Theatre Booking System

KAAREL Part

2025

Contents

[Investigation 2](#_Toc1946522404)

[Desk based research 3](#_Toc1834474569)

[Stakeholders 3](#_Toc914167759)

[Limitations of current system 3](#_Toc897444257)

[Aims and Objectives 3](#_Toc504015910)

[Design 3](#_Toc315845151)

[Sub programs 4](#_Toc358866346)

[Task List 4](#_Toc162314208)

[Algorithms 4](#_Toc1006551316)

[User Interface 4](#_Toc1302420851)

[UML 4](#_Toc1147300912)

[Data Dictionary 4](#_Toc1571434351)

[Software Development 5](#_Toc1088753802)

[Developmental Testing 6](#_Toc938788938)

[Testing 6](#_Toc2010129018)

[Evaluation 7](#_Toc1944161941)

## Investigation

### Desk based research

Use this to look online for similar solutions, you should print screen your findings identifying any features you like and will be trying to incorporate into your solution as well as features you think don’t work.

A seating chart of a theater

AI-generated content may be incorrect.

**Visual seat availability view:**

* allows for a great way to convey to the user which seat can be booked in a way that is easy to use.
* no legend needed as its simple to understand from a person who has never seen it before.
* easy to store just a coordinate and seat status and customer id needed to find info on each seat.
* No need to make it too flashy with zoom and moving around the seat view. Justa grid needed

A screenshot of a video game

AI-generated content may be incorrect.

**Info and times for each show:**

* Conveys the times that can be booked in simple form to the user

### Stakeholders

What are the [stakeholders](https://www.bbc.co.uk/bitesize/guides/z4gcd2p/revision/1#:~:text=Stakeholders%20are%20individuals%2C%20groups%20or,They%20include%3A&text=Workers%20who%20want%20to%20earn,quality%20products%20at%20reasonable%20prices.) and what are their requirements for the solution, these should be as diverse as possible and will have different requirements.

### Limitations of current system

The current system is paper based with the performing arts teachers taking notes about any tickets sold.

### Aims and Objectives

1. The system must allow tickets to be booked. For each ticket sold or allocated, the system must record the name of the customer, type of customer, phone number, performance date, seat booked, price paid.
2. The system must prevent a seat, for a particular performance, being sold or allocated more than once.
3. The system must allow seats to be blocked so that they cannot be purchased or otherwise allocated.
4. The system must allow a staff member to search for a customer and display which ticket(s) they have booked for which performance(s)
5. The system must display a list of ticket holders (seat number, name, phone number), sorted by surname, for a selected performance
6. The system must save all relevant data to an external file (so that the system can be closed down at the end of each day and used again the following day)
7. The system must produce management information for each performance
8. Display how many tickets have been sold (or otherwise allocated)
9. Display how many tickets remain available for sale
10. Display the total revenue for each performance

## Design

### Sub programs

*Break your objectives into subprograms, for example.*

*10. Since the club sells items such as golf clubs and golf balls, they will need a way to add these sale values to the database.*

*10.1. It will need a feature to search through all the items to get the details of the item you want.*

*10.2. There needs to be a way to allow for both selling and lending items.*

*10.3. They must only be able to add the number of items in stock and can’t go over that stock.*

*10.4. Also, the prices should automatically be worked out by the program when they are added to the cart.*

*10.5. An option to select quantity before adding to cart should be available to save time pressing add multiple times.*

*10.6. You should also be able to remove items from the cart.*

*10.7. Once all the items the user wants to add are added to the item cart there then should be an option to add it to the overall main cart that contains all the things the user has purchased.*

