

---

# Software Design Document

for the Student Representative Database

Andras Balla

Olga Christensen

Pär Eriksson

Fredrik Olsson

# 1 Table of content

<b>1 Table of content</b>	<b>2</b>
<b>2 Introduction</b>	<b>4</b>
2.1 Purpose	4
2.2 Scope	4
2.3 Glossary	6
<b>3 General priorities</b>	<b>7</b>
<b>4 Outline of the design</b>	<b>7</b>
4.1 Software architecture	7
4.2 Technology	8
4.3 Deployment design	8
<b>5 Major design issues</b>	<b>9</b>
<b>6 Details of the design</b>	<b>10</b>
6.1 Design patterns and techniques	10
6.1.1 System/software design	10
6.2 UML diagrams	11
6.2.1 API server	11
6.2.2 Web application	11
6.3 Sequence diagrams	12
6.3.1 Faculty	12
6.3.1.1 Create one faculty	12
6.3.1.2 Get one faculty	13
6.3.1.3 Get all faculties	13
6.3.1.4 Update one faculty	13
6.3.1.5 Delete one faculty	14
6.3.2 Councils	14
6.3.2.1 Create one council	14
6.3.2.2 Find one council	14
6.3.2.3 Update one council	15
6.3.2.4 Delete one council	15
6.3.3 Council Instances	15
6.3.3.1 Create one council instance	15

6.3.3.2 Get one council instance	16
6.3.3.3 Update one council instance	16
6.3.3.4 Delete one council instance	16
6.4 Employees	17
Create employee	17
Get one employee	17
Get all employees	17
Update employee	18
Delete employee	18
6.4.1 Employee position	18
Create employee position	18
Get on employee position	18
Update employee position	19
Delete employee position	19
6.4.2 Users	19
Create user	19
Get one user	20
Get all users	20
Update one user	20
Delete user	21
6.4.3 User positions	21
Create user position	21
Get one user position	21
Get all user positions	22
Update one user	22
Delete one user	22
6.5 DB design	22
6.5.1 ER diagram	23
6.5.1.1 Entities and Relationships	23
6.5.2 Normalization and anomalies	25
6.6 UI design	26
6.6.1 Input validation	26
6.6.2 Error and information messages	27
6.6.3 UI for Use Cases	28
6.6.3.4 UC-02.1 Create new student profile + UC-02.3 Update student profile	28

6.6.3.5 UC-02.2 View student profile	29
6.6.3.6 UC-02.5 Create new staff profile + UC-02.6 Update staff profile	31
6.6.3.7 UC-02.6 View staff profile	31
6.6.3.8 UC-03.1 Create new council profile	33
6.6.3.9 UC-03.2 View council	34
6.6.3.10 UC-03.3 Update council	35
6.6.3.11 UC-03.5 Add staff to council	36
6.6.3.12 UC-03.7 Add applicant to council	36
6.6.3.15 UC-06.1 View election candidates	37
6.6.3.16 UC-06.2 Request confirmation from candidates	38
6.6.3.17 UC-06.3 Elect/appoint representative	38
6.6.3.18 UC-07.1 Add a meeting to council	38
6.6.3.19 UC-07.2 Track meeting participants + UC-07.3 Edit meeting	39
<b>7 Appendix - links and other documents</b>	<b>39</b>
7.1 API Documentation	39
7.2 Application repository	39
7.3 All documents from the project	39

## 2 Introduction

### 2.1 Purpose

The purpose of this document is to provide a detailed description of software architecture and system design for the Student Representative Database application.

This document is intended for the developers that are going to be implementing the basic prototype of the system as well as those who are going to continue working on its expansion.

It can also be used by Linnéstudenterna to present the idea of the Student Representative Database application to their superiors and colleagues.

The main objective of the application is to aid the stakeholders in maintaining and managing a database of student representatives, with focus on their interest and participation in a number of LNU's councils.

### 2.2 Scope

This document will describe how to best accommodate system's requirements and constraints defined in the Software Requirements Specification document. This document focuses on the critical parts of the system, those that are needed to provide a proof of concept for building a full-scale system in the future.

Core features:

- viewing general information as a guest

- list of councils grouped under faculties
- list of council with open SR positions
- detailed information about specific council
- contact information about student representatives of a specific council
- contact information about management of a specific council
- managing faculties
  - creating, viewing, updating, deleting
- managing councils
  - creating, viewing, updating, deleting
  - managing applicants
  - managing representatives
  - managing employees
- managing students
  - creating, viewing, updating, deleting
  - managing applications
- managing employees
  - creating, viewing, updating, deleting
  - managing council positions

Additional features:

- authentication and authorization
- managing council's meetings
- managing the election process

This document will not cover a testing strategy of the prototype/system.

## 2.3 Glossary

Term	Meaning
Linnéstudenterna	the Student Union representing students at LNU
Council	a group with specific responsibility, representing student interests
Employee	LNU employees managing the work of a council. Can have several roles: convener ( <i>sammankallande</i> ), chairman ( <i>ordförande</i> ), secretary.
Student Representative (SR)	Linnéstudenterna have elected this person to be the official student representative for a specific council
Applicant	student that applied for a position in the council, but has not been elected (yet)
Election	Elections during the Linnéstudenterna board's monthly meetings. ( <i>Fyllnadsval</i> )
SRDB	Student Representative Database, the name of the application/system.
Council Instance	a council existing within specific period of time
Admin panel	Website for administrators and council members to manage the registry, requires a login with different access permissions.
Website	Website listing all existing councils and their description, including <ul style="list-style-type: none"><li>- an application form for becoming a SR</li><li>- account management feature for authenticated Users</li></ul>
Guest	Anonymous viewer of the Website
User	LNU student or PhD that at some point has shown interest to be a SR and has an account in SRDB.

### 3 General priorities

The main goal of Student Representative DB application is to give various users from various clients an access to one centralised registry / database. The database itself is relatively complex and it is being updated on regular basis, therefore the users should be able to easily access the accurate and relevant information. The users are divided in three major groups: students, LNU employees, Linnéstudenterna employees. Each group has its own goals in using the system and therefore a specific number of functions should be designated for each group.

Based on this information, we can summarise the following goals and constraints of the system:

- centralised db/registry
- easy access from any type of client (OS/browser independent)
- user-friendly (intuitive UI, responsive design)
- security of the stored information (authentication, authorisation, ssl, server security)

Another substantial constraint is the time - there are only three months allocated for the whole project, during which domain analysis, requirement elicitation, software design and the basic prototype should be completed.

### 4 Outline of the design

When deciding on which archetype of application should be used, the following options have been reviewed:

1. Mobile app
2. Rich Internet Application
3. Web application

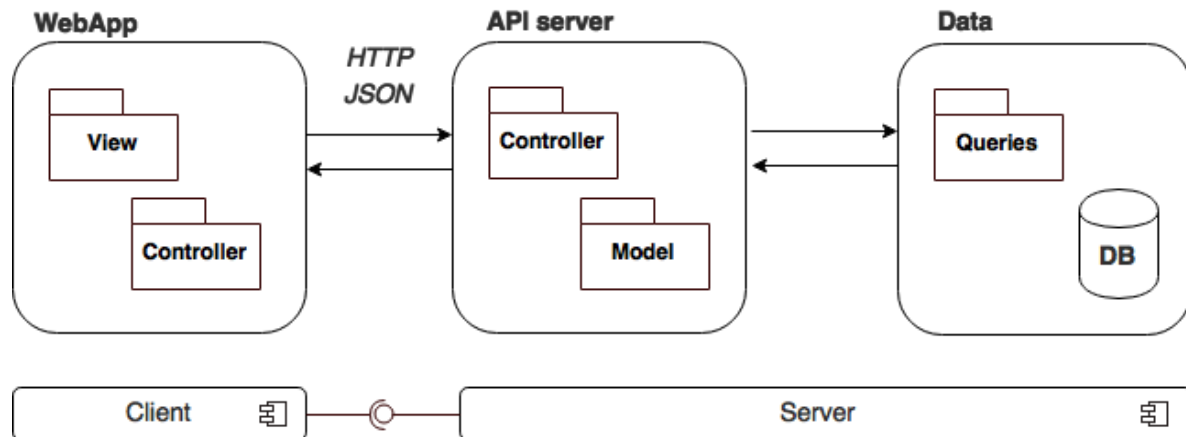
The third option satisfies all the goals and constraints defined earlier. The only potential disadvantage there is - it requires a continuous internet connection, however this was not deemed influential.

We can see that a mobile app can be of interest in the future to stimulate LNU student's interest and engagement in influencing their daily life at university through the channels offered by Student Representatives. However this type of application was not included in the scope of this document.

With all this in mind, a decision was made to create a web application with clearly divided user interface and application's business logic. Thus, the application was divided into two major components: client and server communicating by means of REST API in between. Choosing this approach opens up the possibility to build other types of client applications that uses the same API, such as mobile applications or desktop applications, as well as adding new features and business logic on the client side.

#### 4.1 Software architecture

As the *Student Representative Database* will be a RESTful client/server-type application, there are some architectural patterns to consider. After some discussions and considerations, we have decided to adopt an MVC-like approach, divided into three layers, which we call WebApp (client/presentation), API (application) and Data layers.



This approach enables us to separate concerns in the application using object oriented approaches, while opening up the possibility to expand with different client application types that can consume the same REST API to give the client's users more choices in how they use and access the data in the system.

As we want to make it easy to add different client types in the future by using the RESTful client/server approach, it was logical to go with a thin client versus a fat client, as we want the application logic to be behind the API. And while there are a few client options, our client expressed that they wanted a web application during the domain analysis.

