A2
Full Stack Engineering

Timeline & Evaluation Criteria

Think41
Full-Stack GenAI Company

A2. Full Stack Engineering

Deadline - Tuesday, 25th February, 11:00 am

This project focuses on building a functional end-to-end working application. Deployment and product grade is out of scope for this project. However, the project should be high quality, with automated tests and should demo something of value.

Here are the expected deliverables of this project

- **Demo** - A working end-to end live demo recorded video
- **Technology** - A recorded code walkthrough, showcasing high quality tested code

Both the submissions are essential and will have an evaluation / selection criteria.

Demo Selection Criteria

Record the demo like a live pitch - you are trying to convince a potential client of the **usefulness, value and some technological innovation** of what you built. This should showcase a problem statement and your solution to it in a 2-4 mins video.

Compelling demo pitches would be selected and shown to a live panel for a feedback and Q & A session.

Technology Evaluation Criteria

Apart from the demo, your code would be evaluated on following coding best practices. Here are the evaluation criteria with weightage given to each.

Criteria	**Explanation**	**Weightage**
Testing	Enough detailed test-cases with good coverage	50%
Code Quality	Along the following Axes ● Well Documented ● Modular Functions	30%

	• Proper Logging • Best Practices	
Dev Experience & Process	Along the following Axes • Git Process - Branching, Commits, .gitignore • Containers - Docker & Docker Compose • Configurations - Works on new setup	20%

What does LLM think?

We passed the evaluation criteria to Chat GPT with the following prompt and it created a 20 point checklist on things to do. Feel free to tweak the prompt to get it to better elaborate (or condense) the approach and items you should consider.

I am working on a 4 day React, Tailwind Django project for a training assignment. Here is the evaluation criteria. Please provide a 20 point prioritised checklist on how I should approach this and specific things I should be paying attention to when coding.

Technology Evaluation Criteria

Apart from the demo, your code would be evaluated on following coding best practices. Here are the evaluation criteria with weightage given to each.

Criteria	**Explanation**	**Weightage**
Testing	Enough detailed test-cases with good coverage	50%
Code Quality	Along the following Axes • Well Documented • Modular Functions • Proper Logging • Best Practices	30%

Dev Experience & Process	Along the following Axes ● Git Process - Branching, Commits, .gitignore ● Containers - Docker & Docker Compose ● Configurations - Works on new setup	20%

Prioritized 20-Point Checklist for Your React, Tailwind, and Django Project

1. Testing (50%) - Highest priority

1. **Unit Tests (Backend & Frontend)** – Write Django tests for views, models, and serializers; Jest/React Testing Library tests for components.
2. **Integration Tests** – Test interactions between API and frontend, ensuring smooth data flow.
3. **End-to-End (E2E) Tests** – Use Cypress or Playwright for UI automation covering key user flows.
4. **Mocking & Fixtures** – Use `pytest-django` and `unittest.mock` for DB-independent tests.
5. **Coverage Reports** – Ensure at least 80% test coverage; generate reports using `pytest-cov` and Jest.

2. Code Quality (30%) - Clean, Maintainable Code

6. **Well-Documented Code** – Docstrings for functions, comments for complex logic.
7. **Modular Functions** – Break down large functions into small, reusable ones.
8. **Consistent Naming** – Use clear, descriptive variable and function names.
9. **Logging & Error Handling** – Use Django’s `logging` module and React error boundaries.
10. **Environment Variables** – No hardcoded secrets; use `.env` and `django-environ`.
11. **Adhere to Best Practices** – PEP8 for Python, ESLint & Prettier for JavaScript.
12. **API Response Standardization** – Use DRF serializers to ensure consistent API responses.
13. **Security Measures** – Use Django’s CSRF protection, secure cookies, and rate limiting.

14. **Efficient Database Queries** – Optimize queries with `select_related()` and `prefetch_related()`.
15. **Reusable UI Components** – Use Tailwind's utility classes smartly and extract common components.

3. Dev Experience & Process (20%) - Smooth Setup & Collaboration

16. **Git Workflow** – Follow feature branching, meaningful commit messages, and PR reviews.
17. **.gitignore** – Ensure no sensitive files (`.env`, `__pycache__`, `node_modules/`) are committed.
18. **Dockerized Setup** – Create a `Dockerfile` and `docker-compose.yml` for easy local setup.
19. **Configuration Management** – Use `settings.py` configurations properly for different environments.
20. **README & Setup Guide** – Provide clear setup instructions for new developers.

Following this checklist will maximize your score and ensure a high-quality submission. Let me know if you need help with any of these! 🚀

Appendix

Integrating LLM

1. Use your personal email to create an account at https://console.groq.com/
2. Follow instructions according to your framework - https://console.groq.com/docs/overview
3. You should use Open AI compatibility API because it will allow to switch between other providers as most of them support it - https://console.groq.com/docs/openai

Integrating Web Speech

The easiest way to integrate speech is to use webkit speech APIs. There are two parts of conversational AI

Speech Recognition - Speech to Text - STT

Webkit has an inbuilt API for this.
https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

Speech Synthesis - Text to Speech - TTS

Again there is a built api which is easy to use.
https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

Other Providers

The quality isn't too great for web-kit speech APIs. Instead, Deepgram provides a good amount of free credit to build on top of it. You can build live streaming STT and TTS using this for high quality audio e.g.
https://developers.deepgram.com/docs/getting-started-with-the-streaming-test-suite

There are other providers as well e.g. https://cartesia.ai/

Usually, for a talking bot, the pipeline is STT → LLM → TTS, with the prompts created such that they provide conversational output, rather than too long texts.