Building the AI Tech Lead: An Installable GitHub App

Project Overview & Team Roles

This guide outlines the development of the AI Tech Lead as a professional, installable GitHub App. This approach ensures that any user can add the tool to their repository with a few clicks, rather than manually setting up servers and webhooks. The project is broken down into modules designed to be developed in parallel by a three-person team.

Team Roles and Responsibilities

This project is structured for a three-person team with distinct, specialized roles. Collaboration can be managed effectively using tools like the VS Code Live Share extension.

- **Team Lead / DevOps Engineer (Lead Role):**
  - **Focus:** Overall system architecture, integration, and deployment.
  - **Responsibilities:** Manages the GitHub repository and project board. Registers and configures the GitHub App. Writes the Dockerfile to containerize the application. Manages deployment to a cloud hosting service. Integrates the individual agents into the final CrewAI workflow.
  - **Primary Modules:** Part I (with team), Part III (Lead).
- **AI/Agent Developer:**
  - **Focus:** The core intelligence of the system.
  - **Responsibilities:** Designs and engineers the prompts for the LLM. Implements the Python logic for the Reviewer Agent and the Tester Agent, including all interactions with the Gemini API. Develops the "generate and verify" loop for the unit testing agent.
  - **Primary Modules:** Section 5 (Reviewer Agent), Section 6 (Tester Agent).
- **Platform Integration Developer:**

- **Focus:** Communication between the application and the GitHub platform.
- **Responsibilities:** Builds the Flask web server that acts as the webhook listener for the GitHub App. Implements the Watcher Agent logic to parse incoming pull request data. Implements the Reporter Agent logic to post the final, formatted review back to the pull request using the GitHub API.
- **Primary Modules:** Section 4 (Watcher Agent), Section 7 (Reporter Agent).

Part I: Project Blueprint and Environment Setup

Responsibility: All Team Members

This foundational part establishes the critical groundwork for the project. A well-prepared environment is paramount, as it prevents countless hours of frustration and streamlines the development process. All team members should complete this section to ensure their development environments are identical.

Section 1: Assembling Your Free-Tier Toolkit

This section serves as a map to acquiring all necessary services and API keys without any financial investment, navigating the landscape of student developer benefits to assemble a powerful AI application toolkit for free.

The GitHub Student Developer Pack: The Project's Foundation

The GitHub Student Developer Pack is the cornerstone of this project, providing access to professional-grade tools at no cost for students. To be eligible, an applicant must be at least 13 years old and currently enrolled in a degree or diploma-granting course of study. Verification typically requires a school-issued email address or other official, dated proof of enrollment.[1]

The key benefits from this pack that are essential for the AI Tech Lead project include:

- **Free GitHub Pro:** This grants the ability to use unlimited private repositories, ensuring

the project's source code can be developed in a secure, private setting with advanced collaboration tools.[1]

- **GitHub Copilot Pro:** Verified students receive free access to this powerful AI-powered coding companion. Copilot functions as an expert pair programmer directly within the code editor, offering real-time code suggestions. It will be an invaluable asset for writing the Python code for the AI Tech Lead itself.[1]
- **Cloud Credits:** The pack often includes credits for platforms like Microsoft Azure, Digital Ocean, and Amazon Web Services (AWS).[2] These credits are crucial for our new goal of creating an installable app, as they will allow us to host our application's backend for free.

Choosing The LLM: Google Gemini vs. OpenAI

The selection of the Large Language Model (LLM) is a pivotal decision. Given the strict zero-budget constraint, the choice is determined not just by model capability, but by the accessibility of a free and generous API tier.

- **Google Gemini API (Recommended Path):** The most direct route for this project is to leverage the **Google AI Pro plan for students**. This plan provides a free 12-month subscription to Google's premium AI services for eligible college students (18+) in supported regions.[4] This offer includes expanded access to powerful models like Gemini 2.5 Pro, which demonstrates strong performance on complex tasks such as coding.[4] After signing up for the student plan, a free-tier API key can be generated from the Google AI Studio. This key provides access to the Gemini API with rate limits that are more than sufficient for the development and testing phases of this project.[8]
- **OpenAI API (Alternative Path):** While OpenAI models are powerful, securing free API access is less straightforward. OpenAI does not currently offer a standard, ongoing free API tier for new users simply upon sign-up. The "free tier" often mentioned in documentation typically refers to promotional or trial credits, which are no longer a standard offering.[10] The viable paths for students to gain free access are more conditional:
    - **University Partnerships & Hackathons:** Some universities or specific events may have agreements that provide temporary API access.[12]
    - **Researcher Access Program:** This is a formal program where students and researchers can apply for up to $1,000 in API credits by submitting a research proposal. The proposal must align with OpenAI's research priorities, such as the responsible deployment of AI. This is a legitimate path but involves an application and quarterly review process, introducing a delay and uncertainty that may not be

suitable for a college project with a fixed timeline.[12]

The project's primary constraint of zero cost makes reliable, free API access a non-negotiable requirement. Google's explicit and easily claimable AI Pro plan for students presents a clear and immediate path forward. In contrast, obtaining free OpenAI API access is contingent on factors outside a student's direct control, such as the availability of temporary promotions or the acceptance of a research application. This introduces a significant risk and potential roadblock to the project's timeline and feasibility. Therefore, the most strategic and risk-verse decision is to build the AI Tech Lead primarily on the Gemini API, capitalizing on the guaranteed free access provided to the student development community.

