

5/2/2023

CS 360 - Software engineering

Team 5 - Daniel Snyder, Priyank, Celeste Hendry, Ivie Xue, Andre Centlivre

Client - Sweetwater Music Center

# Room and Gear Booking System 2.0

## Final Documentation

**Table of Contents:**

Part 1 - Introduction	2
Introduction	2
Purpose	2
Goals and Objectives	2
Constraints	3
Part 2 - Glossary	4
Part 3 - Design Models	5
Use Case Diagram	5
Class Diagram	6
Sequence Diagram	7
State Diagram	8
Part 4 - Important Modules	9
Summary	9
Detailed Description	9
Part 5 - Demonstration	21
Part 6 - Testing	32
Part 7 - Developer Tools and Read Me Document	38
Part 8 - Code Design and Standards	40
Part 9 - Member Contributions	41
Part 10 - References	41

## **Part 1 - Introduction:**

### **Introduction**

Our team had the opportunity to work on enhancing an event scheduling and room booking tool that was being used at the Sweetwater Music Center. The tool made use of an Airtable database to keep track of user, gear, and event data, which we visualized and interacted with through a React.js app. We managed source control using Github and deployed the site with Netlify. Our main task was to expand the booking system's capabilities to include not just rooms, but also gear bookings. Before we stepped in, users could see available gear, but could not book it reliably in advance due to the lack of availability validation logic.

### **Purpose**

The main goal of our recently completed project was to broaden the scope of an event scheduling and room booking tool that was actively being used at the Sweetwater Music Center. The tool, which used Airtable as a database for storing user, gear, and event data, was visualized and interacted with through a React.js app. We were successful in implementing a comprehensive gear booking system that included availability validation logic, allowing users to book gear in advance without any hiccups.

### **Goals and Objectives**

Our primary objective for this project was to improve the existing event scheduling and room booking tool at the Sweetwater Music Center by adding a fully-fledged gear booking system. Previously, users could see available gear but couldn't book it reliably in advance because there was no availability validation logic. We built on the existing tool, which used Airtable as a database and was accessed through a React.js app. Our project required us to create a user-friendly interface for booking gear, incorporate availability validation logic to prevent double bookings, and ensure a secure login portal. Additionally, we needed to optimize the system for high-volume requests and quick response times. We used functional and non-functional requirements as our compass to develop a booking system that meets users' needs.

To reach our goals, we gathered user requirements such as booking gear in advance with availability validation logic, an intuitive interface for booking gear and rooms, a system capable of handling high volumes of requests and providing quick response times, and a secure login portal to protect sensitive user data. We came up with a prototype for the improved booking system that included a user-friendly interface for booking gear and rooms, a system for validating gear availability and preventing double bookings, a secure login portal for user authentication, and a scalable system that could adapt to the growing demands of the user base.

On top of that, we were committed to enhancing the app's overall usability and making sure it could manage a high volume of requests from multiple users while maintaining reliability and security. It was important for us to design a system that could grow alongside the needs of an expanding user base.

## Constraints

While working on enhancing the event scheduling and room booking tool for the Sweetwater Music Center, we faced several constraints that impacted our approach and decision-making process:

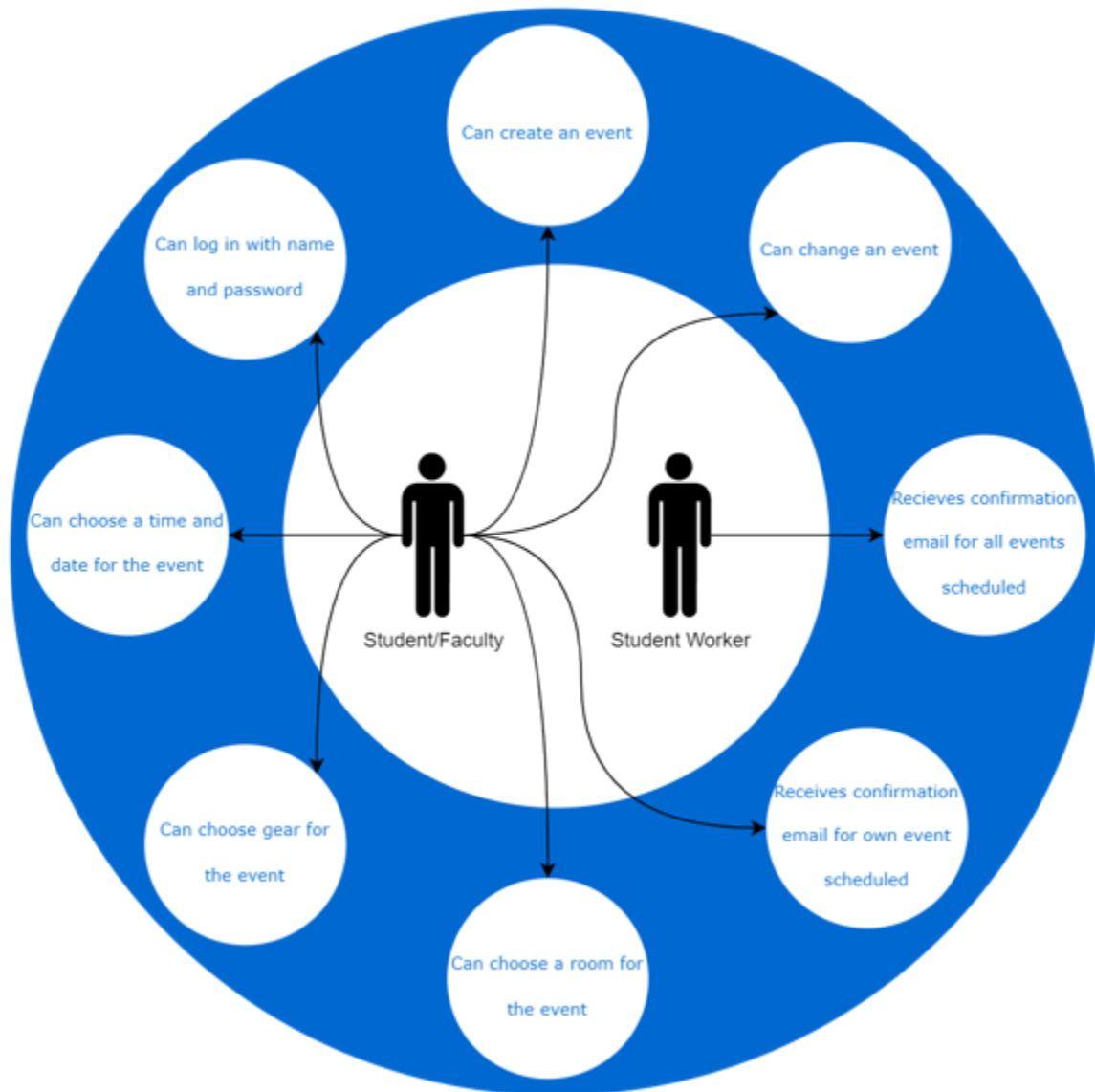
- Time: We had a strict deadline to meet, which pushed us to efficiently allocate resources and prioritize tasks to achieve the project goals within the given timeframe.
- Budget: Our budget was limited, so we had to come up with cost-effective solutions and prioritize features based on their importance and impact on the user experience.
- Technical Limitations: The existing technology stack, including Airtable and React.js, imposed certain limitations on the implementation of new features, so we had to find compatible solutions that worked well with the current setup.

## **Part 2 - Glossary:**

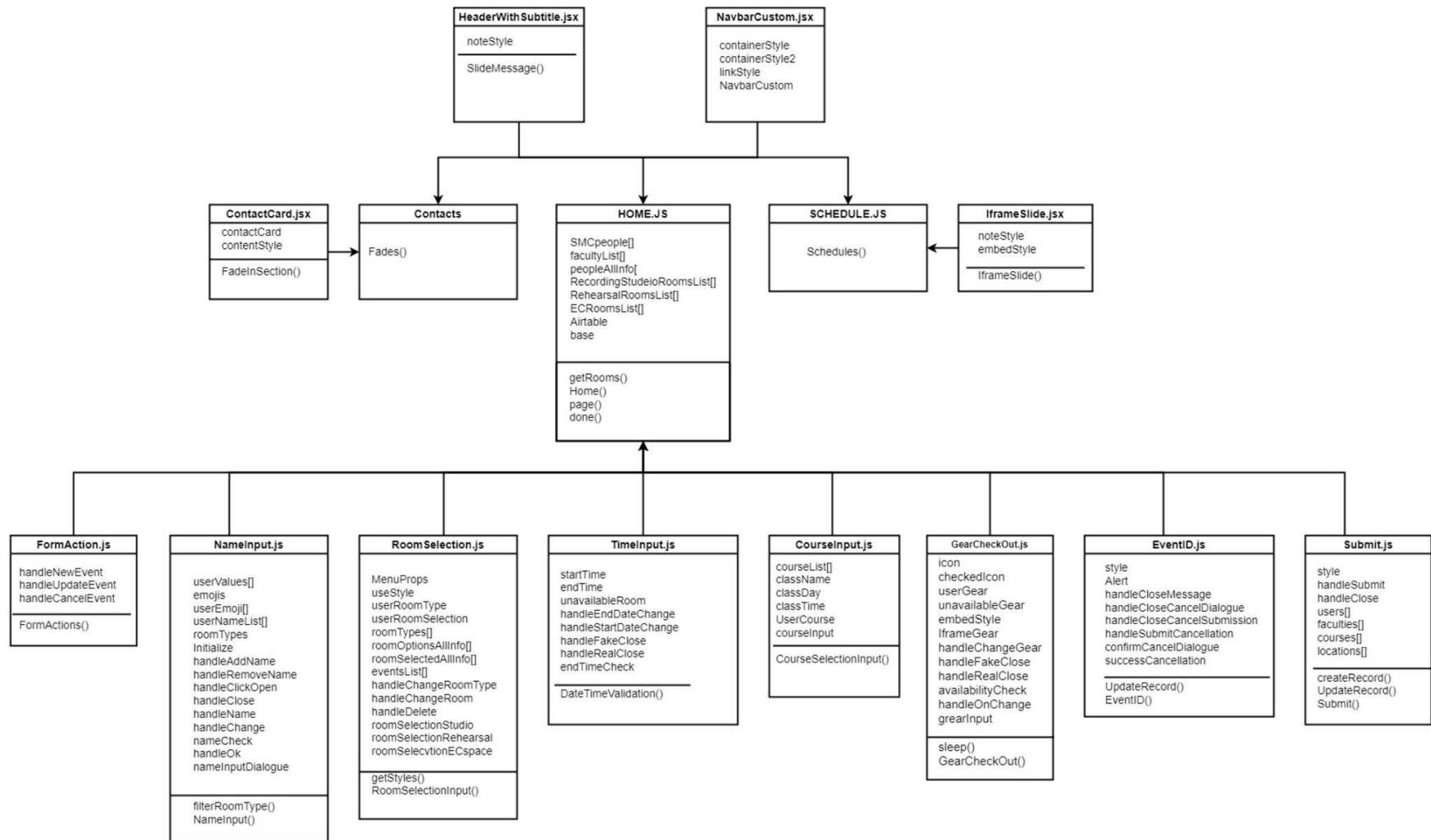
- SMC: Sweetwater Music Center.
- User Interface: The parts of a program that users can interact with, including buttons, menus, and forms.
- User Experience: The overall experience that a user has while interacting with a program, including ease of use and satisfaction.
- Prototype: A working model of a program that is used to test the user interface and user experience before the final product is released.
- Accessibility: The design of a program so that it is usable by people with disabilities, such as hearing impairments.
- Usability Testing: The process of testing a program with real users to evaluate its ease of use, efficiency, and effectiveness.
- Responsive Design: The design of a program to adapt to different screen sizes and devices.
- Component: In React, components are independent and reusable bits of code. You can think of them as classes.
- Airtable: Airtable is used to create databases that hold data for users to access, store, and organize.
- Front-end: The front-end of a program or project is the part that the user can see and includes the code necessary for how the webpage looks and how users can access it.
- Back-end: The back-end of a program is how the program works and functions and includes things that are not accessible to the user, such as the system, logic, and data.
- Javascript: Javascript is a front-end programming language.
- React.js: React is a front-end javascript framework.
- Github: Github is a platform that hosts code and programs. It is accessible from anywhere and allows multiple people to work on a project.
- Netlify: Netlify is a platform used to host websites.
- Gear: As used in this document, refers to audio and video equipment that is available for use by students and faculty at the Sweetwater Music Center.

