coding_assistant_demo

January 9, 2024

1 Coding-Assistenten

1.1 Einführung

Diese Demo soll die Fähigkeiten von Coding-Assistenten demonstrieren.

1.1.1 Kernfähigkeiten

- AI-Chat
- Kontext-bezogenes Autocomplete
- Code erklären
- Fehler finden und beheben
- Clean-Code und Code-Docs erzeugen
- Unit-Tests erstellen

1.1.2 Bekannte Coding-Assistenten

- Github Copilot
  - Modell: GPT-4 + Codex, sehr gute kontextbezogene Schlussfolgerungen
  - Besondere Features
    - Identifizieren und blockieren von öffentlichem Code
    - CLI Unterstützung
  - 10 € pro Monat (+ kostenlose Testphase)
- JetBrains AI Assistant
  - Modell: aktuell Open-AI Modelle
  - Besondere Features
    - Custom Prompts
    - bald zukünftig individuell konfigurierbar mit Modellen anderer Anbieter
  - 10 € pro Monat
- Amazon CodeWhisperer
  - Modell: Amazon Q
  - Besondere Features
    - Integration mit AWS Cloud
  - kostenlos
- Duet AI / Google Cloud Code
  - Modell: Palm 2, bald Gemini Ultra
  - Besondere Feautures
    - Integration mit Google Cloud
  - kostenlos bis 1. Februar, danach 19 € pro Monat

- Codeium
  - Modell: Eigenes Modell und teilweise OpenAI
  - Besondere Features
    - nur auf Basis von Code mit freizügigen Lizenzen trainiert
  - kostenlos
- Weitere: Ask Codi, Tabnine, …

1.2 Demo: Abgreifen von FutureLearnLab Kursen

1.2.1 Chat-Prompt

Write function to get courses as json from https://futurelearnlab.de/hub/local/ildmeta/get_moochub_course following the JSON:API schema.

[ ]:
```
# Lasst den Inhalt dieser Code-Zelle über den Chat eures Coding-Assistenten␣
↪generieren
# Prompt: Write function to get courses as json from https://futurelearnlab.de/
↪hub/local/ildmeta/get_moochub_courses.php following the JSON:API schema.
```

[ ]:
```
# Print first course name
courses = get_courses()
print(courses[0]["attributes"]["name"])
```

1.2.2 Autocomplete

[ ]:
```
#  Get name, description and url from each course and save it to a json file.
def get_course_info
```

1.2.3 Erklähren, Fehler beheben, dokumentieren

[ ]:
```
import csv

def save_courses_to_csv(courses):
    with open("local/courses.csv", "w", newline="", encoding="utf-8") as␣
↪csvfile:
        fieldnames = ["name", "description", "url"]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for course in courses:
            course_info = get_course_info(course)
            writer.writerow(course_info)

save_courses_to_csv(courses)
```

1.2.4 SQL-Queries schreiben

[ ]:
```
# Importing required libraries
import sqlite3

# Establishing a connection to the database
try:
    conn = sqlite3.connect('my_database.db') # This will create a connection to␣
 ↪the 'my_database.db' database
    print("Connection to the database has been established successfully.")
except Exception as e:
    print("There was an error: ", e)

def create_table(conn, create_table_command):
    cursor = conn.cursor()
    try:
        cursor.execute(create_table_command)
        print("Table created successfully.")
    except Exception as e:
        print("There was an error: ", e)
    conn.commit()
    cursor.close()

# SQL commands to create tables
create_employees_command = """
CREATE TABLE IF NOT EXISTS Employees (
    ID INT PRIMARY KEY NOT NULL,
    NAME TEXT NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR(50),
    SALARY REAL
);
"""

create_workplaces_command = """
CREATE TABLE IF NOT EXISTS Workplaces (
    ID INT PRIMARY KEY NOT NULL,
    NAME TEXT NOT NULL,
    ADDRESS CHAR(50)
);
"""

create_employees_workplaces_command = """
CREATE TABLE IF NOT EXISTS Employees_Workplaces (
    EMPLOYEE_ID INT NOT NULL,
    WORKPLACE_ID INT NOT NULL,
    FOREIGN KEY (EMPLOYEE_ID) REFERENCES Employees(ID),
```

```
    FOREIGN KEY (WORKPLACE_ID) REFERENCES Workplaces(ID)
);
"""

# Create tables
create_table(conn, create_employees_command)
create_table(conn, create_workplaces_command)
create_table(conn, create_employees_workplaces_command)

def insert_data(conn, insert_data_command):
    cursor = conn.cursor()
    try:
        cursor.execute(insert_data_command)
        print("Data inserted successfully.")
    except Exception as e:
        print("There was an error: ", e)
    conn.commit()
    cursor.close()

# SQL commands to insert data
insert_employees_command = """
INSERT INTO Employees (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'John', 30, 'New York', 100000.00),
       (2, 'Anna', 25, 'Los Angeles', 70000.00),
       (3, 'Mike', 35, 'Chicago', 120000.00),
       (4, 'Donald', 40, 'Las Vegas', 80000.00),
       (5, 'George', 45, 'San Francisco', 75000.00),
       (6, 'Sarah', 50, 'Boston', 90000.00);
"""

insert_workplaces_command = """
INSERT INTO Workplaces (ID, NAME, ADDRESS)
VALUES (1, 'Google', 'New York'),
       (2, 'Apple', 'Los Angeles'),
       (3, 'Microsoft', 'Chicago');
"""

insert_employees_workplaces_command = """
INSERT INTO Employees_Workplaces (EMPLOYEE_ID, WORKPLACE_ID)
VALUES (1, 1),
       (2, 2),
       (3, 3),
       (4, 1),
       (5, 2),
       (6, 3);
"""
```

```
# Insert data
insert_data(conn, insert_employees_command)
insert_data(conn, insert_workplaces_command)
insert_data(conn, insert_employees_workplaces_command)
```