Table 1: Free-Tier Resource Checklist

This table serves as an actionable checklist for setting up the project's foundational services. It centralizes all required sign-ups, ensuring no steps are missed and progress can be easily tracked.

Service/Program	Key Benefit for Project	Direct Sign-up Link	Action Item	Status
GitHub Student Developer Pack	Free GitHub Pro, Copilot Pro, Cloud Credits	education.github.com/pack	Apply and verify student status.	☐ To Do / ☑ Done
Google AI Pro for Students	Free 12-month access to Gemini 2.5 Pro	gemini.google/students	Sign up with a personal Google account.	☐ To Do / ☑ Done
Google AI Studio	Free-tier Gemini API Key	aistudio.google.com/app/apikey	Generate API Key after signing up for AI Pro.	☐ To Do / ☑ Done

Section 2: Establishing Your Development Environment

This section details the construction of the digital workshop. The necessary software tools will be installed and configured to work together seamlessly, creating a professional and efficient coding environment.

Installing Python

A modern version of the Python interpreter is the first requirement. The CrewAI framework specifies a version of Python 3.10 or newer.[16] Installation packages for all major operating systems (Windows, macOS, and Linux) are available from the official Python website, and detailed setup guides are available for each platform.[18]

The Power of Virtual Environments (venv)

In Python development, using a virtual environment for each project is a critical best practice. A virtual environment is an isolated directory that contains a specific Python interpreter and its own set of installed libraries, independent from the global Python installation and other projects.[19] This practice prevents "dependency hell," a situation where two different projects on the same machine require conflicting versions of the same library.[21] For an AI project, where library versions can be highly specific, this isolation is non-negotiable.

The process involves three simple commands executed in a terminal within the project's root folder:

1. **Create the environment:** This command creates a new directory (commonly named venv) containing the Python installation files.[20]
   Bash
   `python3 -m venv venv`

2. **Activate the environment:** Activating the environment modifies the shell's path to prioritize the executables (like python and pip) within the venv directory. The command

differs slightly by operating system.[19]

- **macOS/Linux:**
  Bash
  source venv/bin/activate

- **Windows:**

.\venv\Scripts\activate```Once activated, the shell prompt will typically be prefixed with (venv).

3. **Deactivate the environment:** When work on the project is finished, the deactivate command returns the shell to the global Python environment.
   Bash
   deactivate

Choosing a Code Editor: The AI-Assisted Advantage

Modern code editors are more than just text editors; they are integrated development environments (IDEs) that can significantly boost productivity, especially with the integration of AI assistants.

- Option A (Recommended for Beginners): Cursor
  Cursor is an "AI-native" code editor designed from the ground up to integrate with large language models. Its installation is straightforward, involving a simple download and execution of the installer.22 For a team new to AI development, Cursor's built-in features provide a gentle learning curve. Core features like "Inline Edit" (activated with Ctrl+K), which allows modifying code via natural language prompts, and "Chat with your Code" (Ctrl+I) can help write, understand, and debug the project's code with AI assistance.[24]
- Option B (Powerful Alternative): Visual Studio Code (VS Code)
  VS Code is a highly popular and extensible code editor. A professional Python AI development environment can be configured by installing a few key extensions.18 The essential extensions for this project are:
  - **Python Extension (from Microsoft):** This provides core Python support, including linting, debugging, and interpreter management.
  - **AI Toolkit Extension:** This extension creates an integrated environment for interacting with AI models directly within VS Code. It allows for exploring model catalogs and testing prompts without leaving the editor, which is useful for the prompt engineering phase of this project.[26]

The development of the AI Tech Lead can be significantly accelerated by leveraging the very technology it seeks to emulate. The GitHub Student Developer Pack provides free access to GitHub Copilot Pro, and modern editors like Cursor and VS Code have powerful, free AI assistance features. This creates a powerful feedback loop where AI serves as a tool to construct a more advanced AI tool. The AI assistant can act as a "pair programmer," generating boilerplate code (like a basic Flask server), explaining complex syntax, suggesting bug fixes, and even helping to write the initial unit tests for the Tester Agent. This approach directly addresses the stated knowledge gap of the team and transforms the development process from a manual struggle into a guided, AI-augmented learning experience.

Section 3: Project Scaffolding and Dependencies

This stage involves laying the digital foundation of the project by creating a clean directory structure and installing the necessary Python libraries that will serve as the project's building blocks.

Creating the Project Structure

A logical folder structure is essential for keeping the project's code organized, maintainable, and easy to navigate. The crewai create command-line tool scaffolds a project with a structure that adheres to Python best practices, which will be adopted here.[17] This structure separates configuration files, source code, and custom tools into distinct directories.

