Exercise 7

Task 1: Compare the two computing models mainframe and cloud. Why did it change over time so dramatically?

Mainframe Computing:

- **Centralized Processing**: Data processing is done on a single, powerful machine in one location.
- **High Reliability and Security**: Commonly used in banking, insurance, and aviation due to their robustness.
- **Efficient Resource Management**: Supports large volumes of simultaneous transactions.
- **High Infrastructure Costs**: Expensive to purchase and maintain.
- **Limited Scalability**: Scaling requires investment in physical hardware.

Cloud Computing:

- **Distributed Processing**: Utilizes many servers across the internet to store and process data.
- **Global Accessibility**: Data and services can be accessed from anywhere.
- **Cost-Efficient**: Pay-as-you-go pricing lowers upfront investment.
- **Flexible & Scalable**: Resources can be adjusted dynamically based on demand.

Why has the shift occurred so dramatically over time?

- **Cost Reduction**: Businesses save on hardware and operational expenses.
- **Scalability & Flexibility**: Cloud platforms allow quick adjustment to workload changes.
- **Technological Advancements**: Faster internet, better virtualization, and improved security have enabled this shift.
- **Increased Agility**: Cloud systems support agile development and faster deployment cycles.

Task 2 : Read the paper *Varia - Cloud Architectures.pdf* pages (see CampUAS).
Summarise the advantages of this architectures compared to you implementation
from exercise 6.

Compared to a traditional implementation like the one from Exercise 6, cloud architecture offers the following benefits:

1. **Almost Zero Upfront Infrastructure Investment**
   → No need to buy servers. My application can be deployed in the cloud without major initial costs.

2. **Just-in-Time Infrastructure**
   → Resources scale automatically depending on current traffic/load. Useful for systems like dynamic notification services.
3. **Efficient Resource Utilization**
   → Cloud platforms avoid overprovisioning. My system can allocate resources only when needed.
4. **Usage-Based Pricing**
   → You pay only for what you use, making the application cost-effective during lower usage times.
5. **Faster Processing via Parallelism**
   → Tasks can run in parallel (e.g., checking multiple websites at once), improving speed.
6. **Scalable and Loosely Coupled Architecture**
   → Components like my observer and strategy pattern remain independent and scalable, which fits cloud principles well.
7. **Resilience & Auto-Recovery**
   → If something crashes, cloud platforms restart services automatically. This boosts reliability.
8. **Support for Distributed Processing**
   → Cloud infrastructure allows for running parts of the system in parallel across multiple machines, which is ideal for complex monitoring tasks.

Task 3: How would you redesign you application using the mentioned cloud services.

Draw a reiterated cloud architecture for your application.

- **The drawing is in a Pdf also in github**

Components and Associated Cloud Services:

1. **User Management**
   – **Cloud Service**: AWS Cognito / Firebase Authentication
   – Handles user login, registration, and secure access.
2. **Website Monitoring Logic**
   – **Cloud Service**: AWS Lambda / Google Cloud Functions
   – Serverless function that checks websites on a schedule.
3. **Data Storage**
   – **Cloud Service**: Amazon DynamoDB / Firestore
   – Stores user preferences, monitoring history, and website data.
4. **Notification Service**
   – **Cloud Service**: AWS SNS / Firebase Cloud Messaging
   – Sends alerts via email, push notification, or SMS.
5. **Frontend/Web Interface**
   – **Cloud Service**: AWS Amplify / Firebase Hosting
   – Hosts the user interface for interacting with the system.
6. **Monitoring & Logging**
   – **Cloud Service**: AWS CloudWatch / Google Cloud Logging
   – Tracks metrics and logs for debugging or alerts.

Task 4: Identify a specific programming task that an AI-assisted coding tool fails to solve correctly. Document the original query and the suggested solution. Provide an explanation of why the solution is incorrect.

Original Task / Prompt tot he AI:

"Write a Python function that returns `True` *if a given string is a palindrome, ignoring spaces, punctuation, and capitalization."*

AI-Suggested Solution:

```
python
KopierenBearbeiten
def is_palindrome(s):
    s = s.lower()
    s = s.replace(" ", "")
    return s == s[::-1]
```

Why the AI-Suggested Solution Is Incorrect:

While the AI correctly lowers the case and removes spaces, it **does not handle punctuation or special characters** (such as commas, periods, etc.). Therefore, this function would fail for palindromes like:

> **"A man, a plan, a canal: Panama"**
>
> This should return `True`, but the function returns `False` because of the commas and colon, which are not removed.

Corrected Solution:

```
import string

def is_palindrome(s):
    s = s.lower()
    s = ''.join(c for c in s if c.isalnum())  # Keep only letters and
numbers
    return s == s[::-1]
```

Explanation of the Fix:

- Instead of only removing spaces, the corrected version removes **all non-alphanumeric characters**, including punctuation.
- This ensures that only the meaningful characters (letters and digits) are considered when checking for a palindrome
- Now the input `"A man, a plan, a canal: Panama"` returns `True` as expected.

AI tools like Copilot and ChatGPT are powerful, but they sometimes overlook edge cases — especially those involving text normalization or data sanitization.

It's important to **critically evaluate** AI-generated code and test it with edge cases.