## 4.2 Technology

Since this is a greenfield project and the client has no specific technological requirements, there is no constraints regarding the choice of technology. The list of technologies chosen is as follows:

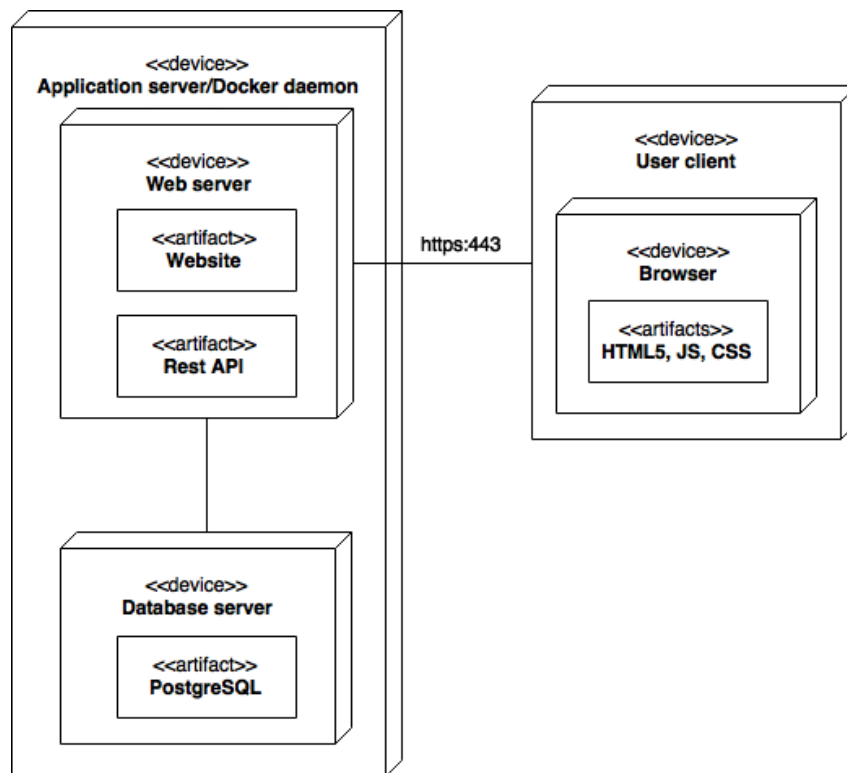
- HTML5 - markup language
- Bootstrap - front-end framework for faster and easier web development, including straightforward implementation of responsive design
- Handlebars - javascript library for building clean templates to simplify web development
- SASS - CSS extension
- TypeScript - class-based and object oriented super-set of the JavaScript language, it can be used both on client and server, compatible with Node and Express
- Node - run-time environment for executing JavaScript code server-side
- Express - web application framework for Node
- PostgreSQL - object-relational database system
- Sequelize - Node's ORM for PostgreSQL, it is promise-based, supports transactions, relations, etc.
- Docker - tool for application deployment automatisations inside software containers
- Nginx - HTTP web server and proxy

## 4.3 Deployment design

At this stage of the project, we don't see the need to distribute the application over several machines: the database is not going to be extremely big, so back-end and database can be located on the same server, i.e. using a non-distributed deployment pattern.

We have chosen to work with a container technology called Docker, which opens up possibilities when it comes to scaling the application up or down to fit future application load. Docker also helps developers have a development environment that very closely resembles the production environment, removing a lot of issues that can be associated with environment configurations.





The web server we have chosen to use is Nginx, which is a lightweight, fast and secure event-based web server, which works really well with Node applications. The web server shall be configured to use a TSL certificate to enforce the use of HTTPS, to protect the data sent between the client and the server.

## 5 Major design issues

**Problem:** one of the major requirements is to be able to store and review the information about the applicants and student representatives for each council for a specific period. Each council has a permanent set of employees, but the students that are involved in it vary greatly from semester to semester. Additionally the same student can be connected to a specific council under several terms.

**Solution:** creating a relation between council and students through a Council Instance, which is unique for each council within a certain period (more on this in 6.5 DB design).

**Problem:** creating a server that is independent of a client implementation, which would allow easy implementation of new clients, as well as possibility to reuse the server in other applications.

**Solution:** creating an interface between the business logic and user interface with help of a REST API (more on this in 6.1 Design patterns)

**Problem:** the application stores and processes personal information of students and employees, which makes the security paramount.

**Solution:** creating an authentication and authorisation components that verify the identity of the user and assign specific limited permissions to read and write data, as well using SSL encryption when working with the application.

**Problem:** the delivered prototype of the system will need implementation of new features as well as general maintenance with possibility of different teams working on the project.

**Solution:** following the principles of object oriented design, as well as producing a feasible API documentation

## 6 Details of the design

### 6.1 Design patterns and techniques

Developing a Node application differs slightly from traditional object oriented development. In Node.js environments, the singleton design pattern is a very common pattern for the different modules used in an application. JavaScript modules can to some extent be seen as classes in object oriented design, but due to the prototypal nature of JavaScript, they are often implemented quite differently from classical object oriented languages, such as Java or C#. Using TypeScript will give the developers a strongly typed language, which catches a good deal of bugs and issues in compile time, and is also more similar to a traditional object oriented programming language than JavaScript.

#### 6.1.1 System/software design

The SRDB system is divided into three subsystems: Web application, API server and a data layer. This design decision was made so that the system's sub parts would be easy to scale and replace if needed.

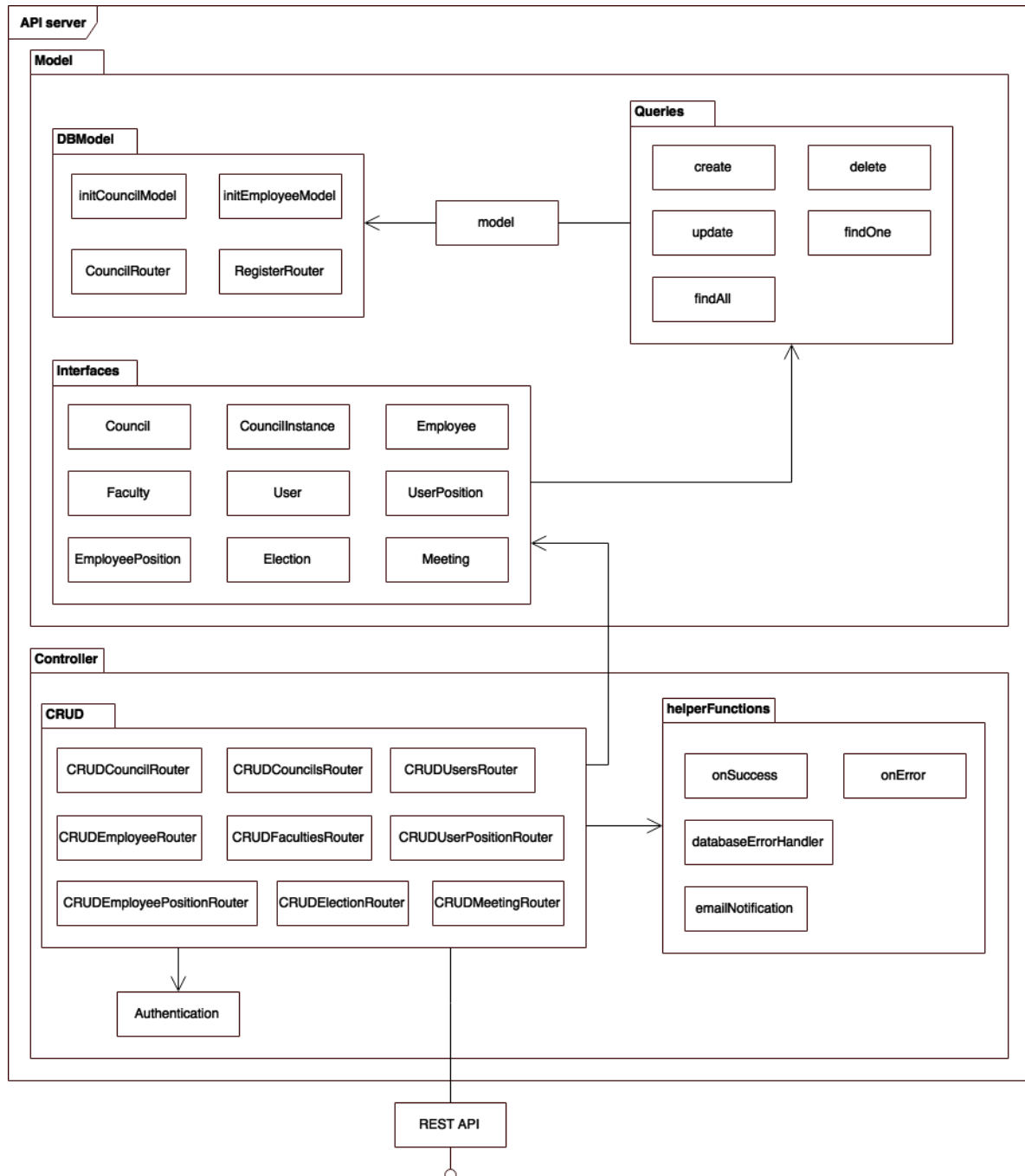
The Web application is responsible for the website to present the information from the API and to accept input for the API. The API is the link between the web app and the database. The API has a pre defined outputs for every input which will make it easy to swap out the web application to any other view. The data layer consists of the database and the data handling classes and queries - it is application's core layer responsible for storing and serving persistent data.

This approach will give us several benefits:

- It is easy to divide the workload within a team during production (there is no need for everybody to understand what's behind the API or for everybody understand all the parts in the web application)
- After the design of the API and the API documentation has been written down it's easier to split to work since work can be done on the web-application even though the API is not fully implemented.
- Additional changes can be made to API/web application without making any changes in other parts as long as there are no changes to what the API returns
- Such decoupling will make it easier to scale the different parts independently when/if the need arises.
- REST API can be easily scaled horizontally due to using HTTP and being stateless
- Allows easy implementation of additional types of clients in the future (e.g., mobile app)
- REST API approach makes the implementation more understandable and thus also easier for a new team to start to work on

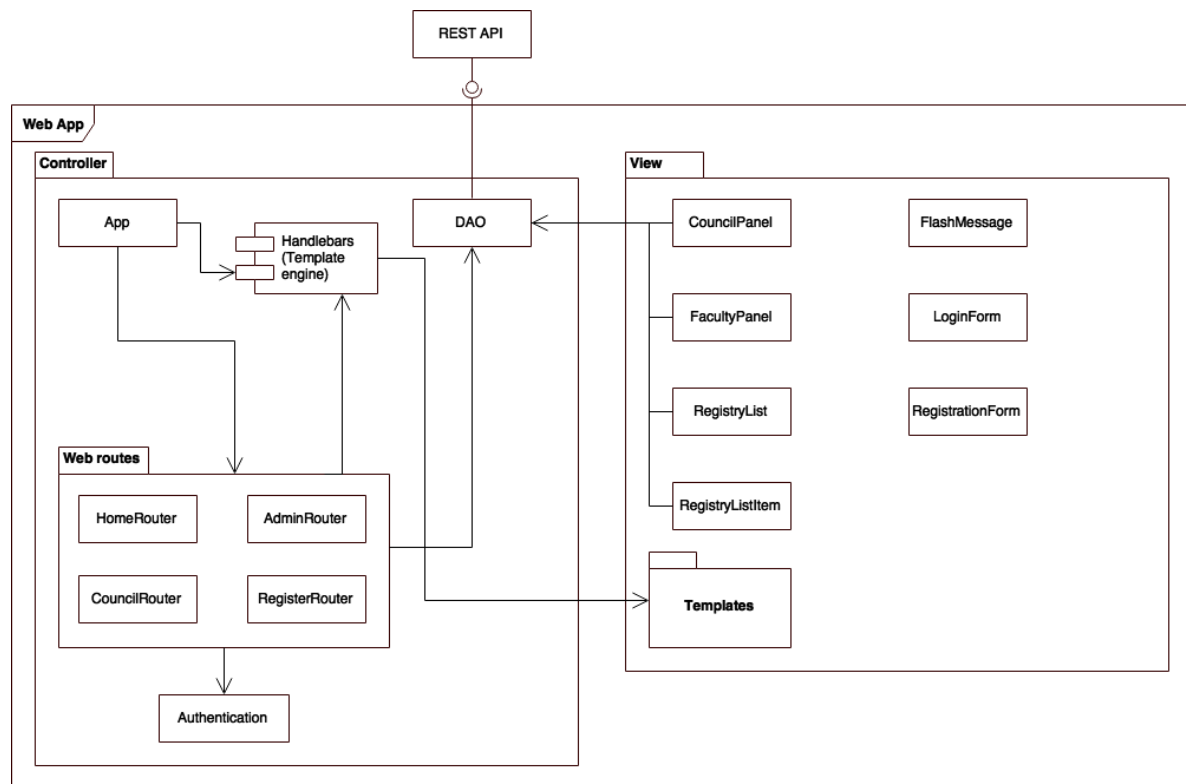
## 6.2 UML diagrams

### 6.2.1 API server



### 6.2.2 Web application

The `view` in the web application is in the end compiled into a single javascript file, which is used by the HTML generated by the template engine. Each module (class) in the view is loaded and activated on different HTML pages in the application.