## Part 3 - Design Models:

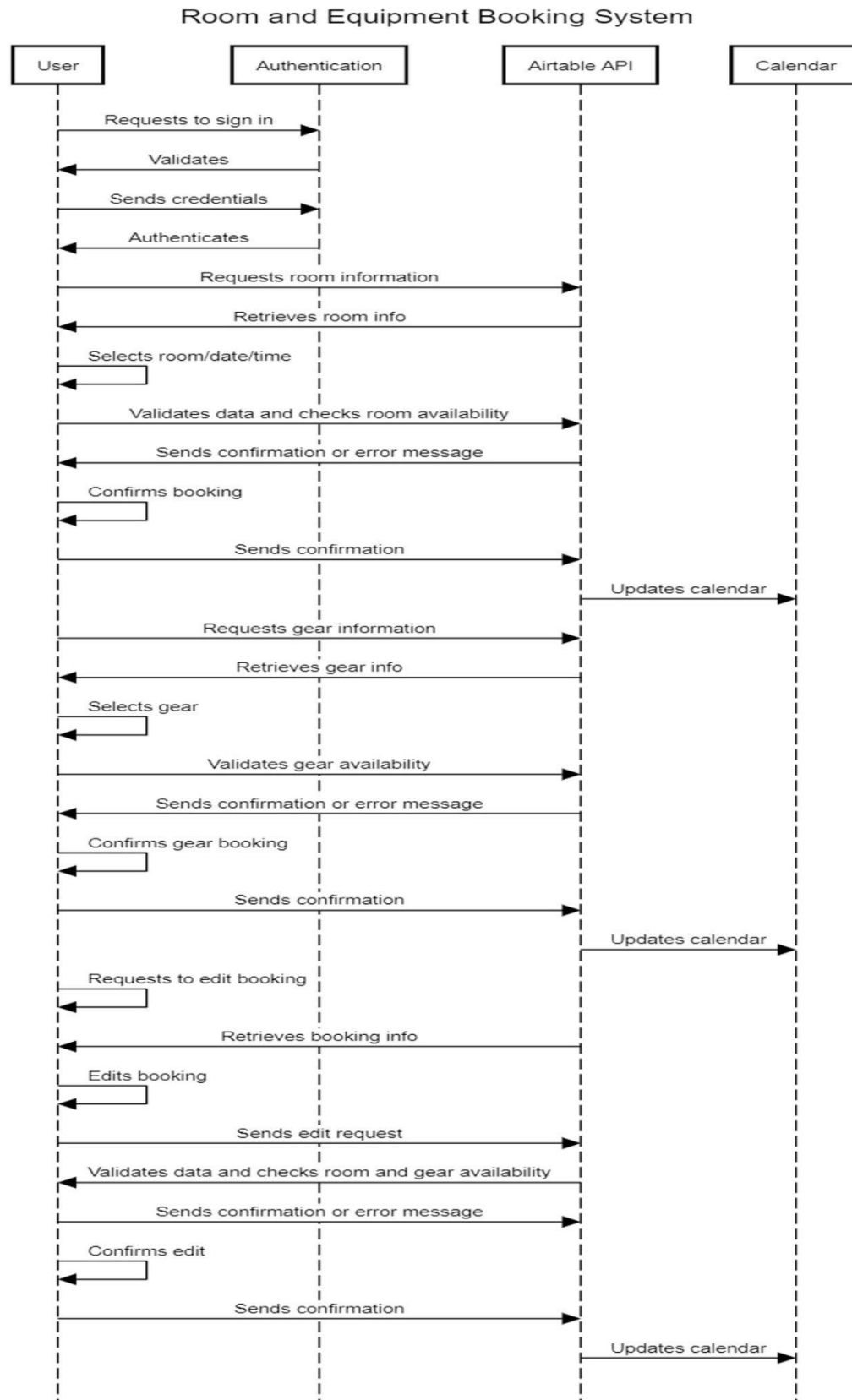
### Use Case Diagram



## Class Diagram

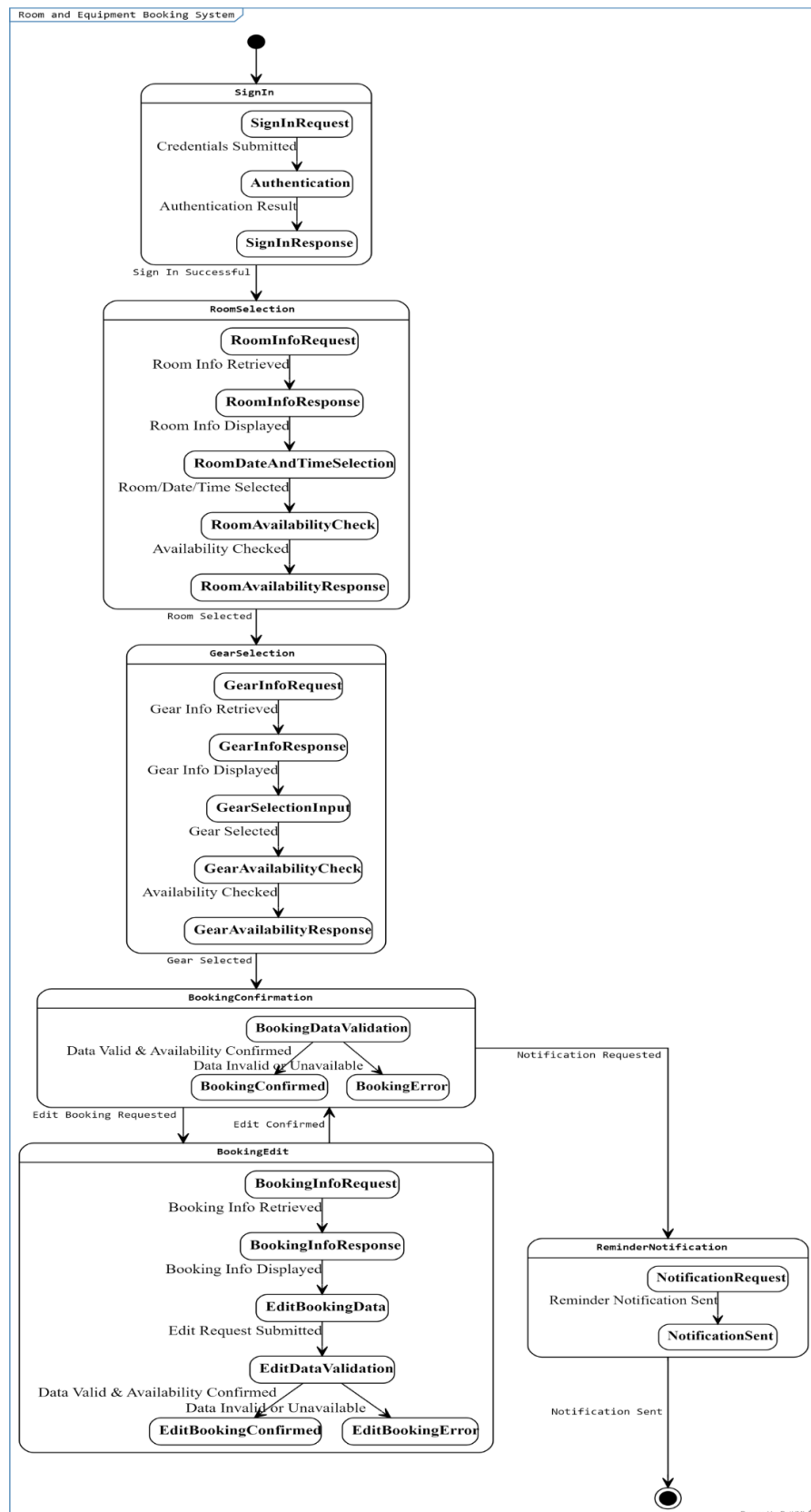


## Sequence Diagram





# State Diagram



## **Part 4 - Important Modules:**

### **Summary:**

1. Pages
  - a. Contact
  - b. Schedule
  - c. Index
  - d. Home
  - e. Gear
  - f. Document
  - g. App
2. Components
  - a. ContactCard
  - b. CourseInput
  - c. EventDetailsInput
  - d. EventID
  - e. FormActions
  - f. GearCheckOut
  - g. HeaderWithSubtitles
  - h. IFrameSlid
  - i. NameInput
  - j. Navbar
  - k. RoomSelection
  - l. Submit
  - m. TimeInput
  - n. airtable

### **Detailed Description:**

#### **1. ContactCard**

- a. Parameters:
  - i. type: string - The type of contact information being displayed. It can be one of the following: "website", "email", "phone", or "sms".
  - ii. url: string - The URL of the website to be opened when the user clicks on the card. Only applicable if type is "website".

