# BookInTime

# Software Design

## CSCI-P465/565 (Software Engineering I)

## Project Team

**Aditya Shahapure**

**Brendan McShane**

**Prathamesh Deshmukh**

**Shanthan Reddy M**

---

# 1. Introduction

We aim to use the object-oriented approach for designing our software. Object-oriented approach is wherein the entire system is viewed as a collection of objects (or entities). The state of the entire system is distributed among the objects; and each object handles its state data.

### 1.1 System Description

For the upcoming sprint, we plan to do the following:

1. Design and build a login page
2. Designing a system which lets the administrator/management team upload the movies data
3. If possible, come up with a system which incorporates the functionality of searching for a movie

## 1.2 Design Evolution

### 1.2.1 Design Issues

We are yet to face any issues related to design. We chose to go ahead with the object-oriented design approach due to the reasons mentioned below in section 1.2.3.

### 1.2.2 Candidate Design Solutions

We thought about two approaches for the software design, viz., function-oriented and object-oriented. In case of a function-oriented approach, the design is decomposed into a set of interacting units where each unit has one clearly defined function. The details of an algorithm are concealed within a function, but the system state information is not hidden.

As for the object-oriented design, the entire system is viewed as a collection of objects (or entities). The state of the entire system is distributed among the objects; and each object handles its state data. The different concepts revolving around this particular methodology are: objects, classes, messages, abstraction, encapsulation, inheritance, polymorphism etc.

### 1.2.3 Design Solution Rationale

We decided to go ahead with the object-oriented approach to designing the software as it comes with several merits. It more or less mimics the real-world scenario wherein everything could be considered as a bunch of different entities connected together, with each entity performing its own individual task. Also, modern programming languages support and encourage the object-oriented design paradigm. This particular approach also helps in reducing development time, along with reducing the time and the number of resources required to maintain existing applications. It also promotes code reusability, and provides a competitive advantage to the organizations that use it.

## 1.3 Design Approach

### 1.3.1 Methods

As was mentioned before, we decided to make use of the object-oriented design approach. Apart from the reasons stated above, usage of object-oriented design methodologies also results in the systems having a natural structure for modular design, which includes several aspects like objects, subsystems, frameworks etc.

The developed systems are thus easier to modify. Further, this approach also makes sure that the system design can be altered in fundamental ways without ever breaking up, as any changes made are neatly encapsulated.

### 1.3.2 Standards

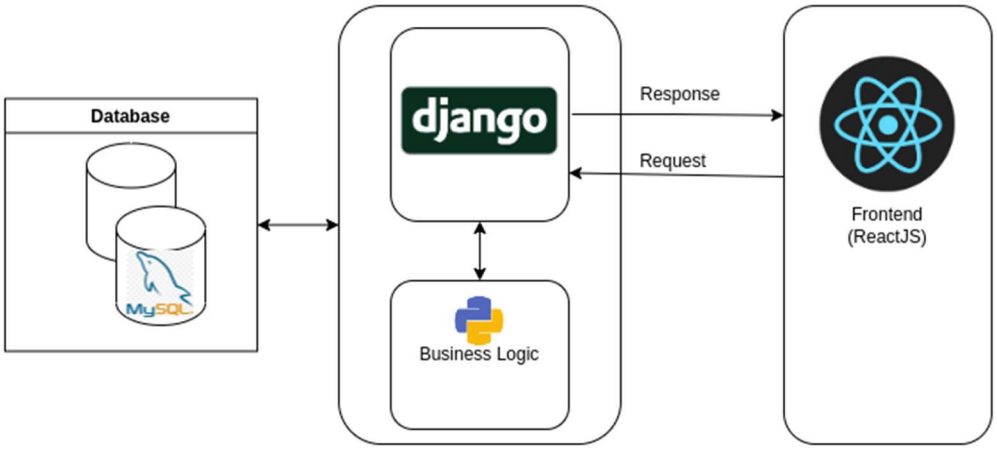
We strive to create a software system which is quite accessible to the users. Also, the name “BookInTime” was chosen to let the users know how quick and simple it will be for them to use our system for booking tickets for their favorite movies. We also aim to do a fair business in that there will not be any discrimination involved as far as any movie theater’s inclusion/exclusion is concerned.

