

**T.C. MALTEPE UNIVERSITY FACULTY OF ENGINEERING AND NATURAL SCIENCES DEPARTMENT OF SOFTWARE ENGINEERING**

**Software Project Management Project Document**

**2024/25 Spring**

**SE 403**

**FıkraBot: A Turkish Joke Generation System with LLM**

**10 May 2025**

**Instructor: Ensar Gül**

**[ensargul@maltepe.edu.tr](mailto:ensargul@maltepe.edu.tr)**

**Prepared By:**

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Melih Küçük | 21 07 06 048 |
| Hüseyin Eray Kızılkaya | 22 07 06 302 |
| Nurşeyda Doğan | 21 07 06 047 |
| Onur Ulaş Canpolat | 20 07 06 314 |
| İdil Öztürk | 21 07 06 026 |
| Afnan Mohammed | 21 07 06 811 |

Table of content

Software Project Management Project Document 1

1. Scope of the project 3

**1.1 Project Objective 3**

**1.2 Project Components and Goals 3**

2. Functionality of the project 4

**2.1 Core Functionalities 4**

**2.2 Unique or Distinguishing Features 5**

**2.3. The Functions Available at the Beginning of the Project 6**

**2.4. Functions Added During the Development 6**

3. Missing parts 7

**1. Use of GPT-4 API 7**

**2. Prompt Input from the User 7**

**3. Joke Categories or Themes 7**

**4. User Feedback or Joke Storage via Database 7**

4. Design documents 8

**4.1 Use Case Diagram 8**

**4.2 Sequence Diagram 9**

**4.3 Activity Diagram (Backend Logic) 10**

**4.4 Frontend Interface (UI Design) 11**

**5.1 System Requirements 12**

**5.2 Project Structure Overview 12**

**5.3 Backend Setup 12**

**5.4 Frontend Setup 13**

**5.5 Model Deployment Logic 13**

**5.6 Additional Notes 13**

**5.7 requirements.txt 13**

6. Responsibilities for each iteration 14

**6.1 Iteration Table 14**

**6.2 Estimated vs Actual Effort 15**

7. Risk management 16

**Risk Table: 16**

8. Tests 17

**Test Case Table – Turkish Joke Generator 17**

**Summary 17**

9. Experience gained 18

11. Project Repository 22

12. References 22

# **1. Scope of the project**

## **1.1 Project Objective**

The main objective of this project is to develop an AI-powered Turkish joke generation system by fine-tuning large language models (LLMs) on a curated dataset of Turkish jokes. The project explores controlled overfitting, efficient fine-tuning techniques, and even training a model from scratch, all in the context of a low-resource language (Turkish). The final deliverables include a fast backend API, a minimal web interface, and an organized training pipeline.

## **1.2 Project Components and Goals**

**LLM Fine-Tuning**

* Fine-tuned two pretrained models:
  + ytu-ce-cosmos/turkish-gpt2-large: used for practical and fast local inference.
  + Mistral 7B: used for experimental performance evaluation.
* A custom LLM was also trained from scratch.
* Overfitting was tested using a 20-joke dataset to evaluate memorization and generation fidelity.
* Main dataset: 4,500 jokes, cleaned down to 3,600 high-quality samples.

**Data Collection & Preprocessing**

* Python was the core language used across all components.
* A web scraping pipeline was created to collect Turkish jokes from public websites.
* Data cleaning and preparation were handled using Pandas, regex, and Hugging Face tokenizers.
* All preprocessing steps were executed in JupyterLab.

**Training Techniques**

* Employed LoRA (Low-Rank Adaptation) for memory-efficient fine-tuning.
* Used full fine-tuning on a custom model to explore foundational behavior.
* Experiment tracking was managed via Weights & Biases (wandb).

**Development Environment**

* Initial training used Google Colab but was later moved to Kaggle Notebooks for stability and longer runtimes.
* MacBook Pro M3 was used for:
  + Local development (data cleaning, backend)
  + Inference testing with MPS (Metal Performance Shaders)

**Backend API**

* Developed with FastAPI to deliver fast, responsive joke generation.
* Features include:
  + Prompt-based input
  + Output filtering with blacklists
  + Post-processing and reranking
  + Configurable token settings (e.g., max\_length, temperature)
  + Integration with Hugging Face Hub

**Web Interface (Frontend)**

* Built with HTML, CSS, and JavaScript
* Provides:
  + A prompt input field
  + Real-time joke generation via API
  + Responsive and minimalistic design

**Project Management**

* The entire project was planned, tracked, and organized using Trello.
* Tasks were divided into sprints and categorized by feature (e.g., data collection, model training, backend, frontend).
* Trello enabled clear visualization of progress, task assignments, and deadlines.gec

This project combines modern NLP techniques, practical model deployment, and real-world data workflows—delivering a humorous AI application powered by robust backend infrastructure and simple user interaction.

