TECHNICAL SPECIFICATION

ASSIGNMENT 2

CSE5006

REPOSITORY:
HTTPS://GITHUB.COM/CSE5006/ASSIGNMENT-2

TABLE OF CONTENTS

OVERVIEW

PLAGIARISM

This is assignment must be completed individually, not in groups. You are welcome to discuss problems with fellow students, but the work that you submit must be your own. Don't try to plagiarise, it's extremely obvious and will only serve to get you into trouble. That's not any fun for you or for us.

DUE DATE

This assignment is due on LMS. Penalties are applied to late assignments (accepted up to 5 days after the due date only). See the departmental Student Handbook for details.

SUBMISSION DETAILS

- Create a short video explaining on how you tackle each task and demonstrate how to run your program. You can use Zoom to record your demo.
- Zip up everything in your project directory into one file
- Submit your zipped file via LMS

Your final submission must run in a clean virtual machine after invoking *docker compose up*, as this is the environment that we will be using to mark your assignment.

DEMO IMPLEMENTATION

I will show the demo again in the next lecture to help you understand about what an implementation of Contactor might look like.

OBJECTIVES

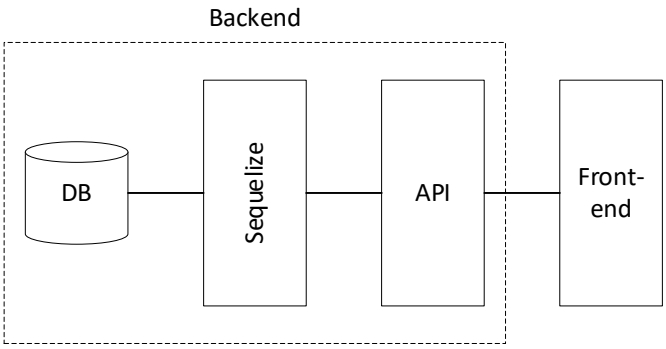
- To combine what you've learnt in labs into a complete web application
- To provide you with a reference web application for future projects

GETTING STARTED

There is a template for the assignment project available for you on GitHub at https://github.com/CSE5006/assignment-2. Clone and fork the repository and use it as a starting point for your assignment. Be sure commit and push regularly to avoid losing any of your work as you develop. **Make sure you fork the repository as a private so other people cannot see your work.**

APPLICATION OVERVIEW

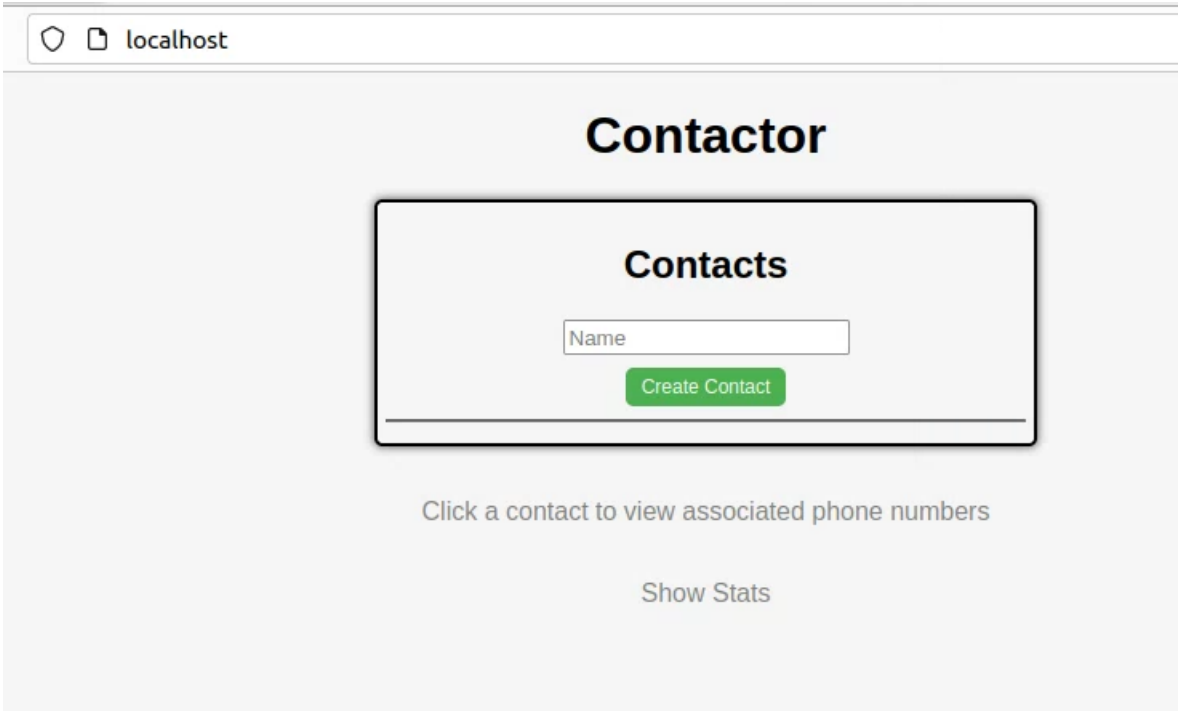
In this assignment, you will create a simple web application that consists of frontend and backend (API and DB) to keep and maintain all of your contacts. The architecture of the application is shown below.



