

DBMS Report

Team

AI23BTECH11013 - Bhuvan Chandra K
AI23BTECH11015 - Mohammed Abdur Rehman
AI23BTECH11022 - Dhadheechi Ravva
AI23BTECH11025 - Sampadram Kumar J

Contents

1	Database Design	2
1.1	bookings.event Table	2
1.2	bookings.theatre Table	2
1.3	bookings.screen Table	3
1.4	bookings.showtime Table	3
1.5	bookings.seat Table	3
1.6	bookings.tier Table	4
1.7	bookings.customer Table	4
1.8	bookings.booking Table	5
1.9	bookings.transaction Table	5
1.10	bookings.cancellation Table	6
1.11	bookings.city Table	6
1.12	bookings.organizer Table	7
2	ER Diagram Analysis	7
2.1	One-to-Many Relationships (1 to Many)	7
2.2	One-to-One Relationships (1 to 1)	9
2.3	Implications of Cardinality Constraints	10
2.4	Relationship Diamonds (Rhombuses) Analysis	10
3	Backend	11
3.1	views.py	11
3.2	models.py	15
3.3	urls.py	16
4	Frontend	17
5	HTML templates	17
5.1	CSS Templates	19

1 Database Design

1.1 bookings_event Table

1.1.1 Table Summary

The `bookings_event` table stores details of the events available for booking, such as movies and concerts. The attributes in the table are: Event name, Event type, Duration, Language, Organizer of the event. This table manages all the data related to events on the website.

1.1.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{event_name, event_type, duration, lang, organizer_id\}$

Possible Organizer Dependency: $organizer_id \rightarrow event_name$

1.1.3 Normalization Analysis

First Normal Form (1NF) The `bookings_event` table is in 1NF form as all columns store atomic values, and there are no multi-valued attributes or repeating groups.

Second Normal Form (2NF) The table is in 2NF form because:

- The primary key is a single column (`id`), so there are no partial dependencies.
- All attributes are fully dependent on `id`.

Third Normal Form (3NF) The table is in 3NF form because:

- No attribute is indirectly dependent on the primary key.
- `event_name`, `event_type`, `duration`, `lang`, and `organizer_id` do not determine each other.

Therefore, the `bookings_event` table is in 3NF form and does not require any changes.

1.2 bookings_theatre Table

1.2.1 Summary of the Table

The `bookings_theatre` table stores information about theatres available for event screenings. Each row represents a unique theatre, with the following attributes: Theatre Name, Location, Total Number of Screens, City ID. This table links each theatre to a specific city using the `city_id`.

1.2.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{theatre_name, loc, total_screens, city_id\}$

City Dependency (Conditional): $city_id \rightarrow loc$

1.2.3 Final Normal Form

The `bookings_theatre` table is in **Third Normal Form (3NF)** because:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

Since it is already in 3NF, no further changes are required.

1.3 bookings_screen Table

1.3.1 Summary of the Table

The `bookings_screen` table stores details about individual screens available in theatres. Each row represents a unique screen with the following attributes: Screen Name, Total Seating Capacity, Theatre ID. This table links each screen to a specific theatre using the `theatre_id`.

1.3.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{screen_name, total_seats, theatre_id\}$

Theatre Dependency (Conditional): $theatre_id \rightarrow screen_name$

1.3.3 Final Normal Form

The `bookings_screen` table is in **Third Normal Form (3NF)** because:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

Since the table is already in 3NF, no changes are needed.

1.4 bookings_showtime Table

1.4.1 Summary of the Table

The `bookings_showtime` table stores information about scheduled showtimes for various events. Each row represents a unique showtime with the following attributes: Show Date, Slot Time, Available Seats, Event ID, Screen ID. This table also links each showtime to a specific event using the `event_id`.

1.4.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{show_date, slot_time, available_seats, event_id, screen_id\}$

Event and Screen Dependency (Conditional): $\{event_id, screen_id, show_date, slot_time\} \rightarrow available_seats$

1.4.3 Final Normal Form

The `bookings_showtime` table is in **Third Normal Form (3NF)** because:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

This table is already in 3NF.

1.5 bookings_seat Table

1.5.1 Summary of the Table

The `bookings_seat` table stores information about individual seats available in a theatre. Each row represents a unique seat with the following attributes: Seat Number, Tier ID. This table ensures each seat has a unique identifier (`seat_number`) and links each seat to a pricing tier using the `tier_id`.