# **2. Functionality of the project**

## **2.1 Core Functionalities**

The project provides a complete backend pipeline for generating Turkish jokes using a fine-tuned large language model. The generation process is fully automated, with no prompt input required from the user. Instead, the system utilizes a carefully crafted fixed prompt that includes several example jokes to guide the model’s style, tone, and structure.

The user simply clicks the "Generate Joke" button on the frontend interface. Upon request, the backend:

* Generates a single new joke based on the internal base prompt
* Filters and validates the generated output using a custom blacklist
* Optionally rewrites the joke using the base (non-finetuned) version of the same model to improve fluency and structure

This approach ensures controlled and stylistically consistent output, maintaining quality while minimizing the risk of inappropriate content.

The backend was developed using FastAPI, a modern Python framework chosen specifically for its:

* High performance with asynchronous request handling (ASGI)
* Ease of use and minimal boilerplate code
* Automatic API documentation generation
* Rapid prototyping and debugging capabilities

The “/generate” endpoint handles all logic in a single request cycle:

* Inference using the fine-tuned model
* Text cleaning and formatting
* Blacklist filtering using regex patterns
* Optional post-editing using the base model
* Return of the final joke in JSON format

All generation parameters such as temperature, top\_k, top\_p, and max\_new\_tokens are preconfigured and cannot be changed by the user through the frontend. This was intentionally enforced to ensure stable and high-quality outputs.

## **2.2 Unique or Distinguishing Features**

* **Promptless user interaction:** All control over joke style and logic is handled within the backend. The user receives a joke without needing to provide any input.
* **Blacklist filtering mechanism:** The system automatically removes jokes that include sensitive or inappropriate content based on predefined regex patterns (e.g., references to violence, years, dates, URLs).
* **Post-processing with base model:** Valid jokes are passed through the base version of the model for restructuring and polishing.
* **Overfitting experimentation:** A separate mini pipeline was created for training on a 20-joke dataset to test model memorization behavior.
* **Apple Silicon optimization (MPS):** The system is compatible with MacBook M-series chips via torch.device("mps") and float32 precision.
* **Minimal frontend design:** Built with HTML, CSS, and JavaScript, the user interface enables one-click joke generation and instant display of results.

## **2.3. The Functions Available at the Beginning of the Project**

At the beginning of the project, only the basic research direction and foundational tools were in place. The following functions and components existed in the earliest stages:

* Access to pretrained models: ytu-ce-cosmos/turkish-gpt2-large and Mistral 7B were identified for use.
* Dataset collection plan: The team had finalized the idea of scraping Turkish jokes from websites.
* Initial backend structure: A FastAPI skeleton was created to simulate joke responses.
* JupyterLab notebooks: Setup for data analysis and cleaning tasks.
* HTML-based frontend draft: A basic UI mockup with a static “Generate Joke” button.
* Manually written joke examples: Used to start testing prompt structure and model response style.

## **2.4. Functions Added During the Development**

As development progressed, the following new functionalities were implemented:

* Full web scraping system to collect and clean Turkish jokes automatically.
* Custom prompt engineering with 10 examples to guide the model's structure.
* Fine-tuning of LLMs using LoRA for efficient parameter training.
* Overfitting pipeline for testing memorization on a 20-joke subset.
* Regex-based blacklist filtering to ensure clean and safe outputs.
* Post-processing logic with the base model to enhance readability and punchlines.
* Frontend-to-backend integration via API calls using JavaScript.
* Retry mechanism in case a generated joke fails the blacklist check.
* MPS compatibility added for Apple devices to support local testing.

# **3. Missing parts**

Several features were originally considered for implementation but were ultimately excluded due to technical limitations, academic constraints, or shifting project priorities. Below are the notable functionalities that were left out:

### **Use of GPT-4 API**

At the beginning of the project, using the GPT-4 API was proposed as a way to improve generation quality and benchmark results. However, due to academic restrictions and the instructor’s preference for open-source or self-managed models, this option was not approved.