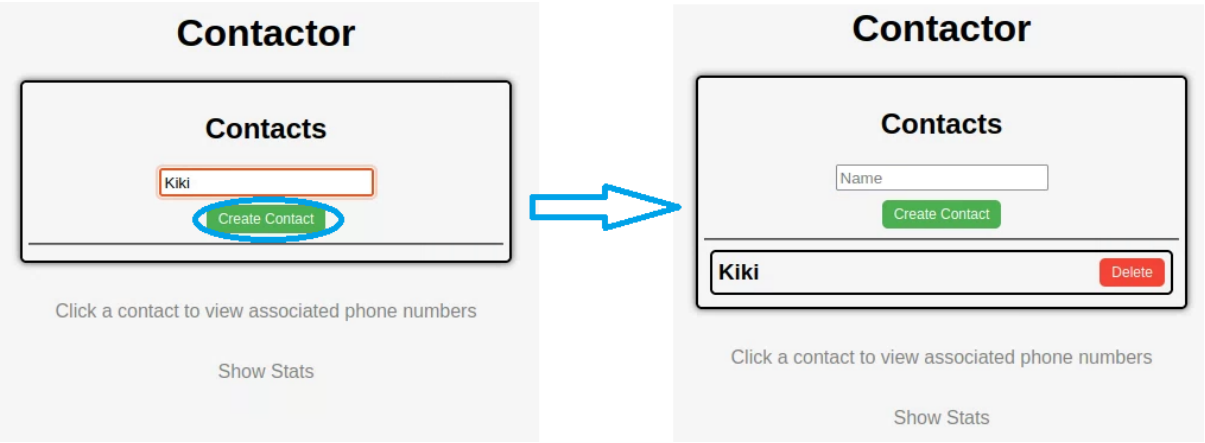
The application has the following functionalities:

- Add new a contact name
- Add new phone numbers
- List all contact names and phone numbers
- Remove a phone number from a contact
- Remove a contact name and all of the phone numbers
- Display the statistics of the data

