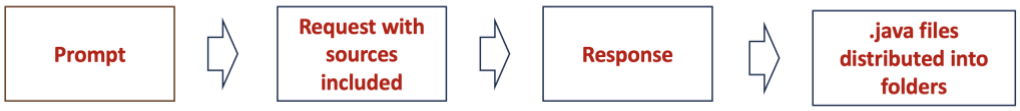
JAIG MAIN IDEAS

1. JAIG lifecycle with source files inclusion and response parsing

The current **lifecycle of JAIG** with the possibility of source files inclusion works this way:



The prompt contains the path to the files or folders, for example:

```
/src/main/**/model/**
Generate a class that populates the domain model with test data.
```

In this case, all files from the folder **model** within **/src/main** directory will be included in the request. The response is retrieved via OpenAI API, and then parsed and distributed to the proper folders within the src/main/java directory. The generated code can be immediately executed and tested.

2. Automatic merge of the changes in code and prompts

The possibility to automatically **merge the changes** in manually updated code and the code generated by AI makes it possible to change the project in 2 ways: by changing the prompts and by changing the code itself.

For example, if we have the prompt and the generated source code:

Generate a Java program “Hello world”	public class HelloWorld {
	  public static void main(String[] args) {
	  System.out.println("Hello, World!");
	  }
	}

Then, we changed the prompt (now we greet with Ciao, not Hello) and at the same time we changed the source code (added exclamation marks):

Generate a Java program “Ciao, world”	public class HelloWorld {
	  public static void main(String[] args) {
	  System.out.println("Hello, World!!!");
	  }
	}

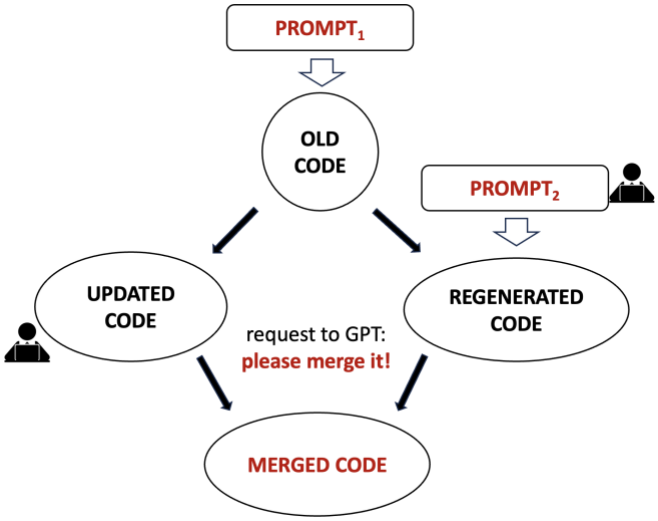
Then, the automatic request asks AI to merge the changes in the regenerated code and the manually updated code. The result looks this way:

```
public class CiaoWorld {
  public static void main(String[] args) {
    System.out.println("Ciao, World!!!");
  }
}
```

Merging means that both changes were preserved:

- Adding exclamation marks (!!!) by changing the source code
- Changing Hello to Ciao by changing the requirements in the prompt

It opens the possibility to independently change the codebase and the project requirements described in the prompts. Below is the diagram describing the process of merging:



This provides a high level of flexibility for the process of code development, especially for the bigger applications.

3. Prompt pipelines

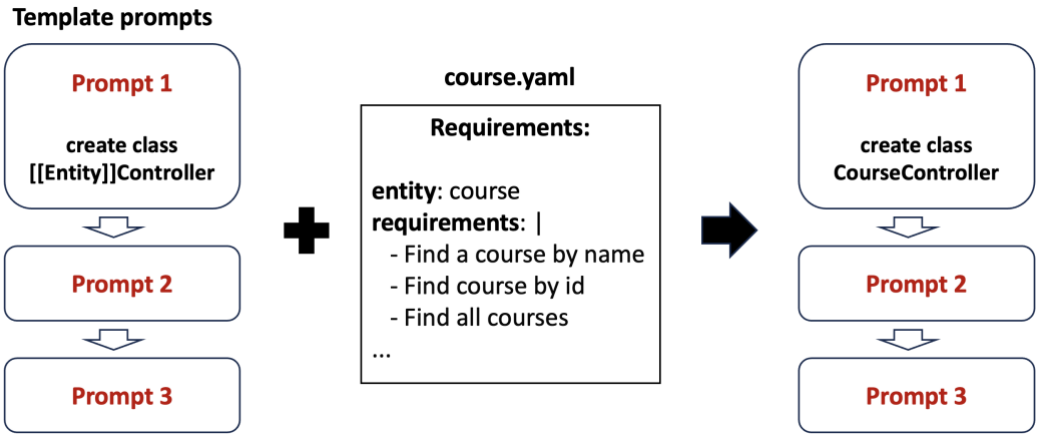
Another feature implemented in Proof of Concept is the possibility to define the **prompt pipelines**. This makes it possible to create a pipeline of several prompts, splitting the complicated problem into multiple steps and sequentially executing each step to receive the final result:

The possibility to run the sequence of prompts in automatic mode is called a **batch**. Running a batch allows running prompts one by one automatically. The result of the previous prompt is used as the input context for the next prompt.

4. Prompt templates

To make prompts more universal and applicable for different tasks, it can use **placeholders**. This feature is named **prompt templates**. This way we can separate prompts from the requirements and apply the same prompt pipeline to another set of requirements, or even to another business domain.

For example, if we have a universal prompt pipeline with the placeholders (like [[Entity]] on the diagram), it can be merged with the requirements to produce the final prompt pipeline:



In addition, this separation of prompts and requirements makes it possible to independently change the code generation workflow and requirements. We discuss this very important feature in more detail.

5. Separation of the concerns: development workflow and business requirements

The prompts pipeline defines the software-building workflow. For example, it may define prompts responsible for:

1. Clarify the requirements
2. Create tests that verify if the code satisfies the requirements
3. Create a domain model
4. Create services
5. Create controllers
6. Verify the correctness of the solution

This flow can be universal and applicable for a very broad set of different use cases. Separately, we define the business requirements.
As a result, we have 2 independent documents: the software building workflow and the requirements.

If the requirements are changing, it's possible to regenerate the code based on the same workflow.
If the workflow is changing, it's possible to regenerate the code based on the same requirements.

This makes the application much easier to support. The role of this separation of concerns is similar to the separation of Code and Configuration in Spring Framework. This idea revolutionized enterprise development 20 years ago, and the JAIG approach can do the same in the AI era.