UML Diagrams for Use Cases

1.1: Create an Account

	Action performed by the actor		**Responses from the system**
1.	Client inputs an email address and a password (twice for confirmation). They then click the “Create Account” button.		
		2a.	If the email address is valid, then a new account is created for them on the database.
		2b.	If not, an invalid error message is given, and they are re-prompted to re-enter their information.

Create An Account
- email address - password
+validateInput(username, password) +createNewAccount(username, password)

1.2: Log In

	Action performed by the actor		**Responses from the system**
1.	Client inputs their username and password and clicks “Log In”		
		2a.	If the username and password are found on the database, the user is sent to the user page.
		2b.	If no username and password are found, an invalid login

			error message is given, and they are re-prompted to re-enter their information.

Log In
- password - username
+validateInput(username, password)

1.3: Manage Account

1.4: Manage Reservations + Bookings

	Action performed by the actor		**Responses from the system**
1.	Client follows the identity flow up to point 5 at which time a username and auth is provided to the ItineraryManager		
		2a.	Checks whether or not the user has data cached on the client side, if they do then it reloads that into the state, otherwise it checks if there's an itinerary stored in the database. If one is the it similarly reloads the state
		2b.	If no data is found then the ItineraryManager initializes an array of regular itineraries and a coordinatedItinerary which the user is assigned to and can recursively store itineraries by group and event.
3.	Client makes a reservation via the ItineraryFlow		
		4	ItineraryManager collects the data and updates the user's locale storage, if the users logged in then the updates are pushed to the database. Any changes made to the client's itinerary state while the flow is in an active state are observed by the Manager which asynchronously performs updates in the background while the client uses the interface.

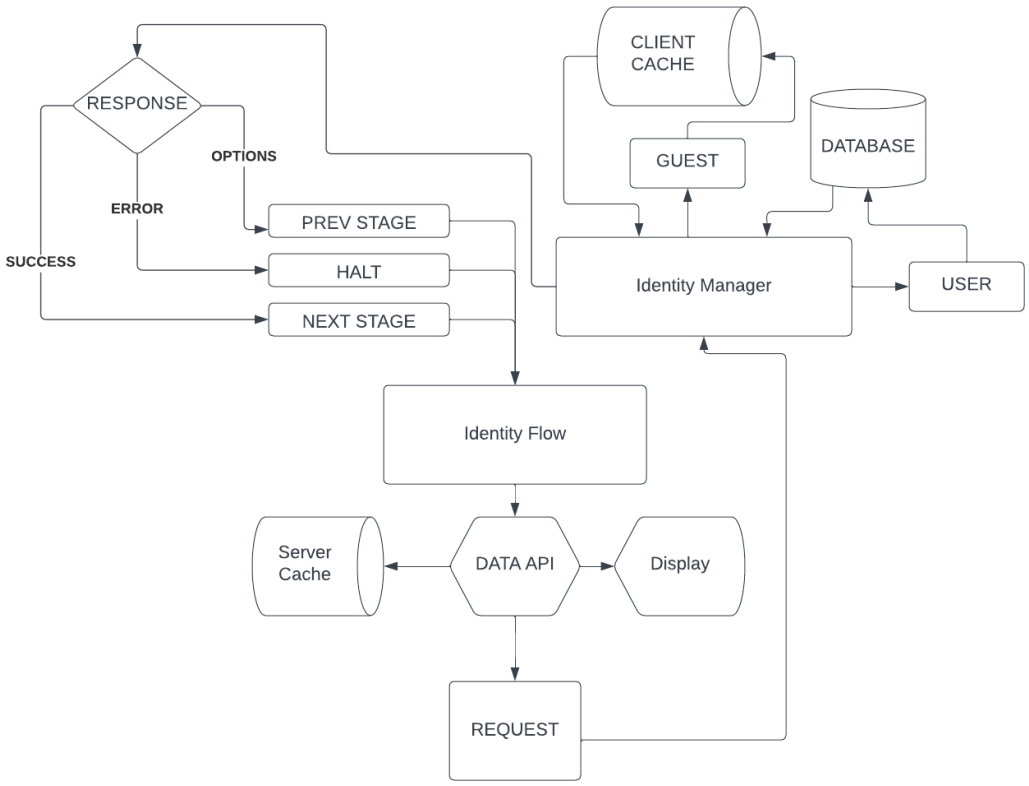
ItineraryManager (Partial)
- state: State *// Itinerary state (the data associated with the Itinerary)* - coordinated: CoordinatedItinerary *//recursive itinerary storage with additional group functionality* - storage: StorageMethod *//the located storage method (cache/database)*
+ createItinerary():Itinerary *// creates a new itinerary* + updateItinerary(intinerary:Itinerary):void *// updates an itinerary* + removeItinerary(itinerary:Itinerary):void *// removes an itinerary from a coordinated itinerary* + getCurrentEvent():Event|null *// get the current active event* + getNextEvent():Event|null *// gets the next booked event* + getEvents():Event[] *// gets all of the available events* + getTransportation(types:[“Uber”|”Lyft”|”Hertz”|...]) *// gets the available transportation methods for the current event* + getStartTime():Date *//gets the start time of the current event* + getEndTime():Date *//gets the end time of the current event* + transitionToNextEvent():void *//transitions the internal state to the next event* + transitionToPrevEvent():void *//transitions the internal state to the previous event* + cancelEvent(event:Event):void *//cancels an event already booked* + addNotifier(notifier:Notifier):void *//adds a notifier (discord, slack, twitter, etc…) to the itinerary* + findUserInDatabase(username:string) : Itinerary|null // *find the user in the database* + reloadUserItineraryFromDatabase(): Itinerary|null// *reloads state into the ItineraryManager from database* + reloadUserItineraryFromCache(): Itinerary|null// *reloads state into the ItineraryManager from cache* + getStorageMethod(type:”local”|”database”):StorageMethod //*auto locates and applies storage method on flow init*