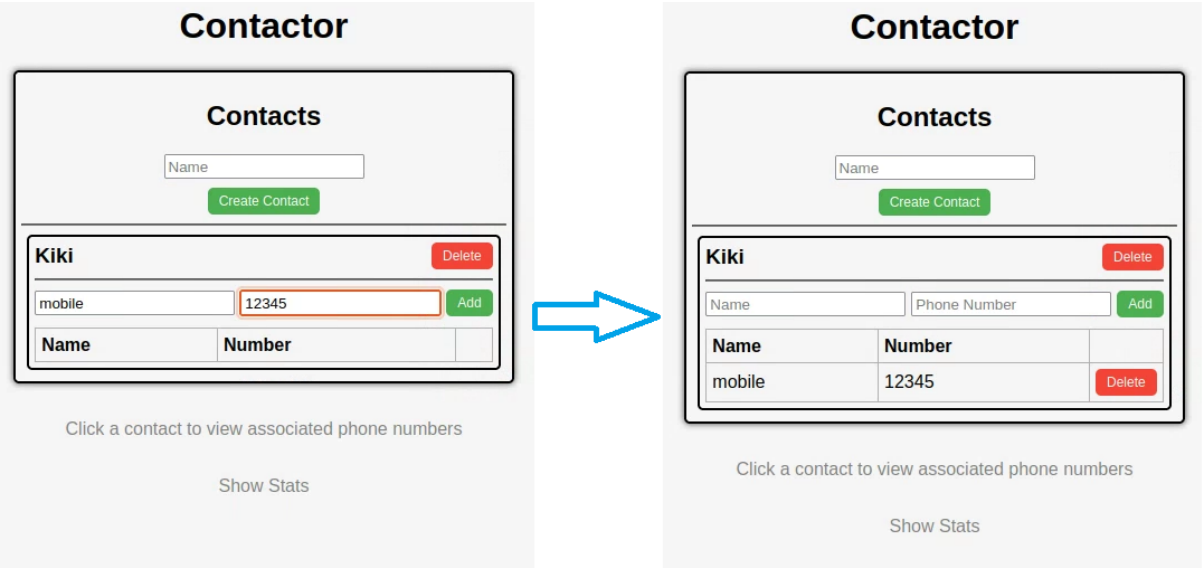
The application is created in a docker container with microservices configuration. The example of fully working application is shown below.



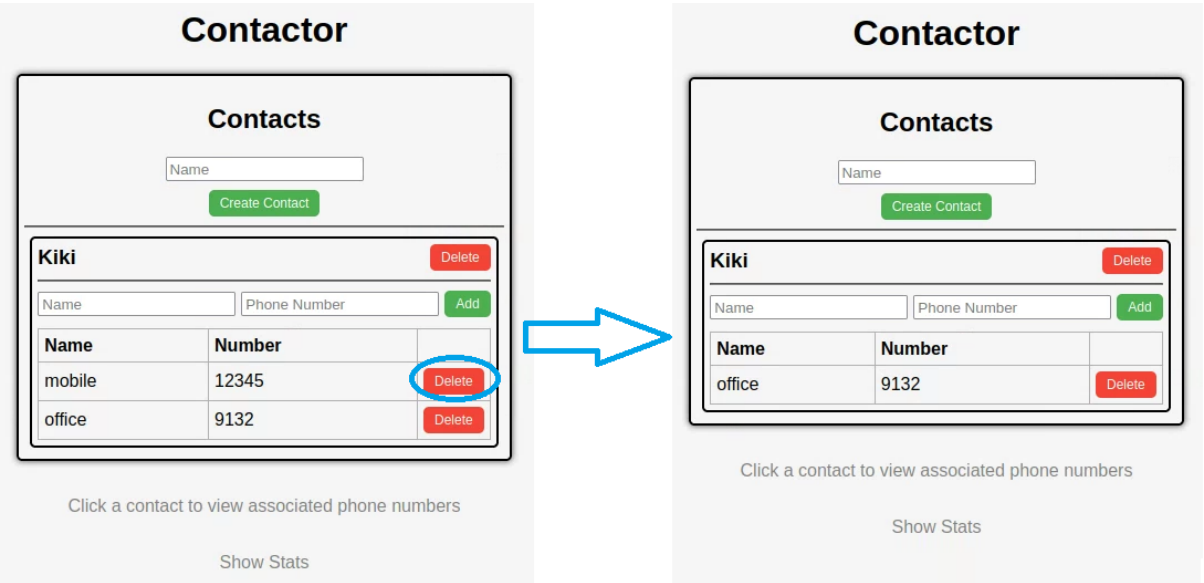
To add a new contact, you could put your name in the name component and click “Create Contact”