### **Prompt Input from the User**

User-provided prompts were initially discussed as a potential feature. However, due to the limited size of the dataset and the unpredictable nature of prompt-based generation in low-resource language contexts, this feature was excluded. Instead, the system was designed to generate jokes randomly using a fixed internal prompt for consistency and safety.

### **Joke Categories or Themes**

The idea of generating jokes by category (e.g., Nasreddin Hoca, school, daily life) was discarded along with user-prompting. The small dataset size and randomness of joke structure made category-based generation unreliable.

### **User Feedback or Joke Storage via Database**

A system for storing generated jokes, collecting user feedback, or rating jokes was briefly discussed. However, upon consulting the course instructor, it was deemed unnecessary for the scope of the project and was thus excluded from implementation.

# **4. Design documents**

This section presents the visual and architectural design elements of the project. Diagrams and interface visuals are provided to illustrate how users interact with the system, how components communicate, and how logic flows within the backend infrastructure.

## **4.1 Use Case Diagram**

The use case diagram illustrates the interaction between the user and different components of the system.

**Actors:**

* User
* Frontend Interface
* Backend (FastAPI)
* Fine-tuned Model
* Base Model (for post-editing)

**Use Cases:**

* Click “Generate Joke”
* Send request to backend
* Generate joke via fine-tuned model
* Run blacklist validation
* Post-process with base model
* Return final joke to frontend
* Display joke to user

diyagram, çizgi, taslak, çizim içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

## **4.2 Sequence Diagram**

This diagram details the step-by-step flow of data when a user triggers the joke generation feature.

**Flow:**

1. User clicks "Generate Joke" on frontend
2. JavaScript sends POST request to /generate endpoint
3. Backend triggers fine-tuned model
4. Output is validated with blacklist rules
5. If invalid, retry logic is applied (up to 5 times)
6. If valid, post-editing is done via base model
7. Cleaned output is sent back to frontend
8. Joke is displayed on the interface

metin, ekran görüntüsü, diyagram, paralel içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

## **4.3 Activity Diagram (Backend Logic)**

This activity diagram visualizes the backend’s internal logic during joke generation.

* Start → Generate via fine-tuned model → Clean → Validate  
  ↳ If invalid → retry (loop)  
  ↳ If valid → post-edit → return joke

taslak, çizim, metin, diyagram içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

## **4.4 Frontend Interface (UI Design)**

A minimal, responsive frontend interface was created using **HTML**, **CSS**, and **JavaScript**. While the backend handles complex generation logic, the frontend ensures ease of access for end-users.

**Key features:**

* Clean and responsive single-page layout
* “Generate Joke” button as the only user action
* Automatic display of returned joke
* Error handling message if generation fails
* Fully connected to backend via Fetch API

metin, yazılım, bilgisayar simgesi, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**5. Deployment**

This section provides comprehensive instructions to set up and run the Turkish Joke Generation System. The application consists of a **FastAPI-based backend**, a **HTML/JavaScript frontend**, and a **locally stored fine-tuned model** built on top of a Hugging Face base model.

## **5.1 System Requirements**

* **OS:** macOS (optimized for Apple Silicon, M1/M2/M3)
* **Python Version:** 3.10+
* **GPU:** MPS backend support (Metal Performance Shaders)
* **Browser:** Any modern browser (Chrome, Firefox, Safari)
* **Internet Access:** Required for initial Hugging Face model download and authentication

## **5.2 Project Structure Overview**

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

## **5.3 Backend Setup**

1. **Clone the Repository**

|  |
| --- |
| git clone https://github.com/Maltepe-University-SWEng/term-project-team-8  cd term-project-team-8 |

1. **Install Required Packages**

|  |
| --- |
| pip install -r requirements.txt |

1. **Login to Hugging Face**

|  |
| --- |
| huggingface-cli login |

Make sure your Hugging Face access token has read permissions for model downloads.

1. **Run the FastAPI Server**

|  |
| --- |
| uvicorn app:app --reload |

The server runs at <http://localhost:8000>

## **5.4 Frontend Setup**

1. Go to the /frontend/ folder.
2. Open index.html using **Live Server** or any local web server.
   * You can also right-click and select **"Open with Live Server"** if using VS Code.
3. Click the **“Fıkra Üret”** button.
4. The frontend sends a POST request to http://localhost:8000/generate and displays the returned joke in a chat-style UI.

The interface includes animations, joke delivery simulation, and character illustrations.