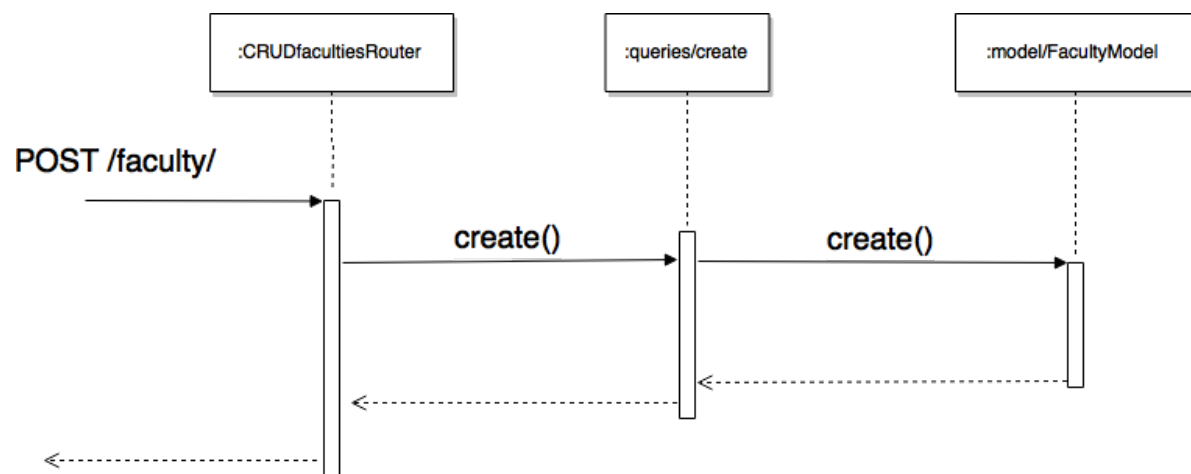


## 6.3 Sequence diagrams

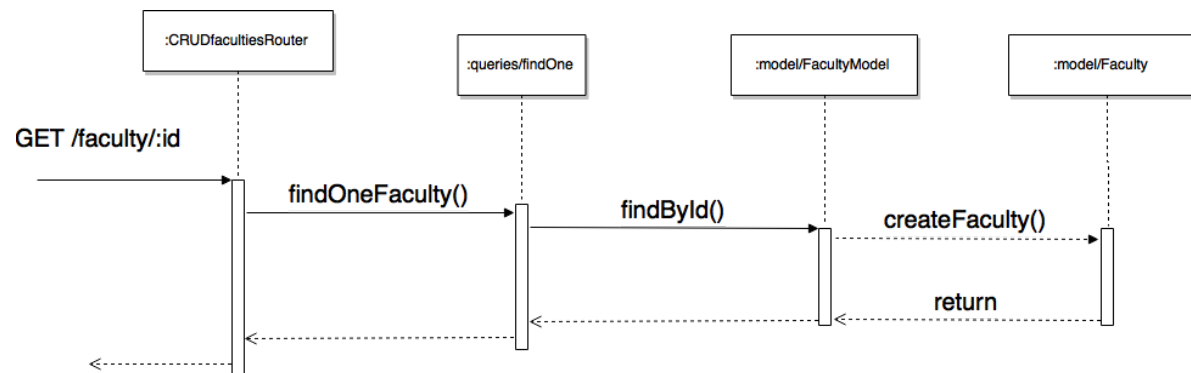
All API calls starts with `/api/v1`, but is left out in the diagrams for brevity. These diagrams depicts how the application handle the API calls to respond with data to the client.

