

# Einen offeneren selbst gehosteten Chatbot machen

## Automatisierte Softwareentwicklung, SoSe 2024

Viton Romane, DFIW, 5006508

Chendjou Dzogoulouh Arold Stephyl, DFIW, 5012980

Setra Thierry Andriamiadanarivo, DFIW, 3880516

Chrislie Briel Mohomye Yotchouen, PIB, 5013415

# Inhalt

- Ziel
- Server-Side: `Llama-gpt` *for Dummies*
- Strukturierung des Verhaltens
- Client-Side: Überblick
- Tests (JUnit, Mockito)
- Statische Codeanalyse
- Site/Reports
- Github Actions
- Zusammenfassung

## Ziel

## Offene Frage ?

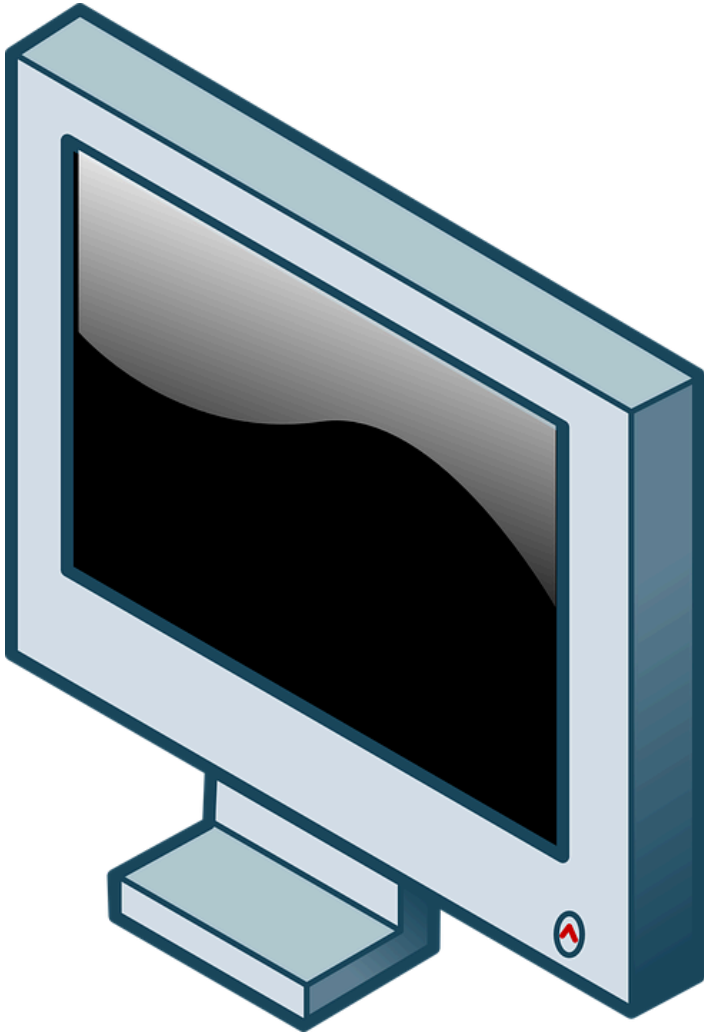
- Wie würdet ihr einen Chatbot erstellen?
- Welches Verhalten hätte er?
- An welchen Stellen könnte die Automatisierung stattfinden?

# Ziel

## Was wollen wir eigentlich?

- Nachrichten:
  - schreiben
  - speichern
  - an einer API schicken
- Die Antwort:
  - bekommen & speichern
  - darstellen

Bild: [pixabay](#)



# Server-Side: *Llama-gpt* for Dummies 🤖

Von: <https://github.com/getumbrel/llama-gpt>

## LlamaGPT

A self-hosted, offline, ChatGPT-like chatbot, powered by Llama 2. 100% private, with no data leaving your device.

