SEMI-FINALS DOCUMENTATION

Lorenzo Anthony Pasco
John Edward Lugod
Reece Tyler Mendoza

CHOSEN APPLICATION THEME

The chosen theme for this application is **Emergency Room (ER) Management**. The application simulates a system that manages patient arrivals, prioritizing them based on the urgency of their condition, which is represented using two data structures: a Stack and a Queue

- **Stack**: Used to represent patients that have been processed or attended to recently in the ER.
- **Queue**: Used to represent the patients waiting for treatment based on their reservation or order of arrival.

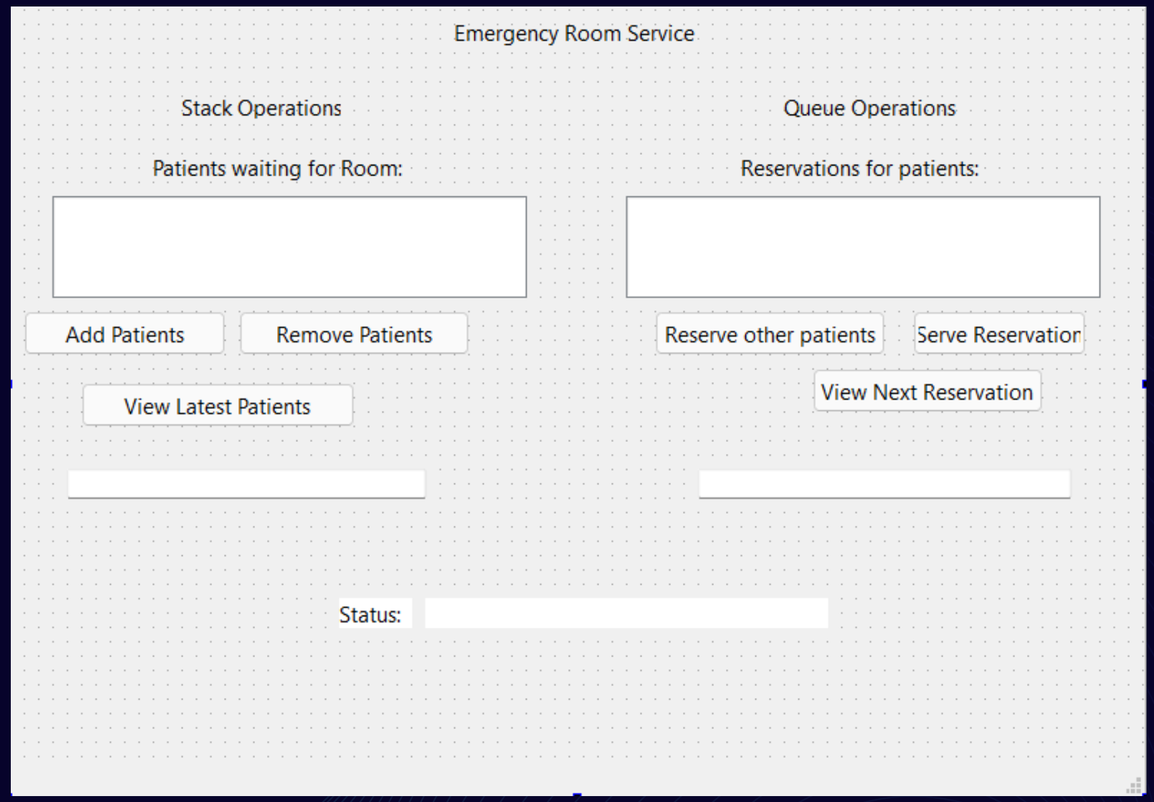
RATIONALE

The rationale behind choosing this theme is based on the critical need for efficient management in emergency rooms, which often handle a high volume of patients requiring urgent attention. The use of a stack and queue allows the application to simulate real-world patient flow:

- **Queue**: Ensures that patients who arrive first are treated first (First-In-First-Out, FIFO), which is important for prioritizing those who are waiting for care.
- **Stack**: Allows for quick retrieval of the most recently attended patients (Last-In-First-Out, LIFO), which reflects the need to quickly manage recently processed patients in a high-paced environment.