- iii. email: string - The email address to be used when the user clicks on the card. Only applicable if type is "email".
  - iv. subject: string - The subject of the email to be sent when the user clicks on the card. Only applicable if type is "email".
  - v. body: string - The body of the email to be sent when the user clicks on the card. Only applicable if type is "email".
  - vi. phone: string - The phone number to be used when the user clicks on the card. Only applicable if type is "phone" or "sms".
  - vii. iconSrc: string - The source URL of the icon to be displayed on the card.
  - viii. title: string - The title of the contact information being displayed.
  - ix. description: string - The description of the contact information being displayed.
  - x. ...rest: object - Any additional props to be passed to the component.
- b. handleClick()
- i. Functionality: Handles the click event on the contact card and performs the appropriate action based on the type of contact information being displayed.
  - ii. Postcondition: The appropriate action will be performed based on the type of contact information being displayed.
  - iii. Returns: void
- c. Returns:
- i. JSX - A styled contact card component that displays the provided contact information.

## 2. CourseSelectionInput



- a. Parameters:
- i. courseSelected: array - An array of objects representing the courses selected by the user. Each object contains the following properties:
    1. key: string - The ID of the course in the Airtable database.
    2. name: string - The display text for the course, which includes the course name, day(s) of the week, and meeting time.
  - ii. setCourseSelected: function - A function that updates the selected course array.
  - iii. addCourse: boolean - A flag that indicates whether the current time slot is for a course assignment.
  - iv. setAddCourse: function - A function that updates the addCourse flag.
- b. fetchCourses()
- i. Functionality: Fetches data of all classes from Airtable and extracts course name, day(s) of the week, and meeting time. Returns an array of objects representing the courses.
  - ii. Postcondition: An array of objects representing the courses will be returned.
  - iii. Returns: Promise that resolves to an array of objects.

- c. `handleCourseAssignmentChange()`
  - i. Parameters: `event` - The event object passed to the function when the checkbox is toggled.
  - ii. Functionality: Updates the `addCourse` flag based on the checkbox value.
  - iii. Postcondition: The `addCourse` flag will be updated.
  - iv. Returns: `void`
- d. `handleCourseSelectionChange()`
  - i. Parameters: `event` - The event object passed to the function when the user selects a course, and `newCourses` - An array of objects representing the courses selected by the user.
  - ii. Functionality: Updates the selected course array based on the courses selected by the user.
  - iii. Postcondition: The selected course array will be updated.
  - iv. Returns: `void`
- e. `renderOption()`
  - i. Parameters: `props` - The props object passed to the function when an option is rendered, `option` - The option object to be rendered, and `{ selected }` - An object containing the selected option.
  - ii. Functionality: Renders the list item for each option in the Autocomplete component.
  - iii. Postcondition: The list item will be rendered for each option in the Autocomplete component.
  - iv. Returns: `JSX` - A styled list item for the option.
- f. `renderInput()`
  - i. Parameters: `params` - The params object passed to the function when the input field is rendered.
  - ii. Functionality: Renders the input field for the Autocomplete component.
  - iii. Postcondition: The input field will be rendered for the Autocomplete component.
  - iv. Returns: `JSX` - A styled input field for the Autocomplete component.
- g. Returns:
  - i. `JSX` - A styled `CourseSelectionInput` component that allows the user to select one or more courses for a given time slot. The component includes a checkbox that indicates whether the current time slot is for a course assignment, and an Autocomplete component that allows the user to search and select courses.

### 3. `EventDetailsInput`

- a. `setSessionTitle`
  - i. Parameters: `string`
  - ii. Functionality: Sets the title of the session.
  - iii. Postcondition: The session title will be set to the string passed as a parameter.

- iv. Returns: void
  - b. setTypeSelected
    - i. Parameters: string
    - ii. Functionality: Sets the selected event type.
    - iii. Postcondition: The selected event type will be set to the string passed as a parameter.
    - iv. Returns: void
  - c. setUsageSelected
    - i. Parameters: string
    - ii. Functionality: Sets the selected event usage.
    - iii. Postcondition: The selected event usage will be set to the string passed as a parameter.
    - iv. Returns: void
  - d. handleEventTypeSelect
    - i. Parameters: event
    - ii. Functionality: Handles the event when an event type is selected.
    - iii. Postcondition: The selectedEventType state will be updated with the value of the selected event type and the eventTypeSelected callback will be called with the same value.
    - iv. Returns: void
  - e. handleEventUsageSelect
    - i. Parameters: event
    - ii. Functionality: Handles the event when an event usage is selected.
    - iii. Postcondition: The selectedEventUsage state will be updated with the value of the selected event usage and the usageSelected callback will be called with the same value.
    - iv. Returns: void
- 4. EventID**
- a. handleCheckID()
    - i. Parameters: None
    - ii. Functionality: Queries the database to check if the given event ID exists in the "Events" table.
    - iii. Postcondition: The IDError and GoodID states will be updated based on the result of the database query. If the updateEvent or CancelEvent props are true, the SuccessMsg state or the openCancelDialog state will be updated accordingly.
    - iv. Returns: void
  - b. handleCloseMessage()
    - i. Parameters: event, reason
    - ii. Functionality: Handles the closing of the success message Snackbar by setting the SuccessMsg state to false.

- iii. Postcondition: The SuccessMsg state will be updated to false.
  - iv. Returns: void
- c. handleCloseCancelDialog()
  - i. Parameters: None
  - ii. Functionality: Handles the closing of the cancel confirmation dialog by setting the openCancelDialog state to false.
  - iii. Postcondition: The openCancelDialog state will be updated to false.
  - iv. Returns: void
- d. handleCloseCancelSubmission()
  - i. Parameters: None
  - ii. Functionality: Handles the closing of the successful cancellation modal by setting the openCancelSuccess state to false.
  - iii. Postcondition: The openCancelSuccess state will be updated to false.
  - iv. Returns: void
- e. handleSubmitCancellation()
  - i. Parameters: None
  - ii. Functionality: Calls the UpdateRecord function to update the status of the event with the given event ID to "Canceled ". It then sets the openCancelDialog state to false and the openCancelSuccess state to true. Finally, it resets the event ID related states to their initial values.
  - iii. Postcondition: The status of the event with the given event ID will be updated to "Canceled  in the "Events" table. The openCancelDialog state will be updated to false and the openCancelSuccess state will be updated to true. The event ID related states will be reset to their initial values.
  - iv. Returns: void
- f. confirmCancelDialog
  - i. Parameters: None
  - ii. Functionality: Returns a Dialog component that serves as a confirmation dialog for cancelling an event. It has two actions: "Yes" and "No". Clicking "Yes" calls the handleSubmitCancellation function, and clicking "No" closes the dialog.
  - iii. Postcondition: A confirmation dialog for canceling an event will be rendered.
  - iv. Returns: Dialog component

## 5. GearCheckOut

- a. GearCheckOut()
  - i. Parameters: { gearSelected, setGearSelected, gearList, addGear, setAddGear }

- ii. Functionality: Renders a form to allow users to select gear to add to their gear list for an event.
  - iii. Postcondition: The form will be rendered and available to the user.
  - iv. Returns: JSX element
- b. `handleGearCheckboxChange()`
  - i. Parameters: event
  - ii. Functionality: Toggles the state of `addGear`.
  - iii. Postcondition: The `addGear` state will be updated.
  - iv. Returns: void
- c. `renderAvailableGearItem()`
  - i. Parameters: { `gearItem`, `handleAddGear` }
  - ii. Functionality: Renders a single gear item that can be selected and added to the gear list.
  - iii. Postcondition: A gear item will be rendered and available to the user.
  - iv. Returns: JSX element
- d. `renderChosenGearItem()`
  - i. Parameters: { `gearItem`, `handleRemoveGear` }
  - ii. Functionality: Renders a single gear item that has already been selected and added to the gear list, along with a delete button to remove it from the list.
  - iii. Postcondition: A gear item will be rendered and available to the user.
  - iv. Returns: JSX element
- e. `AvailableGear()`
  - i. Parameters: { `gearList`, `filterTerm`, `gearSelected`, `setGearSelected` }
  - ii. Functionality: Renders a list of available gear items that can be selected and added to the gear list.
  - iii. Postcondition: A list of available gear items will be rendered and available to the user.
  - iv. Returns: JSX element
- f. `ChosenGear()`
  - i. Parameters: { `gearList`, `gearSelected`, `setGearSelected` }
  - ii. Functionality: Renders the gear list of items that have already been selected and added to the list, along with a delete button to remove them.
  - iii. Postcondition: A list of chosen gear items will be rendered and available to the user.
  - iv. Returns: JSX element

## 6. `SlideMessage`

- a. Parameters: title (string), subtitle (string)
- b. Functionality: Returns a React component that displays a message with a title and subtitle centered on a dark background.
- c. Postcondition: A React component is returned that displays a message with a title and subtitle.

- d. Returns: React component

## 7. IframeSlideComponent

- i. Parameters: {src}
- ii. Functionality: Renders an iframe component that displays content from the specified source.
- iii. Postcondition: An iframe component is rendered with the specified source and styling.
- iv. Returns: JSX code for rendering the iframe component.
- b. React.memo(IframeSlideComponent)
  - i. Parameters: IframeSlideComponent
  - ii. Functionality: Memoizes the IframeSlideComponent so that it only re-renders if the props have changed.
  - iii. Postcondition: IframeSlideComponent is memoized for improved performance.
  - iv. Returns: Memoized IframeSlideComponent.
- c. export default IframeSlide;
  - i. Parameters: None
  - ii. Functionality: Exports the IframeSlide component as the default export for the module.
  - iii. Postcondition: IframeSlide can be imported and used in other modules.
  - iv. Returns: Nothing.

## 8. NameInput

- a. Initialize()
  - i. Parameters: None
  - ii. Functionality: Initializes the userValues and gearList arrays, and sets the roomTypes to an array containing "Recording Studio 🎤", "Rehearsal Spaces 🎧", and "Edit & Collaboration Spaces 🎒". It then filters the gear list and the disabled room types based on the user's gear access and room access.
  - iii. Postcondition: The userValues and gearList arrays are initialized and the roomTypes array is set to contain the appropriate room types based on the user's access level.
- b. handleAddName()
  - i. Parameters: None
  - ii. Functionality: Sets the nameInDisplay state to the full userNameList array.
  - iii. Postcondition: The nameInDisplay state is updated.
- c. handleRemoveName(item)
  - i. Parameters: item (String) - The name of the user to be removed from the list.