### 1.3.3 Tools

We plan to use ‘React’ and ‘Bootstrap’ frameworks for developing the frontend part of our software, while ‘Django’ (or ‘Flask’) for the backend. As for the database, we intend to go ahead with ‘MySQL’.

# 2. System Architecture

## 2.1 System Design:




For the frontend part, we plan to implement the ‘MVC’ (‘model-view-controller’) design pattern. We intend to use ‘React’ and ‘Bootstrap’ as the main technologies for the same. Further, we plan to use Rest-API for the purpose of communicating between the front and the back end. The backend portion of our system will consist of Django and MySQL relational databases. The front end will make calls to create users, log users in, schedule movies, buy tickets, and more, which will all require data storage and retrieval. That will happen with API calls to our MySQL relational database via our Django/Flask backend frameworks. These frameworks will deliver the queried information and allow the front end to display and make use of accurate historical data.

### 2.2 External Interfaces

We plan on connecting with the Google maps API so users can find theatres near them. We also plan on connecting with a Fandango API, or something similar, to find all actively playing movies in the US (so there isn't extra noise surrounding movie title and people must choose from a constrained list).

# 3. Component Design

The Component Design section details the proposed design of each system component. A system component is a functional partition of the system. Components may be organized as you see fit - a component may be a collection of objects, or a single object. However, a system must be composed of multiple components (that is, a system cannot be one component). The layout of this section is at your discretion, but please include the following (at a minimum) information for each component:

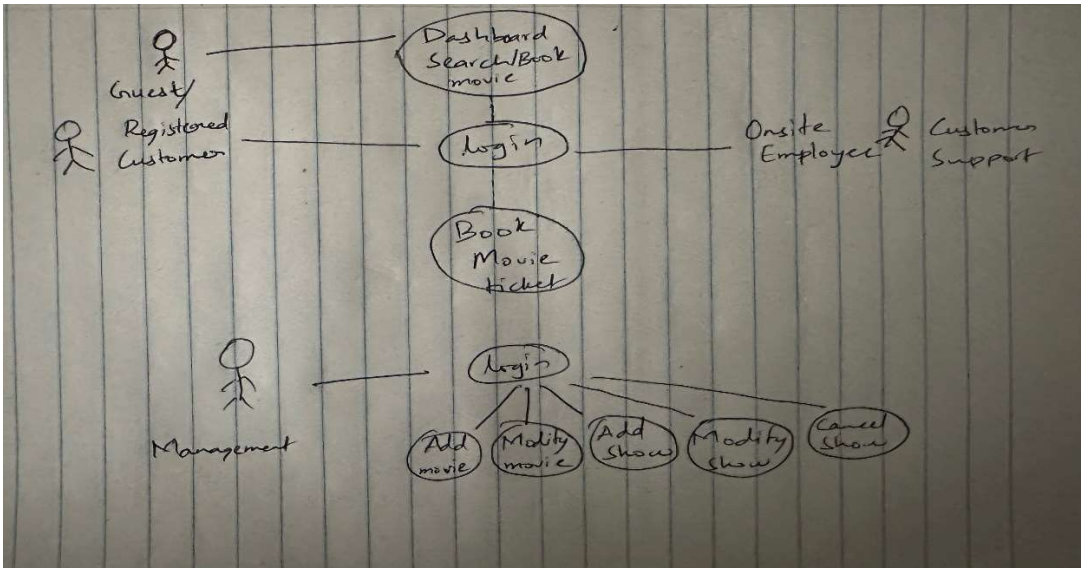
- ## User Interface:

  **Description:**

  This is where the users will be interacting with our system. The user-base consists of customers, management team, onsite employees and customer support team.

  **Responsible Development Team Members:**
  Aditya Shahapure, Shanthan Reddy

**Component Diagram:**




**Component User Interface:**

We plan to design a login page which will let the users log in to the system using their email address and password. The login process can also be done using the user's google or Facebook credentials or also using the mobile phone number. The user interface will also let the users search for cities from where they wish to make the booking. Further, it will have different options to make payments. Further, the user interface will also include a dashboard and an option to add user profiles. Next, we also plan to implement a chat box wherein the users can raise any queries and get responses from employees.

**Component Objects:**

We have one 'customer' class, one 'management' class, one 'on-site employees' class and one 'customer support' class. Further, we also have one 'movie' class, 'city' class, 'movie show' class etc.