## **5.5 Model Deployment Logic**

* **Base Model:** ytu-ce-cosmos/turkish-gpt2-large (fetched via Hugging Face Hub)
* **Fine-tuned Adapter:** Stored locally in ./GPT2 (LoRA format via PEFT)
* **Device Usage:** Model inference runs on torch.device("mps") with float32 precision for MacBooks

## **5.6 Additional Notes**

* If running on Windows/Linux, change torch.device("mps") to cuda or cpu.
* The backend includes filtering, retry logic, and joke post-processing.
* The system is designed to be fully local; no need to upload models online after initial download.
* Make sure ports 8000 (backend) and 5500 (or your live server port) are not blocked.

## **5.7 requirements.txt**

Here is the recommended content for your requirements.txt:

|  |
| --- |
| fastapi==0.100.0  uvicorn==0.23.2  torch>=2.0.0  transformers>=4.35.0  peft>=0.5.0  huggingface\_hub>=0.15.1  pydantic>=1.10.0  regex |

# **6. Responsibilities for each iteration**

## **6.1 Iteration Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **iter no/ developer** | **Melih (leader)** | **Hüseyin** | **Nurşeyda** |
| **iter 1** | Conducted model research and experimentation | Researched technologies and project-related concepts | Researched social engineering concepts and databases |
| **iter 2** | Performed data cleaning, organization, merging, and consolidation  Led model training and iterative improvements | Collected and cleaned data | Collected and organized project data |
| **iter 3** | Managed the AI-related tasks, including training and evaluation  Developed backend functionalities | Participated in training the model  Supported backend development | Designed initial wireframes on Figma  Implemented core frontend components |
| **iter 4** | Improved frontend layout and responsiveness  Tuned generation parameters for better outputs  Contributed to final reporting | Assisted in frontend design improvements (layout + responsiveness)  Created visual diagrams for documentation | Helped with final report preparation |

|  |  |  |  |
| --- | --- | --- | --- |
| **iter no/ developer** | **Onur** | **İdil** | **Afnan** |
| **iter 1** | Contributed to model research | Researched social engineering topics | Researched relevant technologies and concepts |
| **iter 2** | General research | Collected data for the project | Collected data |
| **iter 3** | Performed data cleaning tasks | Assisted with core frontend development | Visual materials for the website |
| **iter 4** | Assisted with presentation preparation | Feedback for Report | Feedback for Report |

## **6.2 Estimated vs Actual Effort**

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Click on the image to go to the Trello Page.**

[logo, yazı tipi, grafik, beyaz içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.](https://trello.com/b/xmd1LbLC/se403-team-project)

# **7. Risk management**

Throughout the development of the Turkish Joke Generation System, various potential and actual risks were identified and addressed. These risks covered technical limitations, model behavior, deployment constraints, and institutional restrictions. The table below outlines each major risk along with its likelihood, impact, and mitigation strategy.

### **Risk Table:**

| **Risk ID** | **Risk Description** | **Likelihood** | **Impact** | **Mitigation Strategy** |
| --- | --- | --- | --- | --- |
| R1 | GPT-4 API usage was not approved by the instructor | Medium | High | Switched to open-source models like GPT-2 and Mistral |
| R2 | Hugging Face API failure due to authentication issues | Medium | Medium | Used huggingface-cli login; tested models locally before deployment |
| R3 | Compatibility issues with MPS backend on macOS | Low | Medium | Optimized inference with torch.device("mps") and float32 precision |
| R4 | Dataset too limited for reliable user-prompt generation | High | High | Switched to fixed internal prompting with random output generation |
| R5 | Inappropriate or unsafe joke outputs | Medium | High | Implemented regex-based blacklist and post-editing with base model |
| R6 | Training interruptions and instability on Google Colab | High | Medium | Migrated training process to Kaggle for longer, more stable sessions |
| R7 | Initial plan to use a database was deemed unnecessary | Low | Low | Confirmed with instructor and excluded database functionality |

# **8. Tests**

The Turkish Joke Generation system was tested manually using tools such as **Postman** for API endpoint validation and **live frontend interaction** for user flow consistency. Since the application relies on generative model outputs, both **functional testing** and **content validation** were performed.

All major functionalities, including output filtering, retry logic, and post-processing, were tested with predefined scenarios. The goal was to verify that generated jokes met expected constraints (e.g., word count, safety, structure), and that error handling worked correctly.