- ii. **Functionality:** Removes the specified user from the `userValues` and `userNameList` arrays. It then filters the gear list and the disabled room types based on the user's gear access and room access.
  - iii. **Postcondition:** The specified user is removed from the arrays and the gear list and disabled room types are updated.
- d. `handleClickOpen()`
  - i. **Parameters:** None
  - ii. **Functionality:** Sets the open state to true, displaying the dialog box.
  - iii. **Postcondition:** The open state is set to true, displaying the dialog box.
- e. `handleClose(event, reason)`
  - i. **Parameters:** `event` (Object) - The event that triggered the `handleClose` function, `reason` (String) - The reason for closing the dialog box.
  - ii. **Functionality:** Sets the open state to false, resetting the `value` and `phoneVal` states to null and the `error` state to false.
  - iii. **Postcondition:** The open state is set to false and the `value`, `phoneVal`, and `error` states are reset.
- f. `handleName(event, newValue)`
  - i. **Parameters:** `event` (Object) - The event that triggered the `handleName` function, `newValue` (String/Object/null) - The value of the input field.
  - ii. **Functionality:** If the `newValue` is null, sets the `value` state to null. If the `newValue` is a string and not found in the `peopleAllInfo` array, returns. Otherwise, sets the `value` state to the `newValue`.
  - iii. **Postcondition:** The `value` state is updated.
- g. `handleChange(newValue)`
  - i. **Parameters:** `newValue` (Object) - The selected option from the Autocomplete component.
  - ii. **Functionality:** If the `value` state is null or the user+password combination is incorrect, sets the `passFail` state to true. Otherwise, sets the `passFail` state to false and adds the selected user to the `userValues` and `userNameList` arrays. It then filters the gear list and the disabled room types based on the user's gear access and room access.
  - iii. **Postcondition:** The `passFail` state is set to true or false, the selected user is added to the arrays, and the gear list and disabled room types are updated.
- h. `nameCheck(recordVal, name, number)`
  - i. **Parameters:** `recordVal` (Object) - The selected user object, `name` (String) - The name of the user, `number` (String) - The last 4 digits of the user's phone number.
  - ii. **Functionality:** Checks if the name and last 4 digits of the phone number match the selected user's name and phone number.

- iii. Returns: Boolean - Returns true if the name and last 4 digits of the phone number match the selected user's name and phone number, otherwise false.
- i. `handleOk()`
  - i. Parameters: None
  - ii. Functionality: Calls the `handleChange` function with the current value state.
  - iii. Postcondition: The `handleChange` function is called with the current value state.

## 9. RoomSelectionInput

- a. `handleChangeRoomType()`
  - i. Parameters: event
  - ii. Functionality: Updates the selected room type and sets the appropriate room options.
  - iii. Postcondition: The selected room type and corresponding rooms are set.
  - iv. Returns: void
- b. `handleChangeRoom()`
  - i. Parameters: event
  - ii. Functionality: Updates the selected room and sets the corresponding room information.
  - iii. Postcondition: The selected room and its information are set.
  - iv. Returns: void
- c. `handleDelete()`
  - i. Parameters: event, value
  - ii. Functionality: Deletes a selected room from the user's input.
  - iii. Postcondition: The selected room is removed from the user's input.
  - iv. Returns: void
- d. `roomSelection()`
  - i. Parameters: label, roomOptions
  - ii. Functionality: Renders the room selection dropdown component.
  - iii. Postcondition: The room selection dropdown is rendered with the appropriate options.
  - iv. Returns: JSX element
- e. `useEffect()`
  - i. Parameters: none
  - ii. Functionality: Checks if the room booking record is not empty.
  - iii. Postcondition: If the room booking record is not empty, it updates the component state.
  - iv. Returns: void
- f. `handleDelete()`
  - i. Parameters: e, value
  - ii. Functionality: Deletes a selected room from the user's input.
  - iii. Postcondition: The selected room is removed from the user's input.
  - iv. Returns: void

- g. RoomSelectionInput()
  - i. Parameters: roomOptionStudio, roomOptionRehearsal, roomOptionECspace, disabledRoomTypes, setRoomTypeSelected, setRoomSelected, roomBookingRecord, setRoomBookingRecord
  - ii. Functionality: Renders the room selection input component.
  - iii. Postcondition: The room selection input component is rendered with the appropriate options and state is updated according to the user's selection.
  - iv. Returns: JSX element.
- 10. createRecord**
  - i. Parameters:
    - ii. - fields: Object - an object containing the fields to be added to a new record in the "Events" table in Airtable
  - iii. Functionality: Creates a new record in the "Events" table in Airtable with the provided fields.
  - iv. Postcondition: A new record is added to the "Events" table in Airtable with the provided fields.
  - v. Returns: void
- b. Function: updateRecord
  - i. Parameters:
    - ii. - eventId: string - the ID of the event record in the "Events" table in Airtable to be updated
    - iii. - fields: Object - an object containing the fields to be updated in the specified event record in the "Events" table in Airtable
  - iv. Functionality: Updates the specified event record in the "Events" table in Airtable with the provided fields.
  - v. Postcondition: The specified event record in the "Events" table in Airtable is updated with the provided fields.
  - vi. Returns: void
- c. Component: Submit
  - i. Props:
    - 1. - userSelected: array - an array of objects representing users selected for the event
    - 2. - sessionTitle: string - the title of the event session
    - 3. - eventTypeSelected: string - the type of event session selected
    - 4. - facultySelected: array - an array of objects representing faculty members selected for the event
    - 5. - usageSelected: string - the intended usage of the event session

- 6. - roomTypeSelected: string - the type of room selected for the event
- 7. - roomSelected: string - the name of the room selected for the event
- 8. - startTimeSelected: string - the start time of the event session
- 9. - endTimeSelected: string - the end time of the event session
- 10. - courseSelected: array - an array of objects representing courses selected for the event
- 11. - gearSelected: array - an array of objects representing gear selected for the event
- 12. - eventID: string - the ID of the event record in the "Events" table in Airtable to be updated (if applicable)
- 13. - newEvent: boolean - a boolean indicating whether this is a new event to be created or an existing event to be updated
- 14. - updateEvent: boolean - a boolean indicating whether this is an existing event to be updated (if applicable)
- 15. - setCancelEvent: function - a function to cancel the event
- 16. - timeCorrect: boolean - a boolean indicating whether the event times are valid
- 17. - setUserCount: function - a function to set the number of users selected for the event
- 18. - setAddCourse: function - a function to add a course to the event
- 19. - setAddGear: function - a function to add gear to the event
- 20. - roomBookingRecord: array - an array of objects representing room booking records for the event
- ii. Functionality: Submits the event session information to Airtable by calling createRecord or updateRecord functions based on whether it is a new event or an existing event respectively. Displays a success message in a modal after the submission is complete.
- iii. Postcondition: The event session information is submitted to Airtable and a success message is displayed in a modal.
- iv. Returns: JSX element.

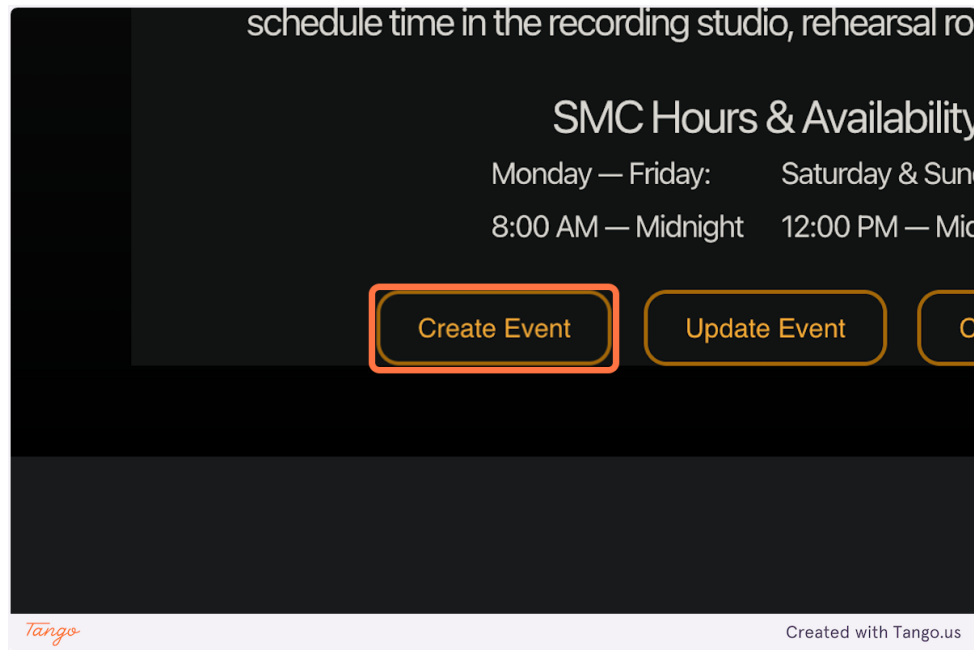
## 11. DateTimeValidation

- a. useTimeSelection()
  - i. Parameters: none
  - ii. Functionality: Custom hook that initializes and manages the start and end date states
  - iii. Postcondition: Returns an object with the start and end date values and a function to handle date changes
  - iv. Returns: Object with the following properties:
    - v. - startDate: Date object representing the selected start date
    - vi. - endDate: Date object representing the selected end date