### 6.3.1 Faculty

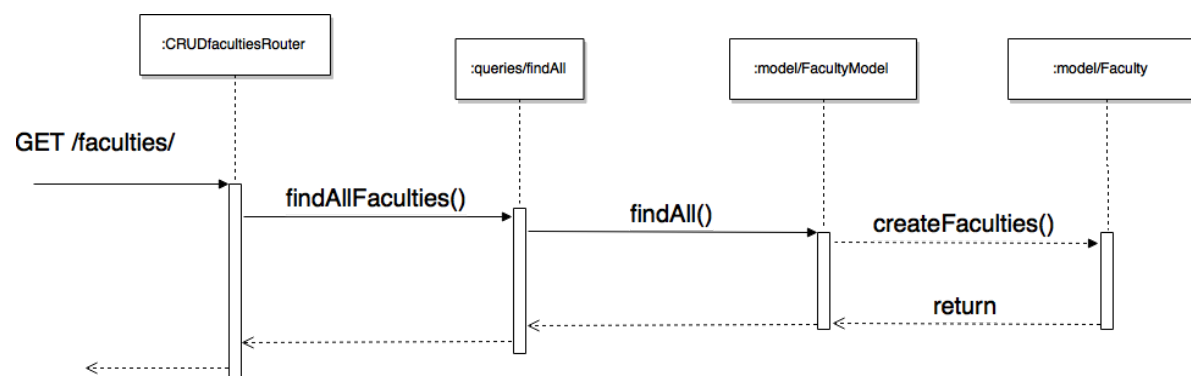
#### 6.3.1.1 Create one faculty



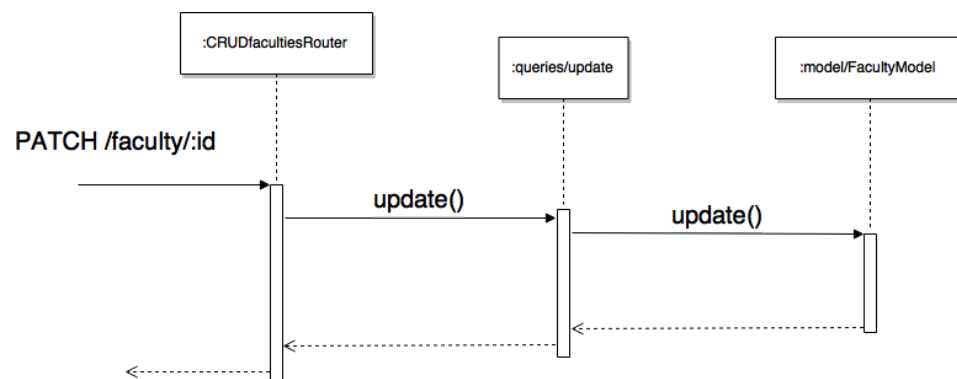
### 6.3.1.2 Get one faculty



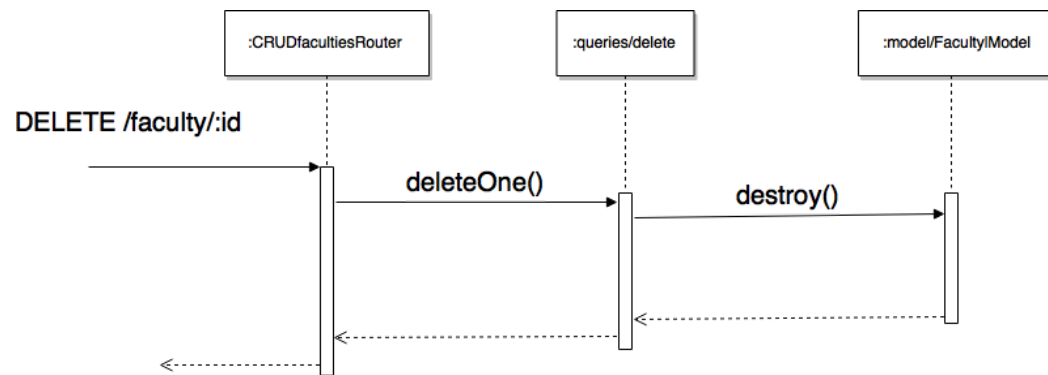
### 6.3.1.3 Get all faculties



### 6.3.1.4 Update one faculty

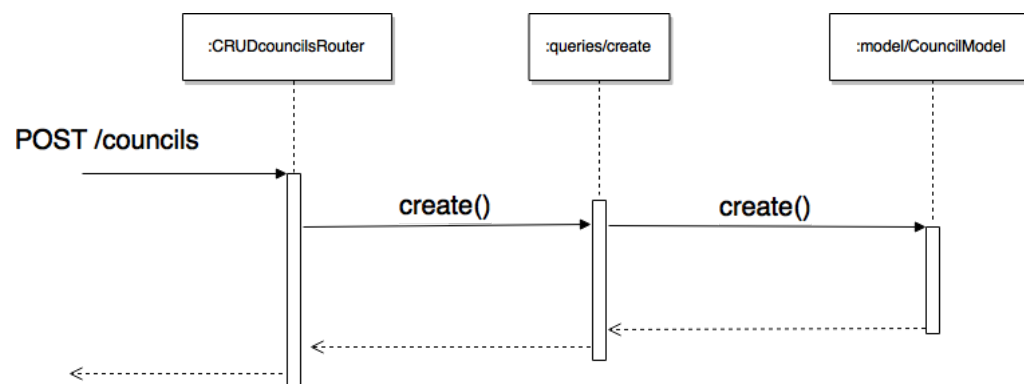


### 6.3.1.5 Delete one faculty

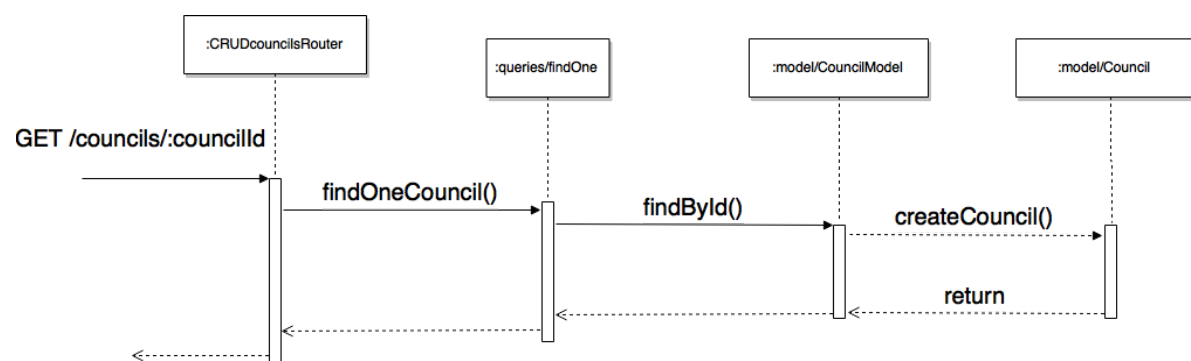


## 6.3.2 Councils

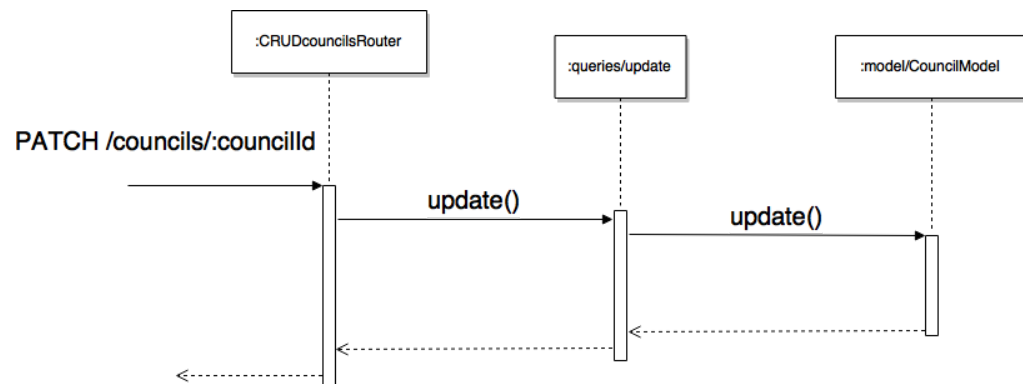
### 6.3.2.1 Create one council



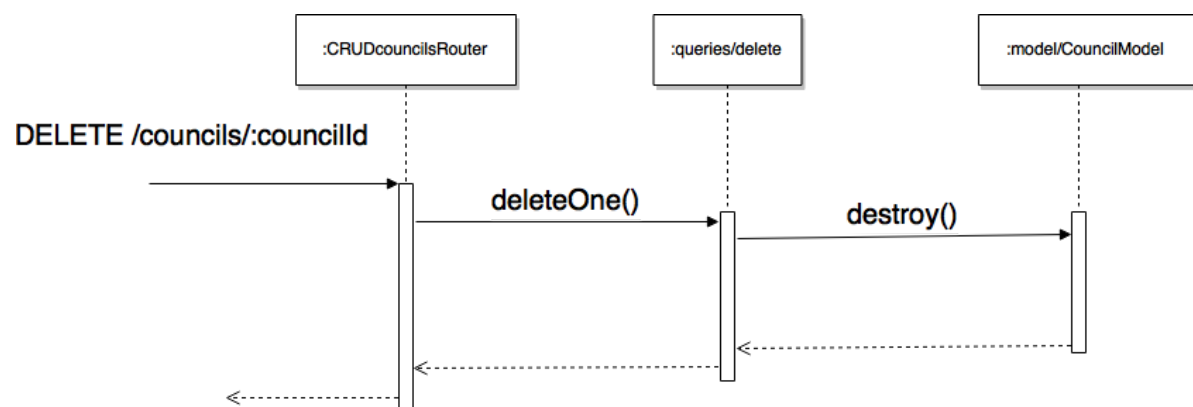
### 6.3.2.2 Find one council



### 6.3.2.3 Update one council

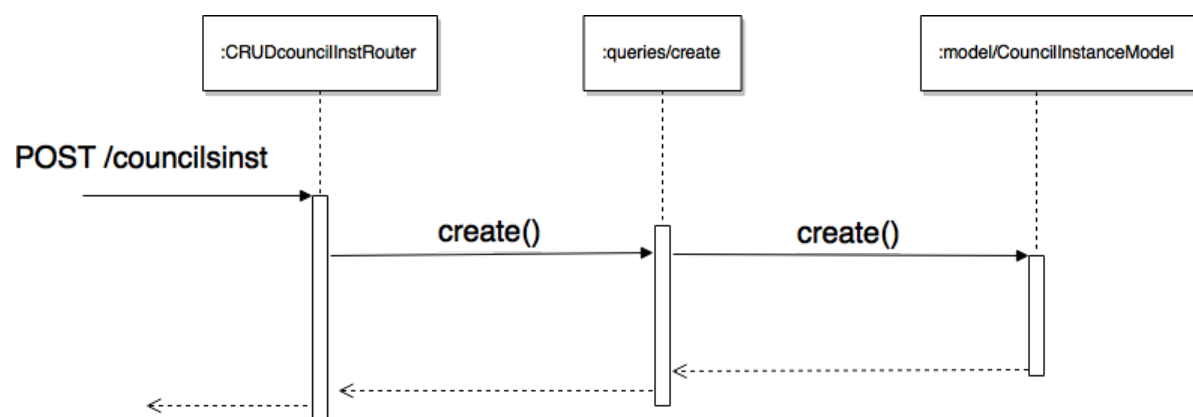


### 6.3.2.4 Delete one council

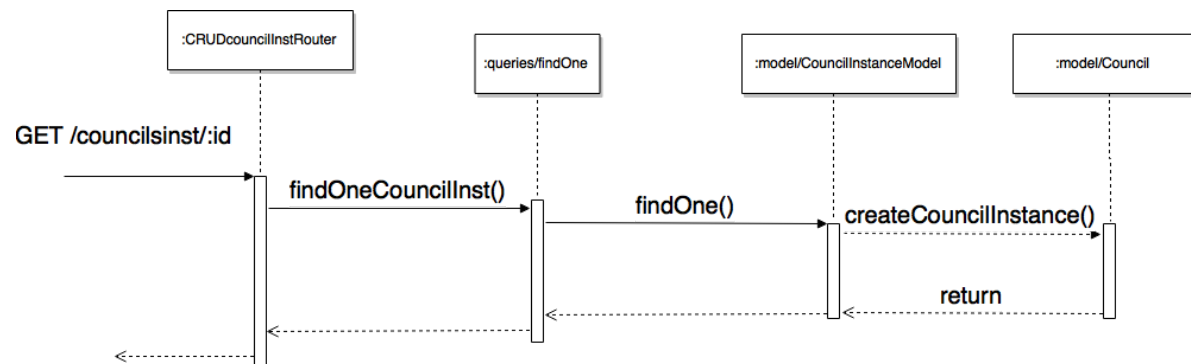


## 6.3.3 Council Instances

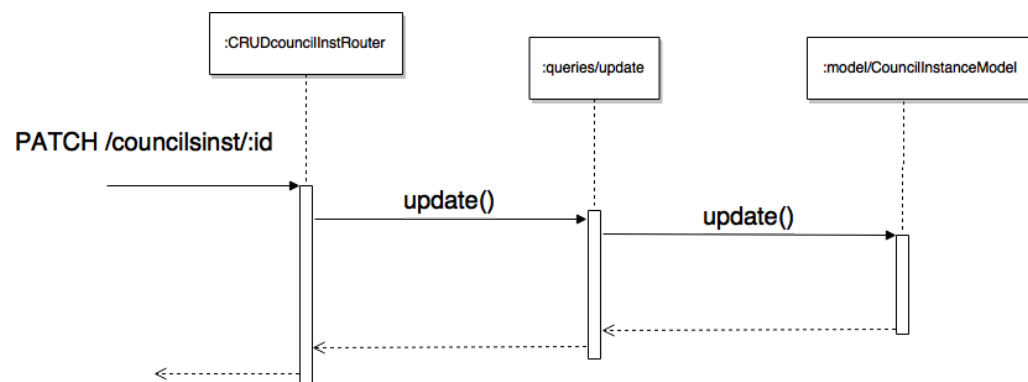
