2021

TAHS – Assignment 1 (DRAFT)

GROUP 13A

AUTHORS :

- ANDY LI
- TANESHWAR PARANKUSAM
- MEHMET SÖZÜDÜZ
- MIHNEA TOADER
- JEGOR ZELENJAK
- JEROEN BASTENHOF

Table of Contents

1 Bounded Contexts

- **Users** – represents the users of the system: *students* (who can become candidate TAs), *candidate TAs* (who can become TAs), *TAs* (who can become students again), *lecturers* (who are responsible for their courses and for hiring TAs), *admin* (a lecturer who has the full access to the system).
- **Authentication** – allows users to be authenticated (identified) and given permissions (access rights) based on their role in the system. The permission management is regulated by passing session tokens to users at authentication. Session tokens that were sent with requests will be checked here for authorization.
- **Courses** – represents the courses that are present within the system. Contains *information* about the course, e.g., name, recruited TAs, number of students.
- **Hiring procedure** – facilitates the *hiring* of students for TA positions by allowing users to *register* for a TA position (taking into account the timing constrains for the application) and allowing lectures to hire new TA's.
- Hour management – allows TAs to declare their hours for a specific course. Also, allows lecturers to approve / reject the declared hours.

Each bounded context is mapped to a single microservice design with identical name. A functional description of the interactions per bounded context can be found in the subsections below.

1.1 Users

The Users microservice:

- Deals with the **functionality for each specific user** (students, candidate TAs, TA, Lecturers).
- Interacts with the Authentication microservice to **give access to authorized users** based on their permissions.
- Interacts with **other microservices** depending on their roles.

1.2 Authentication

The Authentication microservice:

- Deals with **authenticating users** based on their NetID and password, and **passing a session token** to the clients who have been successfully identified.

1.3 Courses

The Courses microservice:

- Lecturers can create / edit courses.
- Grade and pass information will be stored in the database belonging to course.

1.4 Hour Management

The Hour Management microservice:

- Tracks the **TA's hours worked** and hours declared for every specific course.
- Lecturers can **approve the hours** worked for every TA.

1.5 Hiring Procedure

The Hiring Procedure microservice:

- Provides functionality to both candidate TAs and Lecturers to **submit** and **review applications**.
- Lecturers can **choose TAs** from these applications.
- TAs are **notified** of the decision.

1.6 Interactions

- Lecturers can **create courses**, **approve declared hours** by interacting with the Courses microservice. (*Users* → *Courses*)
- Lecturers can **hire TAs**, **see information** about candidates, **ask for recommendations** and **filter applications** by interacting with the Hiring Procedure microservice. (Users → Hiring Procedure)
- Students can **apply to become TAs** and **withdraw their candidacy** (unless the lecturer has started the selection procedure) by interacting with the Hiring Procedure microservice. (Users → Hiring Procedure)
- TAs can **declare hours worked and hours declared** by interacting with the Courses microservice. (Users → Courses)
- TAs can receive contracts to sign when the Hiring Procedure microservice interacts with the Users microservice. (Hiring Procedure → Users)
- Authentication interacts with Users to approve the requested actions. (Authentication → Users).

1.7 Explanations

We chose to build a microservice architecture based around a central microservice (**Users**) that interacts with the other microservices to provide the functionality. Users authenticate themselves

and, based on their roles, requests get sent to the other microservices. Managing different user types is aggregated into a single microservice (Users) that acts as a central hub.

Authentication is specific enough to warrant building a separate microservice for the provided functionality. Every request is passed to this microservice to either assign a token or check for authorization.

Since **Courses** are a core concept in the university learning structure, we chose to integrate the functionality in a separate microservice. This microservice receives requests from Users to modify and add course specific information, such as hours worked or approving the hours worked.

The Hiring Procedure microservice provides most of the functionality that is specific to our scenario, i.e. submitting and withdrawing applications, reviewing them and offering TA recommendations. Since this is a lot of functionality, we chose to put it in a separate microservice.

Our architecture is **vertically scaled enough** to provide modularity and scalability, but **not too large** as to not hamper our understanding of it and to not have interaction overhead. **Maintainability** is also an important factor of our system architecture. Inherently, systems designed around microservices are more easily maintained and this is also the case for us. Since mocking microservices is readily done, good **testability** is also a feature of our system.