1. System must allow tickets to be booked and each ticket sold contains info about the customer and the payment.
   1. The tab is visible to both staff and customers
   2. Contains basic information about the show at the top e.g show name and date/time
   3. Below it allows the customer to input the amount of tickets they want to buy
      1. + to increase – to decrease max of 10 tickets / transaction
   4. 2 different rows the customer can increment or decrement with the keys (child & adult)
      1. £5 for below 18 or above 65
      2. £10 for adults 18 < x < 65
   5. A button below that when clicked continues to the next page where the person chooses their seats.
2. System must prevent a seat for a performance to be blocked from being sold more than once.
   1. For each performance, link the seat coordinate with BookingID therefore each has to be unique
3. System booking control for staff members
   1. Separate tab only visitable for staff members that has the same grid layout of seats except instead of the 3 colours they have on which is fully opaque or disabled which is greyish transparent showing that its blocked. This would be reflected to the customers when confirm at the bottom is pressed.
4. The system must allow a staff member to search for a customer and display which ticket(s) they have booked for which performance(s)
   1. Separate tab only visible to staff
   2. Has bookingID lookup at the top
   3. When a successful one is typed in basic info about their booking is displayed (name of the customer, type of customer, phone number, performance date, seat booked, price paid)
5. The system must display a list of ticket holders (seat number, name, phone number), sorted by surname, for a selected performance
   1. Would part of 4. When a clear search is provided.
   2. Looks like a massive table with columns for each piece of info needed.
   3. To get results for 4. The user must input any of the details in any column in the search to find it.
6. The system must save all relevant data to an external file (so that the system can be closed down at the end of each day and used again the following day)
   1. SQL DB with relevant columns to keep in
   2. 4 tables
7. The system must produce management information for each performance
   1. Display how many tickets have been sold (or otherwise allocated)
   2. Display how many tickets remain available for sale
   3. Display the total revenue for each performance
8. Display how many tickets have been sold (or otherwise allocated)
   1. Summary page with key metrics for each performance
   2. Display the total number of tickets sold, remaining tickets, and total revenue
   3. Filter options to view data by date range or specific performance
9. The system must allow refunds and ticket cancellations
   1. Staff-only function accessible through a secure tab
   2. Staff can search for a booking and process a refund if within policy guidelines
   3. Once refunded, the seat is released and available for booking again
10. The system must log all transactions and changes for auditing purposes
    1. Every ticket sale, refund, or change must be recorded with a timestamp
    2. Staff actions (like manual seat blocking) must be logged for accountability
    3. Logs should be exportable for auditing and review purposes

### Task List

For each task (sub program) you need to identify the inputs, outputs and processing

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Input | Output | Processing |
| 1. Ticket Booking | Customer selects show, date, ticket type, quantity, and payment details | Booking confirmation, ticket details, receipt | Validate input, calculate total price, process payment, store booking in database |
| 2. Prevent Duplicate Seat Bookings | Seat selection, BookingID | Confirmation of seat assignment | Check if seat is available, link seat coordinate to BookingID, update database |
| 3. Staff Booking Control | Staff login, seat selection | Updated seat status | Display seat grid, allow staff to block/unblock seats, update customer view upon confirmation |
| 4. Customer Search | BookingID, customer details | Booking details (name, phone, seat, price, etc.) | Retrieve and display booking details from the database |
| 5. List of Ticket Holders | Performance selection, search query | Table of ticket holders sorted by surname | Fetch ticket data, sort by surname, display as a table |
| 6. Data Storage | Booking details, customer info, payments | Data saved in SQL DB | Insert/update data across relevant tables in database |
| 7. Performance Management Info | Show selection | Ticket sales, remaining tickets, total revenue | Aggregate ticket sales data, calculate revenue, display summary |
| 8. Ticket Sales Summary | Performance/date filter | Summary of tickets sold and remaining | Query and summarize sales data by filters |
| 9. Refunds & Cancellations | BookingID, refund request | Refund confirmation, seat released | Validate refund policy, process refund, update seat availability |
| 10. Transaction Logging | Ticket sales, refunds, seat changes | Log entries for auditing | Record each transaction with timestamp, staff/user ID, action performed |

### Algorithms

Each subtask should have its own algorithm written in pseudocode, these should be clear enough for a third party (fellow students) to follow, consider if someone in the class could make your system from these designs.

These should be lots of separate algorithms, I don’t need to see any algorithms for the user interface as these are designed above.