**New: Support for Code Llama models and Nvidia GPUs.**

# Server-Side: *LLama-gpt for Dummies* 🤖

## Installierung: 3 Zeilen

```
git clone https://github.com/getumbrel/llama-gpt.git  
cd llama-gpt  
./run.sh --model 7b
```

→ UI auf `http://localhost:3000`, API auf `http://localhost:3001`

Aber: das können sehr schwere Modelle sein! (deswegen: Server-Side)

# Server-Side: *Llama-gpt* for Dummies 🤖

## Den Server vom Außen erreichbar machen

- Lösung: *caddy* als Reverse-Proxy
- Konfigurationsdateien sind sehr lesbar  
→ API auf <https://gptapi.oc.romaneviton.fr> erreichbar

# Strukturierung des Verhaltens



default



POST

/v1/completions Create Completion



POST

/v1/embeddings Create Embedding



POST

/v1/chat/completions Create Chat Completion



GET

/v1/models Get Models

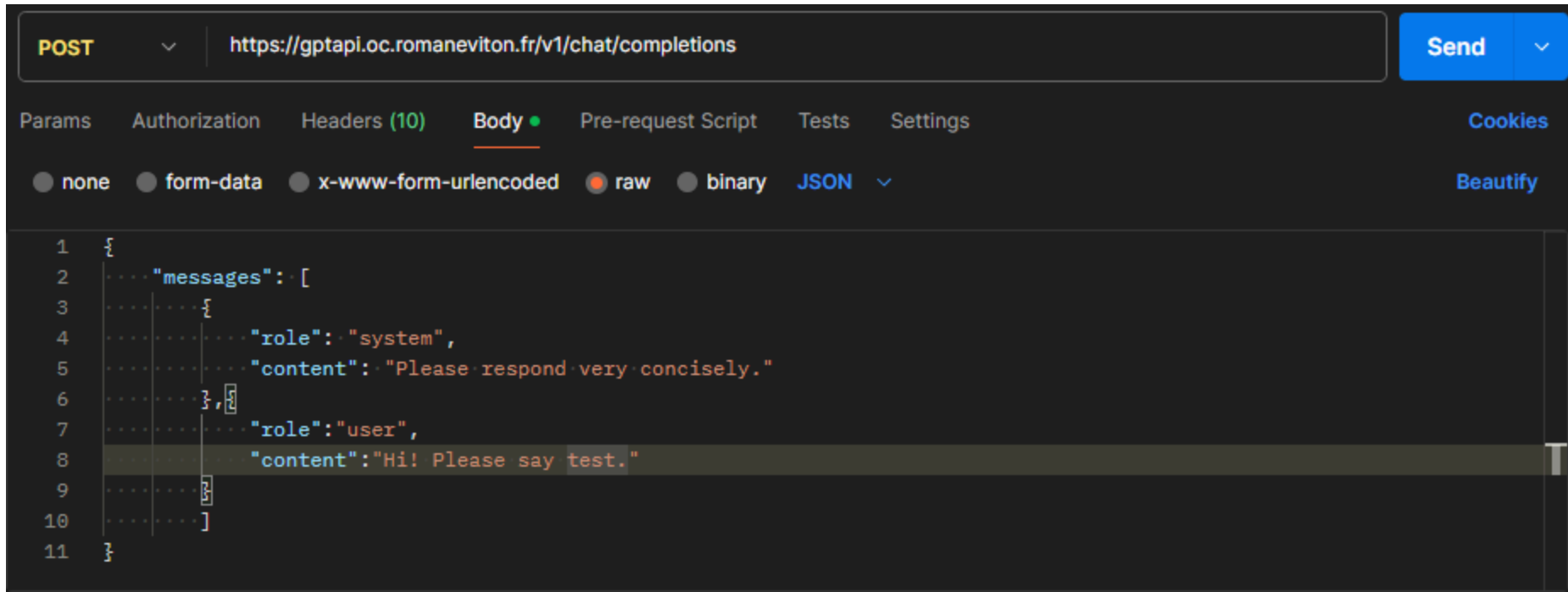


Quelle: [die API selbst](#)