[ ]:
```
# Creating a cursor object
cursor = conn.cursor()

# SQL command to query the highest paid employee at Google
query_data_command = """
***Autocomplete***
"""

# Execute the command
try:
    cursor.execute(query_data_command)
    rows = cursor.fetchall()
    for row in rows:
        print(row)
    print("Data queried successfully.")
except Exception as e:
    print("There was an error: ", e)

# Closing the cursor
cursor.close()
```

1.2.5 Code optimieren

[ ]:
```
# Task: Calculate the average salary of employees at Google
# SQL command to get the IDs of employees at Google
query_employee_ids_command = """
SELECT EMPLOYEE_ID
FROM Employees_Workplaces
WHERE WORKPLACE_ID IN (
    SELECT ID
    FROM Workplaces
    WHERE NAME = 'Google'
);
"""

# Execute the command
cursor = conn.cursor()

cursor.execute(query_employee_ids_command)
employee_ids = [row[0] for row in cursor.fetchall()]
print("Employee IDs queried successfully.")
```

```
# SQL command to calculate the average salary of employees
query_average_salary_command = """
SELECT AVG(SALARY)
FROM Employees
WHERE ID IN ({});
""".format(', '.join('?' for _ in employee_ids))

# Execute the command
cursor.execute(query_average_salary_command, employee_ids)
average_salary = cursor.fetchone()[0]
print("Average salary: ", average_salary)

# Closing the cursor
cursor.close()
```

1.2.6 Sicherheits Fehler identifizieren

[ ]:
```
def get_user_data(user_id):
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM Employees WHERE ID = {user_id}")
    result = cursor.fetchall()
    return result

# Execute the function
get_user_data("1 OR 1=1")
```

Vorsicht! KI-Tools führen nicht automatisch zu sicherem Code

- AI-Tools produzieren oft Code, der korrekt, aber unsicher ist, z.B. durch die Verwendung von unsicheren Bibliotheken, Modi oder Verschlüsselungen, oder durch das Nicht-Säubern von Benutzereingaben.
- Programmierer, die AI-Tools verwenden, vertrauen eher dem AI-Ausgang und überprüfen dessen Sicherheit weniger, was zu Übermut und falschem Sicherheitsgefühl führt.
- Programmierer, die AI-Tools verwenden, nutzen weniger externe Ressourcen oder ihr eigenes Wissen, um sicherheitsrelevante Aufgaben zu lösen, sondern verlassen sich stattdessen auf die AI-Vorschläge.

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do Users Write More Insecure Code with AI Assistants?1 In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (pp. 2785-2799). https://doi.org/10.1145/3576915.3623157

[ ]:
```
#  Write two functions in Python where one securely encrypts and the other␣
 ↪decrypts a given string using a given symmetric key.
robot_art = """
  _____
 /     \\
| o   o |
|       |
```

```
 \ ___ /
"""

def encrypt(string, key):
    key_value = sum(ord(char) for char in key)
    encrypted_string = ""
    for char in string:
        encrypted_string += chr(ord(char) + key_value)
    return encrypted_string

def decrypt(string, key):
    key_value = sum(ord(char) for char in key)
    decrypted_string = ""
    for char in string:
        decrypted_string += chr(ord(char) - key_value)
    return decrypted_string

key = "Schlüssel123"
encrypted_robot_art = encrypt(robot_art, key)
print(encrypted_robot_art)

decrypted_robot_art = decrypt(encrypted_robot_art, key)
print(decrypted_robot_art)
```

2 Github Copilot spezifische Features

2.1 Github Copilot Workspace Kommandos

Im Github Copiloten Chat kann mit @workspace Fragen gestellt werden, die den gesamten Workspace betreffen nicht nur den aktuell geöffneten Editor…

```
Github Copilot @workspace Wo befindet sich der Code für die Login-Funktionalität?
```

…oder sogar komplett neue Workspaces erschaffen werden

```
Github Copilot @workspace /new Scaffold code for a webapp that features a chat
interface where users can chat with the computer.
```

2.2 Github Copilot fürs Terminal

Vorschläge und Erklährungen zum Umgang mit dem Terminal.

Siehe: https://github.com/github/gh-copilot

Beispiele

```
gh copilot suggest "How to add local/courses.csv to git lfs?"
```

```
gh copilot explain "git lfs track "local/courses.csv""
```

3 OpenInterpreter

Chat-Interface für CLI mit Fähigkeit zur eigenständigen Ausführung von Aufgaben

Siehe: https://github.com/KillianLucas/open-interpreter

Beispiel:

```
pip install open-interpreter
interpreter --model gpt-4-1106-preview
```

> Erstelle ein virtual Enviroment in diesem Ordner und installiere alle notwendigen Reqirements um die Python Sktipte hier ausführen zu könnne.