IMPLEMENTED APPLICATION AND ITS FEATURES

The application was built using Qt for the GUI and C++ for the backend logic. The primary goal of this application is to manage two patient lists: one for patients waiting for service (queue) and one for recently processed patients (stack).



IMPLEMENTED APPLICATION AND ITS FEATURES

Its Key Features: **Stack**

- **Add Patient to Stack**: The user can add a patient's name to the recently processed patients' stack using the **Add Patients** button. The patient's name is inputted through a text box and displayed in the stack list widget.

- **Remove Patient from Stack**: The **Remove Patients** button allows the user to remove the most recent patient from the stack (simulating processing or completion of care).

- **View Latest Processed Patient**: The **View Latest Patients** button shows the top patient of the stack (i.e., the most recently processed patient).

IMPLEMENTED APPLICATION AND ITS FEATURES

Its Key Features: **Queue**

- **Reserve a Patient**: The **Reserve Other Patients** button adds a patient to the queue for upcoming service, simulating a reservation system where patients are served based on the order of arrival.

- **Serve a Reservation**: The **Serve Reservation** button allows the user to dequeue a patient from the queue, representing the next patient to be treated.

- **View Next Reservation**: The **View Next Reservation** button shows the first patient in the queue who is next in line for service.

- **Real-time Feedback**: A status label is dynamically updated to show whether an operation was successful (e.g., a patient was added or removed from the stack/queue) or if an error occurred (e.g., the queue is empty).

IMPLEMENTED APPLICATION AND ITS FEATURES

Key Components:

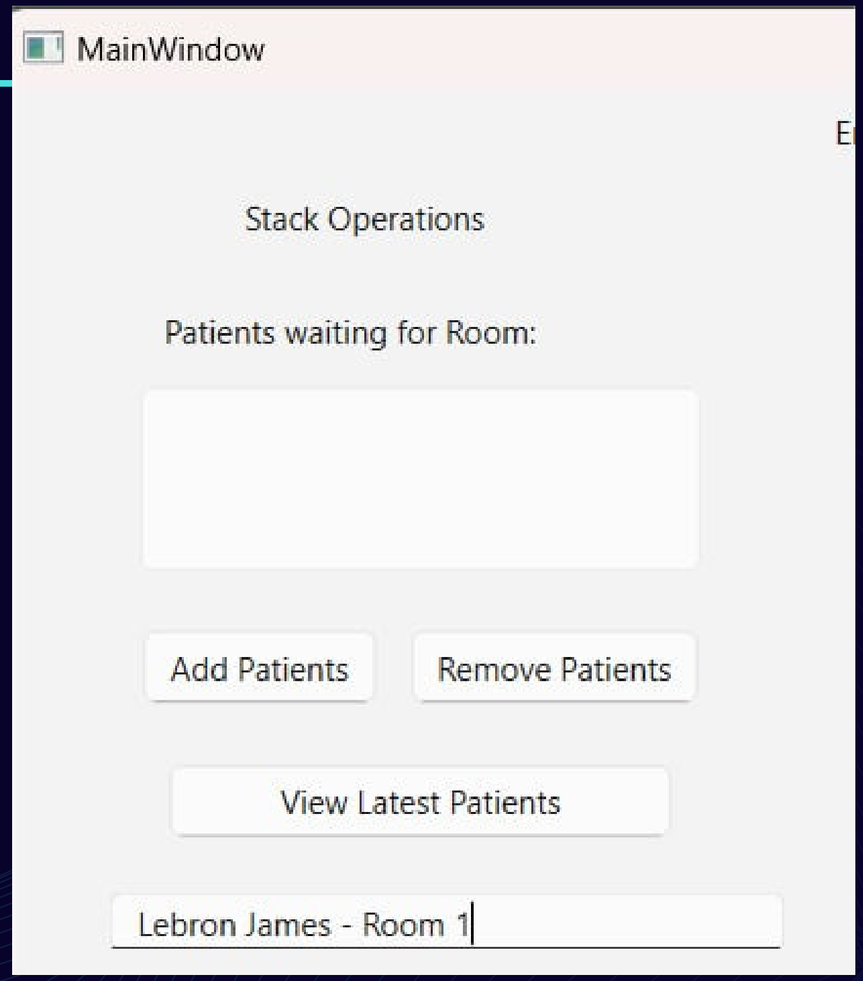
- **QStack**: Stores the names of patients who have been processed and are stored in the stack for quick retrieval.
- **QQueue**: Stores the names of patients who are waiting for service, processed in a first-come-first-serve manner.
- **QPushButton**: Used to trigger actions like adding patients, removing patients, and viewing the top or front patient.
- **QLineEdit**: Provides input fields for entering patient names.
- **QListWidget**: Displays the current patients in the stack or queue.
- **QLabel**: Provides real-time feedback on the status of operations.