1.5.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{seat_number, tier_id\}$

Seat Number Dependency (Conditional): $seat_number \rightarrow id$

Tier Dependency: $tier_id \rightarrow seat_number$

1.5.3 Final Normal Form

The `bookings_seat` table is in **Third Normal Form (3NF)** because:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

Thus, this table is already in 3NF form.

1.6 bookings_tier Table

1.6.1 Summary of the Table

The `bookings_tier` table categorizes seating arrangements into different pricing tiers within a screen. Each row represents a unique tier, and the attributes of the table are: name, price, and the screen to which it belongs. This table also links each tier to a specific screen using `screen_id` and uses dynamic pricing based on seat categories.

1.6.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{tier_name, price, screen_id\}$

Screen Dependency (Conditional): $\{screen_id, tier_name\} \rightarrow price$

1.6.3 Final Normal Form

The `bookings_tier` table is in **Third Normal Form (3NF)**.

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

So this table is already in 3NF form, we do not need to change it.

1.7 bookings_customer Table

1.7.1 Summary of the Table

The `bookings_customer` table stores information about customers who use the event booking system. Each row represents a unique customer, and the attributes of this table are: name, contact details, and associated user account. This table also ensures each customer has a unique email and phone number and links a customer to a user account (if available) via `user_id`.

1.7.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{customer_name, email_id, phone_no, user_id\}$

Uniqueness Constraints: $email_id \rightarrow id$

$phone_no \rightarrow id$

$user_id \rightarrow id$

1.7.3 Final Normal Form

The `bookings_customer` table is in **Third Normal Form (3NF)**.

- 1NF: The table has atomic values with no repeating groups.
- 2NF: All non-key attributes fully depend on the primary key (`id`).
- 3NF: No transitive dependencies exist; all attributes directly depend on `id`.

So this table is already in 3NF form.

1.8 bookings_booking Table

1.8.1 Summary of the Table

The `bookings_booking` table manages ticket reservations for events. Each row represents a unique booking, and the attributes of the table are: booking date, status, associated customer, seat, and showtime. This table also serves the following purposes:

- Links a booking to a specific customer (`customer_id`).
- Associates a booking with a specific seat (`seat_id`).
- Connects the booking to a specific showtime (`show_id`).
- Optionally links a booking to a registered user (`user_id`).

1.8.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{booking_date, status, customer_id, seat_id, show_id, user_id\}$

Seat-Showtime Dependency (Conditional): $\{seat_id, show_id\} \rightarrow customer_id$

User-Booking Dependency: $user_id \rightarrow customer_id$

1.8.3 Final Normal Form

The `bookings_booking` table is in **Third Normal Form (3NF)**.

- 1NF: The table has atomic values with no repeating groups.
- 2NF: All non-key attributes fully depend on the primary key (`id`).
- 3NF: No transitive dependencies exist; all attributes directly depend on `id`.

From this we observe that the table is already in 3NF form.

1.9 bookings_transaction Table

1.9.1 Summary of the Table

The `bookings_transaction` table records financial transactions related to event bookings. Each row represents a unique payment, specifying the amount paid, payment method, status, and associated booking. Some uses of this table are:

- Links each transaction to a specific booking (`booking_id`).
- Ensures each transaction has a unique transaction ID (`transaction_id`).
- Records the timestamp when the transaction occurred.

1.9.2 Functional Dependencies

Primary Key Dependency:

$id \rightarrow \{amount_paid, payment_method, payment_status, transaction_date, booking_id, transaction_id\}$

Transaction ID Uniqueness: $transaction_id \rightarrow id$

Booking Dependency: $booking_id \rightarrow amount_paid$

1.9.3 Final Normal Form

The `bookings_transaction` table is in **Third Normal Form (3NF)**.

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

Since this table is already in 3NF form, no changes are required.

1.10 bookings_cancellation Table

1.10.1 Summary of the Table

The `bookings_cancellation` table records details of cancelled bookings. Each row represents a booking cancellation, and the attributes of the table are: the cancellation date, refund status, and associated booking. It also links each cancellation to a specific booking (`booking_id`).

1.10.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow \{cancel_date, refund_status, booking_id\}$

Booking Dependency: $booking_id \rightarrow cancel_date$

Refund Dependency: $booking_id \rightarrow refund_status$

1.10.3 Final Normal Form

The `bookings_cancellation` table is in **Third Normal Form (3NF)**. To see this, we can verify it as such:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** All non-key attributes fully depend on the primary key (`id`).
- **3NF:** No transitive dependencies exist; all attributes directly depend on `id`.

