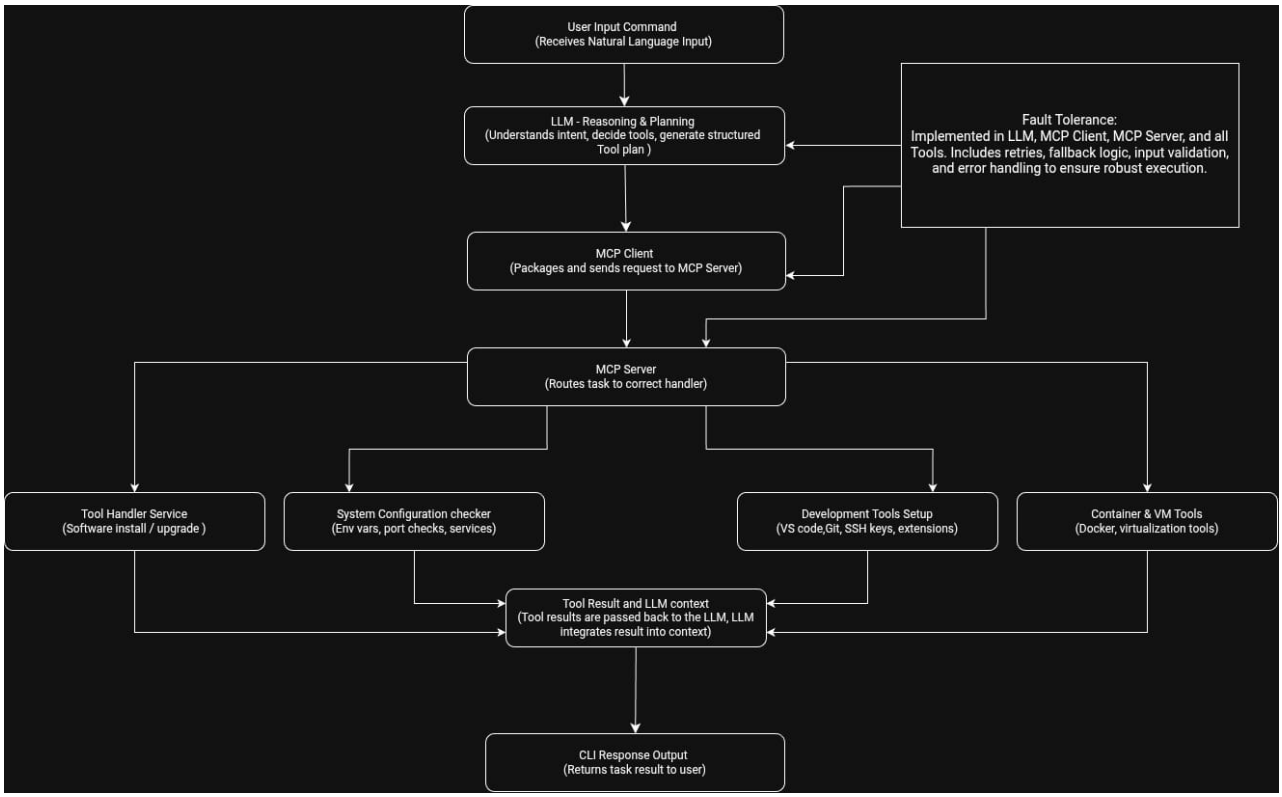
Communications Head – Martins Ejike: Ensures clear information flow between team members and external parties

Resource Manager – Kshitij Sekhar Dutta: Allocates technical resources and optimizes team productivity

Business Analyst – Nishal Koshy Philip: Analyzes requirements and defines system specifications, and ensures the AI operator solution aligns with business objectives and user needs

Software development - Yash, Gokul, Martins, Kshitij, Nishal: Collaboratively develop the AI operator system including frontend interfaces, backend structure , browser automation, and LLM integration components.

