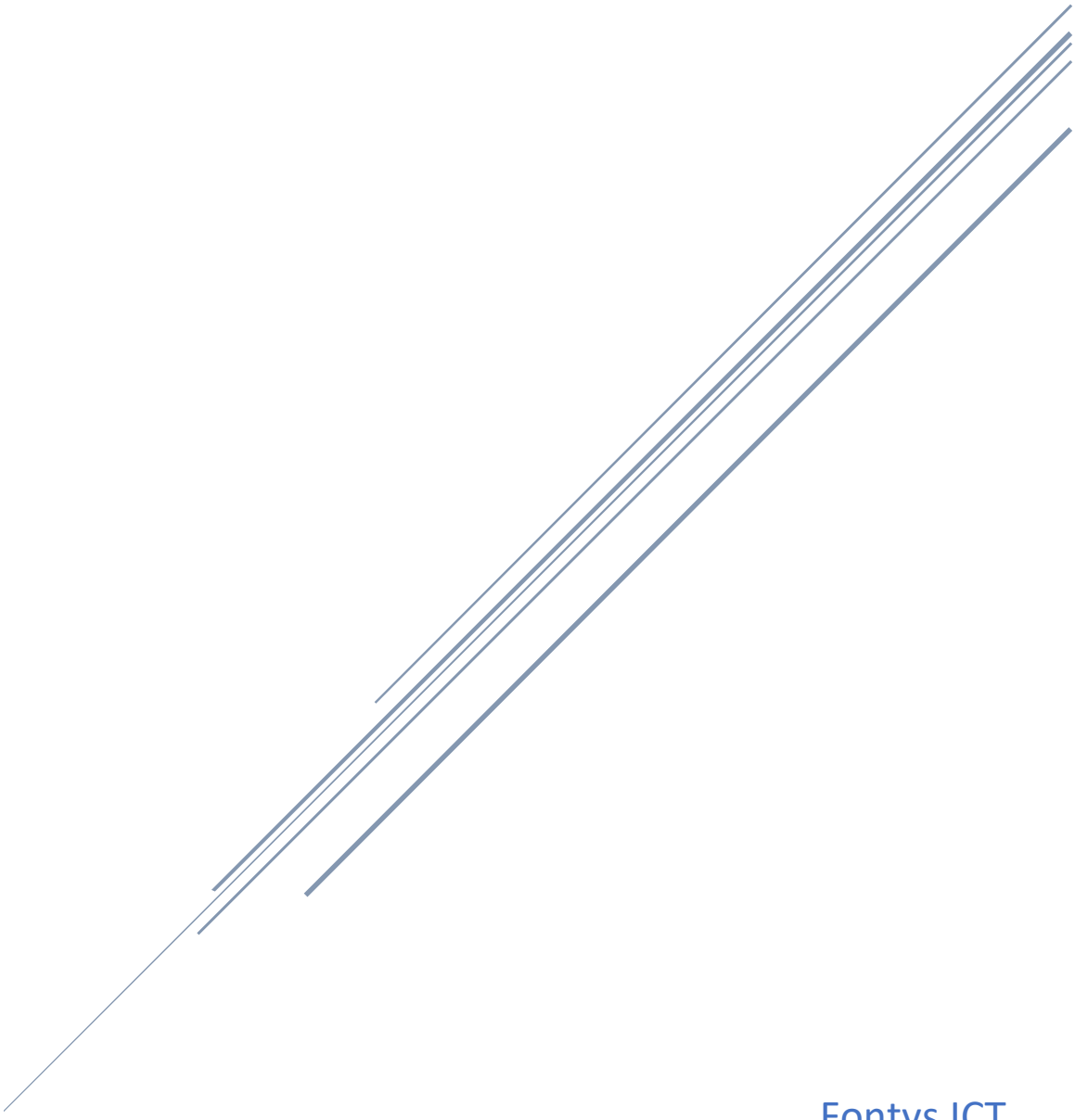
# WORKPLACE ORGANIZATION APPLICATION

## Technical Design Document



Fontys ICT
Tsanko Nedelchev

# Contents

# INTRODUCTION

This technical design document provides an overview of the system architecture, design, and implementation details for the "Workplace Organizer Application". The application consists of multiple microservices, including a user authentication and authorization service, a meeting scheduling service, a notifications service, and a meeting notes service. Each microservice is designed to handle a specific aspect of the application, with communication between the microservices being managed through RabbitMQ messaging. The application will be developed using TypeScript, Node.js, Express, MongoDB, and Mongoose. The aim of this document is to provide a high-level understanding of the architecture and design of the application, as well as to outline the key features of each microservice.