1.11 bookings_city Table

1.11.1 Summary of the Table

The `bookings_city` table stores information about cities where events and theatres are available. Each row represents a unique city that can be selected by users when browsing available events. **Key Functions:**

- Stores city details with a unique identifier (`id`).
- Ensures each city name is distinct.
- Acts as a reference for other tables like `bookings_theatre`, linking theatres to cities.

1.11.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow city_name$

1.11.3 Final Normal Form

The `bookings_city` table is in **Third Normal Form (3NF)**. To verify these, we can manually check the conditions:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** No partial dependencies exist since `id` is the only key.
- **3NF:** No transitive dependencies exist; `city_name` depends directly on `id`.

1.12 bookings_organizer Table

1.12.1 Summary of the Table

The `bookings_organizer` table stores information about event organizers. Each row represents a unique organizer responsible for managing events within the system. This table also ensures that organizer names are unique to prevent duplication and acts as a reference for the `bookings_event` table, linking events to their respective organizers.

1.12.2 Functional Dependencies

Primary Key Dependency: $id \rightarrow organizer_name$

Unique Organizer Constraint: $organizer_name \rightarrow id$

1.12.3 Final Normal Form

The `bookings_organizer` table is in **Third Normal Form (3NF)**. To prove this, check the conditions:

- **1NF:** The table has atomic values with no repeating groups.
- **2NF:** No partial dependencies exist since `id` is the only key.
- **3NF:** No transitive dependencies exist; `organizer_name` depends directly on `id`.

2 ER Diagram Analysis

2.1 One-to-Many Relationships (1 to Many)

1. City to Theatre

- One city (`bookings_city`) can have many theaters (`bookings_theatre`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `city_id` in `bookings_theatre`

2. Theatre to Screen

- One theatre (`bookings_theatre`) can have many screens (`bookings_screen`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `theatre_id` in `bookings_screen`

3. Screen to Tier

- One screen (`bookings_screen`) has many pricing tiers (`bookings_tier`)
- **Cardinality:** 1 to 1..*

- **Foreign key:** `screen_id` in `bookings_tier`

4. Tier to Seat

- One tier (`bookings_tier`) contains many seats (`bookings_seat`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `tier_id` in `bookings_seat`

5. Organizer to Event

- One organizer (`bookings_organizer`) hosts many events (`bookings_event`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `organizer_id` in `bookings_event`

6. Event to Showtime

- One event (`bookings_event`) has many showtimes (`bookings_showtime`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `event_id` in `bookings_showtime`

7. Screen to Showtime

- One screen (`bookings_screen`) hosts many showtimes (`bookings_showtime`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `screen_id` in `bookings_showtime`

8. Customer to Booking

- One customer (`bookings_customer`) makes many bookings (`bookings_booking`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `customer_id` in `bookings_booking`

9. Showtime to Booking

- One showtime (`bookings_showtime`) can have many bookings (`bookings_booking`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `show_id` in `bookings_booking`

10. Seat to Booking

- One seat (`bookings_seat`) can be booked many times (across different showtimes)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `seat_id` in `bookings_booking`

11. Booking to Transaction

- One booking (`bookings_booking`) can have many transactions (`bookings_transaction`)
- **Cardinality:** 1 to 1..*
- **Foreign key:** `booking_id` in `bookings_transaction`

2.2 One-to-One Relationships (1 to 1)

• Booking to Cancellation

- One booking (`bookings_booking`) can have exactly one cancellation (`bookings_cancellation`)
- **Cardinality:** 1 to 1
- **Foreign key:** `booking_id` in `bookings_cancellation`

2.3 Implications of Cardinality Constraints

- The notation “1..*” indicates that each parent entity must have at least one child entity (mandatory relationship)
- The notation “1” indicates exactly one relationship

This structure is maintained such that:

- Every city must have at least one theater
- Every theater must have at least one screen
- Every screen must have at least one pricing tier
- Every tier must have at least one seat
- Customers must have at least one booking
- Every booking must be associated with exactly one seat, one showtime, and one customer

2.4 Relationship Diamonds (Rhombuses) Analysis

In ER notation, rhombuses typically represent the relationships between entities, helping to clarify how different tables are connected. Here’s what these relationship diamonds likely represent in your diagram:

- **LOCATED_IN** - Relationship between City and Theatre
 - *Meaning:* Theaters are located in cities
 - *Connects:* `bookings_city` to `bookings_theatre`
- **HAS** - Relationship between Theatre and Screen
 - *Meaning:* Theaters have screens
 - *Connects:* `bookings_theatre` to `bookings_screen`
- **DIVIDED_INTO** - Relationship between Screen and Tier
 - *Meaning:* Screens are divided into pricing tiers
 - *Connects:* `bookings_screen` to `bookings_tier`
- **CONTAINS** - Relationship between Tier and Seat
 - *Meaning:* Tiers contain seats
 - *Connects:* `bookings_tier` to `bookings_seat`
- **HOSTS** - Relationship between Organizer and Event
 - *Meaning:* Organizers host events
 - *Connects:* `bookings_organizer` to `bookings_event`
- **SCHEDULED_AT** - Relationship between Event, Screen, and Showtime
 - *Meaning:* Events are scheduled at screens for specific showtimes
 - *Connects:* `bookings_event` and `bookings_screen` to `bookings_showtime`
- **MAKES** - Relationship between Customer and Booking
 - *Meaning:* Customers make bookings
 - *Connects:* `bookings_customer` to `bookings_booking`

- **RESERVES** - Relationship between Booking and Seat
 - *Meaning*: Bookings reserve seats
 - *Connects*: `bookings_seat` to `bookings_booking`
- **FOR** - Relationship between Booking and Showtime
 - *Meaning*: Bookings are made for specific showtimes
 - *Connects*: `bookings_showtime` to `bookings_booking`
- **HAS** - Relationship between Booking and Transaction
 - *Meaning*: Bookings have transactions
 - *Connects*: `bookings_booking` to `bookings_transaction`
- **MAY_HAVE** - Relationship between Booking and Cancellation
 - *Meaning*: Bookings may have cancellations
 - *Connects*: `bookings_booking` to `bookings_cancellation`

3 Backend

3.1 views.py

3.1.1 Index View

The index view redirects all visitors from the root URL to the registration page, making user registration the entry point to your application.

3.1.2 book_ticket View

The `book_ticket` view handles the ticket booking process: Needs the user to be logged in. City Verification: Checks if the user has selected a city; redirects to city selection if not. Implements a cascading filter system for:

- Theaters in the selected city
- Events at the selected theater
- Available showtimes for the selected event and date
- Tiers available in the selected screen
- Available seats (excluding already booked ones)

When a `POST` request is received:

- Validates all required parameters
- Creates a unique seat ID to prevent double-booking
- Creates both `UniqueSeatBooking` and `Booking` records
- Redirects to the booking confirmation page

3.1.3 confirm_booking View

The `confirm_booking` view displays booking details for user confirmation:

Needs the user to be logged in

Uses the `unique_seat_id` parameter to:

- Find the specific booking record.
- Retrieve all related entities (seat, tier, show, event, screen, theatre, city).

Assembles all booking details into a context dictionary.

3.1.4 register View

The `register` view handles user registration:

When a `POST` request is received:

- Collects user input (username, email, phone, passwords).
- Validates inputs with basic checks (required fields, password match).
- Checks for existing usernames and emails to prevent duplicates.

User Creation: If validation passes:

- Creates a new Django `User` object.
- Creates an associated `Customer` profile with additional information.
- Saves phone number as required customer data.

Redirects to the login page after successful registration.

3.1.5 user_login View

The `user_login` view handles user authentication:

Form Processing: When a `POST` request is received:

- Extracts username and password from the request.
- Attempts to authenticate the user with the provided credentials.

User Login: If authentication succeeds:

- Logs in the user using Django's `login` function.
- Redirects to the `book_ticket` page.

This view manages the user login process, providing access to authenticated users and handling failed login attempts.

3.1.6 after_login_redirect View

- Requires user authentication (via `@login_required` decorator).
- Logs session data for debugging purposes.
- Redirects user to the city selection page.

3.1.7 city_selection View

- Retrieves all cities from the database.
- Renders the city selection template with the list of cities.
- Provides an interface for users to choose their city.

3.1.8 set_city View

- Processes city selection form submission.
- Validates that a city was selected.
- Stores the selected city ID in the user's session.
- Redirects to the ticket booking page after successful selection.
- Returns to city selection if no city was chosen.

3.1.9 get_events View

- Retrieves events for a specific theatre.
- Filters events based on the provided `theatre_id`.
- Returns a JSON response with event IDs and names.

3.1.10 get_shows View

- Fetches showtimes for a specific theatre, event, and date.
- Filters showtimes based on `theatre_id`, `event_id`, and date.
- Returns a JSON response with show IDs and formatted times.