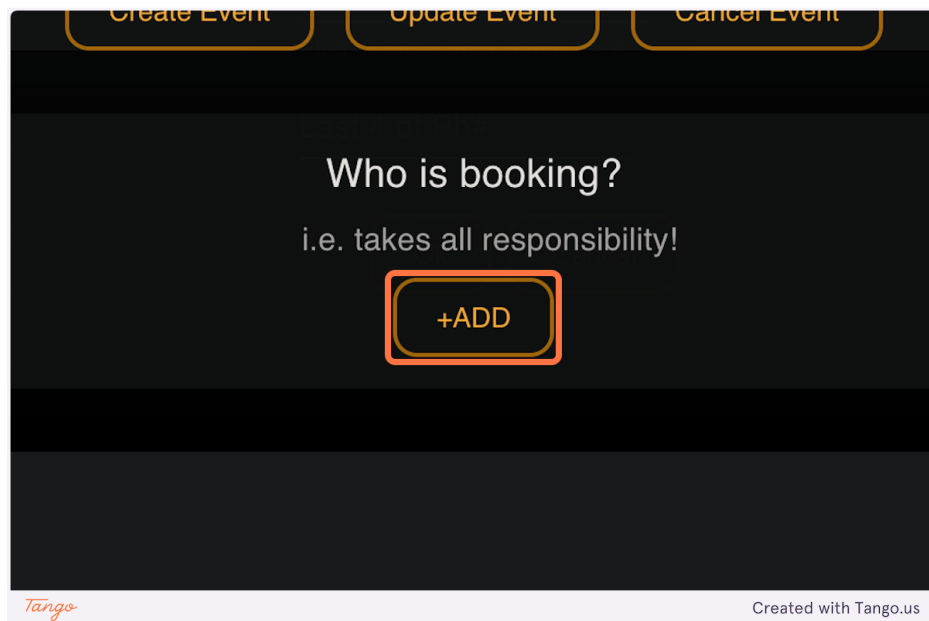
- vii. - `handleDateChange()`: Function to handle date changes
- b. `roundToNearestHalfHour()`
  - i. Parameters: Date object
  - ii. Functionality: Rounds the minutes of a given date object to the nearest half hour
  - iii. Postcondition: Returns a date object with rounded minutes
  - iv. Returns: Date object
- c. `addHours()`
  - i. Parameters: Date object, integer representing number of hours to add
  - ii. Functionality: Adds a given number of hours to a given date object
  - iii. Postcondition: Returns a date object with added hours
  - iv. Returns: Date object
- d. `filterTemporallyUnavailableGear()`
  - i. Parameters: Array of gear objects, selected start time, selected end time
  - ii. Functionality: Filters an array of gear objects to only include those that are available during the selected time range
  - iii. Postcondition: Returns a filtered array of gear objects
  - iv. Returns: Array of gear objects
- e. `DatePickerSection()`
  - i. Parameters: Object with `startDate` and `endDate` properties, function to handle date changes
  - ii. Functionality: Renders the UI for selecting start and end dates
  - iii. Postcondition: Returns a JSX element
  - iv. Returns: JSX element
- f. `NotificationSection()`
  - i. Parameters: Boolean representing room availability, Boolean representing successful room booking, function to handle close events, string representing unavailable room name
  - ii. Functionality: Renders the UI for displaying notifications about room availability and booking success
  - iii. Postcondition: Returns a JSX element
  - iv. Returns: JSX element
- g. `DateTimeValidation()`
  - i. Parameters: Functions to set correct time selection, start time selection, end time selection, room booking record, array of gear objects, function to set filtered gear list
  - ii. Functionality: Renders the UI for selecting start and end dates and checking room availability, filters gear objects based on selected time range
  - iii. Postcondition: Returns a JSX element
  - iv. Returns: JSX element

## Part 5 - Demonstration:

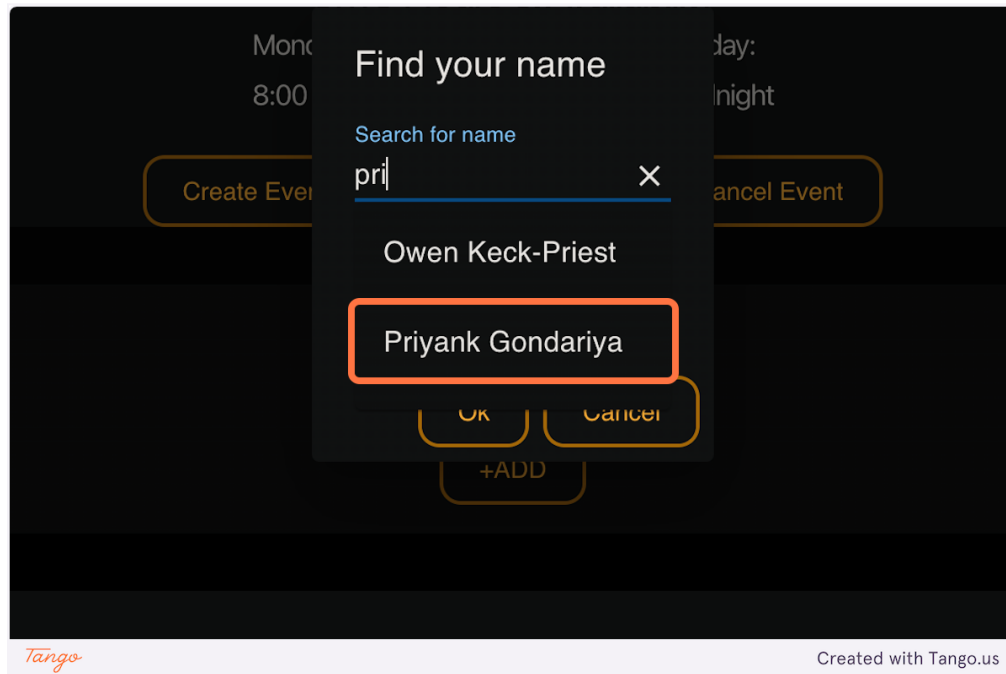
1. Go to [nextp7w.vercel.app](https://nextp7w.vercel.app)
2. Click on *Create Event*