Managing Dependencies with requirements.txt

The requirements.txt file is a standard convention in Python projects. It is a simple text file that lists all of the external libraries and their specific versions that the project depends on. This file allows any team member to create an identical and fully functional development environment with a single command, ensuring consistency and reproducibility.

The initial requirements.txt file for the AI Tech Lead will contain the following:

```
crewai
crewai[tools]
python-dotenv
google-generativeai
openai
PyGithub
Flask
pytest
```

Installing Core Libraries

With the virtual environment activated, all required libraries can be installed at once using pip and the requirements.txt file:

```
Bash
```

```
pip install -r requirements.txt
```

Each library plays a specific role in the architecture of the AI Tech Lead.

Table 2: Core Python Libraries

This table provides a quick reference to the project's key dependencies, explaining the purpose and role of each library within the system.

Library Name	Installation Command	Role in Project	Relevant Agent(s)

crewai	pip install crewai	The main framework for defining and orchestrating the AI agents and their collaborative workflow.[16]	All; used for overall integration.
python-dotenv	pip install python-dotenv	Manages environment variables, allowing for the secure loading of API keys from a .env file.[28]	All; for secure API key access.
google-generativeai	pip install google-generativeai	The official Python SDK for interacting with the Google Gemini API.[29]	Reviewer, Tester
openai	pip install openai	The official Python SDK for the OpenAI API. Included as a backup or for future experimentation.[28]	Reviewer, Tester
PyGithub	pip install PyGithub	A Python library to interact with the GitHub API v3, used to fetch PR data and post comments.[31]	Watcher, Reporter
Flask	pip install Flask	A micro web framework used to create the webhook listener endpoint for the Watcher Agent.[33]	Watcher

pytest	pip install pytest	A powerful testing framework used by the Tester Agent to programmatically run the unit tests it generates.[35]	Tester

---

Part II: Building the Agents, One Module at a Time

With the environment fully prepared, the focus now shifts to constructing the AI Tech Lead, piece by piece. Each team member will take ownership of their assigned modules, developing them in parallel.

Section 4: The Watcher Agent - The System's Eyes and Ears

Responsibility: Platform Integration Developer

This agent serves as the trigger for the entire automated workflow. Its sole responsibility is to listen for notifications from your GitHub App and initiate the code review process. This is achieved by building a simple web service that acts as a webhook receiver.

Understanding Webhooks

Webhooks are an event-driven mechanism for communication between applications. Instead of an application repeatedly polling an API to ask, "Is there anything new?", the webhook model allows the service (in this case, GitHub) to send a notification to a specified URL as soon as an event occurs.[36] This "don't call us, we'll call you" approach is far more efficient and provides real-time updates. A webhook configuration consists of three key components: the

Payload URL (the endpoint that receives the data), a **Secret** (a key to verify the sender's identity), and the specific **Events** to subscribe to.[37]

Building the Listener with Flask

A minimal web application using the Flask framework will serve as the webhook listener. This application will have a single endpoint that accepts POST requests from GitHub.