Architecture



Workflow Breakdown

1. User Input Command

Users begin by entering a natural language instruction into the terminal, such as “Install Docker and set JAVA_HOME.” This input triggers the agent workflow.

2. LLM – Reasoning & Planning

The language model interprets the user’s intent, extracts necessary components, and plans the action. It selects the appropriate tool from the available options and generates a structured function call (MCP-compatible) for execution.

3. MCP Client

The MCP Client receives the tool call from the LLM, packages it into the proper protocol format, and routes it to the appropriate MCP Server. It ensures correct formatting and communication.

4. MCP Server

The MCP Server acts as the central dispatcher. It parses incoming requests and delegates the task to the correct service handler based on the tool name and context.

5. Tool Handler Services

Each tool handler is an independent service responsible for executing specific system-level tasks. The main categories include:

- **Tool Installer:** Installs or upgrades tools like Node.js, Java, Docker, etc.
- **System Config Checker:** Verifies or modifies environment variables, port status, and running services.
- **Dev Tools Setup:** Handles Git configuration, code editor setup, extension installation, and SSH key generation.
- **Container/VM Tools:** Manages Docker, Compose, or virtual machine environments.

6. Tool Result & LLM Context

The result of the tool execution is sent back from the MCP Server through the Client to the LLM. The model integrates this result into its ongoing reasoning process to determine the next step or finalize the output.

7. CLI Output

Finally, the agent prints a human-readable message or result in the terminal, completing the interaction loop with clear feedback to the user.

Data Plan

The project requires several categories of data to function effectively. Primarily, the system will rely on user-provided input in the form of natural language commands, which will be entered through a command-line interface. This data forms the basis for interpreting the user's intent and translating it into actionable development environment setup instructions. Additionally, system-level metadata such as the underlying operating system type, installed software packages, configuration files, and current system state will be programmatically retrieved using Python's standard libraries (such as platform, os, and subprocess) or native shell commands. This metadata is essential for ensuring that the commands generated are compatible with the user's specific machine configuration.

To support the core logic of the application, the system will use a large language model (LLM) accessed via the OpenAI API. This LLM will process user input along with system metadata to generate shell commands, file modifications, or installation procedures. The output from the LLM serves as a critical intermediary data source that guides the agent in automating the environment setup process. Furthermore, as part of MCP (Model Context Protocol) integration, simulated or actual project metadata such as dependency lists and project-specific tooling configurations will be utilized. These may be acquired from structured configuration files or stubbed API responses and are used to customize and optimize the environment for specific use cases.

For evaluation purposes, we will generate a curated set of test commands and their expected outputs, which will act as the benchmark for validating system performance and correctness. This evaluation data will be manually crafted and versioned to ensure repeatable testing.

In summary, the following data types will be required and used:

User input: Natural language commands provided via CLI.

System metadata: OS info, installed packages, current configurations (retrieved locally).

LLM output: AI-generated shell instructions based on user/system context.

Project metadata: MCP-defined configurations, dependency lists (from files or stubs).

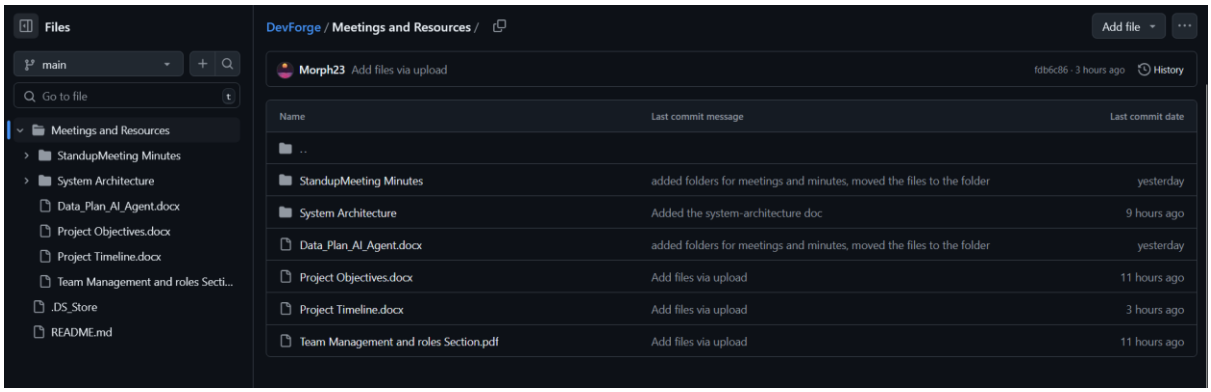
Evaluation data: Manually created test inputs and expected outputs for validation.