TEST CASES FOR EMERGENCY APPLICATION

Test Case 1: Add Patient to Stack

Objective: Verify that a patient can be added to the stack.

- **Input**: Enter "Lebron James" in the "Patient Name" input field and click the "Add Patients" button.
- **Expected Result**: "Lebron James" should appear in the stack list, and the status label should display "name of patient added to recently returned stack."

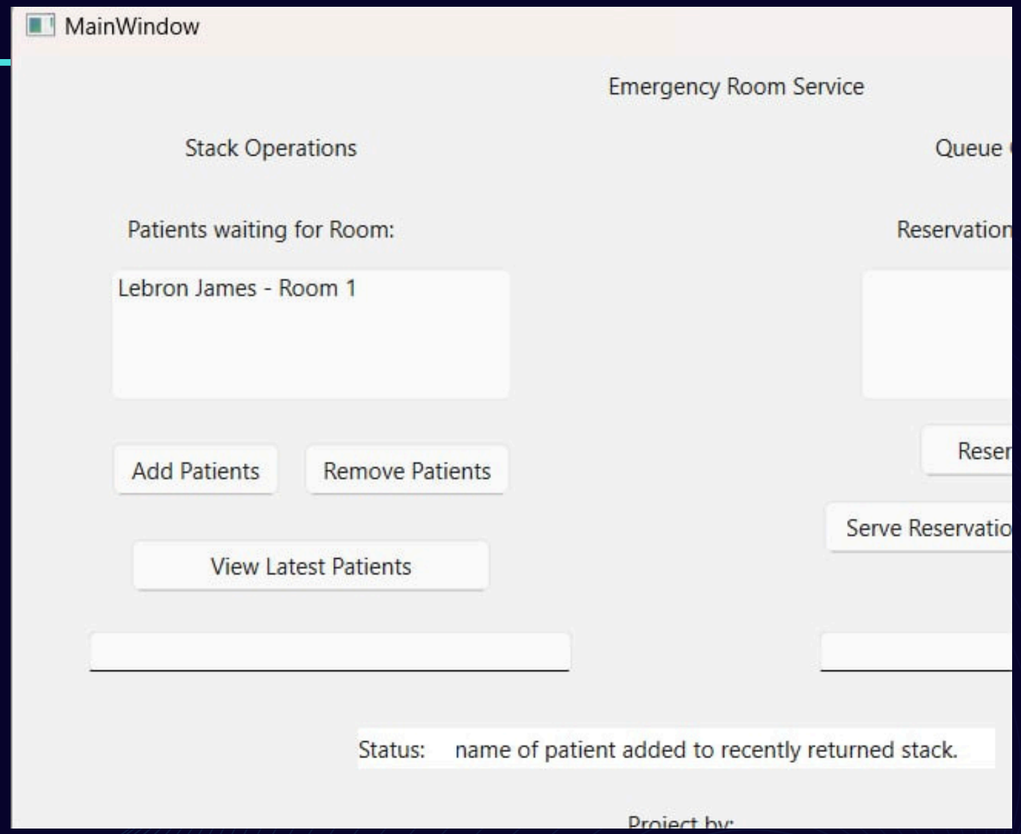


TEST CASES FOR EMERGENCY APPLICATION

Test Case 1: Add Patient to Stack

Objective: Verify that a patient can be added to the stack.