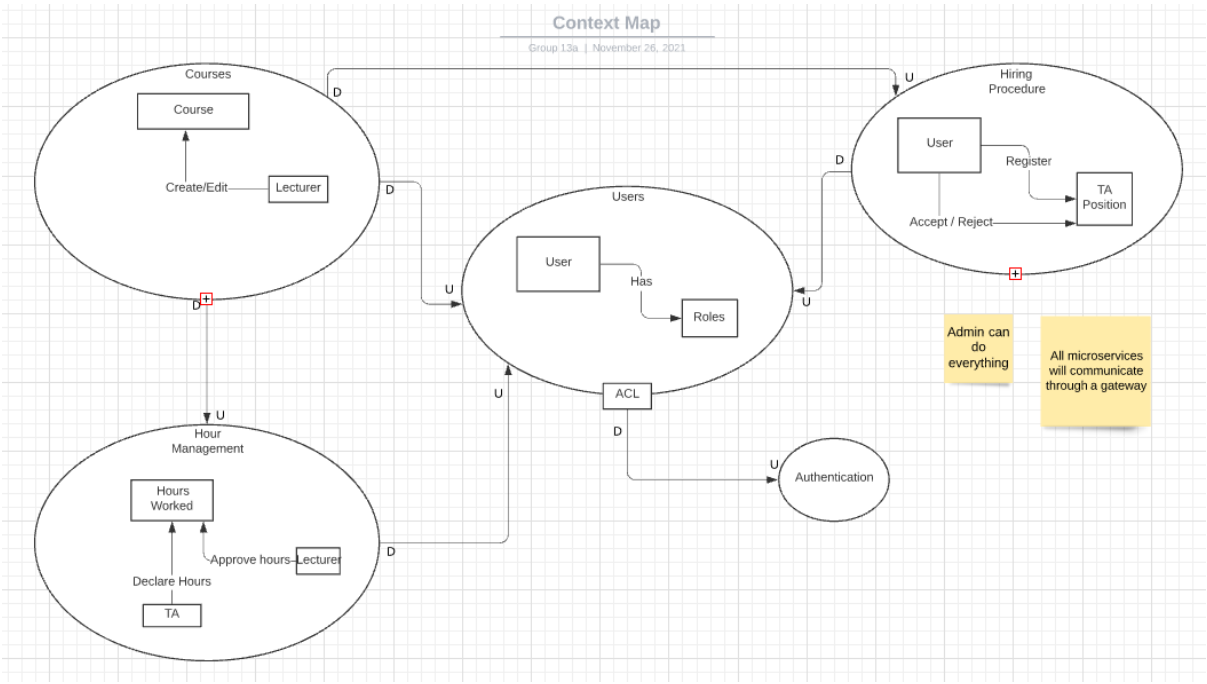

Figure 1: Context map of the derived contexts.

1.8 Component diagram

A high level component diagram of the previously described components can be found in Figure 2. The diagram shows a minimized version of the high-level interaction between microservices. The *REST API* and *DataAccess* interfaces act as groups to represent the overall API and data paths between microservices respectively. Important interfaces have been listed separately to give a visual indication of what the microservice can be used for.

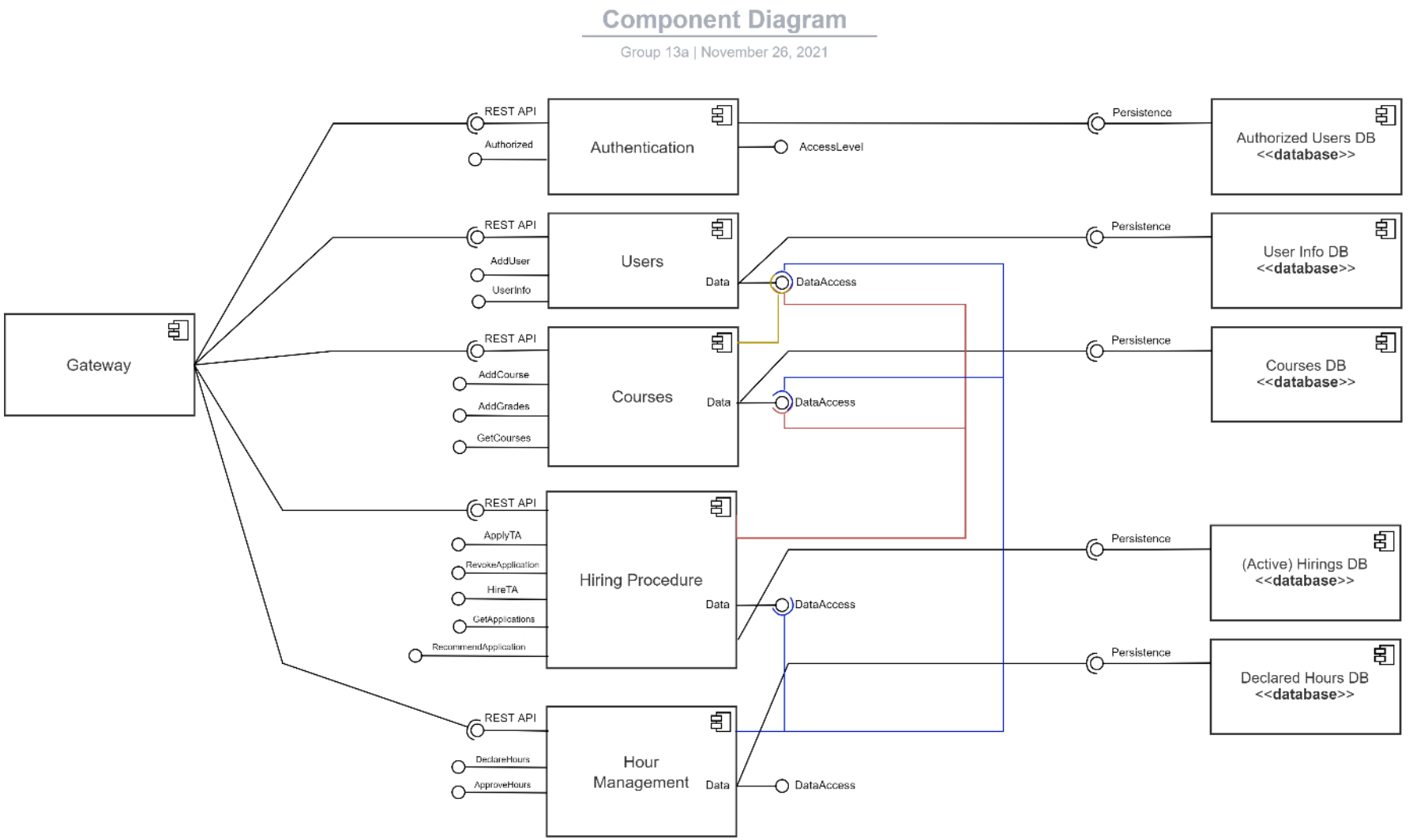

Figure 2: High-level component diagram of the derived microservices.