Python

```
# watcher_agent_server.py
import os
import json
from flask import Flask, request, abort
import hmac
import hashlib

app = Flask(__name__)

# Load the webhook secret from an environment variable for security
GITHUB_WEBHOOK_SECRET = os.environ.get('GITHUB_WEBHOOK_SECRET').encode('utf-8')

def verify_signature(payload_body, signature_header):
    """Verify that the payload was sent from GitHub."""
    if not signature_header:
        raise ValueError("x-hub-signature-256 header is missing!")
    hash_object = hmac.new(GITHUB_WEBHOOK_SECRET, msg=payload_body,
digestmod=hashlib.sha256)
    expected_signature = "sha256=" + hash_object.hexdigest()
    if not hmac.compare_digest(expected_signature, signature_header):
        raise ValueError("Request signatures didn't match!")

@app.route('/webhook', methods=)
def webhook():
    # Verify the request signature for security
    signature_header = request.headers.get('x-hub-signature-256')
    try:
        verify_signature(request.data, signature_header)
```

```
    except ValueError as e:
        print(f"Signature verification failed: {e}")
        abort(403)

    # Check if the event is a pull request
    if request.headers.get('x-github-event') == 'pull_request':
        payload = request.get_json()
        action = payload.get('action')

        # We only care about newly opened or updated pull requests
        if action in ['opened', 'synchronize']:
            pr_data = {
                "repository_name": payload['repository']['full_name'],
                "pull_request_number": payload['number'],
                "pull_request_title": payload['pull_request']['title'],
                "head_sha": payload['pull_request']['head']['sha']
            }
            print(f"Received relevant PR event: {pr_data}")

            # --- TRIGGER CREWAI WORKFLOW HERE ---
            # In Part III, this section will be integrated by the Team Lead
            # to kickoff the CrewAI crew with the pr_data.

            return 'Webhook received and processed.', 200

    return 'Event not processed.', 200

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

This script defines a /webhook route that first verifies the request's signature using the shared secret. It then parses the JSON payload to extract key information about the pull request, such as the repository name and PR number.[33]

Section 5: The Reviewer Agent - The AI's Critical Eye

Responsibility: AI/Agent Developer

This agent forms the core of the code analysis logic. Its effectiveness hinges on the quality of the instructions—the prompt—given to the LLM. This section delves into the principles of prompt engineering to guide the LLM to perform a high-quality, structured code review.

Principles of Effective Prompt Engineering

To elicit the best possible response from an LLM for a technical task, the prompt must be carefully constructed. A robust prompt for code analysis should contain four key components [40]:

1. **Persona:** Assign a role to the AI (e.g., "You are a senior Python developer..."). This primes the model to adopt a specific tone and level of expertise.
2. **Context:** Provide relevant background information, including the code snippet to be reviewed.
3. **Task:** State the objective clearly and specifically (e.g., "Review this code for bugs, style violations, and missing documentation."). Vague requests like "improve this code" lead to generic and unhelpful responses.[40]
4. **Format:** Specify the desired output structure (e.g., "Provide your feedback in a JSON format..."). This is crucial for reliable automation.[41]

Crafting the Code Review Prompt

A master prompt for the Reviewer Agent will be constructed to instruct the LLM to act as an expert software engineer and check for several distinct categories of issues, drawing from established best practices.[40]

You are an expert AI programming assistant acting as a Senior Software Engineer and Tech Lead. Your task is to perform a thorough code review on the following code changes.

Analyze the provided code diff and provide feedback on the following aspects:
1. **Coding Standards Adherence:** Check for compliance with language-specific style guides (e.g., PEP 8 for Python). Identify any formatting or naming convention issues.

2. **Documentation and Comments:** Identify any functions, classes, or complex logic blocks that are missing clear docstrings or explanatory comments.
3. **Potential Bugs and Code Smells:** Analyze the code for common anti-patterns, logic errors, or "code smells" such as duplicated code, overly complex functions, or inadequate variable naming.
4. **Error Handling:** Assess whether the code implements robust error handling. Check for missing try-except blocks for operations that might fail (like file I/O or API calls) and ensure exceptions are handled gracefully.

Output Format:
Provide your complete feedback as a single, well-formed JSON object. The JSON object should have the following keys: "style_issues", "documentation_issues", "potential_bugs", and "error_handling_issues". Each key should correspond to a list of strings, where each string is a specific, actionable feedback point you have identified. If no issues are found for a category, provide an empty list.

Here is the code diff to review:

{code_diff_goes_here}

A key aspect of this prompt is the explicit instruction for a **structured output format**. Requesting a JSON object transforms the LLM's response from a simple, unstructured text block into machine-readable data. This is a critical design choice. If the Reviewer Agent produced a prose response, the Reporter Agent would require complex, brittle parsing logic (or another LLM call) to extract the relevant information. By enforcing a JSON output, the system becomes far more robust and easier to maintain, as the Reporter Agent can simply parse the JSON and iterate through the lists to format its final comment.

Integrating with the Gemini API

The Python code for this agent will use the google-generativeai library to interact with the Gemini model.[29] The process involves:

1. **Fetch Code Changes:** Use the PyGithub library and the pull request data from the Watcher to fetch the "diff" of the PR, which shows the exact lines of code that were added, removed, or modified.
2. **Construct the Full Prompt:** Inject the fetched diff into the master prompt template.

3. **Call the API:** Send the complete prompt to the Gemini API.
4. **Parse the Response:** Receive the model's response, which will be a string containing the JSON object, and use Python's json library to parse it into a dictionary. This dictionary is the final output of the Reviewer Agent.

Section 6: The Tester Agent - The AI as a QA Engineer

Responsibility: AI/Agent Developer

This agent represents the most ambitious and technically complex component of the project. It will be tasked with not only generating unit tests for new code but also executing them and verifying the results. This requires a "generate and verify" loop to overcome the common challenges of AI-generated code.

The Challenge of AI Test Generation

LLMs are capable of generating unit tests, but without careful guidance, these tests can be flawed. Common issues include syntax errors, logical fallacies, testing trivial behavior while ignoring edge cases, or simply failing to compile.[45] The strategy here is not to blindly trust the generated code but to create a system that can programmatically validate it.

Step 1: Code Parsing

The agent's first step is to understand the code it needs to test. It will receive the code diff from the pull request and must parse it to identify new or modified functions. Simple string manipulation or regular expressions can be used to extract function definitions, including their names, parameters, and docstrings.

Step 2: Prompting for pytest Unit Tests

A highly specific prompt is required to guide the LLM in generating useful tests. The prompt will include the source code of the target function and a set of clear instructions.

```
You are an expert Python QA Engineer specializing in automated testing with the pytest framework. Your task is to write a comprehensive suite of unit tests for the following Python function.