1.5: Generate Itinerary

	Action performed by the actor		**Responses from the system**
1.	client opens the website		
		2a.	Flow starts stage 1 and checks for any previous data stored in cookies, if found the the data is reloaded and user is requested if they want to continue or restart -> 3a
		2b.	Flow fails to find any data and initializes a fresh state and is prompted if they want to create a new itinerary -> 3b
$3a_1$	User clicks ‘Continue’	$4a_1$	Flow is resumed from the cached state found from either the database or their client-side cache -> …
$3a_2$	User clicks ‘Restart’	$4a_2$	Skip to 4b.
$3b_1$	Client Clicks ‘Yes’	$4b_1$	Flow is initiated from fresh state -> skip to 5
$3b_2$	Client Clicks ‘No’	$4b_2$	Flow initiates ItineraryManager in a guest state with no storage options and only event search functionality
		5.	IdentityManager is initialized -> Flow returns available functionality to the user
6.	User makes a reservation (see below)		
		7.	Itinerary Flow initializes the Reservation Flow and updates the Manager
This process continues during the entire user experience with different types of stages			

Itinerary Flow (Partial)
- state: State // ItineraryFlow state (the data associated with the stage) - stage: FlowState //current flow stage (login, registration, etc…) - user: { authorization: “Bearer …”, username: <username>}