### 6.3.3.1 Create one council instance



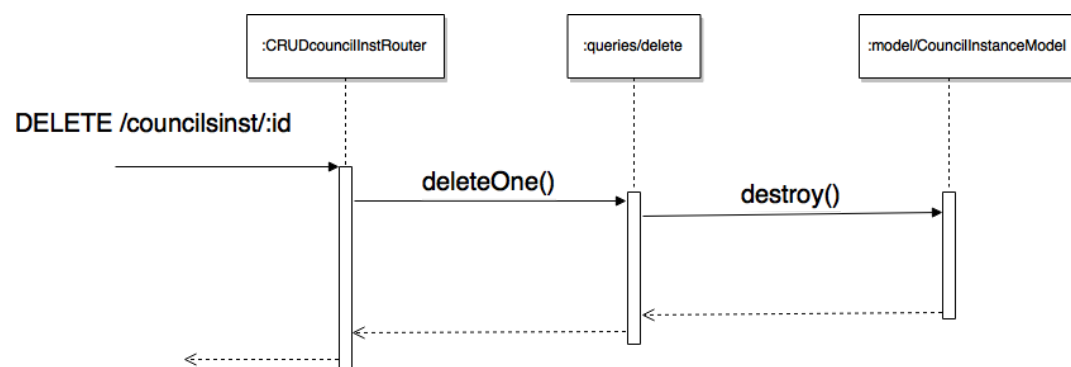
### 6.3.3.2 Get one council instance



### 6.3.3.3 Update one council instance



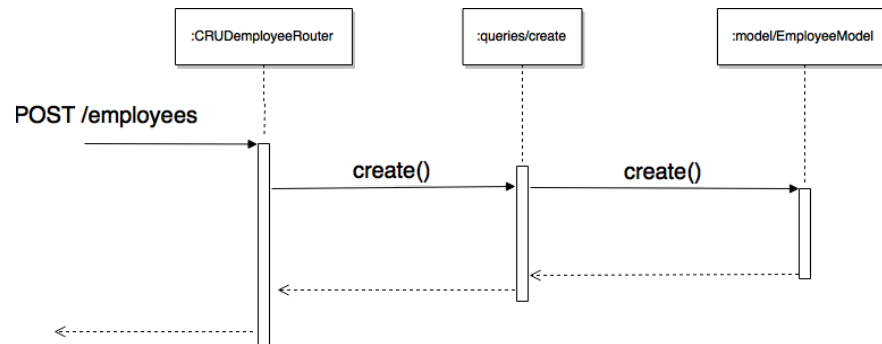
### 6.3.3.4 Delete one council instance



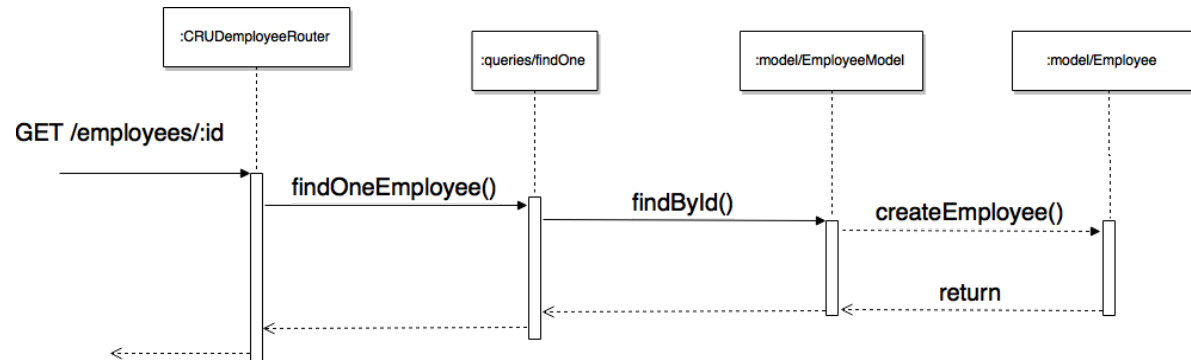


## 6.4 Employees

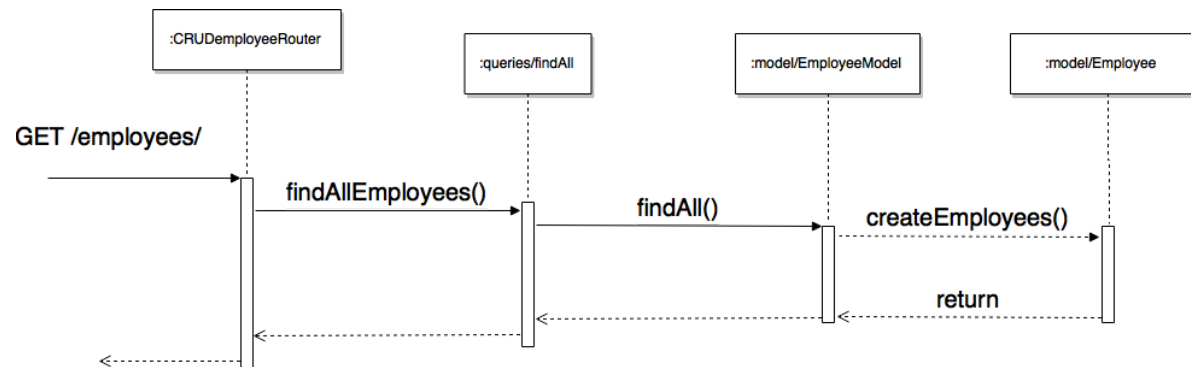
Create employee



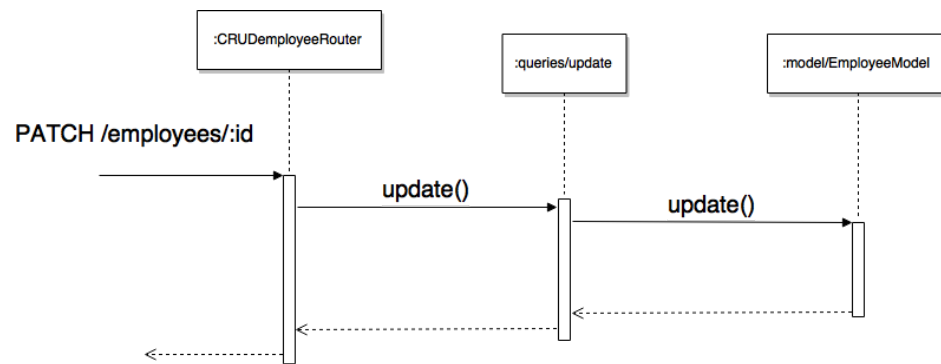
Get one employee



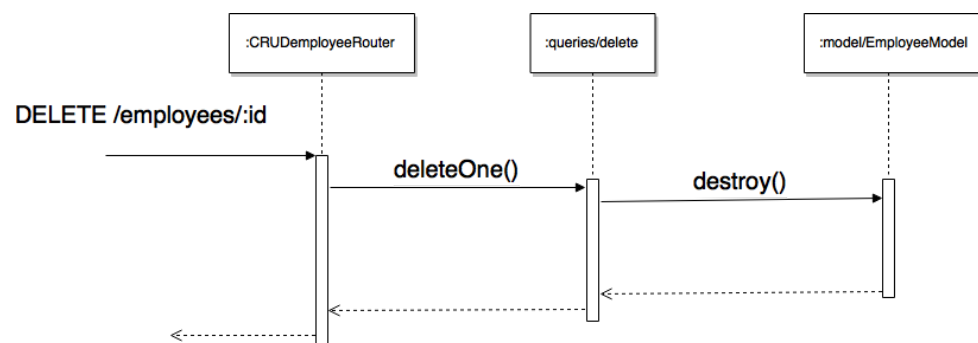
Get all employees



## Update employee

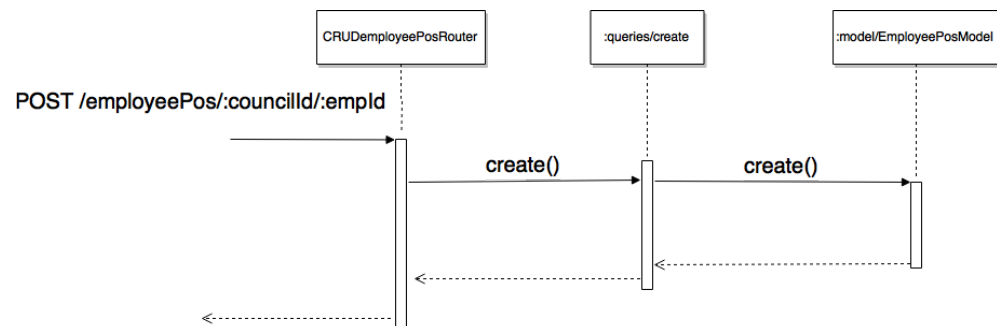


## Delete employee

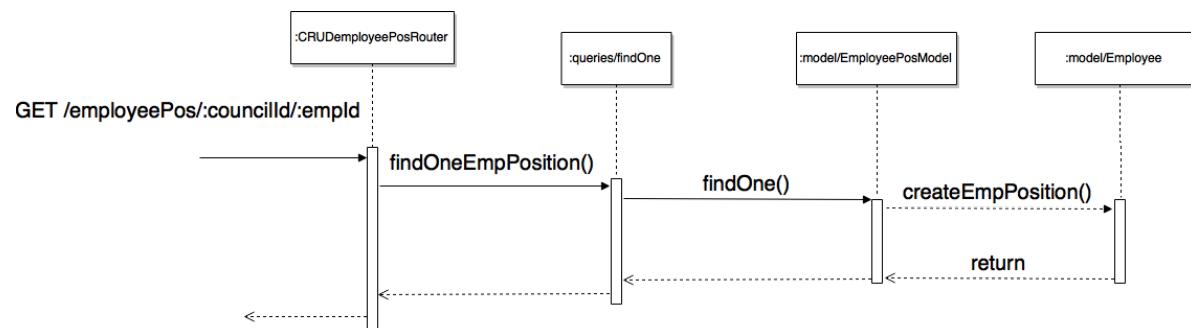


## 6.4.1 Employee position

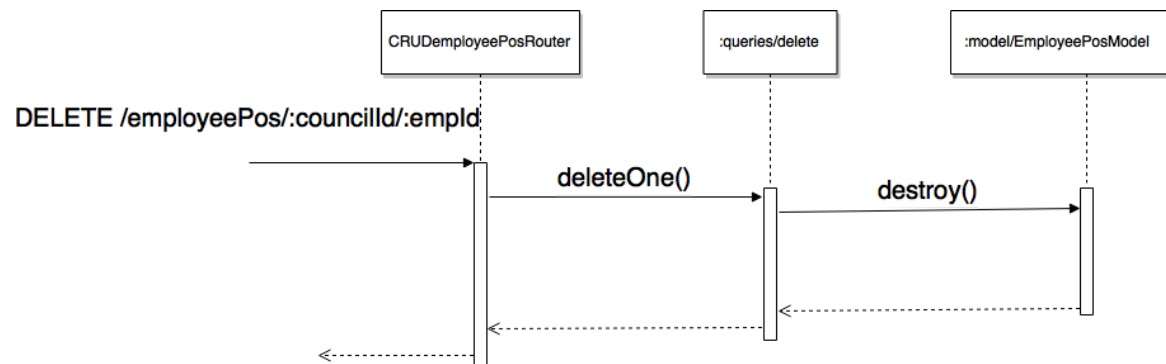
### Create employee position



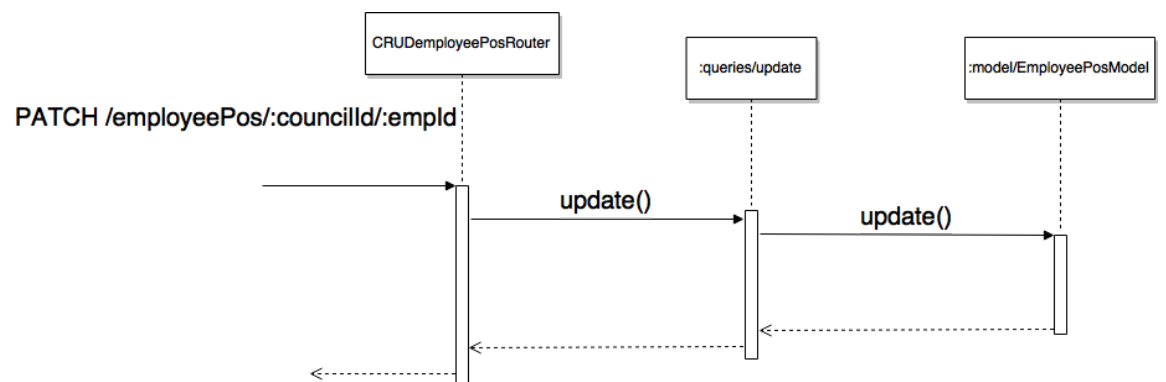
### Get on employee position



Update employee position