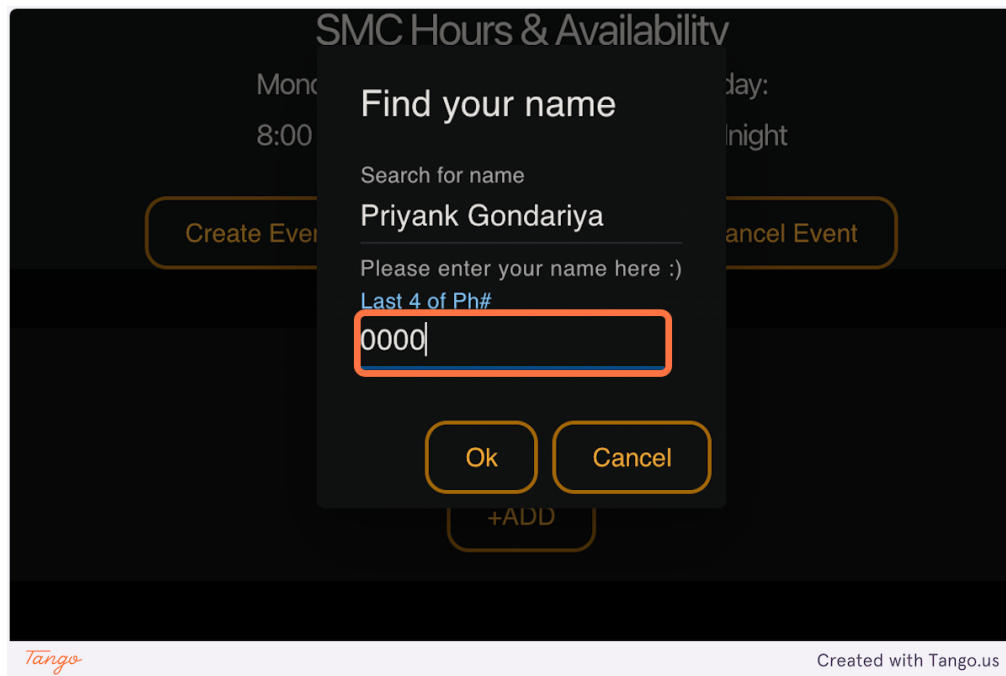
3. Click on *+ADD*



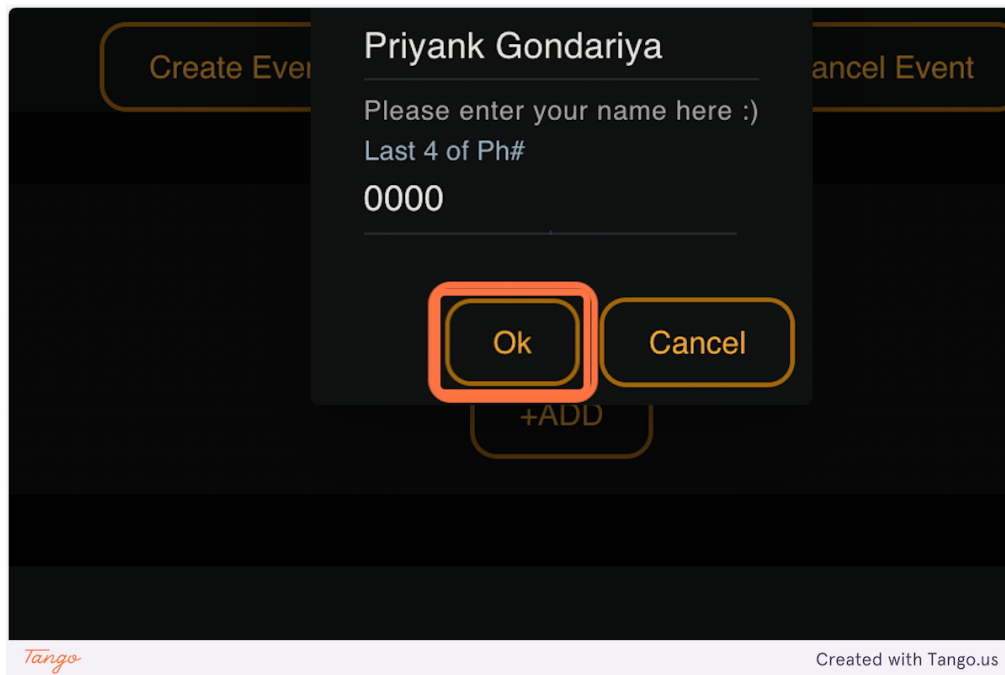
## 4. Type and/or choose name



## 5. Enter last 4 digits of phone number

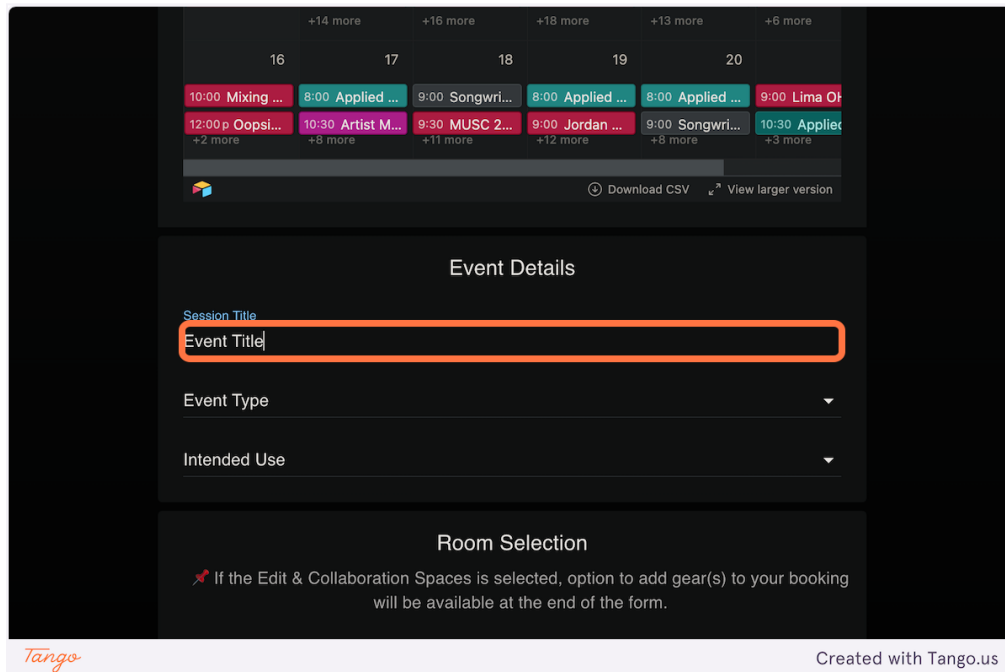


6. Click on *Ok*



The screenshot shows a dark-themed confirmation dialog box. At the top, it says "Priyank Gondariya". Below that, it asks "Please enter your name here :)" and "Last 4 of Ph#" with the input "0000". At the bottom, there are two buttons: "Ok" and "Cancel". The "Ok" button is highlighted with an orange border. Above the "Ok" button is a "+ADD" button. The dialog box is set against a background that shows a "Create Event" button on the left and a "Cancel Event" button on the right. At the bottom of the screen, there is a "Tango" logo and the text "Created with Tango.us".

7. Type in the session title



The screenshot shows a dark-themed "Event Details" form. At the top, there is a calendar view showing dates 16 through 20. Below the calendar, there is a table of sessions with columns for time, session title, and duration. The "Event Title" field is highlighted with an orange border. Below the "Event Title" field, there are two dropdown menus: "Event Type" and "Intended Use". At the bottom, there is a "Room Selection" section with a note: "If the Edit & Collaboration Spaces is selected, option to add gear(s) to your booking will be available at the end of the form." At the bottom of the screen, there is a "Tango" logo and the text "Created with Tango.us".



## 8. Choose event type

### Event Details

Session Title

Event Title

Event Type

Recording Session 🎤

Summer Booking 📅

Recording Session 🎤

Student Project 🎓

Class 📖

Meeting 🗓️

Rehearsal 🎭

Audition 🎤

## 9. Choose the intended use

### Event Details

Session Title

Event Title

Event Type

Recording Session 🎤

Intended Use

Personal Use 👤

Personal Use 👤

Academic 🎓

Sweetwater 🗓️

## 10. Choose room type

**Event Details**

Session Title  
Event Title

Event Type  
Summer Booking 🎨

Intended Use  
Personal Use 👤

**Room Selection**

🔥 If the Edit & Collaboration Spaces is selected, option to add gear(s) to your booking will be available at the end of the form.

Room Type

- ☒ Recording Studio 🎧
- ☐ Rehearsal Spaces 🎧
- ☐ Edit & Collaboration Spaces 🔥

**SUBMIT**

*Tango* Created with Tango.us

## 11. Choose room(s)

Session Title  
Event Title

Event Type  
Summer Booking 🎨

Intended Use  
Personal Use 👤

**Room Selection**

🔥 If the Edit & Collaboration Spaces is selected, option to add gear(s) to your booking will be available at the end of the form.

Room Type  
Recording Studio 🎧

Select studio room(s)

- ☒ 127 SMC (Tracking Room)
- ☐ 126 SMC (Control Room)
- ☐ SMC (Rehearsal Room)
- ☐ 125 SMC (Sound Lock)

*Tango* Created with Tango.us

## 12. Choose start date

127 SMC (Tracking Room) X 126 SMC (Control Room) X

### Session Time

Based on your chosen Session Time, we will notify you with the availability of the room(s) selected above.

Select Start Date

April 15, 2023 7:30PM

Select End Date

April 15, 2023 8:30PM

check availability

Tango Created with Tango.us

## 13. Choose start time

April 2023						Time
Mo	Tu	We	Th	Fr	Sa	
27	28	29	30	31	1	18:30
3	4	5	6	7	8	19:00
10	11	12	13	14	15	19:30
17	18	19	20	21	22	20:00
24	25	26	27	28	29	20:30
1	2	3	4	5	6	

check availability

SUBMIT

Tango Created with Tango.us

## 14. Choose end date

### Session Time

Based on your chosen Session Time, we will notify you with the availability of the room(s) selected above.

Select Start Date

April 15, 2023 8:00PM

Select End Date

April 15, 2023 9:00PM

check availability

### Courses

*Tango* Created with Tango.us

## 15. Choose end time

3	4	5	6	7	8
10	11	12	13	14	15
17	18	19	20	21	22
24	25	26	27	28	29
1	2	3	4	5	6

20:30

21:00

21:30

22:00

### Session Time

Based on your chosen Session Time, we will notify you with the availability of the room(s) selected above.

Select Start Date

April 15, 2023 8:00PM

Select End Date

April 15, 2023 9:00PM

check availability

*Tango* Created with Tango.us

16. Click on *check availability*

room(s) selected above.

Select Start Date

April 15, 2023 8:00PM

Select End Date

April 15, 2023 10:00PM

**check availability**

Courses

Is this time slot for a course assignment?

*Tango* Created with Tango.us

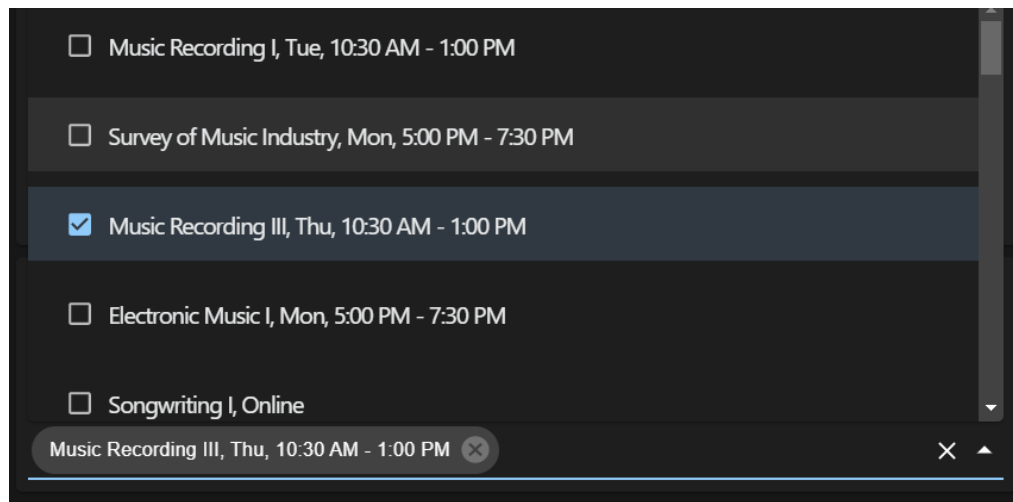
17. Check *Is this time slot for a course assignment?*

☒ Is this time slot for a course assignment?

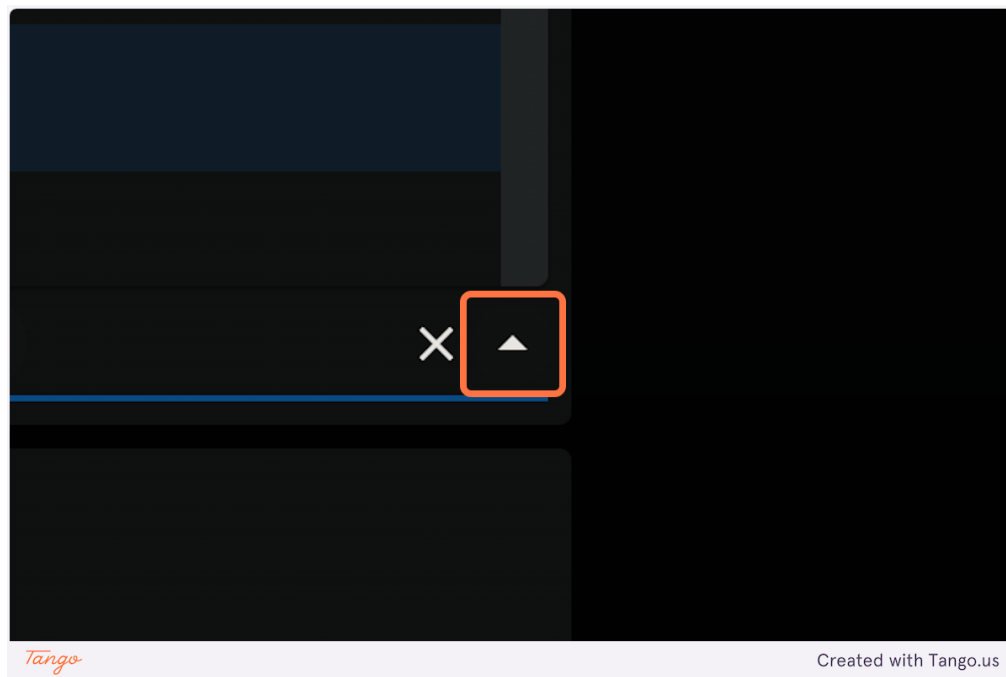
Select course(s)