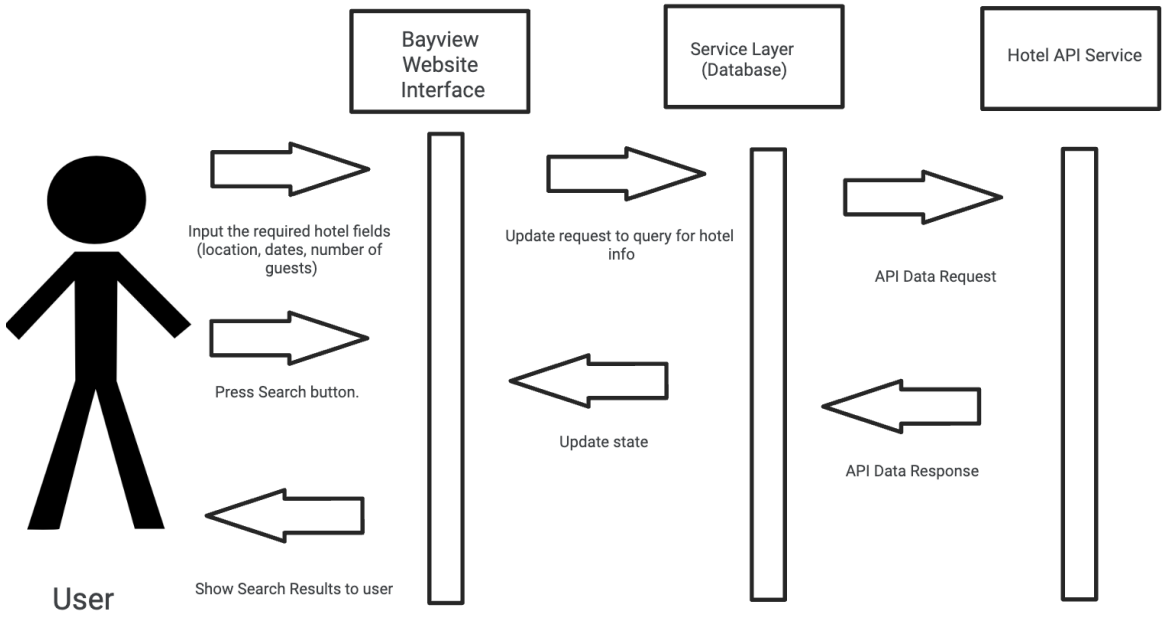
+ createItineraryFlow(user:User) //*initializes the itinerary flow*
+ advanceItineraryFlow() //*state contained within the cache*
+ registerUser(username:string, password:string) //*registers a user*
+ loginUser(auth:string, username:string) //*logs the user in*
+ createIdentity(username:string) //*creates an identity from the bearer auth*
+ assignIdentity(username:string) //*assigns the identity and tracks the user*
+ refreshIdentity(tok:string) //*refreshes the identity from the auth token*
+ getItinerary(username:string) //*gets the itinerary from the Itinerary Manager*



2.2: Search for Hotels

	Action performed by the actor		Responses from the system
1.	Click the “Find Hotels” button after inputting required fields		
		2.	Present a relevant list of hotels with links to book with appropriate third party services, updating the presented itinerary

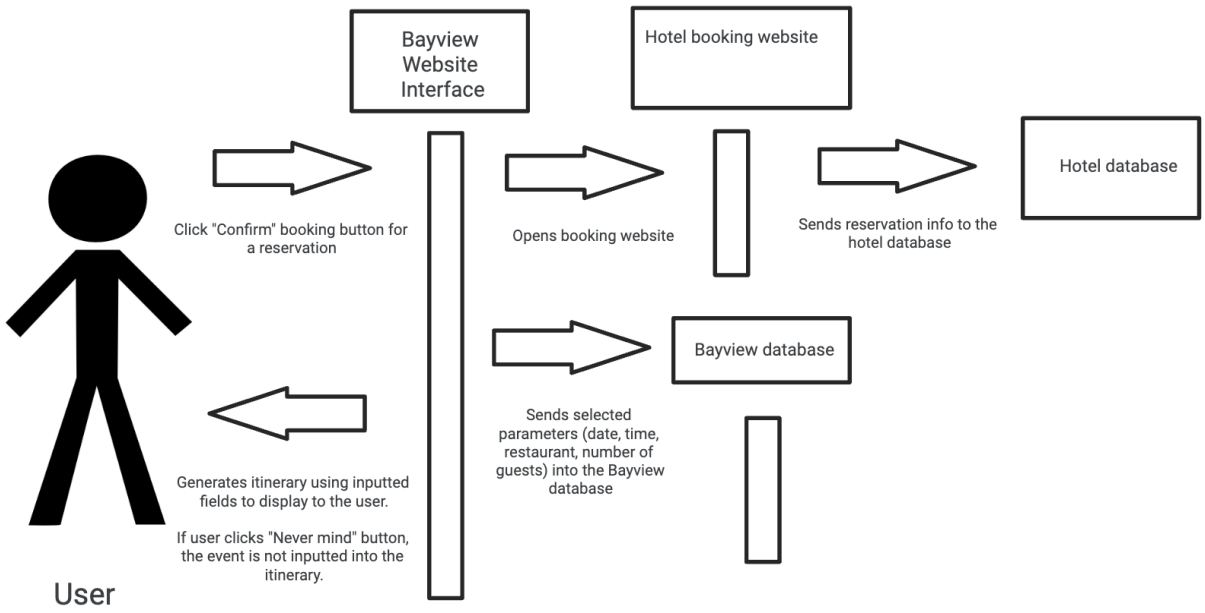
Hotel Reservation
- hotels: hotelsList
+ searchAvailableHotels (String arrival, String departure, String fromDate, String toDate) + getHotels() //fetches hotels from API and returns to user + handleResponse(type:Hotel, payload:Event):Response //Handle User click