- **Input**: Enter "Lebron James" in the "Patient Name" input field and click the "Add Patients" button.
- **Expected Result**: "Lebron James" should appear in the stack list, and the status label should display "name of patient added to recently returned stack."

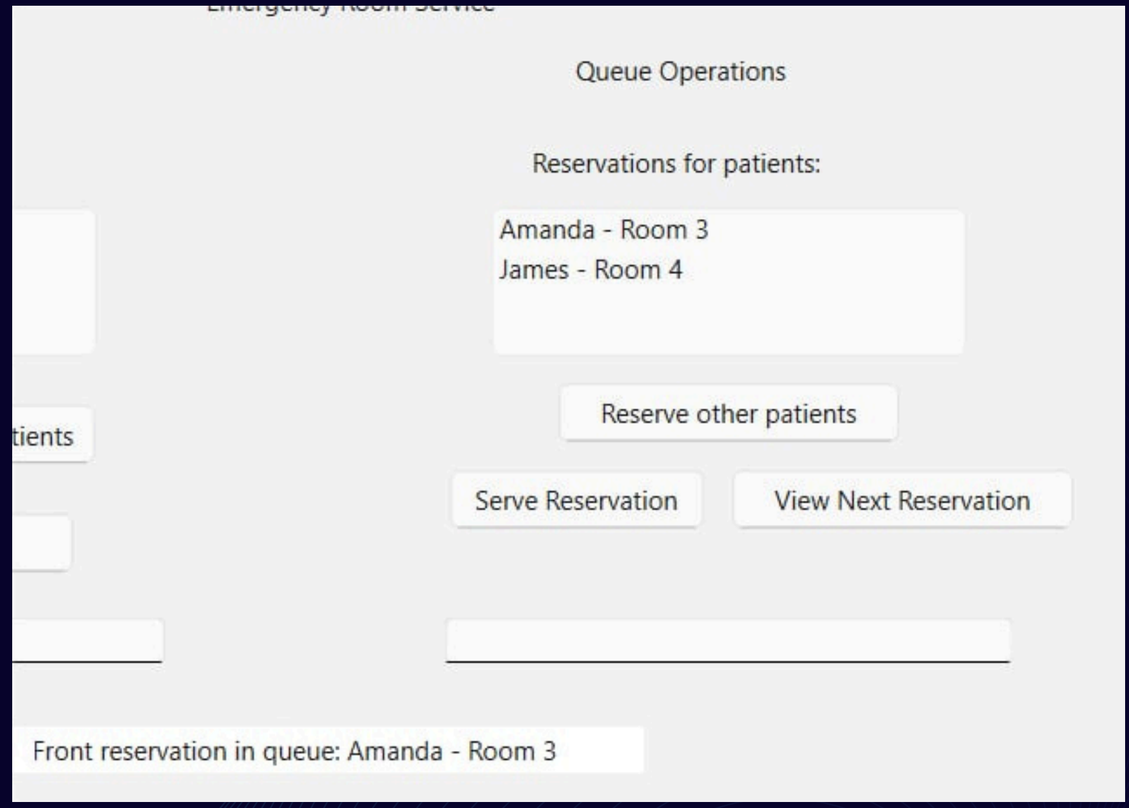


TEST CASES FOR EMERGENCY APPLICATION

Test Case 2: Serve Reservation (Dequeue Patient)

Objective: Verify that the patient at the front of the queue is dequeued properly.

- **Input**: Add a patient "Amanda" to the queue, then click the "Serve Reservation" button.
- **Expected Result**: "Amanda" should be removed from the queue, and the queue list should be updated to reflect the change. The status label should display "Processed front reservation."