# Strukturierung des Verhaltens

Eine API, die *sehr* ähnlich zur OpenAI-API ist



The screenshot shows a REST client interface with a POST request to `https://gptapi.oc.romaneviton.fr/v1/chat/completions`. The request body is a JSON object with a `messages` array. The first message is a system message with the role `system` and content `Please respond very concisely.`. The second message is a user message with the role `user` and content `Hi! Please say test.`.

```
1 {  
2   "messages": [  
3     {  
4       "role": "system",  
5       "content": "Please respond very concisely."  
6     },  
7     {  
8       "role": "user",  
9       "content": "Hi! Please say test."  
10    }  
11  ]  
12 }
```

# Strukturierung des Verhaltens

...und sie funktioniert!

```
1  {
2    "id": "chatcmpl-3fba9fb7-5a0f-424d-b70f-9d4c5e75cc13",
3    "object": "chat.completion",
4    "created": 1720797338,
5    "model": "/models/llama-2-7b-chat.bin",
6    "choices": [
7      {
8        "index": 0,
9        "message": {
10          "role": "assistant",
11          "content": "Test."
12        },
13        "finish_reason": "stop"
14      }
15    ],
16    "usage": {
17      "prompt_tokens": 27,
18      "completion_tokens": 2,
19      "total_tokens": 29
20    }
21  }
```

# Client-Side: Überblick

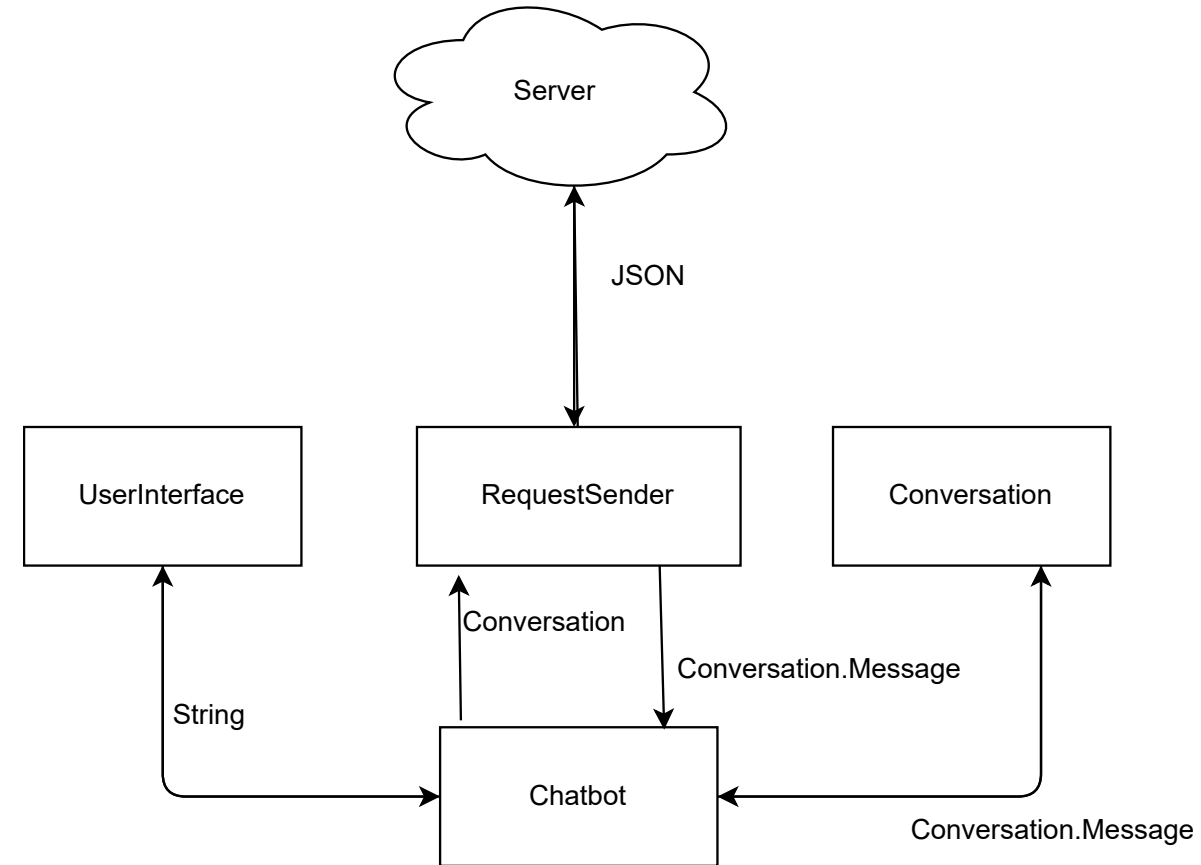
- 3 unterschiedliche Aufgaben
  - Kommunikation mit dem:der Nutzer:in
  - Nachrichtenverlauf speichern
  - Kommunikation mit dem Server