2.2: Book Hotel

	Action performed by the actor		Responses from the system
1.	User requests reservation data from the API and follows the itinerary flow. The user is provided with a reservation url.		
2.	Client clicks on the “confirm reservation url”		
3.	User is redirected to the reservation website in a secondary window.	3.	The ItineraryManager assumes the user is going to complete the reservation and asynchronously adds the event to the itinerary while the user confirms the reservation
		4.	The Website provides the user a “nevermind” button in the background. If the user chooses not the confirm the reservation, they can click this button
5.	User clicks “nevermind”		
		6.	The ItineraryManager removes the event from the expected itineraries and returns the user back to the norm itinerary flow by popping the event off the list and restoring the state

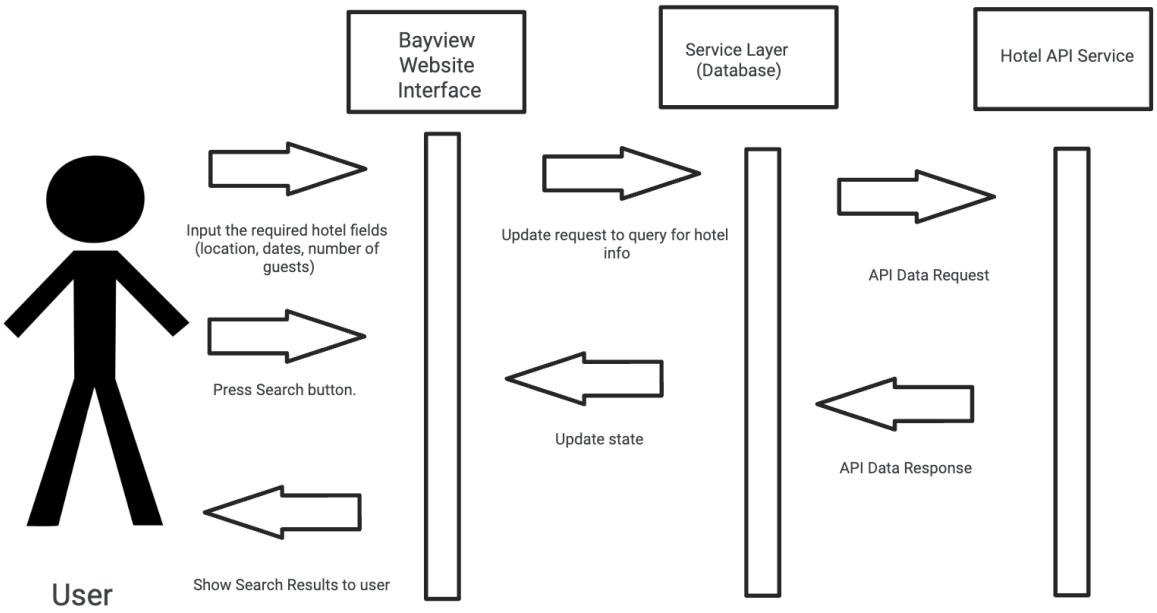
Hotel Reservation
- reservation - hotels: hotelsList
+ displayReservation(reservation) + updateItinerary(hotelsList)