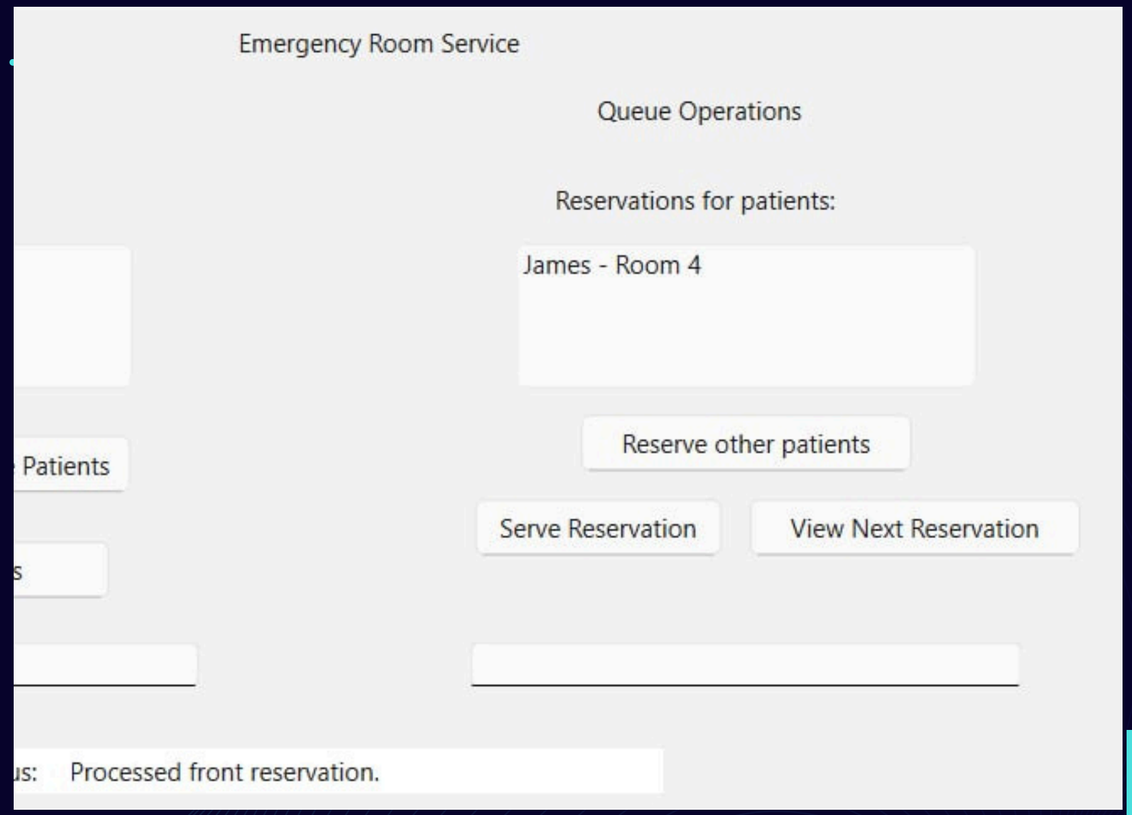


TEST CASES FOR EMERGENCY APPLICATION

Test Case 2: Serve Reservation (Dequeue Patient)

Objective: Verify that the patient at the front of the queue is dequeued properly.

- **Input**: Add a patient "Amanda" to the queue, then click the "Serve Reservation" button.
- **Expected Result**: "Amanda" should be removed from the queue, and the queue list should be updated to reflect the change. The status label should display "Processed front reservation."