**Component Interfaces:**

The login component will consist of several sub-components like email-ID, password, Facebook/Google login ID and mobile phone number. Another component will be the 'login' or 'signup' button, which will send an authentication request to the backend system; a successful response from which will let the user login without any error. Also, the administrator team will be

able to upload data related to different movies to the system. The exact design is yet to be planned.

**Component Error Handling:**

Error check case 1: Check if the email ID entered is valid

Error check case 2: Check if the entered password matches with the corresponding account.

Error check case 3: Check if the entered password satisfies all the requirements of a valid password. (Password validation check)

Error check case 4: If the user logs in using his/her mobile number, then check if the same is valid.

- # Database

  **Description:**

  Our database for storing movies currently playing in the US, current users (of all types), and movie theatres utilizing our services (potentially more tables to come). This component will be a part of the main execution loop. When users try to search for movies or theatres try to schedule upcoming showings, the title will be checked against movies actively playing in the US. When users try to find movie theatres near them, their location will be checked against the locations of the movie theatres utilizing our services. When a user tries to log in, their usernames/passwords will be checked against the user's data table. There will also be separate home screens depending on what type of user the current one is (customer, movie theater employee, etc). We'll be able to send promotional offers and allow users to reset usernames/passwords by utilizing the user's data table.

  **Team Member(s):**

  Brendan McShane, Prathmesh Deshmukh

**User Interface**:

There won't be a direct user interface for our database

**Objects:**

- Movie title data table: this will be used to store the list of all movies actively playing in US theaters today. Customer searches as well as movie theatre schedule additions will be checked against this table. Customers will be able to filter by genre. Will receive data from and send data to the front-end interface of our services.
- User data table: will be a list of all active users of our services. Users will be able to login, change usernames/passwords, receive promotional offers and notifications via email, and check movie theaters near them. Will receive data from and send data to the front-end interface of our services.
- Movie theater data table: this will store all movie theaters actively using our services. This will be part of the process for users checking which movie theaters are near them (we'll store their location). This table is also what will be used to check if a given theater is allowed to schedule movies. Will receive data from and send data to the front-end interface of our services.

**Error Handling:**

- Movie title data table: all titles will need to be within a certain number of characters, will need to fit within a confined list of genres, the air date must make sense, ticket cost must be a positive integer (and likely within a given range).
- User data table: users will need to have valid email addresses, passwords fitting certain specifications, they'll need to be in the
- Movie theater data table: theaters will need to belong to a confined list of companies, have geological locations that make sense, a positive integer number of screens on which to play movies/represent capacity, etc.

- # Django REST API

  **Description:**

  This component acts as a REST API interface to handle requests from the front-end client and fulfills those requests by returning an appropriate response. It accesses the MySQL database to fetch/insert requested data. Different requests (GET/PUT/POST) are handled by their specific 'views' ( Django views are Python functions that takes http requests and returns http response, like HTML documents). Requests can be of different types: authenticating users, registering new users, requesting lists of movies for specific theater, searching for movies/nearby theaters, booking etc. Based on the type of request, Django framework either fetches the data directly from the database, or it triggers certain logic/algorithm which does the computation and returns the data.

  **Team Member(s):**

  Brendan McShane, Prathmesh Deshmukh

  **User Interface:**

  None

  **Objects:**

  - Views: Logic for handling different types of fronted requests goes into these component objects (.py files). It contains function definitions for get/put/post methods to handle http requests for that specific view.
  - URLs: This object parses the request url and resolves it to one of the views. The request parameters are then forwarded to that view module.
  - Models: Model objects are used to access and manage data. Every table in the database is mapped to a corresponding model class. Django uses objects of these model classes to interact with(read/write) the database tables.

**Component Interfaces:**

Django API framework uses REST API interface to interact with other components, and it exchanges data as 'json' objects.

**Error Handling:**

will be developed as framework is built

- # Business logic module

**Component Description:**

This component contains modules to execute different algorithms to fulfill requests (such as movie recommendations, or any requests that require external interfaces such as google maps API etc.

**Component Objects:**

Every component object will be a python module(scripts) that executes specific type of logic.

**Component Interfaces:**

This component's functionality can be used via python class/object interface by Django framework.