3.1.11 get_seats View

- Retrieves available seats for a specific tier and show.
- Excludes already booked seats for the given date.
- Returns a JSON response with available seat IDs and numbers.

3.1.12 get_tiers View

- Fetches tiers for a specific show.
- Retrieves the screen associated with the show.
- Returns a JSON response with tier IDs, names, and prices.

3.1.13 `initiate_payment` View

- Requires user authentication (via `@login_required` decorator).
- Processes payment initiation for a specific booking.
- Retrieves the unique seat booking using the provided `unique_seat_id`.
- Fetches the associated booking record for the current user.
- Creates a new transaction record with **pending** status.
- Renders the payment page with transaction, booking, and unique seat ID.
- Includes comprehensive error handling for various failure scenarios.
- Redirects to `booking_failed` page with appropriate error messages if issues occur.

3.1.14 `process_payment` View

This view handles the payment processing for a booking:

- **OTP Validation:**
 - Checks if the entered OTP matches the hardcoded value "356473".
 - Redirects to `booking_failed` if OTP is invalid.
- **Booking Retrieval:**
 - Fetches the `UniqueSeatBooking` and associated `Booking` using `unique_seat_id`.
 - Handles potential exceptions if bookings are not found.
- **Transaction Creation:**
 - Creates a new `Transaction` record with **success** status.
 - Sets a dummy amount and payment method.
- **Booking Confirmation:**
 - Updates the booking status to `CONFIRMED`.
 - Saves the updated booking.
- **Success Handling:**
 - Redirects to `booking_success` page with the booking ID.
- **Error Handling:**
 - Catches and logs various exceptions.
 - Redirects to `booking_failed` with appropriate error messages.

3.1.15 `booking_success` View

- Requires user authentication (`@login_required`).
- Retrieves confirmed booking for the current user.
- Fetches associated `UniqueSeatBooking` record.
- Gathers all related booking information (event, show, seat, tier, city, screen, theatre).
- Renders booking confirmation page with comprehensive booking details.
- Includes extensive error handling with appropriate redirects and messages.

3.1.16 booking_failed View

- Simple view that renders a booking failure page.
- Displays a generic error message encouraging users to try again.

3.2 models.py

3.2.1 City

- Represents cities where events take place.
- **Fields:** `city_name`.

3.2.2 Theatre

- Represents theatres in cities.
- **Fields:** `city` (FK), `theatre_name`, `loc`, `total_screens`.

3.2.3 Screen

- Represents screens within theatres.
- **Fields:** `theatre` (FK), `screen_name`, `total_seats`.

3.2.4 Organizer

- Represents event organizers.
- **Fields:** `organizer_name`.

3.2.5 Event

- Represents events (movies or concerts).
- **Fields:** `event_name`, `event_type`, `duration`, `lang`, `organizer` (FK).

3.2.6 Showtime

- Represents specific showings of events.
- **Fields:** `event` (FK), `screen` (FK), `show_date`, `slot_time`, `available_seats`.

3.2.7 Tier

- Represents seating tiers in screens.
- **Fields:** `screen` (FK), `tier_name`, `price`.

3.2.8 Seat

- Represents individual seats.
- **Fields:** `seat_number`, `tier` (FK).

3.2.9 Customer

- Represents customers with user accounts.
- **Fields:** `user` (OneToOne), `customer_name`, `email_id`, `phone_no`.

3.2.10 Booking

- Represents event bookings.
- **Fields:** `customer` (FK), `show` (FK), `seat` (FK), `booking_date`, `status`.

3.2.11 Transaction

- Represents payment transactions for bookings.
- **Fields:** `transaction_id`, `booking` (FK), `amount_paid`, `payment_method`, `payment_status`, `transaction_date`.

3.2.12 Cancellation

- Represents booking cancellations.
- **Fields:** `booking` (FK), `cancel_date`, `refund_status`.

3.2.13 Admin

- Represents admin users.
- **Fields:** `user` (OneToOne), `admin_role`.

3.2.14 Date

- Represents show dates.
- **Fields:** `show_date`.

3.2.15 UniqueSeatBooking

- Represents unique seat bookings.
- **Fields:** `unique_seat_id`, `city_id`, `screen_id`, `event_id`, `show_id`, `date`, `seat_id`, `tier_id`.