**Function to Test:**
---
{function_source_code}
---

**Requirements for the Test Suite:**
1. Use the `pytest` framework.
2. Do not include the function definition in your response; only provide the test code.
3. Write multiple test functions to cover a variety of scenarios.
4. Include tests for valid, expected inputs to verify the "happy path."
5. Include tests for common edge cases, such as `0`, `None`, empty strings, or empty lists.
6. If the function is expected to raise exceptions for invalid inputs, include tests that use `pytest.raises` to verify this behavior.
7. Ensure all test functions have descriptive names, like `test_function_name_with_specific_input`.
```

This prompt leverages best practices by specifying the framework (pytest), providing clear constraints, and guiding the model to consider different test categories, such as edge cases and error conditions.[47]

Step 3: Programmatically Running the Tests

Once the LLM returns the generated test code, the agent must execute it.

1. **Save the Test Code:** The agent will write the generated Python code to a temporary file, for example, temp_test_suite.py. It will also save the source code of the function under test to another file that can be imported by the test suite.
2. **Execute pytest:** The agent will then invoke the pytest framework programmatically. This

can be done using Python's built-in subprocess module to run pytest as a command-line process or by calling the pytest.main() function directly from the script.[50]
3. **Capture Output:** It is critical to capture the standard output (stdout), standard error (stderr), and the final exit code of the pytest process. This data provides the definitive result of the test run.[51]

Step 4: Analyzing the Results

The captured output from the pytest execution determines the success or failure of the generated tests.

- An **exit code of 0** signifies that all generated tests passed successfully.
- A **non-zero exit code** indicates that one or more tests failed or an error occurred during execution.
- The agent will package the result—a simple pass/fail status along with the complete captured output log—as its final deliverable. This information will then be passed to the Reporter Agent.

Section 7: The Reporter Agent - The Voice of the AI Tech Lead

Responsibility: Platform Integration Developer

The final agent in the workflow is responsible for communication. It synthesizes the findings from the Reviewer and Tester agents into a single, coherent, and easy-to-read comment for the human developers on the pull request.

Collecting the Data

The Reporter Agent's inputs are the structured outputs from the preceding agents:

- From the **Reviewer Agent:** A Python dictionary containing categorized lists of feedback (e.g., {'style_issues': [...], 'potential_bugs': [...]}).
- From the **Tester Agent:** A status (e.g., "PASS" or "FAIL") and the full text log from the

pytest execution.

Formatting the Report

The agent will use Python's string formatting capabilities to construct a report in Markdown, which allows for rich formatting like headings, bullet points, and code blocks in GitHub/GitLab comments.

The report will have a clear, hierarchical structure:

🤖 AI Tech Lead Review

Summary: 3 style suggestions found. AI-generated unit tests **PASSED**.

📝 Code Review Analysis

Potential Bugs & Logic Issues:

- The loop on line 42 may result in an off-by-one error under certain conditions.

Style & Readability Suggestions:

- The variable x on line 15 could be renamed to user_index for clarity.
- Consider refactoring the process_data function into smaller, single-responsibility functions.

Documentation:

- The calculate_metrics function is missing a docstring explaining its parameters and return value.

☑️ Unit Test Results: PASSED

Posting the Comment via API

The final step is to post this formatted Markdown string to the correct pull request. This is accomplished using the platform's API via a Python library.

- **GitHub:** The PyGithub library will be used. A key detail is that general comments on a pull request are made through the "Issues" API, as GitHub treats every pull request as a special type of issue.[52] The agent will use the repository name and PR number to get the PullRequest object and then call its create_issue_comment() method.[54]
- **GitLab:** The python-gitlab library will be used. The agent will get the ProjectMergeRequest object and use its notes.create() method to post the comment.[56]

Part III: Integration, Deployment, and Showcase

Responsibility: Team Lead / DevOps Engineer

In this final part, the individual agent modules are brought together into a single, cohesive application. The focus shifts from individual component development to system-level architecture, deployment, and preparing the project for a compelling demonstration.

Section 8: From Webhook to Installable GitHub App

To make the AI Tech Lead a tool that anyone can easily use, we will package it as a GitHub App instead of using a simple repository webhook. GitHub Apps are the preferred method for building integrations as they offer fine-grained permissions, better security, and a professional installation flow for users.

Registering Your GitHub App

1. **Navigate to Developer Settings:** In your GitHub profile, go to Settings > Developer

settings > GitHub Apps.
2. **Create New App:** Click "New GitHub App".
3. **App Name & Homepage:** Give your app a unique name (e.g., "AI Tech Lead Assistant") and set the homepage URL to your project's GitHub repository.
4. **Webhook Configuration:**
    - **Activate Webhooks:** Check the "Active" box.
    - **Webhook URL:** This is the public URL where your deployed Flask application will be running. During development, you can use a service like ngrok to get a temporary public URL that forwards to your local machine.[62] For the final product, this will be the URL from your cloud hosting provider.
    - **Webhook Secret:** Create a strong, random string. This secret is used to verify that webhook payloads are genuinely from GitHub.[63]
5. **Repository Permissions:** This is the most critical step for security. Grant only the permissions your app absolutely needs. For this project:
    - **Pull Requests:** Set to Read & write. This allows the app to read PR data (like the code diff) and write comments.
6. **Subscribe to Events:** Tell GitHub which events your app cares about.
    - Check the box for **Pull request**. This will send a notification whenever a pull request is opened, updated, closed, etc.
7. **Create and Install:** Click "Create GitHub App". After creation, you will need to install the app on the repository you want it to monitor.

Section 9: Containerizing with Docker

To ensure the application runs consistently anywhere, we will package it into a Docker container. A container bundles the application code, the Python interpreter, and all its dependencies into a single, portable unit.

Creating the Dockerfile

In the root directory of your project, create a file named Dockerfile (no extension) with the following instructions:

Dockerfile

```
# Use an official, lightweight Python image as the base
FROM python:3.10-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the dependencies file first to leverage Docker's layer caching
COPY requirements.txt requirements.txt

# Install the Python dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application's source code into the container
COPY..