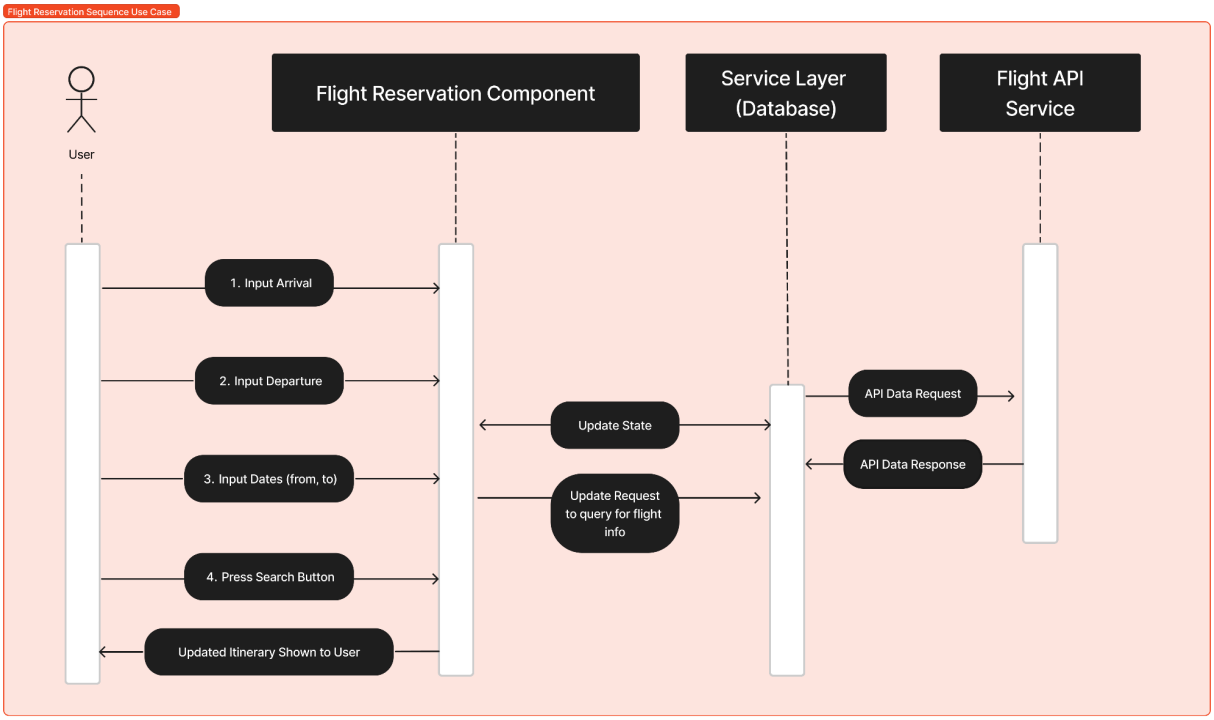
2.3: Search for Flights

	Action performed by the actor		**Responses from the system**
1.	Click the “Find Flights” button after inputting required fields		
		2.	Present a relevant list of flights with links to book with appropriate third party services, updating the presented itinerary

Flight Reservation
- flights: flightsList - reservation
+ searchAvailableFlights (String arrival, String departure, String fromDate, String toDate) + getFlights() //fetches flights from API and returns to user + handleResponse(type:Flight, payload:Event):Response //Handle User click



