Jeremy Tan’s Project Design

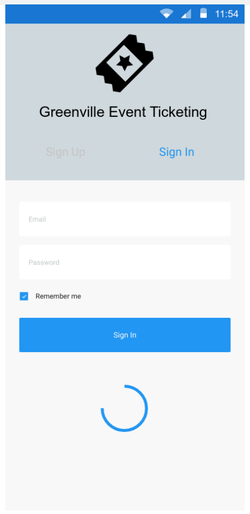
Table of Contents

Screen Design

Login Page

Default Page of App
Verifies username and password with server
Navigate to next page if validation successful

API call: open_gate()
Foreground



Profile Page

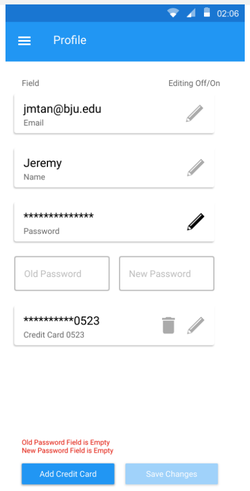
Retrieves following stored data on sever

- Email
- Name
- Credit Card No
- Password Placeholder

Functionality

- Change name
- Add, edit, remove credit cards
- Change password
  - Enter old password
  - Enter new password
- Verify Old password for new password to be updated
- (Concurrency) Prevent simultaneous editing in case two people are logged into the same account

API Call: get_userprofile()
Foreground

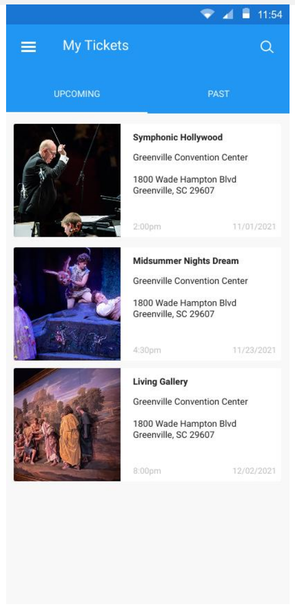


My Tickets Page

Displays list of bought tickets (sever-stored)

Navigate to specified ticket from selected ticket.

API Call: get_usertickets()
Foreground



Ticket Detail Page

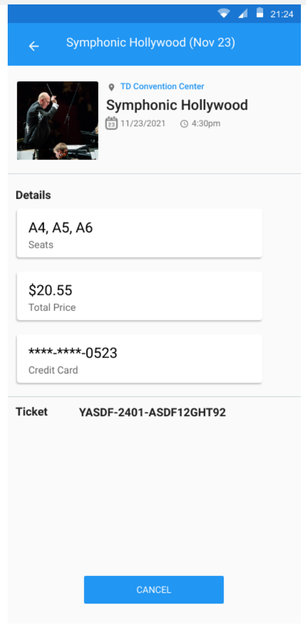
Shows the following sever-stored information:

- Cinema & Address
- Date
- Begin Time
- ~~Auditorium~~
- Seat(s) Chosen
- Credit Card Used
- Ticket ID & Barcode
- Price
- Cancel Button

Funcitonality

- Cancel ticket if 24 hours minimum before showing
- Release hold on seats if cancellation successful
- (Concurrency) make sure that entry still exists in case two people logged into the same account cancel the same ticket simultaneously

API Call: get_userticketdetail()
Foreground



Events List Page
Displays list of available and non-expired event showings

Navigate to event detail page from selected Event

API Call: get_eventlist()
Foreground

Seating Chart Page
Displays the following information

- Selected Event Title
- Theater
- Show Date and Time
- Seating Diagram
    - Available Seats
    - Reserved Seats
    - Chosen Seats
    - Selected Seats
- Confirm and Checkout Button

Validation Checks

- Check if seats are still available

Change label of 'Buy Tickets' to 'Buy More Tickets' if user already bought tickets for that Event showing

API Call: get_eventseating()
Foreground

Order Verification Page
Display the following information:

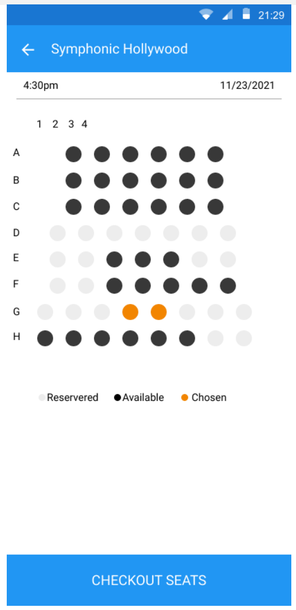
- Selected Event Title
- Theater & Address
- Selected Seat(s)
- Total Price
- Selected Credit Card to buy tickets
- Confirm Purchase Button

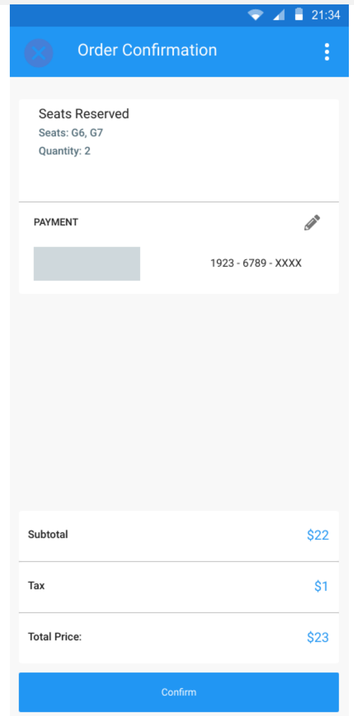
Validation Checks

- (Concurrency) Prevent two simultaneous orders of the exact same seat(s) by processing request in a lock
- (Concurrency) Process checkout request to prevent duplicating the request and get successful validation of order by assinging idempotentcy token

API Call: hold_eventseatbyuser(),
Background
Error handling: Show mini activity-indicator and grey out 'confirm button' until app receives confirmation from the api call hold_eventseatbyuser().

API Call: reserve_eventseatbyuser()
Foreground





Server API

Route	API Calls	Input Values	Processing Work	Return values
"users/#userid#" POST	open_gate() *Non-idempotent* Creates a timed session token for each login.	str username str password	Retrieve & parse dictionary request Validate credential entries against db Send back confirmation message Or send back rejection message Generate session token with timer	bool success/failure int sessionid int response code str error_messsage
	close_gate() *Local Server Operation* Destroys any expired session token.	N/A	Check any expired timers Delete respective session tokens	N/A
	handle_error() *Local Server Operation* Handles all error message formatting	Exception e	Format purposely-thrown errors Format unhandled exceptions	int error code str error_message

Route	API Calls	Input Values	Processing Work	Return values
"/users/##userid##/tickets/" POST	get_usertickets() *Idempotent* Allows clients to retrieve and view list of purchased tickets	sessionid	Retrieve & parse dictionary request Check if session-id is still valid Check if session-id is non-expired Retrieve ticket collection info w/ user-id Return array of ticket-collection defined in a dictionary	array TicketCollection [0] str event-name [1] str address [2] start time [3] start date [4] png thumbnail int response code str error_messsage

Route	API Calls	Input Values	Processing Work	Return values
"/users/##userid##/tickets/##ticketid##" POST / DELETE	get_userticketdetail() *Idempotent* Allows clients to retrieve and view details of chosen ticket	sessionid ticketid	Retrieve & parse dictionary request Check if session-id is still valid Check if session-id is non-expired Retrieve ticket detail info w/ ticket-id Return ticket details defined in a dictionary	str event-name str address str start time str start date png thumbnail str seat assignments int price-paid int last4creditcardnum int response code str error_message
	cancel_userticket() *Non-idempotent* Cancels (deletes) and refund selected ticket by the user.	sessionid ticketid	Retrieve & parse dictionary request Check if session-id is still valid Check if session-id is non-expired Check if chosen ticket is not expired Check if chosen ticket still has >24 hours before event If pass, then cancel ticket and send back confirmation If fail, then send back error message	int response code str error_message