**Algorithms**

**1. Ticket Booking Algorithm**

START  
DISPLAY available shows and dates  
PROMPT user to select a show and date  
PROMPT user to enter number of tickets (max 10)  
PROMPT user to select ticket type (Child/Adult)  
CALCULATE total price  
PROMPT user to enter payment details  
IF payment successful THEN  
 GENERATE BookingID  
 STORE booking details in database  
 DISPLAY booking confirmation  
ELSE  
 DISPLAY payment error message  
ENDIF  
END

**2. Prevent Duplicate Seat Booking Algorithm**

START  
PROMPT user to select a seat  
IF seat is already booked THEN  
 DISPLAY error message "Seat already taken"  
ELSE  
 ASSIGN seat to BookingID  
 UPDATE database with seat selection  
 DISPLAY seat confirmation  
ENDIF  
END

**3. Staff Booking Control Algorithm**

START  
VERIFY staff login  
DISPLAY seat grid with blocked/available seats  
PROMPT staff to block/unblock seats  
IF changes made THEN  
 UPDATE seat availability in database  
 REFLECT changes to customer view  
ENDIF  
END

**4. Customer Search Algorithm**

START  
PROMPT staff to enter BookingID  
SEARCH database for BookingID  
IF found THEN  
 DISPLAY customer details (name, phone, seat, etc.)  
ELSE  
 DISPLAY "Booking not found" message  
ENDIF  
END  
**5. List of Ticket Holders Algorithm**

START  
PROMPT staff to select performance  
RETRIEVE ticket holder list from database  
SORT by surname  
DISPLAY table with ticket holder details  
END  
**6. Data Storage Algorithm**

START  
RECEIVE booking/payment details  
STORE in SQL database (4 tables: customers, bookings, seats, payments)  
CONFIRM data saved successfully  
END  
**7. Performance Management Info Algorithm**

START  
PROMPT user to select a performance  
RETRIEVE ticket sales data  
DISPLAY number of tickets sold, remaining, and total revenue  
END  
**8. Ticket Sales Summary Algorithm**

START  
PROMPT user to enter date range or specific performance  
RETRIEVE sales data for the selected period  
DISPLAY total tickets sold, remaining, and revenue  
END  
**9. Refunds & Cancellations Algorithm**

START  
VERIFY staff login  
PROMPT staff to enter BookingID  
CHECK refund policy validity  
IF valid THEN  
 PROCESS refund  
 RELEASE seat for rebooking  
 UPDATE database  
 DISPLAY refund confirmation  
ELSE  
 DISPLAY "Refund not allowed" message  
ENDIF  
END  
**10. Transaction Logging Algorithm**