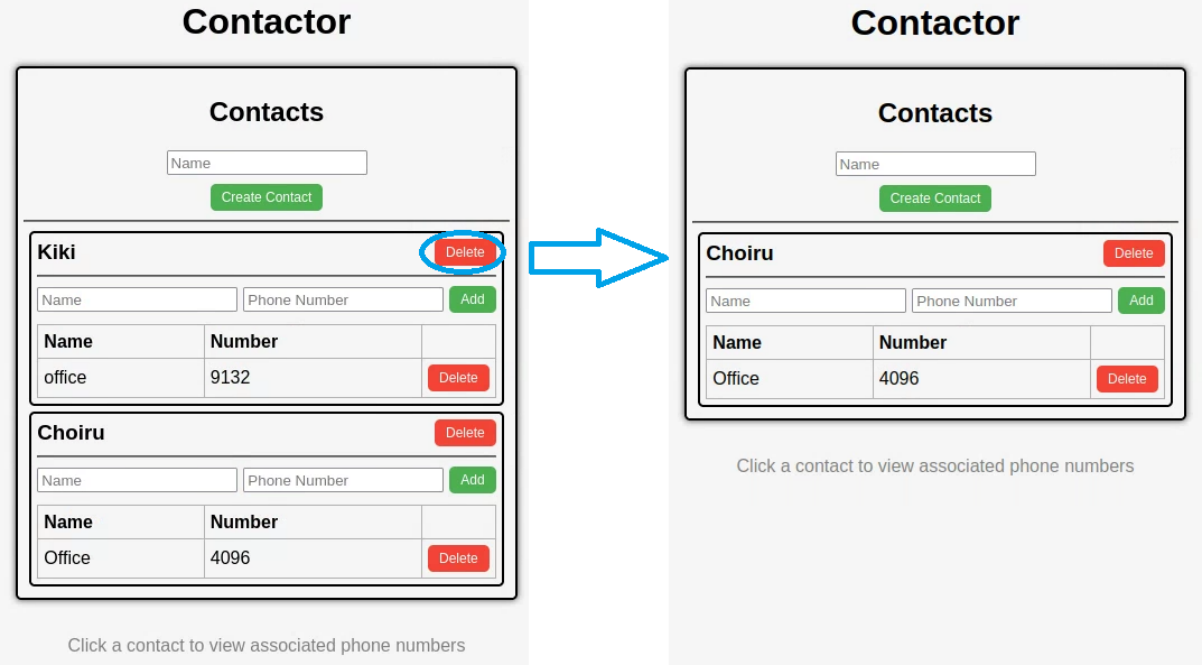
To add a new contact phone number, you could click the name and provide the contact type and the phone number.



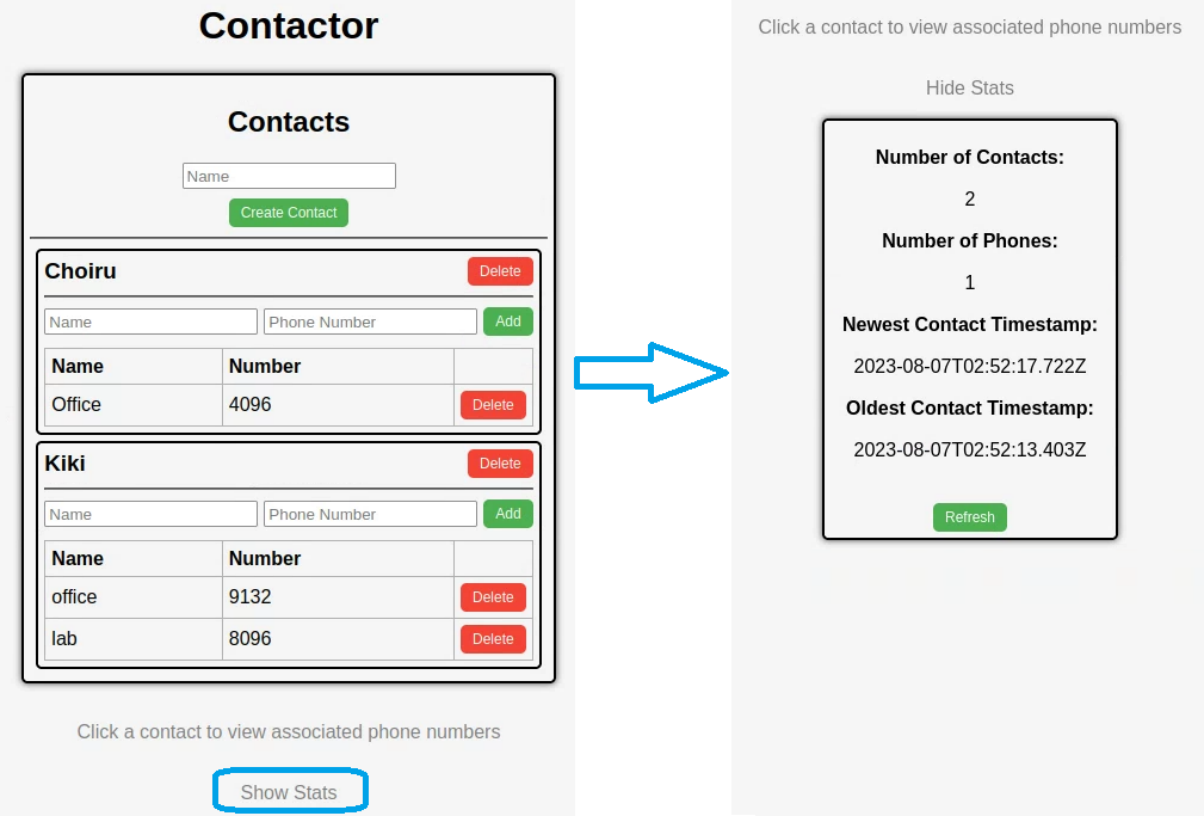
To delete a phone number, click the “Delete” button on a phone number row.