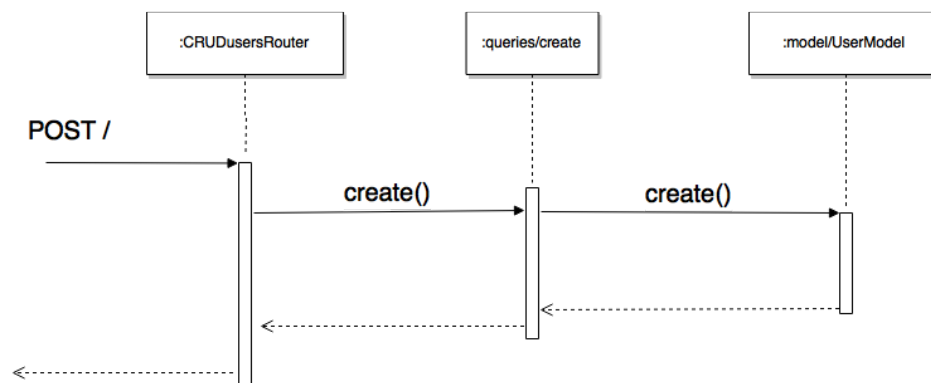


Delete employee position

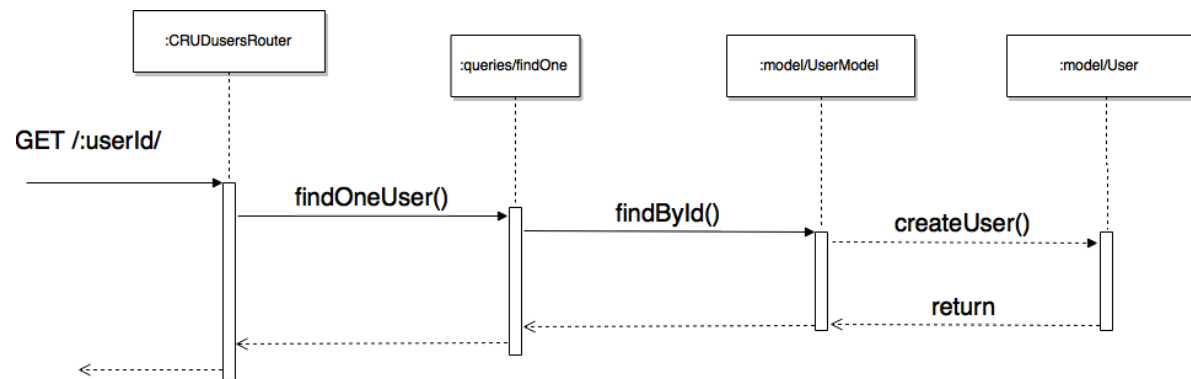


## 6.4.2 Users

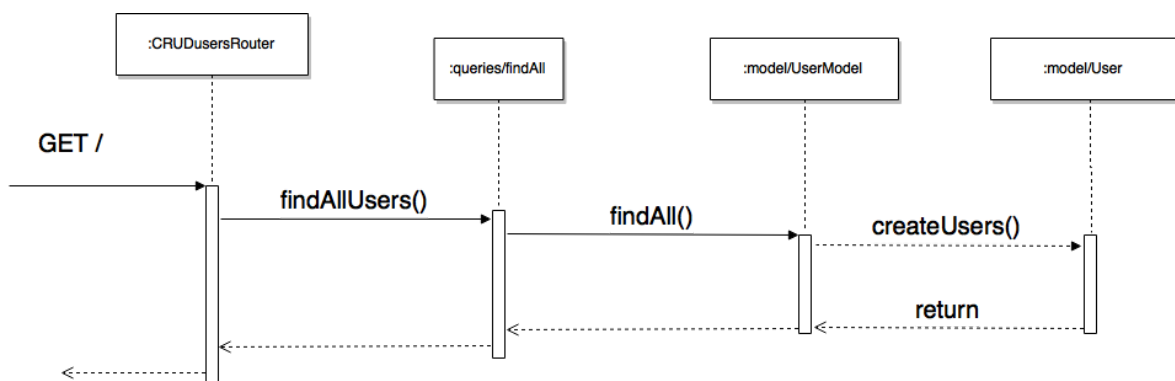
Create user



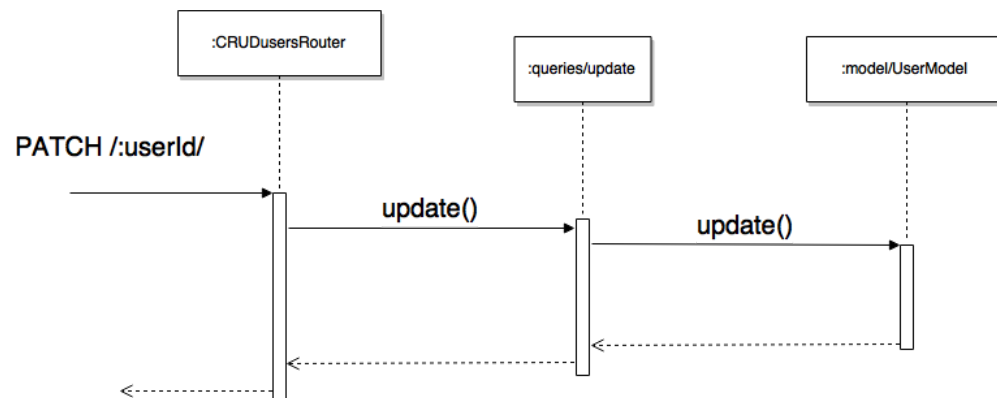
### Get one user



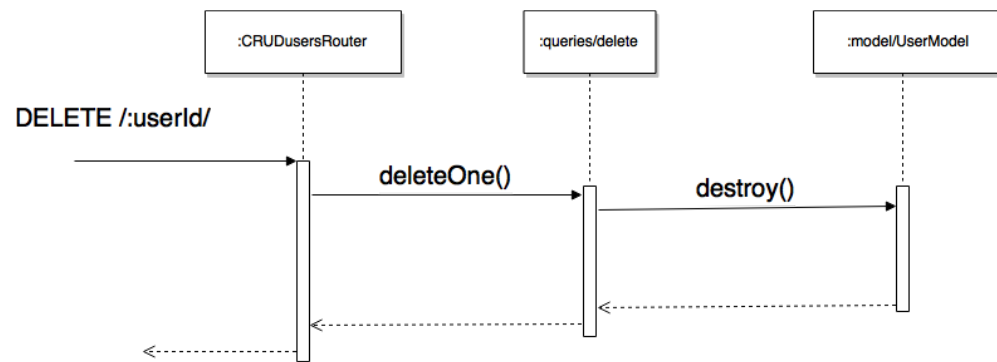
### Get all users



### Update one user

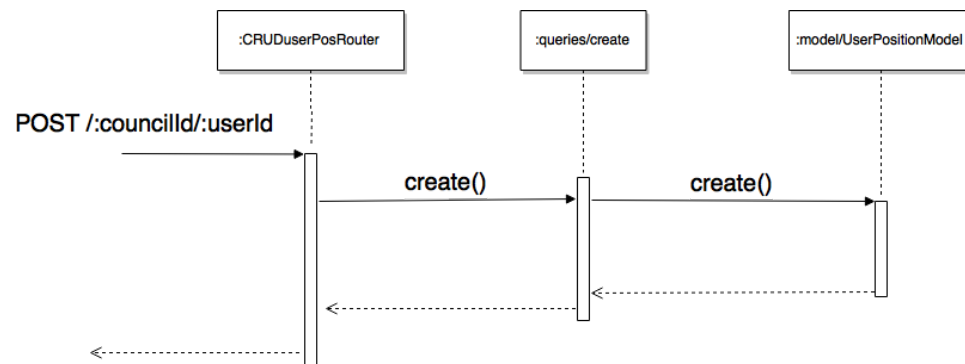


Delete user

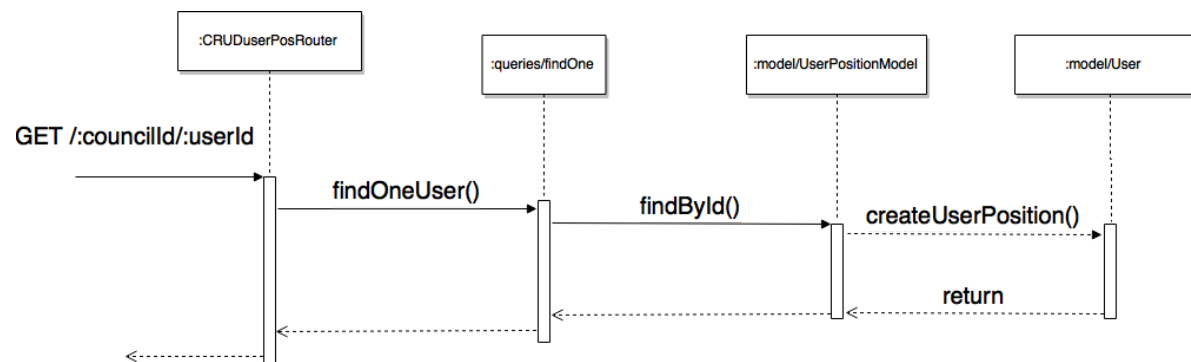


### 6.4.3 User positions

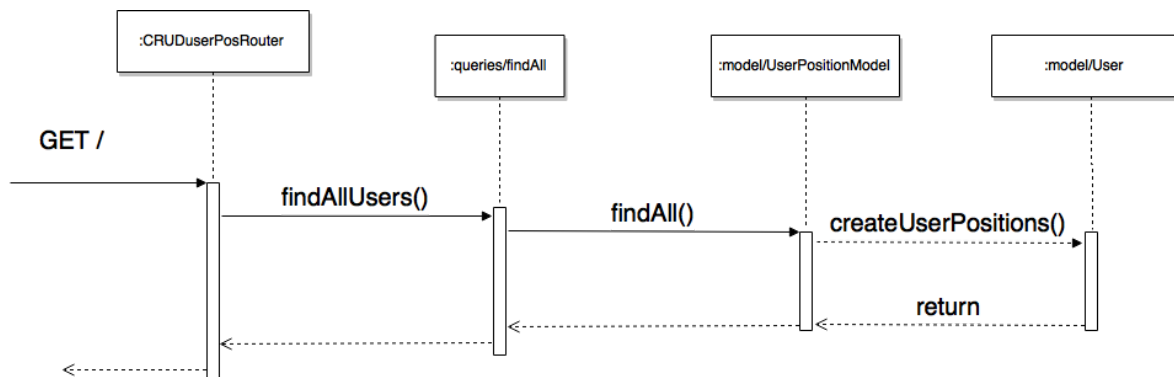
Create user position



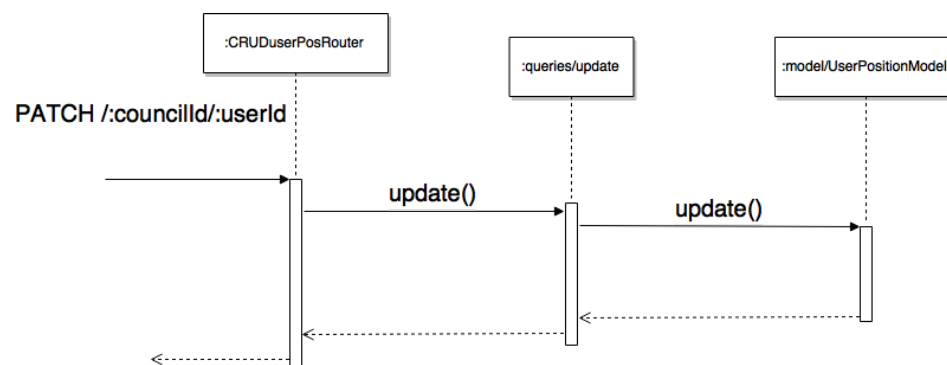
Get one user position



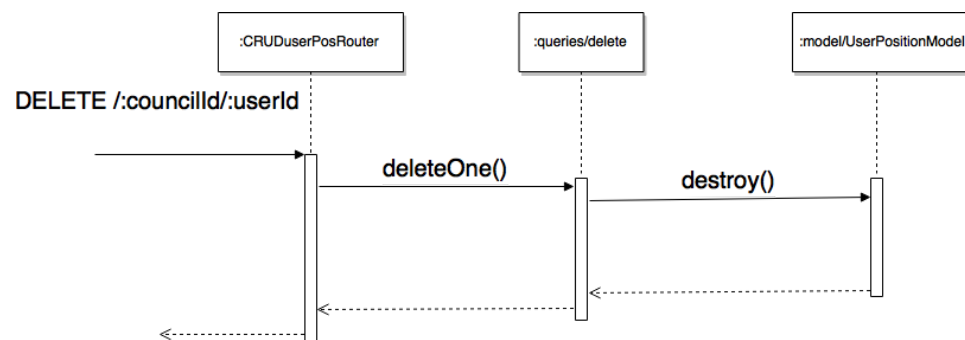
Get all user positions



Update one user



Delete one user



## 6.5 DB design

When choosing a RDMS, the following aspects were taken into consideration:

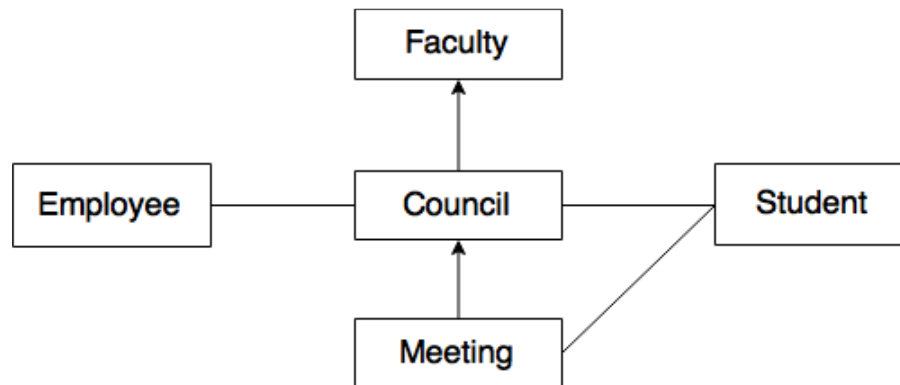
- Object-relational db
- Node and Express compatibility
- SQL compliance
- Data integrity
- User management
- Multi-user application support
- Integration
- Community support

After comparing the most popular DBs used with Node.js, the obvious solution was using PostgreSQL (it scored highest on the priority list) with Sequelize as ORM due to its advancement, support and popularity.

## 6.5.1 ER diagram

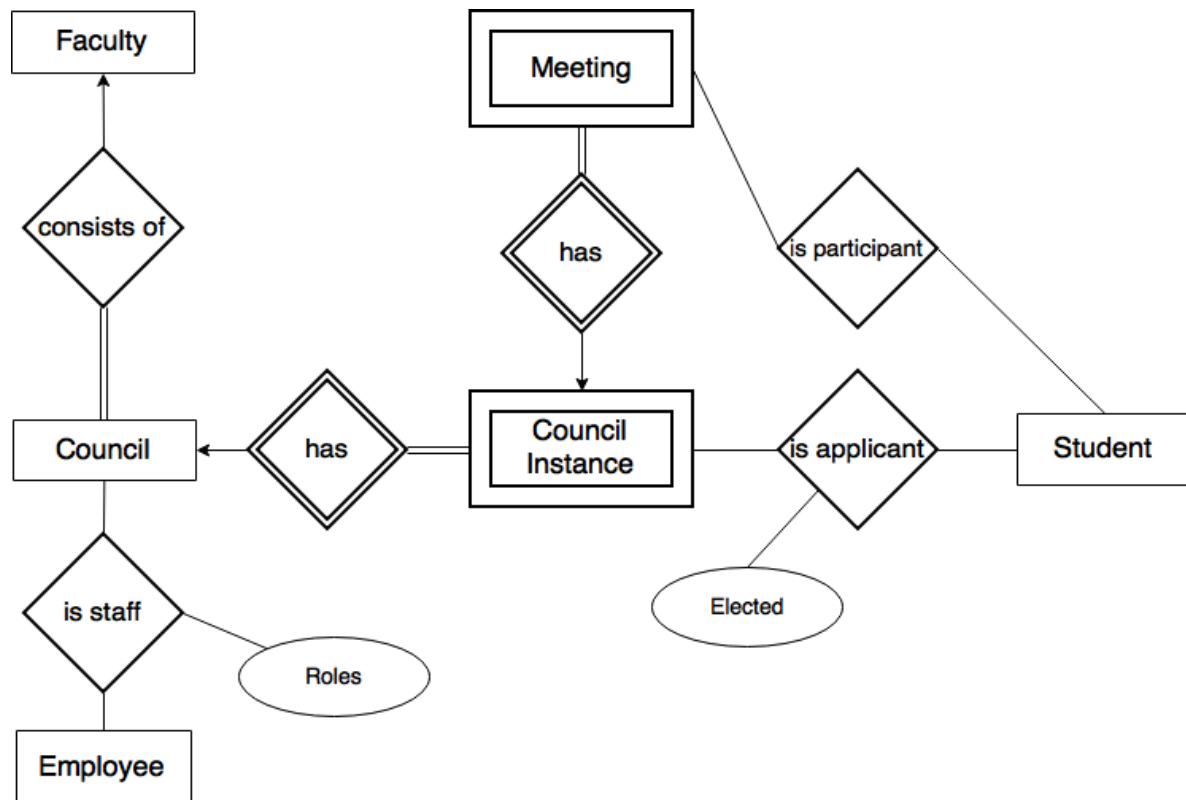
### 6.5.1.1 Entities and Relationships

There are five major concepts in the data structure of the application:



- Faculty - Council -> 1:N
  - One council can belong only to one faculty
  - One faculty can have many councils
- Employee - Council -> N:M
  - One employee can work in many councils
  - One council can have many employees
- Student - Council -> N:M
  - One student can be connected to many councils
  - One council can have many students
- Meeting - Council -> 1:N
  - One council can have schedule many meetings
  - One meeting can be scheduled only for one council
- Student - Meeting -> N:M
  - One student can participate in many meetings
  - One meeting can have many participants (students)

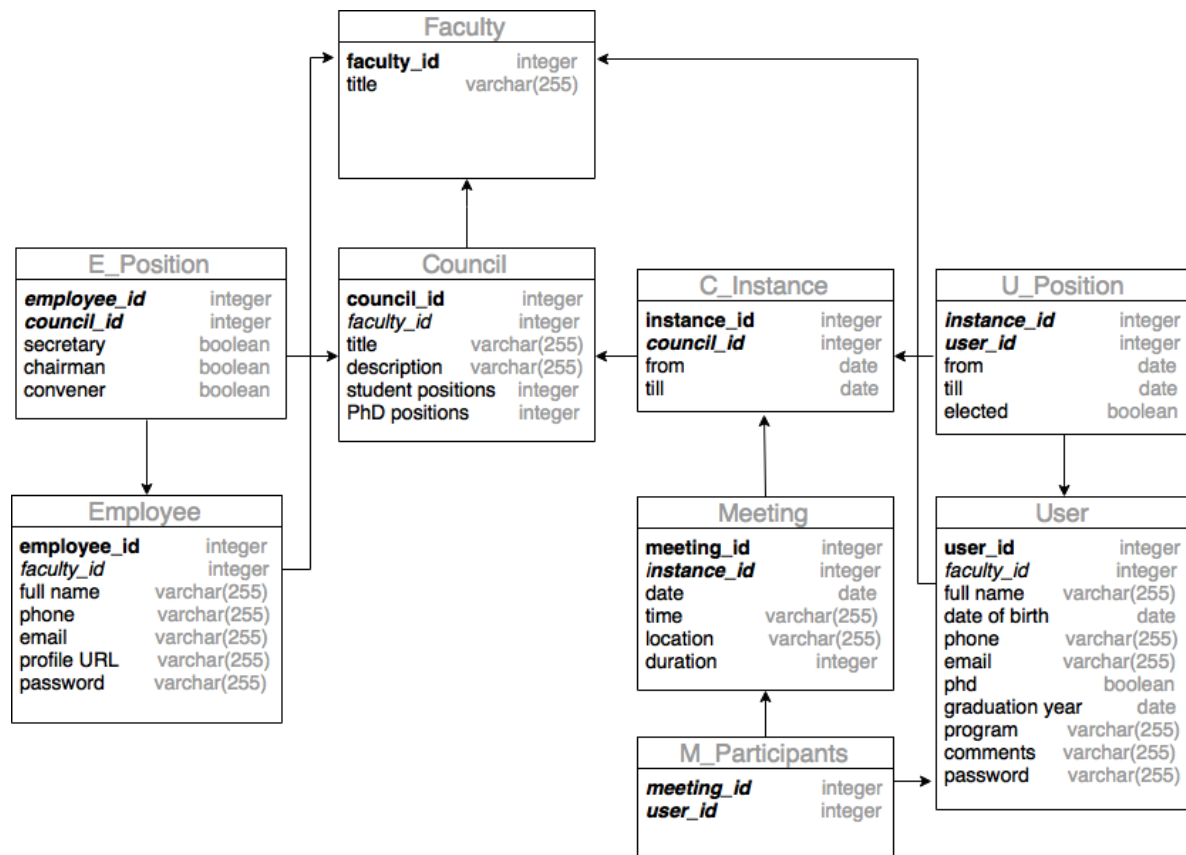
To remove many-to-many relationships, new entity is introduced - Council\_Instances, along with some key relationships. This will also allow us to track the history of students involvement in different councils and their meetings.



- Council cannot exist without a Faculty (total participation)
- Council Instance is a weak entity of Council
- Meeting is a weak entity of Council Instance
- Employee works for a Council and has defined roles (chairman, secretary or convener)
- Student applies to work for council and can be elected as Student Representative and can participate in Meetings



These entities and their relations can be represented as a following database:



**Primary keys** are in bold.

*Foreign keys* are in italic.

No cells are allowed to have a NULL value, except from *faculty\_id* in User table. This can be fixed with creating a separate database of all courses and programs offered by LNU, so even if a user doesn't know the name of the faculty s/he studies under, the system would be able to query that information.