Route	API Calls	Input Values	Processing Work	Return values
"/users/##userid##/" GET / PATCH	get_userprofile() *Idempotent* Allows clients to view their user profile	userid sessionid	Retrieve & parse dictionary request Check if session-id is still valid Check if session-id is non-expired Retrieve user info w/ user-id Return user info defined in a dictionary	str email str name str password??? str credit-card-no int response code str error_message
	edit_userprofile() *Non-idempotent* Allows clients to update changed fields in their profile	userid sessionid bool edit_name str newname bool edit_creditcard str newcreditcard	Retrieve & parse dictionary request Determine what fields to edit Validate new credit-card value Replace old number with new number Replace old name with new name	int responsecode str error_message

Route	API Calls	Input Values	Processing Work	Return values
"/eventlist" GET	get_eventlist() *Idempotent* Return list of events to client	N/A	Retrieve list of events Parse info into array of dict Return list of events defined in an [] of dictionaries	array EventCollection [0] str event-name [1] str start-date [2] str start-time [3] png thumbnail
	update_eventlist() *Local Server Operation* 'Deletes' (catalogs) all expired events	N/A	Runs once a day Change status of expired events	N/A

Route	API Calls	Input Values	Processing Work	Return values
"/eventlist/##eventid##/seating" GET	get_eventseating() *Idempotent* Return seating chart of selected event	eventid	Select event from eventid Retrieve seating info Parse seating info as an array[][] Return seating info	array[][] seats int0 = available int1 = unavailable

Route	**API Calls**	**Input Values**	**Processing Work**	**Return values**
"/eventlist/##eventid##/seating/confirm" *POST*	hold_eventseatbyuser() *Non-Idempotent* Change status of chosen seats to 'hold' (not yet bought but still blocked-off to others) if conditions are met	userid sessionid eventid array[] chosen-seats int seat-id-1 int seat-id-2 int seat-id-…	Retrieve & parse dictionary request Start Lock Check if requested seats are taken If available, hold seats (change seats status) Start Hold Timer	bool success/failure str feedback
	reserve_eventseatbyuser() *Non-Idempotent* Changes status of chosen seats to 'reserved' if conditions are met	userid sessionid eventid array[] chosen-seats int seat-id-1 int seat-id-2 int seat-id-… credit-card info	Retrieve & parse dictionary request Start Lock Start Timer Check if hold seats belong to user If yes, then reserve seats (change seat status) Scrap Hold Timer (if not expired)	bool success/failure str feedback int ticketid (if successful)
	check_holdtimers() *Local Server Operation* Releases back seats on hold if their respective timers are expired	N/A	If hold-timers exist, then run periodically For every cycle, check for expired timers Delete expired timers if found	N/A
	update_holdtimerbyuser() *Non-Idempotent* Postpones time limit of held seats if client presses the 'non-idle' prompt	userid bool userpostpone	Retrieve & parse dictionary request Check if timer has not exceeded postponement limit If not, then extend timer and send confirmation If yes, then send failure indicator	bool success/failure

Object Model

(Server) User Table	Instance Variables	Storage
Collection of all registered user profiles	ID Email Username Password Credit Card Ticket Collection	SQLAlchemy

(Server) Ticket Table	Instance Variables	Storage
Collection of all tickets	ID UserID EventID Seats Reserved Price Paid	SQLAlchemy

(Server) Seat Object	Instance Variables	Storage
Entity that represents each seat	ID Row # Col # Status	SQLAlchemy

(Server) Login Session	Instance Variables	Storage
Session Token for each logged in user	SessionID UserID Time-Start Time-End	SQLAlchemy

(Server) Event Table	Instance Variables	Storage
Collection of all events	ID Name Day Time Duration Price Selected Seats Collection	SQLAlchemy

(Server) Seating Table	Instance Variables	Storage
Collection of seating charts for each event	ID EventID Seat Collection	SQLAlchemy

(Server) Ticket Object	Instance Variables	Storage
Entity that represents each ticket	ID UserID EventID ChosenSeats PricePaid	SQLAlchemy

(Server) Seat-Hold Session	Instance Variables	Storage
Session token for each seat(s) placed on hold	SessionID Time-Start Time-End	SQLAlchemy