To delete a contact name, click the “Delete” button on a name block.



To show the statistic, click the “Show Stats” link



There is a template for the assignment project available for you on GitHub at https://github.com/CSE5006/assignment-2. Clone and fork the repository and use it as a starting point for your assignment. Be sure commit and push regularly to avoid losing any of your work as you develop. **Make sure you fork the repository as a private so other people cannot see your work.**

TASKS AND RELATED TOPICS

Here you can check the tasks and their highly related topics or activities in the subject. Please use this as a guidance only.

Task	Lecture Activities	Lab Activities
Task 1	Week 7	Week 9
Task 2	Week 5, 6	Week 7,8,9
Task 3	Week 4	Week 4, 5,9
Task 4	Week 4,5,6	Week 4,5,6,7,8

HINTS

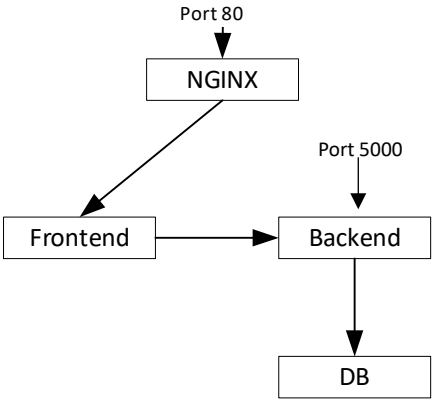
- Start working with the frontend section by creating a static frontend using basic HTML tags. Once you have finished with it, gradually convert the HTML components into React components.
- Create all frontend components without considering the data. Once you have created all frontend components, you can connect the data when the API is complete.
- Make sure only ports that have been specified in the specification are exposed.
- You can only use the method that we have learned in this subject to develop the project.

TASK 1 – MICROSERVICES (10 MARKS)

Provide the connections among all of the microservices

SPECIFICATIONS

The project has several microservices that are connected to each other. These microservices are NGINX, Frontend, Backend and Database. The microservices architecture of the application is shown below:



- Update the docker-compose.yml to provide the connections as shown in the microservice architecture

TASK 2 – BACKEND API (40 MARKS)

Create the backend database and REST API for the Contactor application. The backend contains the API and Sequelize scripts to handle the request.

