****

Department of Computer Science Engineering

Certificate

This Is to Certify That Mr./Ms. Mohammed Gouse.H Bearing USN: TEMPB-Tech-CSE034 Has Completed Internship Entitled “PYTHON” During 9th To 28th September 2024 for The Partial Fulfilment of Requirements for The Award of Bachelor’s Degree In B-Tech from KISHKINDA UNIVERSITY.

Signature of Internal Guide Signature of External Guide

Signature of HOD Signature of Principal

****

Internship Program on Python for BE-3rd Sem students from 9th to 28th September 2024 (During 3rd semester vacations).

Student Name: Mohammed Gouse.H USN No: BTech-CSE034 Branch: CSE

|  |  |  |  |
| --- | --- | --- | --- |
| **INDEX PAGE** | | | |
| **Day** | **Date** | **Content Covered** | **Signature of the**  **Faculty in-charge** |
| **1** | **09.09.24** | **Overview of Python-IO Statements** |  |
| **2** | **10.09.24** | **Operators basic Problem Solving** |  |
| **3** | **11.09.24** | **Conditional & Looping Statements** |  |
| **4** | **12.09.24** | **List, Tuple with Problem Solving** |  |
| **5** | **13.09.24** | **Set & Dictionary with Problem Solving** |  |
| **6** | **14.09.24** | **Overview of Strings** |  |
| **7** | **15.09.24** | **Strings with Problem Solving** |  |
| **8** | **16.09.24** | **DSA Overview- Stack &Queue (List & LinkedModel)** |  |
| **9** | **18.09.24** | **Linked List-Type Single & Circular** |  |
| **10** | **19.09.24** | **Linked List-Types Double &Double Circular** |  |
| **11** | **20.09.24** | **Binary Tree with Traversal** |  |
| **12** | **21.09.24** | **Binary Search Tree with Traversal** |  |
| **13** | **23.09.24** | **Graph-Build Matrix & Adj.List Model** |  |
| **14** | **24.09.24** | **Graph-BFS,DFS** |  |
| **15** | **25.09.24** | **Project Overview-Submit Project Title** |  |
| **16** | **26.09.24** | **Code Development** |  |
| **17** | **27.09.24** | **Report & PPT Development** |  |
| **18** | **28.09.24** | **Project review-PPT Presentation for Each team** |  |

****

**KISHKINDA UNIVERSITY**

**MINI PROJECT REPORT**

**PROJECT TITLE: Stadium seating Management (POC)**

**PROJECT TEAM MEMBERS**

|  |  |
| --- | --- |
| Akash C | **TEMPBTech-CSE014** |

**Saniya Khanam TEMPBTech-CSE068**

**V. Deekshitha TEMPBTech-CSE027**

**Mohammed Gouse.H TEMPBTech-CSE034**

**Mohammed Israr.S TEMPBTech-CSE039**

**Abdul Rehman TEMPBTech-CSE065**

**TABLE OF CONTENTS**

1. Introduction

2. Objective

3.Methodology

4.Results/Findings

5.Conclusion

6.References

# Introduction :

This project focuses on building a stadium seating management system. It allows users to manage seats in different sections of a stadium, including assigning seats, tracking their availability, and optimizing seating arrangements. The system is implemented using object-oriented programming principles, where classes represent seats, sections, and the stadium itself. The primary goal of the project is to provide a structured and scalable way to handle seating in large venues, such as stadiums or theatres.

* **Brief Background of the Project:**

In any large-scale event venue, managing seating arrangements is crucial for maximizing space, ensuring customer satisfaction, and adhering to safety regulations. Traditionally, managing stadium seating has been done manually or through non-flexible systems that can lead to inefficiencies such as unoccupied seats in prime locations or overly complicated booking systems. This project aims to address these issues by offering a programmatic solution to optimize seat allocation and track occupancy in a more efficient and systematic manner.

* **Problem Statement or Purpose of the Project:**

The primary problem addressed by this project is the difficulty of managing seating arrangements in large venues. Manually handling seating allocations, checking seat availability, and ensuring that optimal usage of space is achieved can be error-prone and time-consuming. The purpose of this project is to create a system that automates seat management, allowing for real-time updates, visual representations of seat occupancy, and simple integration of optimization algorithms to ensure that seats are filled efficiently, from the front rows to the back.

* **Scope of the Project**

**Seat Management:** The project will allow the creation, updating, and deletion of seats in specific sections.

**Section Management:** The system supports the addition and removal of sections, allowing flexible handling of various sections in a stadium.

**Seat Allocation Optimization:** A simple optimization method is included to efficiently fill empty seats from the front to the back of the stadium.

**Unit Testing:** Comprehensive tests ensure the correct functionality of the seating system, including CRUD operations and optimization.