Grading Levels

80 LEVEL FEATURES	
CLIENT	Primitive outline of navigation pages of client-side app Be able to navigate from events list page to order ticket confirmation page • Query and receive list of events via event-list page • Query and receive details of chosen event (using event ID) • Query and receive available seating for chosen event via seating-chart page • Send data about chosen seating via order verification page • Send confirmation about reserving seating Be able to change name and credit card information via profile page
SERVER	Base foundation of routes and objects to store Be able to receive JSON requests and append data. Validation and concurrency handling will not be the main focus • Change the status of the chosen seats in the collection to their appropriate status (available, reserved, not-available) • Hold (block) off seats that the user(s) choose when navigating to the order confirmation page Validate that email is unique and not already existing within the database of registered emails. • Send error / rejection message if there is already a pre-existing email • Send confirmation / acceptance message if server side registration is successful
NOTES	Core Functionality of server and client is the focus Exception and concurrency handling is not the focus of the 80 level design

90 LEVEL FEATURES	
CLIENT	Online/offline indicators: • Error banner messages indicating offline status • Restoration indicator when app gets back internet connection Allow the user to cancel a ticket if the event has not already started or expired • Replace the cancel button with 'Expired' label if the event is already expired (if the current time exceeds the start time plus the event duration) • Send query about the cancellation and receive confirmation of cancellation from the server Client-side validation • Providing instant error feedback when user leaves entry fields blank when signing up/registering for something
SERVER	Perform 2nd layer of validation on server-side to prevent malformed and malicious calls to server API (sql injection) via CURL commands When the user is on the seating chart page … • Process logic that holds the seats in a lock • Send appropriate feedback response if client attempts to hold seat that is already held by another client When client-side users hold seats via order verification page… • process logic that reserves the seats in a lock • start timer to hold seats • renew seats when user hits 'not idle' pop-up box
NOTES	Exception and Concurrency Handling is the focus of this level

100 LEVEL FEATURES	
CLIENT	Be able to create an account and receive confirmation about successful account creation via Sign-up page
SERVER	Handle requests to hold and reserve seats with assigning idempotency tokens

BONUS FEATURES	
REAL LIFE USABILITY	Offline functionality • Remain logged in (if the user selects 'remember me' checkbox in the Login Page • Display the most recently retrieved list of events stored in a cache • When navigated to the seating chart page of the app, show the most recent seating chart obtained from the server **but** grey out and block the user from selecting seats Using the PayPal API call from Cps 404, be able to register valid test credit card numbers When booking seats, have server be able to make PayPal API calls to subtract the balance of the test account Have the app send the user's current location to the server and have the server calculate the distance between the user's location and venue's location (in miles) During registration of email, instead of only checking for existing emails of the entered email, have an email sent out to the entered email. Users must click URL link to confirm validity of email. Publish the app on both the Google Play Store and Apple App Store

Road Map

Deliverables		
Deliverable 1	Client: Have the following navigation pages working (at a bare-bone level excluding graphics and ui design) • Login Page • Profile Page • My Tickets Page • Ticket Detail Page • Event List Page Server: Have the following API calls working • get_userprofile() • get_usertickets() • get_userticketdetail() • get_eventlist() **Exception and concurrency handling will be ignored at this stage** **Session tokens/timers are also ignored at this stage**	September 16
Deliverable 2	Client: Have the following navigation pages working in addition to the previous ones: • Seating Chart Page • Order Verification Page In addition, each navigation page should demonstrate client-side exception handling and appropriate reporting Server: Have the following API calls working in addition to the previous ones: • get_eventseating() • hold_eventseatbyuser() • reserve_eventseatbyuser() In addition, have exception reporting appropriately formatted and handled. All concurrency handling should be done	December 02
Deliverable 3	Server: Have the following features working in addition to the previous deliverables: • Session-token/timers • Session maintenance / repeated methods	December 09

Time Log (Nov 4, 2021)

Date	Description	Time Spent
Oct 25	Project Proposal Draft	2.5 hours
Oct 26	Project Proposal Revisions	0. 5 hours
Nov 3	Project Design: Grading Rubric Breakdown	1 hours
Nov 4	Proj Design: Obj Model Design, Screen Design, Server API	4 hours
Total	8 hours	