⇒ Antwort: **verschiedene Klassen nutzen**

→ *Wie würden diese Klassen miteinander kommunizieren ?*

# Client-Side: Überblick

- 4 Hauptklassen
  - Chatbot
  - CommandLineInterface  
(implementiert `UserInterface`)
  - RequestSender
  - Conversation



```
/** Testing the conversation. */
public class ConversationTest {
    //first, test Conversation.Message

    @Test
    public void messageEqualityTest() {
        Conversation.Message baseMsg = new Conversation.Message(
            role:"system",
            content:"A sample message.");
        Conversation.Message identicalBaseMsg = new Conversation.Message(
            role:"system",
            content:"A sample message.");
        Conversation.Message changedContentMsg = new Conversation.Message(
            role:"system",
            content:"A different message.");
        Conversation.Message changedRoleMsg = new Conversation.Message(
            role:"user",
            content:"A sample message.");

        assertEquals(baseMsg, identicalBaseMsg);
        assertNotEquals(baseMsg, changedContentMsg);
        assertNotEquals(baseMsg, changedRoleMsg);
    }
}
```

# Tests

## Unit-Tests mit JUnit

- Teile individuell testen, um das Verhalten zu sichern
- Dauerhafte Sicherheit

# Tests

## Mockito: Klassen simulieren

- Erlaubt, einfachere Klassen für Testzwecke zu nutzen

```
//actually create the mocks
UserInterface ui = mock(classToMock:UserInterface.class);
when(ui.getInput()).thenReturn(userInput);

RequestSender rs = mock(classToMock:RequestSender.class);
when(rs.requestNextMessage(sentConversation)).thenReturn(assistantMsg);

//use the mocks
Chatbot chatbot = new Chatbot(ui, rs);
chatbot.conversationRound();

//theck method calls on mocks
verify(ui).getInput();
verify(ui).show(assistantResponse);
```



# Tests

## JaCoCo: Testabdeckung

- Kann helfen, Stellen zu finden, die nicht getestet sind
- "Welche Tests soll ich jetzt schreiben?"

Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">RequestSender</a>	<div><div></div></div>	21 %	<div><div></div></div>	30 %
<a href="#">Main</a>	<div><div></div></div>	0 %	<div><div></div></div>	0 %
<a href="#">Conversation.Message</a>	<div><div></div></div>	76 %	<div><div></div></div>	70 %
<a href="#">CommandLineInterface</a>	<div><div></div></div>	0 %		n/a
<a href="#">Conversation</a>	<div><div></div></div>	93 %	<div><div></div></div>	83 %
<a href="#">ChatbotException</a>	<div><div></div></div>	0 %		n/a
<a href="#">Chatbot</a>	<div><div></div></div>	89 %		n/a
Total	206 of 481	57 %	14 of 34	58 %

# Statische Codeanalyse

- Den Code vor der Laufzeit prüfen
  - `errorprone`, um häufige Fehlermuster zu erkennen ()
  - `checkstyle`, um zu sichern, dass der Code an den Stilregeln angepasst ist ()

```
[INFO] -----  
[INFO] BUILD FAILURE  
[INFO] -----  
[INFO] Total time: 18.708 s  
[INFO] Finished at: 2024-07-12T18:31:47+02:00  
[INFO] -----  
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-checkstyle-plugin:3.4.0:check (test)  
on project java-chatbot: You have 1 Checkstyle violation. -> [Help 1]
```