* **Limitations of the Project**

**Basic Optimization Algorithm**: The current seat optimization function fills seats from front to back, but it doesn't take into account more complex factors such as customer preferences (e.g., aisle seats, proximity to the stage), pricing categories, or seat groupings for multiple attendees.

**Single Stadium Context:** The system is designed to manage a single stadium's seating. It doesn't support multiple venues or complex layouts such as multi-tiered seating or different seating zones within the same section.

**No GUI:** The project currently does not include a graphical user interface (GUI), so all operations need to be handled programmatically or via text output. Integrating a more user-friendly interface could make the system easier to use for non-technical users.

**No Real-Time Integration:** The system is not connected to a real-time booking or reservation system, meaning that any seat updates or optimizations have to be manually triggered in this version of the project.

This project provides a foundational structure for managing seating arrangements but can be further extended to address more complex requirements in a real-world stadium or event management system.

**2. Objective:**

The main goals or aims of the stadium seating management system are as follows.

* **Efficient Seat Management:**

Create, update, and track the status (occupied or unoccupied) of individual seats within different sections of a stadium.

* **Section Management:**

Allow for the creation and removal of seating sections, with each section containing a customizable number of rows and seats per row.

Provide the ability to easily manage multiple sections within a single stadium.

* **Real-Time Seat Allocation:**

Enable real-time updates to seat availability based on reservations or seat assignments, making it easy to track which seats are occupied during an event.

* **4. Seat Optimization:**

Implement a simple seat optimization function that automatically fills available seats from the front rows to the back, maximizing the efficient use of space.Lay the foundation for more advanced optimization algorithms that could be added later.

* **Systematic Tracking of Seating Arrangements:**

Provide a structured and scalable approach for managing seating in large venues, ensuring that seat occupancy and availability can be easily monitored.

* **Unit Testing and System Validation:**

Incorporate automated unit tests to ensure the system performs CRUD operations (Create, Read, Update, Delete) correctly, as well as to verify the correctness of the seat optimization functionality.

* **Extendibility for Future Features:**

Design the system to allow for future enhancements, such as integration with a booking platform, customer preferences, or tiered seating, making it adaptable for more complex seating arrangements.

These objectives aim to address the core challenges of managing seating arrangements in large venues and provide a scalable solution that can be expanded and refined based on future needs.

# 3. Methodology:

The development of the stadium seating management system involves several tools, techniques, and processes aimed at creating an efficient and maintainable software solution. The following outlines the tools, algorithms, and workflows used in the project.

* **Tools, Software, and Techniques Used:**

**1. Programming Language: Python:**

Python was chosen for its readability, simplicity, and extensive library support, making it well-suited for developing object-oriented systems like this one.

**2. Object-Oriented Programming (OOP):**

The system is designed using OOP principles, allowing for modularity, code reuse, and scalability.

Key components (Seats, Sections, Stadium) are represented as Python classes to encapsulate data and behaviour related to seat management.

**3. Unit Testing Framework: unittest:**

Python's built-in unittest framework is used to write and run test cases to ensure the system operates as expected.

This allows for automated testing of CRUD operations and seat optimization, validating the system's functionality throughout development.

**4. Data Structures:**

**Dictionaries:** Used to represent seating sections, where each section contains a dictionary mapping row numbers to lists of seats.

**Lists:** Each row in a section is a list of Seat objects, which store information about individual seats such as row number, seat number, and occupation status.

* **Description of the Process, Algorithms, and Workflows Followed:**

**1. Seat and Section Creation Workflow:**

**Objective:** Create seating sections and individual seats for each section, defining the stadium structure.

**Process:** A Stadium class is responsible for managing multiple Seating Section objects. Each Seating Section object contains multiple rows of seats, with each row represented as a list of Seat objects .The Seat class represents individual seats, with attributes such as row number, seat number, and occupancy status.

**Algorithm:** When a new section is added to the stadium, a Seating Section object is created .For each section, the rows dictionary is populated with lists of Seat objects, with the number of rows and seats per row passed as parameters.

**2. Seat Status Update Workflow**:

**Objective:** Update the occupancy status of a specific seat (e.g., when a reservation is made).

**Process:** The update\_seating method in the Stadium class is used to change the occupancy status of a seat.The method takes a section ID, row number, seat number, and the desired occupancy status (True for occupied, False for unoccupied).

**Algorithm:** Locate the section by its section\_id in the sections dictionary. Retrieve the seat using the get\_seat method from the SeatingSection class and update its is\_occupied attribute.

**3.Seat Optimization Workflow:**

**Objective:** Automatically fill unoccupied seats from the front rows to the back to optimize the seating arrangement.

**Process:** The optimize\_stadium\_seating function iterates through all the sections in the stadium.It checks each seat’s occupancy status and marks empty seats as occupied starting from the front row.