2.4: Book Flight



	Action performed by the actor		Responses from the system

1.	User requests reservation data from the API and follows the itinerary flow. The user is provided with a reservation url.		
2.	Client clicks on the “confirm reservation url”		
3.	User is redirected to the reservation website in a secondary window.	3.	The ItineraryManager assumes the user is going to complete the reservation and asynchronously adds the event to the itinerary while the user confirms the reservation
		4.	The Website provides the user a “nevermind” button in the background. If the user chooses not the confirm the reservation, they can click this button
5.	User clicks “nevermind”		
		6.	The ItineraryManager removes the event from the expected itineraries and returns the user back to the norm itinerary flow by popping the event off the list and restoring the state

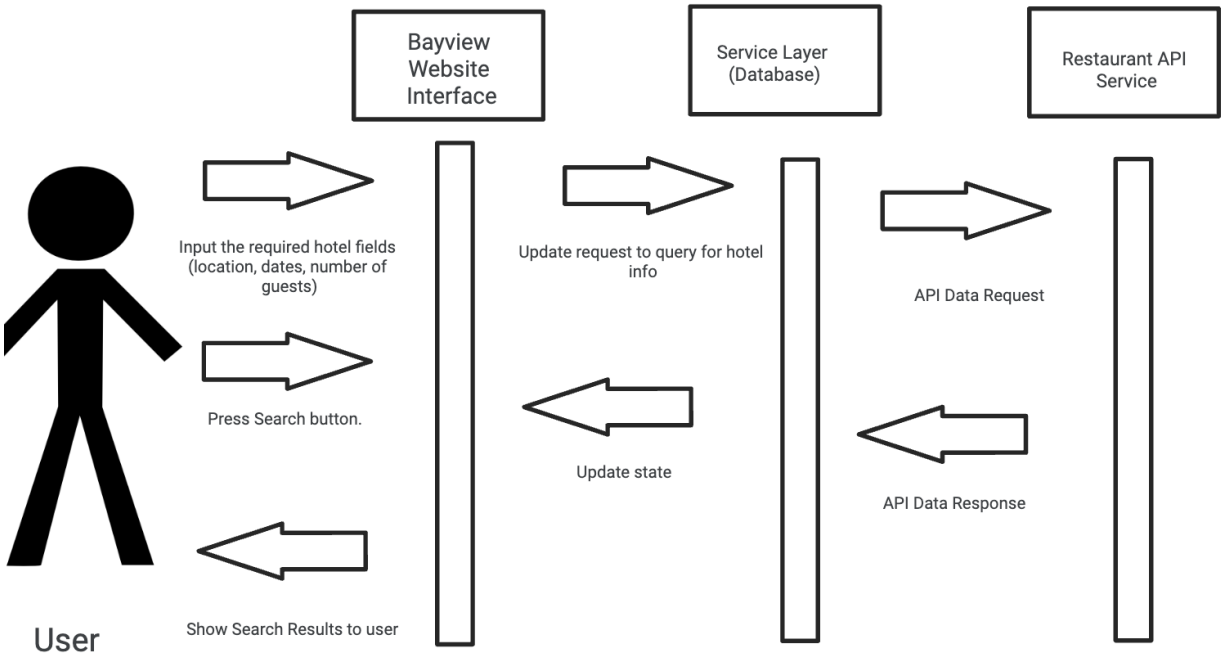
Flight Reservation
- flights: flightsList
+ searchAvailableFlights (String arrival, String departure, String fromDate, String toDate) + displayReservation(reservation) + updateItinerary(flightsList)

2.5: Search for Restaurants

	Action performed by the actor		**Responses from the system**
1.	Click the “Find Restaurants” button after inputting required fields		
		2.	Present a relevant list of restaurants with a link to book through the official Disney

			website, updating the presented itinerary.