The following table summarizes the manual test cases executed during development:

## **Test Case Table – Turkish Joke Generator**

| **Test ID** | **Requirement** | **Precondition** | **Test Steps** | **Expected Result** | **Result** |
| --- | --- | --- | --- | --- | --- |
| TC01 | Should return a meaningful joke via valid POST request | Server running, model loaded | Send POST to /generate with JSON payload | A logical and relevant joke should be returned | ✓ |
| TC02 | Joke must not include blacklisted content (e.g. links) | Validation function active | Check output for words like http, tweet, etc. | No blacklisted words should appear in output | ✓ |
| TC03 | Joke must not exceed 30 words | Tokenizer and model configured | Count number of words in the joke | Joke should be within 30 words | ✓ |
| TC04 | Retry logic should trigger and exit gracefully | Retry limit set, force invalid outputs | Trigger up to 5 failed generations | Should return: “No suitable joke could be generated” | ✓ |
| TC05 | Edited joke should be shorter and more humorous | Base model editing enabled | Run generation + post-editing step | Final joke should be clearer, shorter, and funnier | ✓ |
| TC06 | Only POST requests should be allowed | FastAPI app running | Send GET request to /generate | Should return HTTP 405 Method Not Allowed | ✓ |
| TC07 | Model loading failure should be logged, not crash app | Faulty adapter path used | Launch server with invalid model path | Error should be logged, app must not crash | ✗ |

### **Summary**

All tests except one were successful. Functional coverage included both frontend and backend. The retry mechanism, blacklist validation, and format enforcement worked as expected.  
**Test TC07 failed**, indicating that the system does not currently log model loading errors gracefully if the adapter path is incorrect. This will be addressed in future improvements.

# **9. Experience gained**

The Turkish Joke Generation project aimed to develop a functional and culturally relevant generative AI system for producing humorous short-form content in Turkish. While not every output was flawless, the results were promising—frequently producing coherent, humorous jokes that matched user expectations. The final product is a **usable and deployable system,** equipped with an interactive frontend and a FastAPI-based backend integrated with a fine-tuned transformer model.

Throughout development, key milestones were achieved:

* A complete data collection, cleaning, and formatting pipeline was implemented.
* Fine-tuning was successfully performed on both **GPT-2 (YTU-Cosmos)** and **Mistral v1** models.
* Various training strategies were explored, including multiple epochs, dataset sizes (1K, 2K, 5K), and prompt-based generation.
* Blacklist filtering and base model post-editing mechanisms significantly improved output safety and readability.
* A minimal, functional web interface was developed and integrated with backend logic.

Though we encountered challenges, the team adapted quickly:

* **Model research** proved more technically intensive than expected, but the team's existing knowledge in AI provided a helpful foundation.
* **Data collection** was difficult due to limited high-quality, open-source Turkish joke datasets. Creating and cleaning a usable dataset took considerable effort.
* **Training process issues** included environment crashes, resource limitations, epoch repetition struggles, and hyperparameter tuning.
* Despite initial plans to use advanced prompting and GPT-4 APIs, **institutional restrictions** redirected the project toward fully local, open-source alternatives.

These constraints shaped the final solution in a meaningful and realistic way. In short, the outcome is a lean, functional, and tested generative system tailored for Turkish humor.

## **1. Melih**

As As a team member responsible for AI workflows, I was actively involved in model research, dataset cleaning and consolidation, and the implementation of fine-tuning pipelines. I contributed directly to the model training process and supported both backend and frontend development, particularly focusing on layout structure and responsiveness. I improved the frontend’s responsiveness and handled prompt design and parameter tuning within the backend.

During the project, I not only worked with pretrained models but also learned how to train a language model from scratch, gaining hands-on experience in preparing datasets, configuring training loops, and interpreting output quality across multiple epochs. This deepened my understanding of the end-to-end training process and model behavior under limited resources.

In addition, my project management and Scrum-related skills were significantly strengthened. I helped coordinate tasks, tracked sprint progress using Trello and GitHub, and contributed to planning and documentation. I naturally assumed responsibilities similar to a Scrum Master, facilitating collaboration and ensuring steady team progress.

Overall, this project provided me with a comprehensive full-stack experience covering both machine learning and deployment. It boosted my confidence in managing AI projects from start to finish and enhanced my technical expertise, teamwork, and documentation skills.

## **2. Hüseyin**

During this project, I focused on technology and concept research as well as core data operations, including collection and cleaning. I contributed directly to the model training process and supported both backend and frontend development, especially in layout and responsiveness.

One of the most valuable parts of the experience was understanding the full pipeline—starting from raw data to a working, user-friendly AI system. I also designed UML diagrams and learned how critical system planning is during rapid development cycles.

This project helped me improve my coding discipline, collaboration workflow, and design thinking within AI-based projects.