3.3 urls.py

3.3.1 URL Patterns

- `/` - Home page (`views.index`)
- `/book/` - Ticket booking page (`views.book_ticket`)
- `/register/` - User registration (`views.register`)
- `/login/` - User login (Django's built-in `LoginView`)
- `/logout/` - User logout (`views.user_logout`)
- `/select-city/` - City selection page (`views.city_selection`)
- `/after-login/` - Post-login redirect (`views.after_login_redirect`)
- `/set-city/` - Process city selection (`views.set_city`)

3.3.2 API Endpoints

- `/get-events/` - Fetch events for a theatre (`views.get_events`)
- `/get-shows/` - Fetch showtimes for an event (`views.get_shows`)
- `/get-seats/` - Fetch available seats (`views.get_seats`)
- `/get-tiers/` - Fetch pricing tiers (`views.get_tiers`)

3.3.3 Booking Flow

- `/book/confirm/` - Confirm booking (`views.confirm_booking`)
- `/book/confirm/download/` - Download booking PDF (`views.download_booking_pdf`)
- `/book/payment/` - Initiate payment (`views.initiate_payment`)
- `/book/process-payment/` - Process payment (`views.process_payment`)
- `/booking-success/` - Booking success page (`views.booking_success`)
- `/booking-failed/` - Booking failure page (`views.booking_failed`)

4 Frontend

5 HTML templates

5.0.1 register.html

This HTML template displays a user registration page for **Booket Ticking** with:

- Split-screen design (welcome message on the left, form on the right)
- Registration form collecting:
 - Username
 - Email
 - Phone
 - Password
- Error message display capability
- Login link for existing users
- Animated gradient background and modern styling

5.0.2 login.html

This HTML template displays a login page for a **BookMyShow Clone** with:

- Centered login form with username and password fields
- Dark gradient animated background
- Error message display capability via Django's message framework
- Registration link for new users
- Modern styling with Bootstrap 5.3.0
- Responsive design with custom CSS animations

5.0.3 `book_ticket.html`

This HTML template displays a multi-step ticket booking interface with:

- Sequential booking flow through 6 steps:
 - Theatre selection
 - Event selection
 - Date selection
 - Show time selection
 - Tier/price selection
 - Seat selection
- Interactive UI with collapsible sections and "back" navigation
- AJAX-powered dynamic content loading for each step
- Visual seat map with availability indicators
- Form submission with hidden fields to capture all selections
- Modern styling with Bootstrap and custom CSS
- Screen indicator in the seat selection step

5.0.4 `confirm_booking.html`

This HTML template displays a booking confirmation page that:

- Extends a base template.
- Shows a comprehensive summary of the booking details organized in four sections:
 - **Event Details:** Name, date, show time.
 - **Venue Details:** City, theatre.
 - **Seat Details:** Seat number, tier, price.
 - **Booking Reference:** Unique booking ID.
- Provides two action buttons:
 - **"Proceed to Pay"** button that links to the payment initiation page.
 - **"Return to Home"** button that links back to the homepage.
- Includes custom CSS styling for a clean, organized layout with:
 - Responsive grid layout for details sections.
 - Styled section boxes with shadows and borders.
 - Prominent action buttons.
 - Consistent spacing and typography.

5.0.5 `payment_page.html`

This HTML template displays a payment page for completing a ticket booking transaction with:

- Clean payment interface with animated green gradient background.
- Payment details section showing booking ID and amount to pay.
- OTP verification form for secure payment confirmation.
- Client-side validation that checks for the correct OTP (356473).
- Action buttons for confirming or canceling the payment.
- Responsive design with Bootstrap 5.3.0 styling.
- JavaScript functionality that automatically displays the OTP in an alert when the page loads.
- Form submission to the `process_payment` view with the `unique_seat_id` parameter.

5.0.6 `booking_success.html`

This HTML template displays a booking success page that:

- Extends a base template.
- Shows a confirmation message .
- Presents key booking details in a clean, centered container:
 - Event name.
 - Show time.
 - Seat number.
 - City.
 - Booking ID.
- Features a "Book Another Ticket" button that links back to the booking page.
- Uses custom styling with:
 - Green background for success indication.
 - Responsive container with shadow.
 - Consistent typography and spacing.
 - Blue action button with hover effect.

5.1 CSS Templates

5.1.1 `city_selection.css`

This CSS creates a full-screen modal for city selection with:

- Animated gradient background
- Centered white content box
- Search input field
- Scrollable grid of city options
- Hover effects on city items
- Custom-styled scrollbar
- Responsive design elements

The layout uses flexbox and grid for alignment, with transitions and animations for a smooth user experience.