**Algorithm:** For each section in the stadium, rows are sorted by their number (to prioritize front rows).For each row, iterate over its seats and set the is\_occupied flag to True for unoccupied seats.

**Workflow:** This function can be invoked to "fill" the stadium seats automatically, making it easy to visualize how an event might progress when attendees are seated.

**4. Unit Testing Workflow:**

**Objective:** Ensure that the system functions as intended and that all operations (create, read, update, delete) behave correctly.

**Process:** Unit tests are written using the unittest framework to validate core functionality.Tests include checking seat creation, updating seat occupancy, deleting sections, and ensuring that the seat optimization function works as expected.

**Algorithm:** Tests are run in isolation using the setUp method, which creates a sample stadium with two sections. Specific CRUD operations and the optimization function are tested to ensure they perform as required.

**Summary of Workflows:**

**1. Seat Creation:**

Each section consists of rows, and each row contains seats. When a section is added, seats are generated automatically based on the number of rows and seats per row.

**2. Seat Status Updates:**

Seat statuses (occupied or unoccupied) are updated through the update\_seating method, making it easy to track and manage seating in real-time.

**3. Optimization:**

Seats are filled in an optimal manner (front to back) using the optimize\_stadium\_seating function, ensuring maximum utilization of available seats.

**4. Testing:**

The system is thoroughly tested to validate that the seating operations and optimizations work correctly and that sections and seats can be created, updated, and deleted as expected.

These methodologies ensure that the project is built in a modular, maintainable, and scalable manner while incorporating robust testing and optimization techniques.

**4. Results/Findings:**

The stadium seating management system was successfully implemented, and the key functionalities were tested using unit tests to ensure correctness and reliability. The following presents the results obtained from testing the core features: seat management, section management, and seat optimization.

**1. Seat and Section Creation Results:**

**Test:** Creation of sections with specific rows and seats per row.

**Outcome:** Sections were successfully created with the correct number of rows and seats. Each seat was initialized with the correct row number, seat number, and was unoccupied by default.

**Example:**

python

stadium.add\_section(101, 10, 10)

stadium.add\_section(102, 5, 20)

**Section 101:** 10 rows with 10 seats per row.

**Section 102:** 5 rows with 20 seats per row.

**Representation (example for Section 101):**

Section 101: 1-1, 1-2, 1-3, ..., 10-10

Section 102: 1-1, 1-2, ..., 5-20

**2. Seat Status Update Results:**

**Test:** Update the occupancy status of specific seats (e.g., marking them as occupied when reserved).

**Outcome:** Seats were successfully updated when a reservation or change in occupancy was applied. The system correctly reflected the updated seat status.

**Example:**

python

stadium.update\_seating(101, 1, 1, True)

Seat 1 in Row 1 of Section 101 was marked as occupied (1-1\*).

**Representation:**

Section 101: 1-1\*, 1-2, 1-3, ..., 10-10

**3. Seat Optimization Results:**

**Test:** Automatically fill unoccupied seats from the front rows to the back using the seat optimization algorithm.

**Outcome**: The algorithm successfully filled the seats in each section from the front row to the back row. All unoccupied seats were marked as occupied after the optimization process.

**Before Optimization:**

Section 101: 1-1, 1-2, 1-3, ..., 10-10

Section 102: 1-1, 1-2, ..., 5-20

**After Optimization\* (all seats marked as occupied):**

Section 101: 1-1\*, 1-2\*, 1-3\*, ..., 10-10\*

Section 102: 1-1\*, 1-2\*, ..., 5-20\*

**4. CRUD Operations Results:**

**Test**: Create, read, update, and delete operations for sections and seats.

**Outcome:** Seats and sections were successfully created, retrieved, updated, and deleted without errors.

The system correctly handled the deletion of sections and ensured that no data remained for deleted sections.

**Example:** After deleting Section 102:

Python

stadium.remove\_section(102)

**Result:** Section 101: 1-1\*, 1-2\*, ..., 10-10\*

**Unit Test Results:**

All unit tests were run using Python's unittest framework, and they validated the following:

**Test Seat Creation:** Successfully created sections with the correct number of rows and seats.

**Test Seat Update**: Seat status was updated correctly when reserved.

**Test Seat Optimization**: Unoccupied seats were correctly marked as occupied from the front rows to the back.

**Test Section Deletion:** Sections were deleted properly without leaving any residual data.

**Example Test Output:**

bash

Ran 2 tests in 0.001s

OK

**Analysis of the Results:**

**1. Efficiency in Seat Management:**

The system allows efficient seat and section creation, updates, and deletion. The use of dictionaries and lists ensures quick access to seat data and easy modification of the stadium structure.