# Expose the port that the Flask app will run on
EXPOSE 5000

# Define the command to run the application when the container starts
CMD ["python", "src/ai_tech_lead_project/watcher_server.py"]
```

This file provides a complete blueprint for building a portable image of your application.

Section 10: Orchestration and Final Integration

The Team Lead's final coding task is to assemble the components built by the team into a cohesive workflow using CrewAI.

Assembling the Crew

The CrewAI framework will be used to manage the dependencies and flow of information between the specialized agents, turning them from isolated components into a collaborative team.[58] This work will be done in a central

crew.py file.

Python

```
# src/ai_tech_lead/crew.py
from crewai import Agent, Task, Crew, Process
from.agents import reviewer_agent, tester_agent, reporter_agent
from.tasks import review_task, test_task, report_task

# Define the crew
code_review_crew = Crew(
    agents=[reviewer_agent, tester_agent, reporter_agent],
    tasks=[review_task, test_task, report_task],
    process=Process.sequential,
    verbose=True
)
```

The review_task, test_task, and report_task will be defined as Task objects. Crucially, task dependencies will be established using the context parameter. For example, the report_task will have a context that includes the outputs of both the review_task and the test_task, ensuring the Reporter Agent has all the necessary information before it begins its work.[60] The

Process.sequential setting ensures that the agents execute in the correct, logical order: Reviewer -> Tester -> Reporter.

Kicking off the Crew

The Flask endpoint in the Watcher Agent's server will be modified. Instead of simply printing the pull request data, it will now be responsible for initiating the entire agentic workflow. Upon receiving a valid webhook, it will instantiate the Crew and call its kickoff() method, passing the pull request details (like repository name and PR number) as the initial input for the first task.[60]

Section 11: Deployment and Showcase Preparation

This concluding section focuses on making the project robust, secure, and ready for presentation.

Securely Managing API Keys

Hardcoding sensitive information like API keys directly into source code is a major security vulnerability. The project will be refactored to use a .env file to store these secrets. The python-dotenv library will be used to load these keys into the application's environment at runtime. This practice ensures that keys are not committed to version control.[9]

The .env file (which should be added to .gitignore) will look like this:

```
#.env
GEMINI_API_KEY="your_google_ai_studio_api_key"
GITHUB_APP_ID="your_github_app_id"
GITHUB_APP_PRIVATE_KEY="your_github_app_private_key"
GITHUB_WEBHOOK_SECRET="your_webhook_secret_string"
```

Writing a Comprehensive README.md

A high-quality README.md file is essential for any software project, especially for an academic submission. It serves as the project's front page and user manual. A template will be provided, including sections for:

- **Project Title and Description:** A brief overview of the AI Tech Lead and the problem it solves.
- **Features:** A bulleted list of the system's capabilities (e.g., Automated PEP 8 checking, AI-powered bug detection, automated unit test generation).
- **Tech Stack:** A list of the key technologies and libraries used.

- **Setup and Installation:** Step-by-step instructions on how to clone the repository, create a virtual environment, install dependencies, and set up the .env file.
- **How to Run:** The exact commands needed to start the Flask server and run the application.

Preparing Your Demonstration

A live demonstration is the most effective way to showcase the project's functionality. The following sequence provides a compelling narrative:

1. **Set the Stage:** Briefly explain the problem: code reviews are a bottleneck. Introduce the AI Tech Lead as the solution.
2. **Show the GitHub App:** Show the registered GitHub App and the repository where it has been installed.
3. **Introduce a Change:** In a local clone of the repository, create a new branch. Make a small, deliberate code change that includes both a style violation and a subtle bug.
4. **Open the Pull Request:** Commit and push the changes, then open a new pull request on GitHub.
5. **Show the "Magic":** Switch to the terminal where the deployed application's logs are visible. Point out the log message indicating that the webhook has been received and the crew has been kicked off.
6. **The Reveal:** Switch back to the pull request page in the browser. After a minute, refresh the page to show the newly posted comment from the AI Tech Lead GitHub App.
7. **Walk Through the Results:** Read through the AI-generated comment. Point out how the Reviewer Agent correctly identified the issues and how the Tester Agent's results are displayed. This powerfully demonstrates the value and successful implementation of the entire agentic system as an installable tool.

Works cited

1. Students - GitHub Education · GitHub, accessed August 26, 2025, https://github.com/education/students
2. If you're a student, you should be taking full advantage of the GitHub Student Developer Pack - Reddit, accessed August 26, 2025, https://www.reddit.com/r/learnprogramming/comments/exuh5w/if_youre_a_student_you_should_be_taking_full/
3. GitHub Student Developer Pack - Ahmed Musaad, accessed August 26, 2025, https://ahmedmusaad.com/github-student-developer-pack/
4. Google's best AI tools for college students for free - The Keyword, accessed

August 26, 2025, https://blog.google/products/gemini/google-ai-pro-students-learning/
5. Google AI Plans and Features, accessed August 26, 2025, https://one.google.com/about/google-ai-plans/
6. College students can get Google's AI Pro plan for free now. Here's how | ZDNET, accessed August 26, 2025, https://www.zdnet.com/article/college-students-can-get-googles-ai-pro-plan-for-free-now-heres-how/
7. gemini.google, accessed August 26, 2025, https://gemini.google/students/#:~:text=You%20need%20to%20sign%20up%20for%20this%20specific%20free%20Google,2.5%20Pro%20capabilities%20and%20NotebookLM).
8. Gemini Developer API Pricing | Gemini API | Google AI for Developers, accessed August 26, 2025, https://ai.google.dev/gemini-api/docs/pricing
9. Using Gemini API keys | Google AI for Developers, accessed August 26, 2025, https://ai.google.dev/gemini-api/docs/api-key
10. Pricing | OpenAI, accessed August 26, 2025, https://openai.com/api/pricing/
11. API Access using free tier - OpenAI Developer Community, accessed August 26, 2025, https://community.openai.com/t/api-access-using-free-tier/710656
12. How to Get Free OpenAI API Keys in 2025: Safe and Practical Methods - Protiviti, accessed August 26, 2025, https://www.protiviti.com/sites/default/files/webform/students_and_graduates/_sid_/openai-api.pdf
13. Researcher Access Program FAQ | OpenAI Help Center, accessed August 26, 2025, https://help.openai.com/en/articles/10139500-researcher-access-program-faq
14. OpenAI, accessed August 26, 2025, https://openai.smapply.org/
15. OpenAI Researcher Access Program, accessed August 26, 2025, https://openai.smapply.org/prog/openai_researcher_access_program/
16. crewai - PyPI, accessed August 26, 2025, https://pypi.org/project/crewai/
17. How to build a crew for CrewAI Enterprise - crewAI+ Help Center, accessed August 26, 2025, https://help.crewai.com/how-to-build-a-crew-for-crewai
18. Getting Started with Python in VS Code - Visual Studio Code, accessed August 26, 2025, https://code.visualstudio.com/docs/python/python-tutorial
19. venv — Creation of virtual environments — Python 3.13.7 ..., accessed August 26, 2025, https://docs.python.org/3/library/venv.html
20. Setting Up Virtual Environments in Python | Information Technology and Computing Support, accessed August 26, 2025, https://it.engineering.oregonstate.edu/setting-virtual-environments-python
21. How does a virtual environment work? : r/Python - Reddit, accessed August 26, 2025, https://www.reddit.com/r/Python/comments/104dfxa/how_does_a_virtual_environment_work/
22. Installation - Cursor, accessed August 26, 2025, https://docs.cursor.com/get-

started/installation
23. Cursor – Welcome, accessed August 26, 2025, https://docs.cursor.com/
24. Quickstart - Cursor Docs, accessed August 26, 2025, https://docs.cursor.com/en/get-started/quickstart