SPECIFICATIONS

The database shall have two tables, one for contacts called **Contacts** and one for phone numbers called **Phones**. You can test these endpoints using HTTPie

- Records from the **Contactor** table shall contain the fields **id** (number, Primary Key), **name** (string)
- Records from the **Phones** table shall contain the fields **id** (number, Primary Key), **name** (string), **number** (string), and **contactId** (number, Foreign Key)
- Foreign keys shall follow the camel-case naming convention (eg a foreign key to the Contacts table would be called contactId). Specify the foreign key name explicitly in associations, as in Lab 07 (there we did foreignKey: 'postId').
- Each phone number shall belong to one contact, but a single contact can have many phone numbers.
- Two Sequelize models shall be defined – **Contact** and **Phone**. These models shall have the correct associations.
- The backend shall expose the following REST API:

HTTP Action	**Description**
GET /contacts	Returns a list of all contacts (does not include phone numbers). **This endpoint has been created for you already.**
GET /contacts/:contactId/phones	Returns a list of all phone numbers for a particular contact.
POST /contacts	Creates a new contact using the posted data. Returns the new contact.
GET /contacts/:contactId	Returns a single contact by ID.
DELETE /contacts/:contactId	Deletes a single contact by ID. All of the contact's phone numbers shall be deleted also. Returns an empty object, {}.
PUT /contacts/:contactId	Updates the attributes of a particular contact. Returns the updated contact. This needs to work even though it will not be used by the frontend.
POST /contacts/:contactId/phones	Creates a new phone number using the posted data. Returns the new note.
GET /contact/:contactId/phones/:phoneId	Returns a single phone number by ID.
DELETE /contact/:contactId/phones/:phoneId	Deletes a single phone number by ID. Returns an empty object, {}.
PUT /contact/:contactId/phones/:phoneId	Updates the attributes of a particular phone number. Returns the updated phone number. This needs to work even though it will not be used by the frontend.

TASK 3 – FRONTEND INTERFACE (25 MARKS)

TASK 3.1

Create a read-only web interface for the Contactor application which displays contacts and phone numbers.

SPECIFICATIONS

- There shall be a view which lists the names of all the contacts.
- Clicking on a contact shall display a list of all of the phone numbers within that contact.

TASK 3.2

Allow users to create and delete phone numbers and contacts via the web interface.

SPECIFICATIONS

- Users shall be able to create and delete phone numbers and contacts.
- Modifications to contacts and phone numbers shall be persisted in the backend database via the REST API. Therefore changes should survive a page refresh.
- You do not need to implement the ability to edit contacts or phone numbers, only to create and delete them

TASK 3.3

Make the interface pretty and nice to use. The Internet is a great place to find inspiration for styles and small snippets of CSS which you can incorporate into your design. It is acceptable for you to use CSS libraries such as Tailwind if you like. It is not acceptable to use component libraries such as Chakra-ui.

TASK 4 – SUMMARY STATISTICS (15 MARKS)

TASK 4.1

Create a new REST API endpoint at *GET /stats*. This endpoint will serve to show statistics about the application. It should meet the following requirements:

- The new /stats API endpoint shall be made available at http://localhost/api/stats
- The stats controller code shall be placed in a new controller file
- The /stats endpoint shall respond to GET requests with a JSON object containing the following information:
    - The number of contacts in the database
    - The number of phone numbers in the database
    - The time of the most recently created contact
    - The oldest contact creation time

TASK 4.2

Modify the frontend to display summary statistics on the Contactor home page. The statistics themselves shall be retrieved using the /stats API endpoint implemented in Task 3.1. It should meet the following requirements:

- A new “Statistics” React component shall be created which is connected to the application store and displays summary statistics
- The Statistics component shall be displayed on the Contactor home page

TASK 4.3

Add a refresh button which updates the displayed statistics when clicked.

For example, if the statistics show a count of 3 contacts, and then you add another one, clicking the refresh button should update the displayed statistics to indicate that there are now 4 contacts. This means that:

- A refresh button shall be visible in the statistics component
- Clicking the refresh button shall update the stats by accessing the /stats API endpoint

TASK 5 –PRESENTATION VIDEO (10 MARKS)

TASK 5.1

Create a 5-minute (maximum) presentation video that explains how you solved/addressed the tasks and demonstrates how the complete project runs.

SPECIFICATIONS

- Your video should not be longer than 5 minutes.
- Use Zoom to record your video in mp4 format. Make sure you show your face when recording the video.
- Make sure you explain how to address the challenges in each tasks, including your program flow.
- Demonstrate how to run the project and how all tasks have been covered.
- Compressed the video together with your project folder.