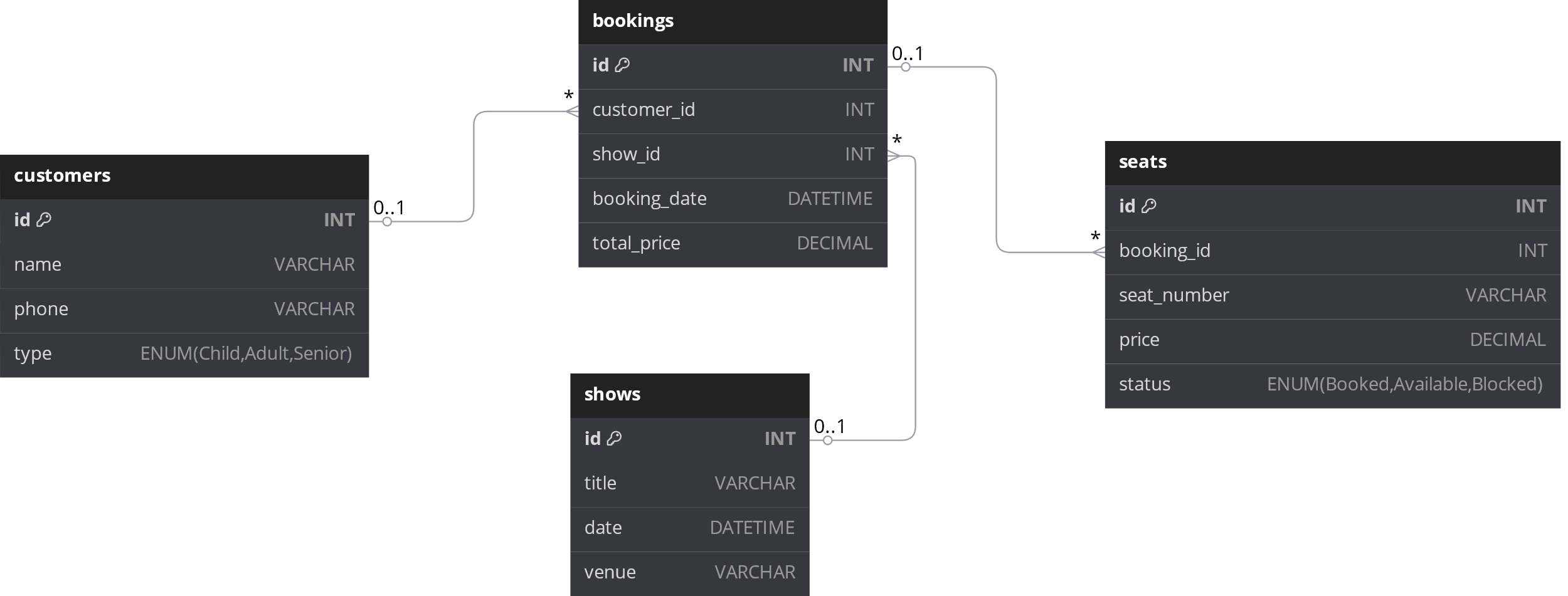
START  
RECORD every sale, refund, or seat change with timestamp  
STORE staff/user actions in log database  
ALLOW exporting logs for auditing  
END

### User Interface

Draw annotated designs of your user interface. These should be detailed enough that another person could make your system and it will look as you envision.

### UML

Include all your UML here, this could be an ERD, DFD or class diagram



### Data Dictionary

Fill in the table for each of your entities.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute/Field | Data type | Validation | Key | Comments |
| customerID | INT | NOT NULL, UNIQUE | PK, FK (Customers → Bookings) |  |
| Name | VARCHAR |  |  |  |
| Phone | VARCHAR |  |  |  |
| Type | ENUM | Child,Adult,Senior |  |  |
| showID | INT | NOT NULL, UNIQUE | PK, FK (shows→ Bookings) |  |
| Title | VARCHAR |  |  |  |
| Date | DATETIME |  |  |  |
| Venue | VARCHAR |  |  |  |
| SeatID | INT | NOT NULL, UNIQUE | PK, FK (seats→ Bookings) |  |
| Seat\_number | VARCHAR |  |  |  |
| Price | VARCHAR |  |  |  |
| Status | ENUM | Booked,Available,Blocked |  |  |
| bookingID | INT | NOT NULL, UNIQUE | PK, FK (Bookings→Seats) |  |
| Booking\_date | DATETIME |  |  |  |
| Total\_price | DECIMAL |  |  |  |

## Developmental Testing

Identify at least 5 different problems you had and your solutions, there should be before and after images.

## Testing

Complete the testing table, you must comprehensively test your solution using a range of testing criteria. Testing should include:

* Valid data – data with expected outcomes that should be accepted by any validation you have included
* Invalid data – data that should be rejected with an error message
* Erroneous data – symbols etc in text fields that could break everything
* Extreme data – only used for range checks, data that should be accepted but is on the edges of acceptable.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Test type | Test data | Expected result | Actual result | Commentary |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Evaluation

You should write a detailed evaluation covering the following points

* How effective the language including libraries etc you chose is for solving this and what features you used.
* Compared the system you have made with those you researched for your desk based research.
* Identified strengths and weaknesses of your system with potential improvements, you don’t have to program these so you can be imaginative
* Your own strengths and weaknesses, consider each stage of the process
* Identify any personal changes you would make for the future