25. How to Set Up VS Code for Data Science and AI - Ultimate Guide - GeeksforGeeks, accessed August 26, 2025, https://www.geeksforgeeks.org/data-science/how-to-set-up-vs-code-for-a-data-science-project/
26. Get started with AI Toolkit for Visual Studio Code | Microsoft Learn, accessed August 26, 2025, https://learn.microsoft.com/en-us/windows/ai/toolkit/toolkit-getting-started
27. AI Toolkit for Visual Studio Code - Visual Studio Marketplace, accessed August 26, 2025, https://marketplace.visualstudio.com/items?itemName=ms-windows-ai-studio.windows-ai-studio
28. openai - PyPI, accessed August 26, 2025, https://pypi.org/project/openai/
29. Gemini API quickstart | Google AI for Developers, accessed August 26, 2025, https://ai.google.dev/gemini-api/docs/quickstart
30. LangChain 02: Pip Install OpenAI | Python - YouTube, accessed August 26, 2025, https://www.youtube.com/watch?v=pvrRrqrgHDw
31. PyGithub - PyPI, accessed August 26, 2025, https://pypi.org/project/PyGithub/
32. Introduction — PyGithub 2.7.1.dev16+gde7b0497f documentation, accessed August 26, 2025, https://pygithub.readthedocs.io/en/latest/introduction.html
33. How do I receive Github Webhooks in Python - Stack Overflow, accessed August 26, 2025, https://stackoverflow.com/questions/14536992/how-do-i-receive-github-webhooks-in-python
34. Building a Real-Time GitHub Webhook Listener with Flask, MongoDB, and React - Medium, accessed August 26, 2025, https://medium.com/@achu1997singh/building-a-real-time-github-webhook-listener-with-flask-mongodb-and-react-0ce5bac43ac0
35. Effective Python Testing With pytest, accessed August 26, 2025, https://realpython.com/pytest-python-testing/
36. Intro to Webhooks with Python - DEV Community, accessed August 26, 2025, https://dev.to/rahulbanerjee99/intro-to-webhooks-with-python-1ikh
37. Creating webhooks - GitHub Docs, accessed August 26, 2025, https://docs.github.com/en/webhooks/using-webhooks/creating-webhooks
38. Webhooks | GitLab Docs, accessed August 26, 2025, https://docs.gitlab.com/user/project/integrations/webhooks/
39. Professional Webhooks with Flask in Python - YouTube, accessed August 26, 2025, https://www.youtube.com/watch?v=iLV65sD8_eY
40. Prompt Engineering: Part 2 – Best Practices for Software Developers ..., accessed August 26, 2025, https://blogs.sw.siemens.com/thought-leadership/prompt-engineering-part-2-best-practices-for-software-developers-in-digital-industries/

41. Prompt Engineering for AI Guide | Google Cloud, accessed August 26, 2025, https://cloud.google.com/discover/what-is-prompt-engineering
42. Effective LLM Prompting: Getting the Code You Actually Need | by Dor Amram | Medium, accessed August 26, 2025, https://medium.com/@doramram210/effective-llm-prompting-getting-the-code-you-actually-need-8d5c2cf12503
43. AI Prompts for Code Reviews - Faqprime, accessed August 26, 2025, https://faqprime.com/en/ai-prompts-for-code-reviews/
44. I Built 5 Prompts for Better Code Analysis : r/ChatGPTCoding - Reddit, accessed August 26, 2025, https://www.reddit.com/r/ChatGPTCoding/comments/1i807g4/i_built_5_prompts_for_better_code_analysis/
45. Generating unit tests with LLMs : r/learnprogramming - Reddit, accessed August 26, 2025, https://www.reddit.com/r/learnprogramming/comments/1i168we/generating_unit_tests_with_llms/
46. An Empirical Study of Unit Test Generation with Large Language Models. - arXiv, accessed August 26, 2025, https://arxiv.org/html/2406.18181v1
47. Write Unit Tests for Your Python Code With ChatGPT – Real Python, accessed August 26, 2025, https://realpython.com/chatgpt-unit-tests-python/
48. How to generate unit tests with GitHub Copilot: Tips and examples, accessed August 26, 2025, https://github.blog/ai-and-ml/github-copilot/how-to-generate-unit-tests-with-github-copilot-tips-and-examples/
49. Leveraging Large Language Models for Python Unit Test - ResearchGate, accessed August 26, 2025, https://www.researchgate.net/publication/384336412_Leveraging_Large_Language_Models_for_Python_Unit_Test
50. How to invoke pytest - pytest documentation, accessed August 26, 2025, https://docs.pytest.org/en/stable/how-to/usage.html
51. How to capture stdout/stderr output - pytest documentation, accessed August 26, 2025, https://docs.pytest.org/en/stable/how-to/capture-stdout-stderr.html
52. Working with comments - GitHub Docs, accessed August 26, 2025, https://docs.github.com/en/rest/guides/working-with-comments
53. REST API endpoints for issue comments - GitHub Docs, accessed August 26, 2025, https://docs.github.com/rest/issues/comments
54. PullRequest — PyGithub 0.1.dev50+gbccc5aa documentation, accessed August 26, 2025, https://pygithub.readthedocs.io/en/latest/github_objects/PullRequest.html
55. github - Create comment on pull request - Stack Overflow, accessed August 26, 2025, https://stackoverflow.com/questions/16744069/create-comment-on-pull-request
56. Discussions - python-gitlab v6.2.0, accessed August 26, 2025, https://python-gitlab.readthedocs.io/en/stable/gl_objects/discussions.html

57. Merge requests - python-gitlab v6.2.0 - Read the Docs, accessed August 26, 2025, https://python-gitlab.readthedocs.io/en/stable/gl_objects/merge_requests.html
58. Introduction - CrewAI, accessed August 26, 2025, https://docs.crewai.com/
59. Crew AI Crash Course (Step by Step) - Alejandro AO, accessed August 26, 2025, https://alejandro-ao.com/crew-ai-crash-course-step-by-step/
60. Build Your First Crew - CrewAI, accessed August 26, 2025, https://docs.crewai.com/guides/crews/first-crew
61. Quickstart - CrewAI Documentation, accessed August 26, 2025, https://docs.crewai.com/quickstart
62. Building a GitHub App that responds to webhook events, accessed August 26, 2025, https://docs.github.com/en/apps/creating-github-apps/writing-code-for-a-github-app/building-a-github-app-that-responds-to-webhook-events
63. Webhook events and payloads - GitHub Docs, accessed August 26, 2025, https://docs.github.com/en/webhooks/webhook-events-and-payloads