Restaurant Reservation
- restaurants: restaurantsList
+ searchAvailableRestaurants (String Restaurant, String date) + showAvailableRestaurants(String date)

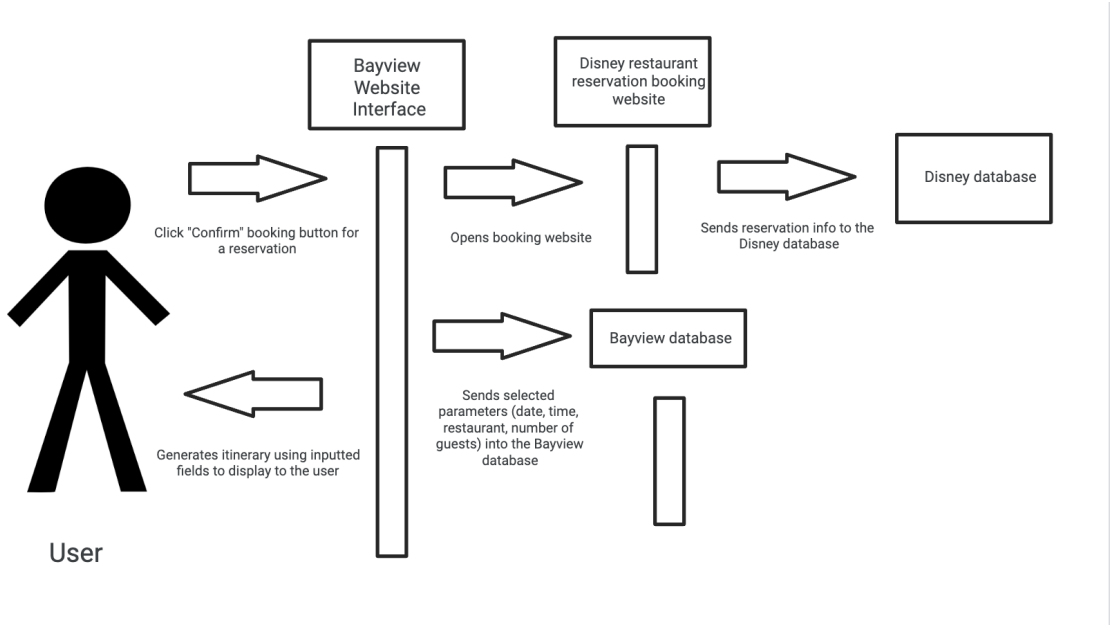


2.6: Book Restaurant

	Action performed by the actor		Responses from the system
1.	User requests reservation data from the API and follows the itinerary flow. The user is provided with a reservation url.		
2.	Client clicks on the “confirm reservation url”		
3.	User is redirected to the reservation website in a secondary window.	3.	The ItineraryManager assumes the user is going to complete the reservation and asynchronously adds the

			event to the itinerary while the user confirms the reservation
		4.	The Website provides the user a “nevermind” button in the background. If the user chooses not the confirm the reservation, they can click this button
5.	User clicks “nevermind”		
		6.	The ItineraryManager removes the event from the expected itineraries and returns the user back to the norm itinerary flow by popping the event off the list and restoring the state

Restaurant Reservation
- restaurants: reservationsList - reservation
+ displayReservation(reservation) + updateItinerary(flightsList)



2.7: Data Collection API

	Action performed by the actor		**Responses from the system**
			Website initializes a websocket connection and authenticates through disney's oauth platform
2.	User requests dining reservation availability, pass prices, hotel reservations, etc..		
		3.	API gathers requested information and parses irrelevant data out
		4.	Response is formatted and provided back to the client
5.	Client selects the available reservation date they want	5.	Server asynchronously caches the data based on the provided cookie expiration times and associates it with similar searches for later use.
		6.	Server requests previously cached data made from the same host and collects the necessary url and provides it to the client
7.	If the client is not logged in then the data is cached client-side and a request is made to the ItineraryManager to begin the Itinerary Flow, if they are logged in then the Itinerary Flow was initiated from the API, otherwise nothing.	7.	If the client is logged in then the server sends an itinerary request to the ItinteraryManager and begins the Itinerary Flow
		8.	At this point the Itinerary Manager is initialized with a user's previous itinerary if they have one stored in the database, OR the previously cached itinerary on the client side if one is available. Otherwise the ItineraryManager is initialized to use the Database if logged in, or the client's localStorage cache otherwise.