Wandering and getting lost: the architecture of an app activating local communities on dementia issues

Nicklas S. Andersen, PhD Student in CS
Marco Chiarandini, Asc. Prof. in CS
Jacopo Mauro, Asc. Prof. in CS

SEH 2021, June, 2021
Outline

1. Sammen Om Demens
2. Implementation
3. Experimental Setup
4. Results
5. Conclusion & Future Work
Sammen Om Demens (SOD)
- Introduction

What is SOD?
- An app implemented for a Danish municipality
What is SOD?
- An app implemented for a Danish municipality

Motivation:
- Improve the handling of cases where people with dementia get lost
- Use new technological innovations in doing so
Sammen Om Demens (SOD)

- Introduction

What is SOD?
- An app implemented for a Danish municipality

Motivation:
- Improve the handling of cases where people with dementia get lost
- Use new technological innovations in doing so

Goals:
- Create awareness about dementia among ordinary citizens
- Involve ordinary citizens in helping persons with dementia
- Alleviate the anxiety of persons with dementia and caregivers
The goals are accomplished through features of the SOD application.
Sammen Om Demens (SOD)

- Overview

- The goals are accomplished through features of the SOD application
Sammen Om Demens (SOD)

- Overview

The goals are accomplished through features of the SOD application:
The goals are accomplished through features of the SOD application.
Sammen Om Demens (SOD)
- Overview

- The goals are accomplished through features of the SOD application

- Backend system requirements:
  - It should be scalable and able to process data efficiently and reliably
  - It should be maintainable and structurally flexible
The goals are accomplished through features of the SOD application:

- A knowledge bank
The goals are accomplished through features of the SOD application:
- A knowledge bank
- A recreational activity calendar
Sammen Om Demens (SOD)
- Overview

The goals are accomplished through features of the SOD application:
- A knowledge bank
- A recreational activity calendar
- A help component

Main SOD features
The goals are accomplished through features of the SOD application:

- A knowledge bank
- A recreational activity calendar
- A help component
Sammen Om Demens (SOD) - Overview

The goals are accomplished through features of the SOD application:
- A knowledge bank
- A recreational activity calendar
- A help component

Backend system requirements:
The goals are accomplished through features of the SOD application:
- A knowledge bank
- A recreational activity calendar
- A help component

Backend system requirements:
- It should be scalable and able to process data efficiently and reliably
The goals are accomplished through features of the SOD application:
- A knowledge bank
- A recreational activity calendar
- A help component

Backend system requirements:
- It should be scalable and able to process data efficiently and reliably
- It should be maintainable and structurally flexible
Outline

1. Sammen Om Demens
2. Implementation
3. Experimental Setup
4. Results
5. Conclusion & Future Work
Implementation

- The entry point of the system is an Ingress Controller.
  - It redirects traffic to the Orchestrator.
  - It also reserves a direct route to Additional Services.
Implementation

- The entry point of the system is an **Ingress Controller** ①
Implementation

- The entry point of the system is an Ingress Controller ①
- It redirects traffic to the Orchestrator ②
Implementation

- The entry point of the system is an **Ingress Controller** ①
- It redirects traffic to the **Orchestrator** ②
- It also reserves a direct route to **Additional Services** ⑥
Implementation

- The Orchestrator routes requests to other microservices through Redis.

Diagram: Implementation flowchart showing the interaction between the Orchestrator, Ingress Controller, Redis, and other services.
Implementation

- The **Orchestrator** routes requests to other microservices through **Redis**
- It handles the creation, activation, deletion, update, and retrieval of users.
Implementation

- The **Orchestrator** routes requests to other microservices through **Redis**
- It handles the creation, activation, deletion, update, and retrieval of users
- It makes it easy to authorize and authenticate a user in a single place
Implementation

- The **Orchestrator** routes requests to other microservices through **Redis**
- It handles the creation, activation, deletion, update, and retrieval of users
- It makes it easy to authorize and authenticate a user in a single place
- It handles WebSocket connections
The help component is implemented by 3 and 4.
Implementation

- The help component is implemented by ③ and ④.
- Microservice ③ acts as a database buffer and handles bulk operations on raw data.
Implementation

- The help component is implemented by 3 and 4.
- Microservice 3 acts as a database buffer and handles bulk operations on raw data.
- Microservice 4 handles coordination of function execution triggered through the OpenFaaS Gateway 7.
Implementation

- The help component is implemented by ③ and ④.
- Microservice ③ acts as a database buffer and handles bulk operations on raw data.
- Microservice ④ handles coordination of function execution triggered through the OpenFaaS Gateway ⑦.
- OpenFaaS provides the infrastructure for implementing the detection algorithms.
The recreational activity calendar is implemented by ⑤
Outline

1. Sammen Om Demens
2. Implementation
3. Experimental Setup
4. Results
5. Conclusion & Future Work
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
  - Kubernetes autoscaling configuration:
    - Min & max number of replicas
    - CPU and memory resource requests
    - Autoscaling triggered based on:
      - CPU utilization for microservices
      - Requests per second for OpenFaaS functions