## 6.5.2 Normalization and anomalies

The final database design does not violate the first three Normal Forms:

1. each cell contains a single value and there are no repeating groups;
2. every non-key column is fully dependent on the primary key (the set of primary keys)
3. every non-key columns are independent from each other.

Consequently, there are no insertion, deletion or update anomalies present in any table.

## 6.6 UI design

The user interface of the application should be intuitive and easy to comprehend. It will be divided into two major groups:

- Guest and student interface, where any anonymous users can view general information about faculties, councils and representatives, as well apply for a student representative position him/herself. This interface will be also used by registered students to login into their account for purposes of editing their profile and submitting new applications for SR position.
- Administrative interface, available only to authenticated administrators of Linnéstudenterna and employees of LNU. Here all major work with the registry is going to take place, e.g. creating and editing councils, student and employee profiles, etc.

The guest and student interface shall follow the graphical theme already established at the current website for Linnéstudenterna (linnestudenterna.se).

The primary background color is a light grey (#f5f5f5) with orange as a primary highlight (#f49a12). The primary font color is a dark grey (#444444) and primary hyperlink are the same as the orange highlights. The main navigation bar at the top has a white background (#ffffff).

The two main font families used are Brandon Grotesque N7 for headings and Source Serif Pro N4 for paragraph texts. For the prototype we will use the fonts that closest resembles these at Google Fonts.

### 6.6.1 Input validation

The validation of user input will take place both on the client and the server sides of the application to ensure security and usability. The client-side validation should be done with JavaScript, and the server-side - with Sequelize model validation feature.

- JavaScript is responsible for making sure that all the required fields in various forms are filled in and are not submitted empty.
- Sequelize double-checks that the input is not null, as well that it is of correct type and/or format before inserting the data into the database.

E.g., validation of email input field:

- client:

```
if (typeof inputData === 'string' && inputData === '') {
    throw new Error ('Fyll i alla fält med stjärna...')
}
```
- server:

```
email: {
  type: Sequelize.STRING,
  allowNull: false,
  validate: { isEmail: true }
}
```

### 6.6.2 Error and information messages

The application will display eloquent flash messages in case of user's successful action, if an error occurred or to show some general information. To make these messages more user-friendly, each type of the message will be colored appropriately. All messages will be short, but precise, stating the reason and/or solution for the situation.

Success messages will be shown on green background and will state the status of the action:

**OK!** The application was sent successfully.



Error messages will be shown on red background and will clearly state the reason of the error:

**Error!** Please fill in all fields marked with asterisk.



Information messages will be shown on blue background:

**Information!** You can add applicants later.



Warning messages will be shown on yellow background:

**Warning!** This is sensitive information.



If a user should decide to delete an item from the registry, the system will present a confirmation dialog to make sure that the action was intended and not a mistake.

**Are you sure want to delete the record?**

Delete

Cancel

## 6.6.3 UI for Use Cases

*Admin view:*

6.6.3.4 UC-02.1 Create new student profile + UC-02.3 Update student profile

**Student - edit**

Name:

Last name:

Birthdate:

Telephone:

Email:

Faculty:

Program / Course:

Graduation year:

About:

☒ PhD

### 6.6.3.5 UC-02.2 View student profile

**Student**

<b>Name:</b>	Bob
<b>Last name:</b>	Husky
<b>Birthdate:</b>	01.01.1990
<b>Phone:</b>	072123456
<b>Email:</b>	bh111zx@student.lnue.se
<b>Faculty:</b>	Economy
<b>Program / Course:</b>	Who knows
<b>Graduation year:</b>	2020
<b>About:</b>	this is info about me. And a little more info about me. And a little more. And a little more. And a little more. And a little more. And a little more. And a little more. And a little more.
<b>PhD:</b>	<input checked="" type="checkbox"/>

[View applications](#)[Edit](#)[Remove](#)



**Student - view applications**

Council	From ▼	Till ▼	Status
Council 1	2017.01.01	2017.06.01	Pending
Council 2	2017.01.01	2017.06.01	Elected
Council 3	2016.09.01	2016.12.31	Closed
Council 2	2016.09.01	2016.12.31	Elected

[New application](#)[Close](#)

Status:

- pending - council hasn't expired (till date is in the future), student is not elected
- elected - student is/was elected
- closed - council has expired (till date in the past), student was not elected



**Student - new application**

Choose faculty

Choose council

Search...

Council 1 (2017.01.01 - 2017.06.01)

**Council 2 (2017.09.01 - 2017.12.31)** ✓

Council N (from - till)

Period:

from

till

Apply

Close

Councils: show only not expired instances (with open positions)

Period: pre-filled with dates of chosen council instance

Period validation:

- from  $\geq$  council's from
- till  $\leq$  council's till
- from  $<$  till

#### 6.6.3.6 UC-02.5 Create new staff profile + UC-02.6 Update staff profile

**Employee - edit**

Name:

Last name:

Faculty:  
 ▼

Telephone:

Email:

LNU profile:

#### 6.6.3.7 UC-02.6 View staff profile

**Employee**

<b>Name:</b>	Bob
<b>Last name:</b>	Husky
<b>Faculty:</b>	Economy
<b>Phone:</b>	072123456
<b>Email:</b>	bob.husky@lnue.se
<b>LNU profile:</b>	www.lnu.se/staff/bobhusky

**Employee - new position**

Economy

Choose council

Search...

Council 1

**Council 2** ✓

Council N

Position:

☐ Chairman

☐ Convener

☒ Secretary

Save

Close

Position: user should select minimum 1 position.



#### 6.6.3.8 UC-03.1 Create new council profile

**New Council**

Choose faculty

Search...

Faculty 1

**Faculty 2** ✓

Faculty N

Choose council

Period:

from

till

OR CREATE NEW

Choose faculty

Name:

Description:

A very long description of this council

Student position:

3

PhD position:

2

Period:

from

till

Save

Cancel

The first (top) option will create a new Council Instance.

The second option will first create a new Council and immediately its new Instance.

After confirming the submission, the user will be presented a council view interface (UC-03.2) where blocks Staff, Student Representatives, Applicants and Meetings will be empty.

### 6.6.3.9 UC-03.2 View council

https://srd.linnestudenterna.se/admin/teknik/programrad

LinnestudenternaFacultiesCounsilMembersProfileLog Out

Council

Faculty: Economy Faculty

Name: Some council

Description: A very long description

Student positions: 0 out of 3

PhD positions: 0 out of 2

Period: 2017.01.10 - 2017.06.10

EditRemove

Staff

Name LastNameChairman, ConvenerEditRemove

Name LastNameSecretaryEditRemove

Add new

SR

Name LastNameStudentRemove

Name LastNamePhDRemove

Add new

Applicants

Name LastNameStudentRemove

Name LastNameStudentRemove

Add new

Meetings

Date	Time	Location	Status	
2017.01.01	12:30	Kalmar, Nyckel, Dxxx	Pending	Edit
2017.01.31	14:00	Kalmar, Villa, Sal 3	Completed	Edit

Add new

#### 6.6.3.10 UC-03.3 Update council

**Council edit**

Faculty:

Economy

Name:

Choose

Description:

A very long description of this council

Student positions:

3

PhD positions:

2

Period:

from

till

< October 2014 >

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

When clicking on From or Till input fields, a calendar should pop out with date preset for “today”.

#### 6.6.3.11 UC-03.5 Add staff to council

**Staff - add new**

Lastname1 Name

Search...

Lastname Name

**Lastname1 Name** ✓

Lastname2 Name

Positions:

☐ Chairman

☐ Convener

☒ **Secretary**

Save

Cancel

Multiple positions should be allowed for selection.

#### 6.6.3.12 UC-03.7 Add applicant to council

**Applicants - add new**

Lastname1 Name

Search...

Lastname Name

**Lastname1 Name** ✓

Lastname2 Name

Period:

from

till

Save

Cancel

Period fields should be pre-filled with dates of chosen council instance.

From: can NOT be before council instance "from"

Till: can NOT be after council instance "till"

### 6.6.3.15 UC-06.1 View election candidates

First user must select the period for viewing applicants.

**Get applicants list**

Period:

from

till

Submit

Cancel

Then user is presented with the list of applicants.

https://srd.linnestudenterna.se/admin/election

LinnestudenternaFacultiesCouncilsMembersElectionProfileLog Out

Council ▼	Applicant ▼	S/PhD ▼	Applications ▼	From	From	Print
Council 1	John Boo	S	3	01.04.2017	01.06.2017	<input checked="" type="checkbox"/>
Council 1	Michael Robinson	PhD	5	01.02.2017	01.06.2017	<input checked="" type="checkbox"/>
Council 1	Alexander Robson	S	1	10.02.2017	10.05.2017	<input checked="" type="checkbox"/>
Council 2	Jennifer Pinsker	S	2	01.03.2017	01.05.2017	<input checked="" type="checkbox"/>
Council 2	Bob Robson	S	2	01.02.2017	01.06.2017	<input type="checkbox"/>
Council 3	Michael Robinson	PhD	3	01.02.2017	01.06.2017	<input type="checkbox"/>

Print listPrint selected profiles

Applications column - how many applications student has submitted all together for this period.

Rows highlighted with red - student is participating in the meetings without being officially elected.

Date fields highlighted with yellow - the date is different from council's from/till date.


Print option opens a print-friendly page version with option to save as PDF.


Printing profiles (checked in the table) saves all profiles in one single PDF.

#### 6.6.3.16 UC-06.2 Request confirmation from candidates

**Confirm applications**

Period:

from 

till 

Confirm applications

Cancel

#### 6.6.3.17 UC-06.3 Elect/appoint representative

**SR - add new**

#	First Name	Last Name	Elected
1	John	Boo	<input checked="" type="checkbox"/>
2	Mary	Brown	<input checked="" type="checkbox"/>
3	James	Mooray	<input type="checkbox"/>


Save

Cancel


#### 6.6.3.18 UC-07.1 Add a meeting to council

**Meetings - add new**

Date:

from 

Time:

14:00 

Location:

Save

Cancel

### 6.6.3.19 UC-07.2 Track meeting participants + UC-07.3 Edit meeting

Meetings - edit

Date:  
from

Time:  
14:00

Location:

Duration:  
Short

Participants:  

#	First Name	Last Name	Participated
1	John	Boo	<input checked="" type="checkbox"/>
2	Mary	Brown	<input checked="" type="checkbox"/>
3	James	Mooray	<input type="checkbox"/>

Add unofficial participant

Save Cancel

Meetings - add unofficial participant

Lastname1 Name

Search...

Lastname Name

Lastname1 Name ☒

Lastname2 Name

If student profile doesn't exist - **create new** and start over

Save Cancel

## 7 Appendix - links and other documents

### 7.1 API Documentation

The full API documentation is available at:

<https://student-representative-database.github.io>

### 7.2 Application repository

The prototype applications source code is available at:

<https://github.com/student-representative-database/studDB>

### 7.3 All documents from the project

This and the other documents developed during this project is available at:

<https://github.com/student-representative-database/documentation>