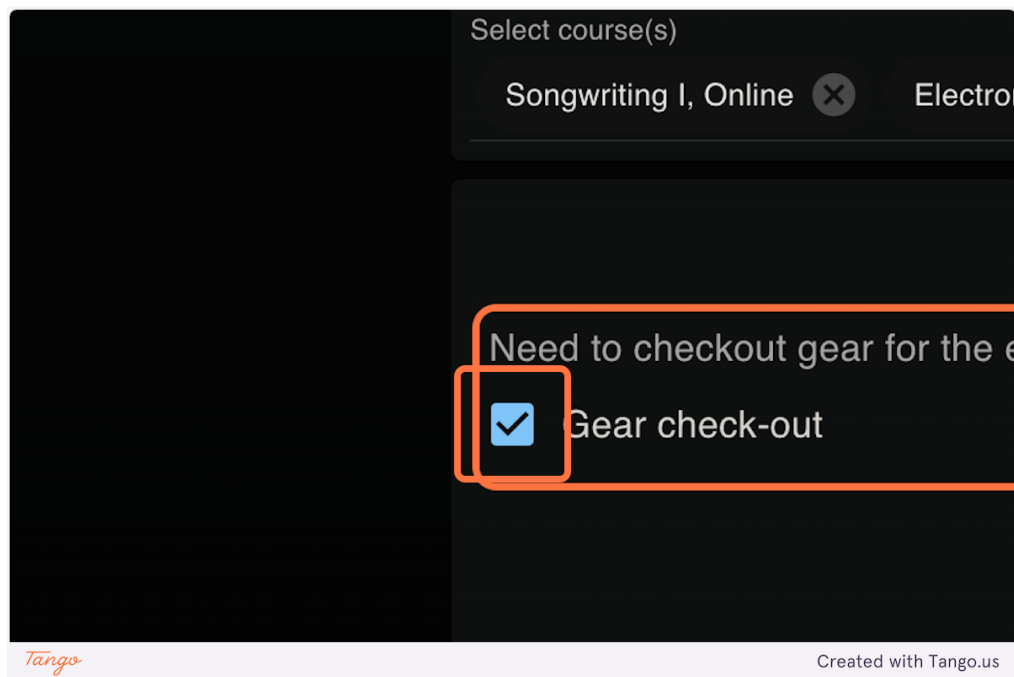
*Tango* Created with Tango.us

## 18. Select a course

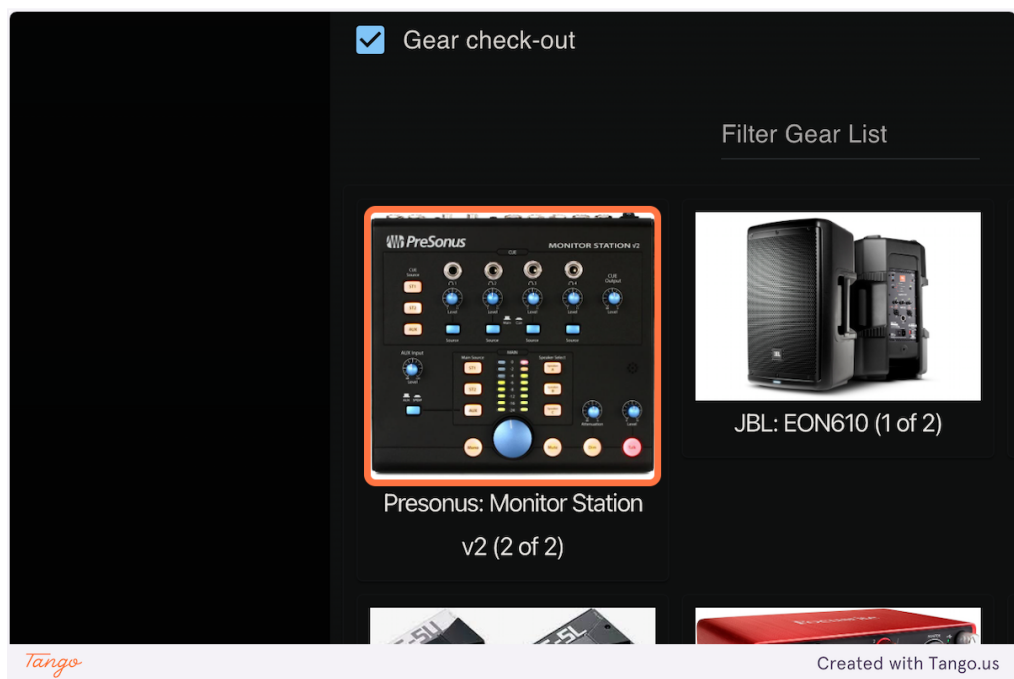


## 19. Minimize dropdown

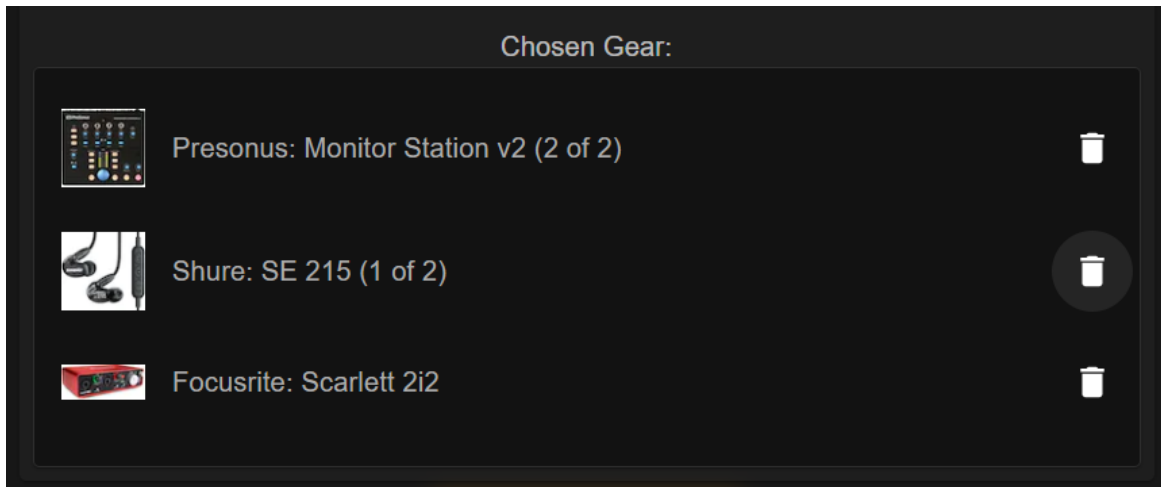


20. Check *Gear check-out*

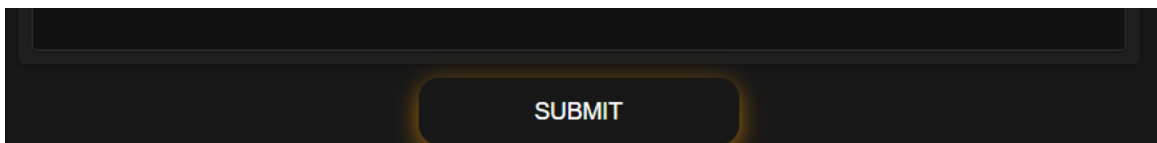
## 21. Choose all desired gear



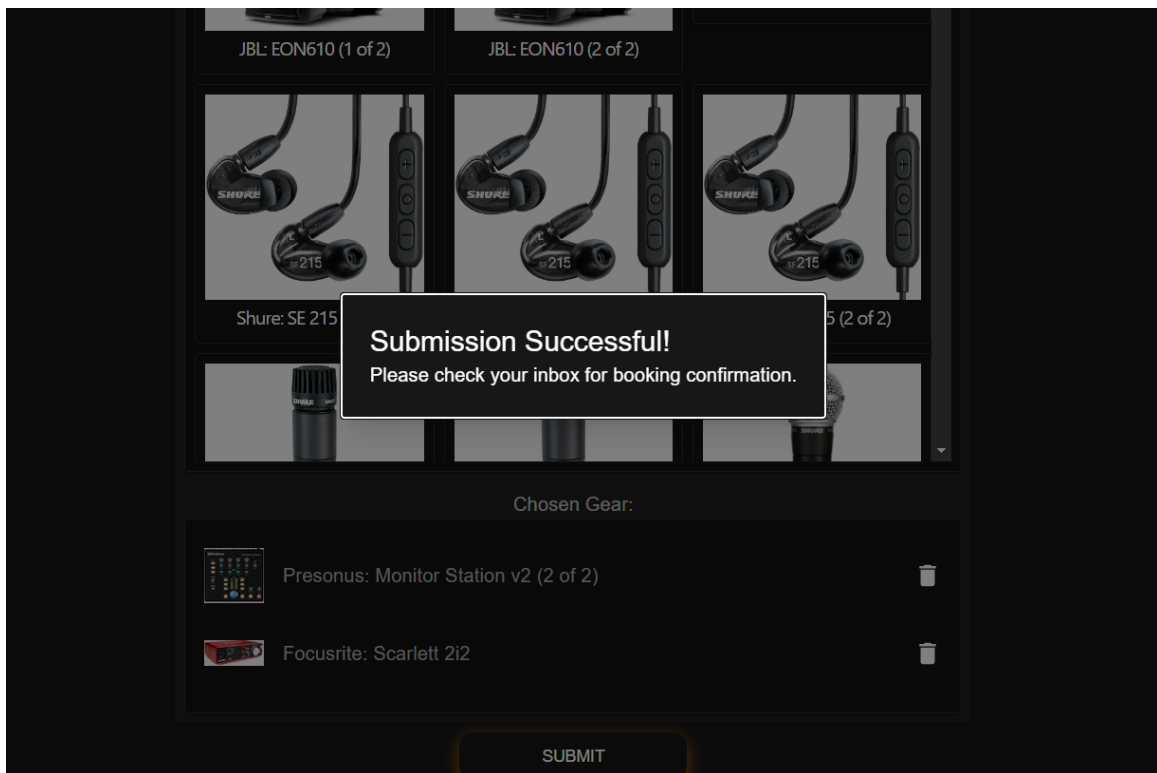
22. Remove gear from chosen gear list if needed