## **3. Nurşeyda**

I contributed to social engineering and database research at the early stages of the project. I worked on data collection and cleanup, designed the first interface drafts in Figma, and later implemented the main structure of the frontend. I also helped refine parts of the project report and kept track of the user flow.

This project helped me understand the importance of structure and user interaction in AI systems. I gained confidence in designing and converting mockups into actual code. I also learned how backend and AI logic impacts frontend behavior and response. I’m now more comfortable working with design-to-code tools and enjoyed playing a central role in shaping the user experience.

## **4. Afnan**

My responsibilities involved researching technologies and concepts to support the development team. I worked on collecting Turkish joke data and finding visual elements for the frontend.

Though my role was not code-heavy, I gained valuable insight into dataset preparation, visual consistency in UI, and supporting an AI project from a content and design angle. I also learned how small details (like illustrations and formatting) greatly influence final user experience. This project showed me the importance of teamwork across both technical and non-technical domains.

## **5. Onur**

I focused primarily on model research and data cleaning, as well as helping with the presentation materials. I worked on identifying reliable models for the Turkish language and helped test some of the early results.

Through this project, I learned how to evaluate model architectures and gained experience in preparing datasets for fine-tuning. I also learned how to structure and present AI project findings in a way that’s accessible to non-technical audiences.

## **6. İdil**

My main role was in researching social engineering contexts and helping with data collection. I also assisted in coding key sections of the frontend interface.

This was my first hands-on experience working in an AI-powered project with real deployment goals. I learned how UI design, content filtering, and ethical concerns all play a role in building responsible AI systems. I enjoyed working on the visual and structural aspects of the interface and seeing it come to life through code.

## **Future Work**

Several features and improvements are planned for the next stages of development:

* **Use of more powerful models:** Integrating APIs like GPT-4 (if permitted) or open-weight high-capacity models for higher quality and creativity.
* **Larger and richer dataset:** Increasing dataset diversity to allow more accurate category-based joke generation.
* **User-provided prompting:** Allowing the user to provide joke themes or keywords to enable targeted humor generation.
* **Joke categorization feature:** Supporting themes such as Nasreddin Hoca, school, daily life, etc.
* **Database integration:** Saving user-generated jokes, feedback, or ratings, based on instructor approval.
* **UI enhancements:** Improving frontend animations, multilingual support, and accessibility.

This project also involved intensive documentation and sprint management using **Trello** and **GitHub**, where Gantt charts, risk tables, and schedule-effort breakdowns were provided. The sprint reports included model comparisons (e.g., GPT-2 vs. Mistral), detailed analysis of outputs, and collaborative model training workflows.

While initial experiments with prompting were rejected by the instructor, the team adapted by focusing on fine-tuning and controlling generation logic through fixed templates. This ultimately led to a more robust and stable system.

The final implementation includes:

* All backend logic via FastAPI
* A fully working frontend
* Clean codebase and organized folder structure
* A carefully fine-tuned model for production
* Full open-source availability via GitHub

The Turkish Joke Generator is both a technical achievement and a demonstration of adaptive, resilient development within academic constraints.

# **11. Project Repository**

**Click on the image to go to the repository.**

[logo, yazı tipi, grafik, metin içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.](https://github.com/orgs/Maltepe-University-SWEng/teams/team-8)

# **12. References**

1. Hugging Face. “Transformers Documentation.”  
   <https://huggingface.co/docs/transformers>
2. Hugging Face Hub – Model Page. ytu-ce-cosmos/turkish-gpt2-large.  
   https://huggingface.co/ytu-ce-cosmos/turkish-gpt2-large
3. Google Colab. “Colaboratory Documentation.”  
   https://colab.research.google.com
4. Kaggle. “Kaggle Notebooks and Dataset Platform.”  
   <https://www.kaggle.com>
5. GitHub. “FastAPI Official Repository.”  
   <https://github.com/tiangolo/fastapi>
6. PEFT: Parameter-Efficient Fine-Tuning Library.  
   <https://github.com/huggingface/peft>
7. PyTorch. “Torch Documentation.”  
   https://pytorch.org/docs/stable/index.html
8. ChatGPT (OpenAI). Used interactively for debugging assistance, language modeling support, and report drafting suggestions.
9. Stack Overflow Community. Various posts regarding FastAPI, JSON handling, training issues, and PyTorch deployment.
10. MDN Web Docs. “Frontend responsiveness and layout strategies.”  
    <https://developer.mozilla.org/>