# ARCHITECTURE OVERVIEW

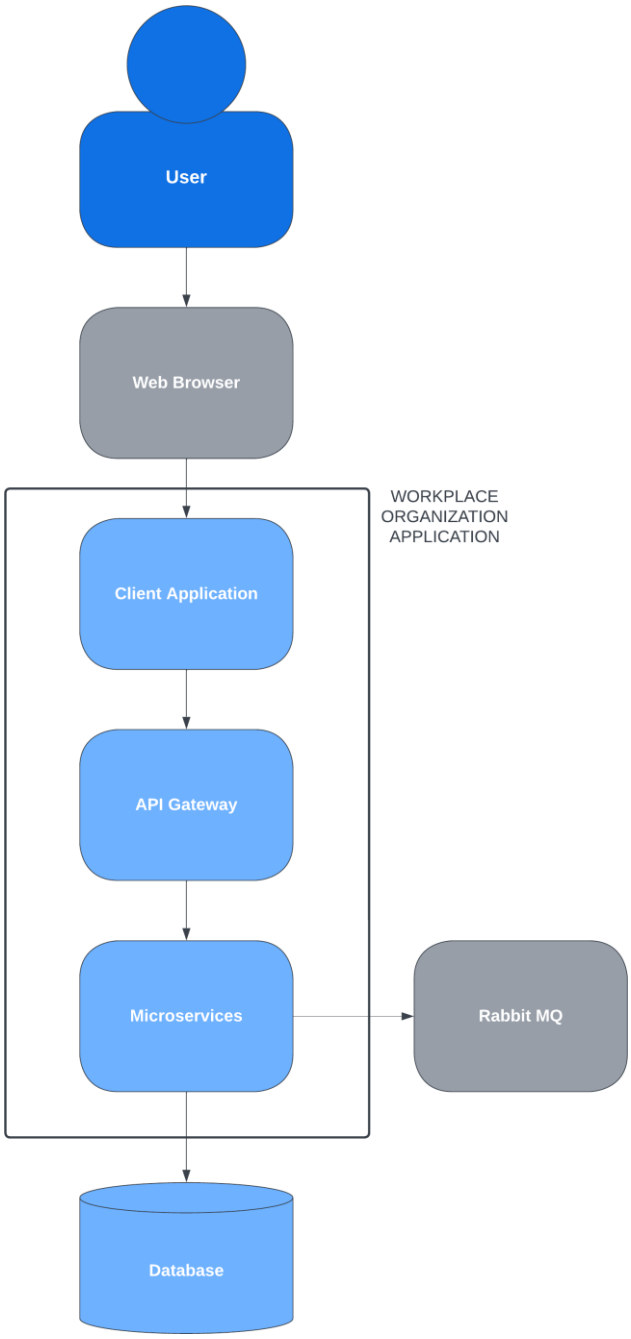



*Figure 1 - C4 diagram - level 2*

The application's architecture is designed to be a microservices-based architecture, consisting of several microservices that interact with each other via Rabbit MQ messaging. A gateway is used to manage the external API interface and to route requests to the appropriate microservices.