**2. Successful Seat Optimization:**

The seat optimization function worked as intended, filling unoccupied seats from the front rows to the back. This feature could be beneficial for maximizing space usage during events where assigned seating is necessary.

**3. Scalability:**

The project demonstrates potential scalability. By adding more sections and seats, the system can easily handle large-scale stadiums with thousands of seats.

**4. Modularity:**

The system's design follows good object-oriented principles, allowing for easy extension. For instance, more complex seat optimization algorithms or customer preferences (such as choosing aisle seats) can be added in the future.

**5. Limitations:**

The seat optimization process does not currently handle advanced seat allocation strategies (e.g., pricing tiers or groups of attendees).

The absence of a graphical user interface (GUI) limits user interaction to text-based input/output, making it less user-friendly for non-technical users.

**5. Conclusion:**

The stadium seating management system successfully addresses the core challenges of efficiently managing seating arrangements in large venues. By leveraging object-oriented programming principles, the system enables the creation, updating, and deletion of seats and sections, while also providing functionality for optimizing seat allocation from the front rows to the back. The system was rigorously tested using Python's unittest framework to validate its performance and correctness in performing CRUD operations and seat optimization.

**Main Points:**

**1. Efficient Seat and Section Management:**

The system allows easy creation and management of multiple seating sections, each with customizable rows and seats per row.

Seats can be individually updated to reflect their occupancy status, providing real-time seat tracking.

**2. Seat Optimization:**

A basic seat optimization algorithm was implemented to ensure that unoccupied seats are filled in an organized manner, starting from the front of the stadium.

**3. Testing and Validation:**

Comprehensive unit tests confirmed that the system operates as expected, handling all CRUD operations (Create, Read, Update, Delete) and optimizing seat arrangements effectively.

**Lessons Learned:**

**Modularity and Scalability:**

The project demonstrated the importance of modularity in software design. By breaking down the problem into smaller components (Seats, Sections, Stadium), the system is flexible and easy to maintain or expand.

**Optimization Complexity:**

While the basic seat optimization function works well for simple scenarios, more advanced seat allocation strategies (e.g., handling customer preferences, pricing tiers, or group seating) would require more complex algorithms. This highlights the need for careful planning when implementing features that involve real-world variables.

**Testing for Reliability:**

Incorporating unit tests during development ensures that the system remains stable and reliable as new features or changes are introduced. Testing is an essential part of building a robust system.

**Future Work:**

**1. Advanced Seat Optimization:**

Develop more sophisticated algorithms that take into account customer preferences, proximity to exits, and dynamic pricing to enhance seat allocation strategies.

**2. Graphical User Interface (GUI):**

Implement a user-friendly GUI to make the system accessible for non-technical users, allowing for visual seat selection and real-time updates.

**3. Real-Time Booking Integration:**

Integrate the system with a real-time booking or ticketing system, enabling users to reserve seats online and automatically updating seat availability.

**4. Group Seat Management:**

Add functionality to handle group reservations, ensuring that multiple seats can be reserved together, taking into account their proximity.

**5. Multi-Stadium Support:**

Extend the system to support multiple venues or stadiums, each with its own seating arrangement and booking system, allowing it to manage events at different locations simultaneously.

In conclusion, this project provides a strong foundation for a stadium seating management system that can be expanded and adapted to meet more complex real-world requirements. Future improvements will make the system more sophisticated, user-friendly, and capable of handling large-scale events efficiently.

**6. References:**

**1. Python Documentation:** Python Software Foundation. (n.d.). Python 3 Documentation. Retrieved from [https://docs.python.org/3/](https://docs.python.org/3/)

Used as a reference for understanding Python syntax, object-oriented programming principles, and the unittest framework.

**2. Unit Testing with Python:**

Beazley, D. M., & Jones, B. K. (2013). Python Cookbook. O'Reilly Media, Inc. Chapter on testing and debugging in Python, particularly using the unittest framework, helped in designing test cases.

**3. Object-Oriented Programming in Python:**

Hetland, M. L. (2005). Beginning Python: From Novice to Professional. Apress. Provided insights into implementing OOP concepts in Python, which were key to designing classes like Seat, SeatingSection, and Stadium.

**4. \*eat Optimization Concepts:**

Larson, R. C. (1987). Perspectives on Queues: Social Justice and the Psychology of Queueing. Operations Research, 35(6), 895–905. Helped in understanding basic optimization techniques for efficiently filling seats in an organized manner.

**5. Software Design Patterns:**

Freeman, E., & Robson, E. (2004). Head First Design Patterns. O'Reilly Media, Inc. Provided general design patterns used in object-oriented programming, influencing the overall system design.

These references provided foundational knowledge and practical guidance for developing the stadium seating management system.