# Site/Reports

- `site` -Phase des Maven-Lifecycles
- `pom.xml` enthält Projektdaten
- Möglichkeit, den Stil in `src/site/site.xml` zu definieren
- Reports können hinzugefügt werden (`reportSets` oder Binding an einer Phase)
  - Beispiel: *JaCoCo*

# Site/Reports

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.12</version>
  <executions>
    <execution>
      <id>default-prepare-agent</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>verify</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

# Github Actions: die nächste Ebene

- Was kann Github Actions bringen?
  - Sicherheit?
  - Einen geringeren Zeitaufwand?

# Github Actions: mehr Sicherheit 🚧

- Wir Menschen sind sehr oft vergesslich
- Eine Ausführung von `mvn clean test` kann schnell fehlen und gepusht werden
- Sicherheitsnetze mit Github Actions erlauben, jeden Push zu prüfen

# Github Actions: mehr Sicherheit

Aus [.github/workflows/maven.yml](#)

```
jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Set up JDK 21
        uses: actions/setup-java@v3
        with:
          java-version: '21'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn -B clean package site --file pom.xml
```

# Github Actions







## Package/Pages erstellen

Aus [.github/workflows/maven.yml](#)

- name: Configure GitHub Pages  
uses: actions/configure-pages@v3
- name: Upload GitHub Pages artifact  
uses: actions/upload-pages-artifact@v1  
with:  
  path: target/site

# Github Actions

Artefakte herunterladen 

<b>Artifacts</b> Produced during runtime			
Name		Size	
 Package		15.5 MB	 
 github-pages		440 KB	 

# Github Actions: weitere Automatisierungen

## Dependabot

The screenshot shows a GitHub pull request interface. At the top, the title reads "Bump org.apache.maven.plugins:maven-compiler-plugin from 3.11.0 to 3.13.0 #1". To the right of the title are "Edit" and "<> Code" buttons. Below the title, a purple "Merged" badge is followed by the text "rom-vtn merged 1 commit into main from dependabot/maven/org.apache.maven.plugins-maven-compiler-plugin-3.13.0 4 days ago".

A navigation bar below the merge info shows "Conversation 0", "Commits 1", "Checks 1", and "Files changed 1". To the right of this bar is a small bar showing "+1 -1" with a color-coded progress indicator.

The main content area shows a comment from the "dependabot" bot, stating it commented on behalf of github last week. The comment text says: "Bumps [org.apache.maven.plugins:maven-compiler-plugin](#) from 3.11.0 to 3.13.0." Below this text are expandable sections for "Release notes" and "Commits". A "compatibility 87%" badge is displayed. The comment continues: "Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting @dependabot rebase .". At the bottom of the comment is a section for "Dependabot commands and options".

On the right side of the pull request, there is a sidebar with various settings: "Reviewers" (No reviews), "Assignees" (No one—assign yourself), "Labels" (dependencies), "Projects" (None yet), and "Milestone" (No milestone). Each item has a gear icon for settings.



## Andere nützliche Tools

`versions:use-latest-releases`, `dependency:analyze`

```
[INFO] --- dependency:3.6.1:analyze (default-cli) @ java-chatbot ---  
[WARNING] Unused declared dependencies found:  
[WARNING]    com.google.guava:guava:jar:33.2.1-jre:compile  
[WARNING]    org.apache.logging.log4j:log4j-api:jar:3.0.0-beta2:compile  
[WARNING]    org.apache.logging.log4j:log4j-core:jar:3.0.0-beta2:compile  
[WARNING]    org.codehaus.mojo:versions-maven-plugin:jar:2.17.0:compile
```

# Zusammenfassung

- Automatisierung ist:
  - effizienter
  - sicherer
  - wartbarer

⇒ *Sie erlaubt, sich darum zu kümmern, was wirklich zählt*

# Danke für Ihre Aufmerksamkeit!

## Noch Fragen?

Alles zum Code (inkl. Präsentation) befindet sich im Github-Repo:  
<https://github.com/rom-vtn/java-chatbot>