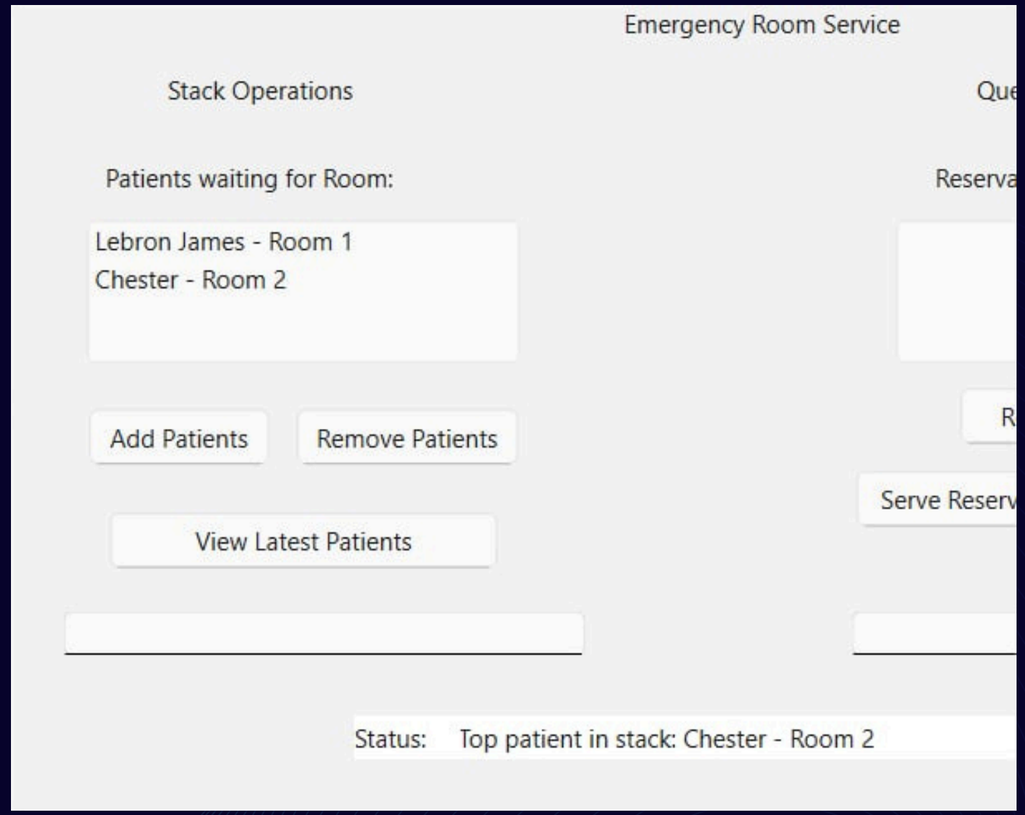


TEST CASES FOR EMERGENCY APPLICATION

Test Case 3: View Latest Patient in Stack (Peek Stack)

Objective: Verify that the top patient in the stack is displayed correctly.

- **Input**: Add a few patients to the stack (e.g., "Lebron James", "Chester"), and click the "View Latest Patients" button.
- **Expected Result**: The status label should display the name of the last added patient (e.g., "Top patient in stack: Chester").



CHALLENGES FACED DURING DEVELOPMENT

- **Stack and Queue Management**: Handling stack and queue operations correctly required careful tracking of the order in which patients were added and processed. Ensuring that the UI reflected the changes in real-time posed challenges that required continuous updates and checks.

- **Real-time Updates to UI**: Keeping the **QListWidget** updated dynamically when the stack or queue was modified required careful management of the **updateStackDisplay()** and **updateQueueDisplay()** methods. Ensuring smooth updates without performance degradation was a challenge.

- **Error Handling**: Proper error messages had to be provided when attempting to pop from an empty stack or dequeue from an empty queue. This involved integrating appropriate checks before performing these operations.

CHALLENGES FACED DURING DEVELOPMENT

- **UI Design**: Ensuring the UI was intuitive for users while also accommodating the logic required for both stack and queue operations involved careful planning, especially in terms of layout and widget management.

ROLES OF MEMBERS AND THEIR CONTRIBUTIONS

- **Lorenzo Pasco** (**Developer**): Responsible for implementing the core logic for stack and queue operations in mainwindow.cpp. Handled the functions that manipulate the stack and queue, ensuring that the operations are correctly executed when buttons are clicked.

- **Edward Lugod** (**UI Designer**): Designed the layout of the application in Qt Designer, including the placement of buttons, input fields, and list widgets. Ensured that the interface was user-friendly and visually appealing.

- **Reece Mendoza** (**Tester**): Responsible for testing the application’s functionality, ensuring that the stack and queue operations worked as expected, and verifying that the status messages were correctly displayed. Reported any bugs or UI inconsistencies.