# Zero-Shot Text Classification Coding Exercise

In this project, you will create a Python-based zero-shot text classification RESTful web service using FastAPI and Poetry. The web service will leverage the BERT-based pretrained models available in the Hugging Face's transformers library.

# Assignment Steps

**Setup the Environment**

Initialize a new project using Poetry and set up a virtual environment.

Add the required Python packages to the pyproject.toml file.

**Load the Pretrained Model**

Load a pretrained model suitable for zero-shot classification, such as BertForSequenceClassification or T5ForConditionalGeneration, from the transformers library.

**Implement Zero-Shot Classification**

Implement a function that takes text and a list of candidate labels as input, processes it using the loaded model, and outputs the predicted label. This function should handle tokenization, input processing, model inference, and post-processing.

**Create the RESTful Web Service with FastAPI**

Use FastAPI to create the RESTful web service. Define an API endpoint (e.g., /classify) that accepts a JSON payload with text and a list of candidate labels. Return the predicted label and optional confidence scores as a JSON response.

Use Pydantic models to define the request and response schemas.

**Testing**

Run the web service using Uvicorn. Test the web service using tools like curl or postman to ensure that the predictions are accurate and meaningful. Evaluate the web service's performance with different types of text and candidate labels.

**Documentation**

Write a README file that explains how to install, run, and use the web service using Poetry and FastAPI, as well as a brief explanation of the model and zero-shot classification approach used. Use FastAPI's automatic API documentation features (Swagger UI and ReDoc) to provide interactive API documentation.

**Optional Enhancements**

Implement a confidence score or probability for each label prediction.

Provide an API for batch processing of multiple texts.

Implement proper error handling and status codes for the API responses.

# What we look for

* **Readability**: Coding style, method/variable/class/etc. names, encapsulation.
* **Structure**: Object design and code architecture. Try not to overarchitect your solution.
* **Documentation**: Clarity of communication in your documentation.
* **Testing**: Approach to testing, unit testing, etc.

# General Notes

* Keep the exercise simple. This is not something that should take more than **8-12 hours** of effort for someone familiar with the technologies.
* Create a github repository that contains the complete solution and a README file on how to build it and run it.
* Prepare to discuss your code in the final interview.

# Delivery Date

You should submit the project before 10AM Thursday 14 September 2023