All data will either be interactively provided, locally inspected, fetched via APIs, or manually constructed. These data sources collectively support the development, functionality, customization, and evaluation of the AI agent as defined by the project scope.

GitHub

Thus far, every single member of the team has updated the GitHub repository of the project with their respective contributions. The project is accessible here.

As a part of this document itself, we have defined a set of goals given by the Project objectives. This was prepared by Yash. The project timeline represents our rough view at how we can progress with the project week after week and the tasks we need to perform to be successful. This timeline was made by Kshitij. The system architecture lays the groundwork for the project, showing us the complex configurations needed for our agent to work. This architecture was completed by Gokul. We will be operating on a large amount of data throughout the project and the methods to acquire this data are highlighted in the data plan done by Nishal. We are following a strict schedule of daily sprints wherein every team member is asked to be present and talk about their progress. Martins has been recording and maintaining the minutes for our meets so far, helping us track our progress. Overall, the team has been giving each other feedback and listening to feedback from our mentor for this set of initial commits.



Team Management

Our team manages the project by holding regular meetings and using tools to stay organized and on track. We meet every day on Google Meet for quick “stand-up” meetings. In these meetings, each person shares what they’ve done, what they’re working on, and if they’re facing any problems. This helps us keep up with everyone’s progress and support each other.

We also have weekly meetings where we plan the next sprint, look at what has been done, and make changes if needed. All meetings are recorded in PDF files that include the date, who was there, what we talked about, and any tasks we agreed on.

To manage our tasks, we use Jira. In Jira, we:

- Create and organize sprints.

● Assign tasks to team members.

● Track progress by moving tasks from “To Do” to “In Progress” to “Done.”

● See how the team is doing and if we’re on schedule.

For proof of our team management, we’ve included:

● PDF files with meeting minutes on our github.

Date and time of the meeting	Thursday 7th June at 10am
Names of attendees and absentees	Gokul - Present Nishal - Present Kshitij - Unavailable Yash - Present Martins - Present
Key discussion points	Main Topic • Make Project Plan • Add stories to dashboard jira • Make commit to github (architecture diagram), (minutes document) • Assign roles • Document work for the architecture • Discussion of the roles: Project management, software developer, user experience evaluation communication Feedback to incorporate from Probal “Please document all your work. All your thoughts and processes you want to follow and work. That's very important.”
Decisions made	• Meeting daily • To do Weekly sprints • Research how we’ll do different parts of our project, main technology being used will be Python • Project objectives - Yash • Project Plan - Kshitij

	• Roles • Architecture - Gokul • Data Plan - Nishal • Github - Everyone • Team Management - Martins
Next meeting details (if set)	(Standup Call) each day proposed at 10am.

Date and time of the meeting	Thursday 8th June at 11am
Names of attendees and absentees	Gokul - Present Nishal - Present Kshitij - Present Yash - Present Martins - Present
Key discussion points	Main Topic • Standup calls • Project Plan template • Progress update • Assign roles
Decisions made	• Standup calls continue every day • Template being created for project plan • Roles - Project Management - Yash Software development - Yash, Gokul, Martins, Kshitij, Nishal User experience & Evaluation - Gokul

	Communication - Martins Resource Manager - Kshitij Business Analyst - Nishal • Get parts done and put into the template as early as you can, so we can show probal
Next meeting details (if set)	(Standup Call) each day proposed at 10am.