Master Node
Worker Node

Standard D16v4 VM:
- 16 vCPUs
- 32 GB RAM
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
  - Kubernetes autoscaling configuration:
    - Min & max number of replicas
    - CPU and memory resource requests
    - Autoscaling triggered based on:
      - CPU utilization for microservices
      - Requests per second for OpenFaaS functions
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage

- Kubernetes autoscaling configuration:
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage

- Kubernetes autoscaling configuration:
  - Min & max number of replicas
Experimental Setup
- System Configuration

- Microsoft Azure Setup:
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage

- Kubernetes autoscaling configuration:
  - Min & max number of replicas
  - CPU and memory resource requests
Experimental Setup
- System Configuration

Microsoft Azure Setup:
- Kubernetes cluster consisting of a master and a worker node
- General-purpose VMs running Ubuntu and using standard HDD storage

Kubernetes autoscaling configuration:
- Min & max number of replicas
- CPU and memory resource requests
- Autoscaling triggered based on:
  - CPU utilization for microservices
  - Requests per second for OpenFaaS functions
Experimental Setup
- System Configuration

**Microsoft Azure Setup:**
- Kubernetes cluster consisting of a master and a worker node
- General-purpose VMs running Ubuntu and using standard HDD storage

**Kubernetes autoscaling configuration:**
- Min & max number of replicas
- CPU and memory resource requests
- Autoscaling triggered based on:
  - CPU utilization for microservices
Experimental Setup
- System Configuration

- **Microsoft Azure Setup:**
  - Kubernetes cluster consisting of a master and a worker node
  - General-purpose VMs running Ubuntu and using standard HDD storage

- **Kubernetes autoscaling configuration:**
  - Min & max number of replicas
  - CPU and memory resource requests
  - Autoscaling triggered based on:
    - CPU utilization for microservices
    - Requests per second for OpenFaaS functions
Experimental Setup

- Load Test Description

Load tests target the infrastructure used by the help component. They measure the performance of the system under the following conditions:

- HTTP POST requests are sent to the backend system every $\tau \in U(1,5)$ seconds
- Invoked functions compute a moving average of the incoming location data
Experimental Setup
- Load Test Description

- Load tests target the infrastructure used by the help component
Experimental Setup
- Load Test Description

- Load tests target the infrastructure used by the help component
- They measure the performance of the system under the following conditions:
Experimental Setup
- Load Test Description

- Load tests target the infrastructure used by the help component
- They measure the performance of the system under the following conditions:
  - HTTP POST requests are sent to the backend system every $\tau \in U(1, 5)$ seconds
Experimental Setup
- Load Test Description

- Load tests target the infrastructure used by the help component

- They measure the performance of the system under the following conditions:
  - HTTP POST requests are sent to the backend system every $\tau \in U(1, 5)$ seconds
  - Invoked functions compute a moving average of the incoming location data
Outline

1. Sammen Om Demens
2. Implementation
3. Experimental Setup
4. Results
5. Conclusion & Future Work
Outline

1 Sammen Om Demens
2 Implementation
3 Experimental Setup
4 Results
5 Conclusion & Future Work
Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - Structurally flexible and maintainable system

- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques
  - Build out the OpenFaaS function execution pipeline
Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing

- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques
  - Build out the OpenFaaS function execution pipeline
Conclusion & Future Work

■ We designed the backend system of SOD relying on:
  ▶ Microservices and serverless computing
  ▶ Structurally flexible and maintainable system
Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - Structurally flexible and maintainable system

- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques
  - Build out the OpenFaaS function execution pipeline
Conclusion & Future Work

■ We designed the backend system of SOD relying on:
  ▶ Microservices and serverless computing
  ▶ Structurally flexible and maintainable system

■ We confirmed with load tests that:
  ▶ The architecture is able to cope with different types of load and scale appropriately
Conclusion & Future Work

We designed the backend system of SOD relying on:

- Microservices and serverless computing
- Structurally flexible and maintainable system

We confirmed with load tests that:

- The architecture is able to cope with different types of load and scale appropriately
- The architecture is able to handle data efficiently and reliably

We need to extend and improve the functionalities:

- Implement artificial intelligence techniques
- Build out the OpenFaaS function execution pipeline
Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - Structurally flexible and maintainable system

- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

- We need to extended and improve the functionalities:
Conclusion & Future Work

- We designed the backend system of SOD relying on:
  - Microservices and serverless computing
  - Structurally flexible and maintainable system

- We confirmed with load tests that:
  - The architecture is able to cope with different types of load and scale appropriately
  - The architecture is able to handle data efficiently and reliably

- We need to extended and improve the functionalities:
  - Implement artificial intelligence techniques
Conclusion & Future Work

We designed the backend system of SOD relying on:
- Microservices and serverless computing
- Structurally flexible and maintainable system

We confirmed with load tests that:
- The architecture is able to cope with different types of load and scale appropriately
- The architecture is able to handle data efficiently and reliably

We need to extend and improve the functionalities:
- Implement artificial intelligence techniques
- Build out the OpenFaaS function execution pipeline
Thank you for your attention!