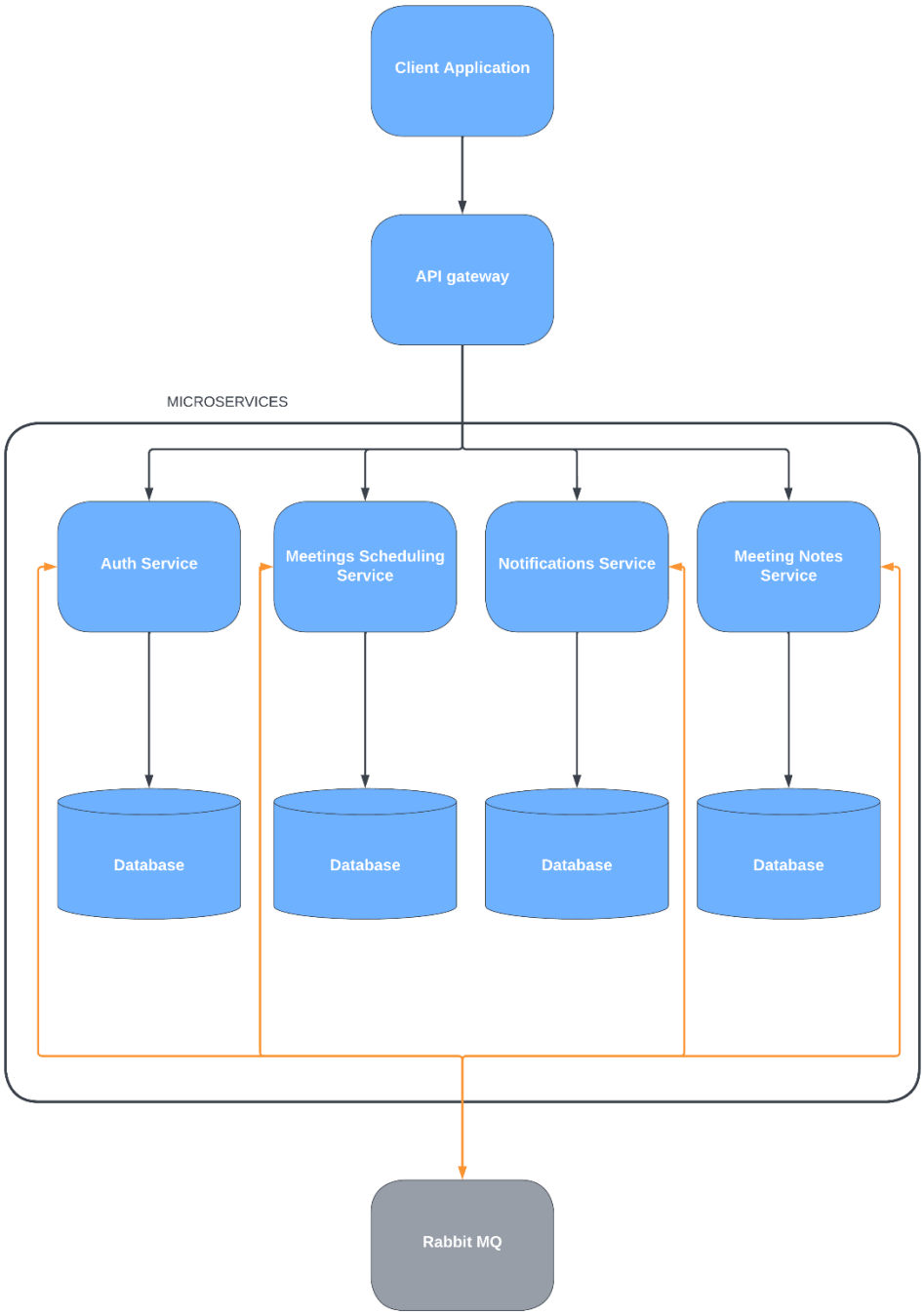



*Figure 2 - C4 diagram - level 3*

The back end system consists of the following microservices:

**Authentication and authorization microservice -** provides user authentication and authorization functionality, including login, logout, and access control.

**The Meetings Scheduling microservice -** provides functionality for scheduling meetings, including creating, updating, and deleting meetings.

**The Notifications microservice -** responsible for sending notifications to users when they have upcoming meetings, as well as sending notifications for other system events, such as new team invitations.

**The Meeting Notes microservice -** provides functionality for creating and editing meeting notes, as well as storing and retrieving them from the database.

## MICROSERVICE ARCHITECTURE

Each microservice is designed to be loosely coupled and independently deployable which allows easy scaling and maintenance. The microservices interact with each other via Rabbit MQ – a message broker, with each microservice consuming messages from another to perform tasks.

The microservices are implemented using NodeJS and TypeScript with the Express framework and the TSOA open API documentation library for easy automation of API swagger documentation.

## DATABASE

Each microservice is connected to an instance of MongoBD database providing a flexible and scalable database management solution.