23. Click on SUBMIT



24. Submission Successful!





## Part 6 - Testing:

Test Case ID	Test Case	Pre-condition	Steps	Data	Expected Result	Actual Result	Status
Login_Test_1	Enter name not in database	1. Name does not exist in database	1. Navigate to Home page, 2. Click Add Name, 3. Enter nonexistant name, 4. Try to add participant	John Doe	Participant cannot be added	Participant could not be added	Pass
Login_Test_2	Enter incorrect password	1. Name and phone number exist in database	1. Navigate to Home page, 2. Click Add Name, 3. Enter correct name, 4. Enter incorrect phone number	name: Kyle Andrews, password: 1234	Password rejected	Password was rejected	Pass
Login_Test_3	Okay button on name entry popup	n/a	1. Navigate to Home page, 2. Click Add Name, 3. Enter correct name, 4. Enter correct phone number, 5. Click Okay button	name: Kyle Andrews, password: 6100	Popup closes, name accepted	Popup closes, name added	Pass
Login_Test_4	Cancel button on name entry popup	n/a	1. Navigate to Home page, 2. Click Add Name, 3. Enter correct name, 4. Enter correct phone number, 5. Click Cancel button	name: Kyle Andrews, password: 6100	Popup closes, name not accepted	Popup closes, name not added	Pass
Login_Test_5	Delete username	1. User has been added	1. Click Delete icon	n/a	User removed, row with user name dissapears	User removed, row with user name dissapears	Pass
Login_Test_6	Enter multiple people	n/a	1. Navigate to Home page, 2. Enter 2 different people, 3. Fill out rest of fields, 4. Submit	name1: Kyle Andrews, password1: 6100; name2: Priyank Gondariya, password2: 0000	The event shows up in the database with the both participants. The room and gear availability was filtered	The event shows up in the database with the both participants. The room and gear availability was filtered	Pass

					correctly.	correctly.	
Login_Test_7	Remove a participant, room and gear filtering	Multiple participant have been added with different levels	1. Navigate to Home page, 2. Enter 2 different people, 3. Fill out other filelds. 4. Verify that high level gear and rooms are visible. 5. Remove a high level participant.	Participants: Kyle Andrews, Priyank Gondariya	After removing higher level participant, hogher level rooms and gear should become unavailable.	I was able to schedule higher level gear without a higher level participant	Pass
EventDetails_Test_1	Enter name with special characters	1. Other fields filled out correctlty	1. Navigate to Home page, 2. Enter participant, 3. Enter event name with special characters, 4. Enter rest of filelds. Submit.	Name: "!@#\$\$%^&*+_)(* &^%Cć\$zł123**"	The event shows up in the database with the proper name	The event shpwed up in the database with the proper name	Pass
EventDetails_Test_2	Enter name with escape characters	1. Other fields filled out correctlty	1. Navigate to Home page, 2. Enter participant, 3. Enter event name with excape characters, 4. Enter rest of filelds. Submit.	Name: "\t \n \{ \} \\' \+\"	The event shows up in the database with the proper name	The event shpwed up in the database with the proper name	Pass
EventDetails_Test_3	All event types and uses can be chosen	1. Participant entered	1. Navigate to Home page, 2. Enter participant, 3. Verify options for event type and use	n/a	All options are available	All options were available	Pass
RoomSelection_Test_1	Room Type selection filter on level	1. Other fileds filled out correctlty	1. Navigate to Home page, 2. Enter participant, 3. Verify room types, 4. Repeat for participant of different levels		Room type dropdown filters correctly based on participant level	Room type dropdown filtered correctly based on participant level	Pass
RoomSelection_Test_2	Room selection filter on type	1. Other fileds filled out correctlty	1. Navigate to Home page, 2. Enter participant, 3. Choose room type, 4. Verify room dropdown, 5. Repeat for different room types	n/a	Room dropdown filters correctly based on room type	Room dropdown filtered correctly based on room type	Pass

TimePicker_Test_1	End time before or same time as start time	1. Other fields filled out correctly	1. Navigate to Home page, 2. Fill out Participant and Room Details, 3. Choose start time, 4. Try to choose end time before or at the same time as the chosen start time.	Start: April 4, 2023 12:00PM, End1: April 4, 2023 12:00PM, End2: April 4, 2023 11:00AM, End3: April 3, 2023 12:00PM	Time Picker does not allow such end time to be chosen	Time Picker did not allow such end time to be chosen	Pass
TimeValidation_Test_1	Schedule event on that overlaps with existing event towards end	1. Existing event in database	1. Navigate to Home page, 2. Choose same room as existing event, 3. Schedule new event ending an hour into existing event.	Existing event: April 14, 2023 12:00PM - 2:00PM, New event: April 14, 2023 11:00AM - 1:00PM	Scheduling rejected	Scheduling was rejected	Pass
TimeValidation_Test_2	Schedule event that overlaps with existing event towards beginning	1. Existing event in database	1. Navigate to Home page, 2. Choose same room as existing event, 3. Schedule new event starting an hour into existing event	Existing event: April 14, 2023 12:00PM - 2:00PM, New event: April 14, 2023 1:00PM - 3:00PM	Scheduling rejected	Scheduling was rejected	Pass
TimeValidation_Test_3	Schedule event that overlaps with existing event perfectly	1. Existing event in database	1. Navigate to Home page, 2. Choose same room as existing event, 3. Schedule new event with same start and end times as existing event	Existing event: April 14, 2023 12:00PM - 2:00PM, New event: April 14, 2023 12:00PM - 2:00PM	Scheduling rejected	Scheduling was rejected	Pass
TimeValidation_Test_4	Schedule event that does not overlap	1. Existing events in database	1. Navigate to Home page, 2. Choose same room as existing event, 3. Schedule new event	Existing event: April 14, 2023 12:00PM - 2:00PM, New	Scheduling accepted	Scheduling was accepted	Pass

				event: April 14, 2023 3:00PM - 5:00PM			
CoursePicker_Test_1	Course Selection drives dropdown visibility	1. Other fields filled out correctly	1. Navigate to Home page, 2. Fill out participant, 3. Verify that checkbox drives dropdown visibility	n/a	When checkbox unchecked, the dropdown is not visible. When checkbox checked, the dropdown is visible	When checkbox was unchecked, the dropdown was not visible	Pass
CoursePicker_Test_2	All courses in the database are available	1. Other fields filled out correctly	1. Navigate to Home page, 2. Fill out participant, 3. Check Course Checkbox, 4. Verify presents of courses on dropdown	n/a	All courses in database are available in the dropdown	All courses in database were available in the dropdown	Pass
GearFiltering_Test_1	Gear filtering on participant level	1. Other fields filled out correctly	1. Navigate to Home page, 2. Enter participant, 3. Verify available gear based on participant level, 4. Repeat for participant of different levels	Participant: Kelly Anderson, Level 2	Gear is filtered based on participant level	Gear was filtered based on participant level	Pass
GearFiltering_Test_2	Gear filtering on times chosen	1. Other fields filled out correctly	1. Navigate to Home page, 2. Enter participant, 3. Choose start and end times, 4. Verify that the Available Gear component filters based on chosen gear	Start: April 4, 2023 12:00PM, End: April 4, 2023 2:00PM	Available Gear component filters correctly	Available Gear component filtered correctly	Pass

GearSelection_Test_1	Click of gear on Available Gear component adds to Chosen Gear Component. Removes from Available Gear Component	1. Other fields filled out correctly	1. Navigate to Home page 2. Enter participant 3. Choose start and end times 4. Click on gear in Available Gear component 5. Verify gear is added to Chosen Gear Component and removed from Available Gear Component	Gear:"Presonus: Monitor Station v2 (2 of 2)"	Gear added to Chosen Gear Component and removed from Available Gear Component	Gear was added to Chosen Gear Component and removed from Available Gear Component	Pass
GearSelection_Test_2	Remove Chosen Gear, adds gear back to Available Gear component	1. Other fields filled out correctly	1. Navigate to Home page 2. Enter participant 3. Choose start and end times 4. Click on gear in Chosen Gear Component 5. Verify gear is removed from Chosen Gear Component and added back to Available Gear Component	Participant: Kelly Anderson, Event: Music Session, Gear:"Presonus: Monitor Station v2 (2 of 2)"	Gear removed from Chosen Gear Component and added back to Available Gear Component	Gear was removed from Chosen Gear Component and added back to Available Gear Component	Pass
GearSelection_Test_3	Chosen gear appear in database with event	1. Other fields filled out correctly	1. Navigate to Home page 2. Enter participant 3. Choose start and end times 4. Select gear 5. Submit event 6. Verify chosen gear appears in database with event	Participant: Kelly Anderson, Event: Music Session, Gear:"Presonus: Monitor Station v2 (2 of 2)"	Chosen gear appears in database with event	Chosen gear appeared in database with event	Pass
Submission_Test_1	Can Submit with proper data	1. Other fields filled out correctly	1. Navigate to Home page 2. Fill out all required fields with proper data 3. Click Submit	Participant: Kelly Anderson, Event: Music Session	Event is successfully submitted	Event was successfully submitted	Pass
Submission_Test_2	Cannot submit without proper data	n/a	1. Navigate to Home page 2. Fill out required fields with improper data 3. Attempt to click Submit	Participant: Kelly Anderson, Invalid phone number: 1234	Submission is not allowed	Submission was not allowed	Pass

Submission_Test_3	Cannot submit without required fields	n/a	1. Navigate to Home page 2. Fill out some but not all required fields 3. Attempt to click Submit	Participant: Kelly Anderson	Submission is not allowed	Submission was not allowed	Pass
GearPage_Test_1	Schedule event on Gear Checkout page	n/a	1. Navigate to Home page, 2. Schedule new event	Participant: Kelly Anderson, Event: Music Session, Gear:"Presonus: Monitor Station v2 (2 of 2)"	Scheduling same as on home page, but without room booking	Event was successfully submitted	Pass
UpdateCancel_Test_1	Can edit old event	1. Pre-existing event	1.Navigate to Update Event 2. Update start and end times 3. Update gear in Available Gear component 4. Verify gear is added to Chosen Gear Component and removed from Available Gear Component 5.Verify it updated in Airtable	Participant: Kelly Anderson, Event: Music Session, Gear:"Presonus: Monitor Station v2 (2 of 2)"	Gear removed from Chosen Gear Component and added back to Available Gear Component	Chosen gear appeared in database with event	Pass
UpdateCancel_Test_2	Can delete old event	1. Pre-existing event	1.Navigate to Delete Event 2. Delete Event	Old event ID	Event was removed from Airtable	Event was successfully removed from Airtable	Pass

## **Part 7 - Developer Tools and Read Me Document:**

### **Developer Tools:**

- Material UI: A ReactJS library that we used for easier and more visually appealing customization of object location and anchor points.
- NextUI: A ReactJS library that we used for easier and more visually appealing customization of objects such as buttons in React.
- Tailwind: A CSS framework that we used for in-text CSS coding inside HTML code to prevent spending time searching for where a component is being modified in the CSS files.
- Airtable: An API used to display the SMC Airtable Database information on the website.
- ESLint: A coding tool we used to assist in identifying problems within the Javascript code.

### **Read Me:**

Guide for CS360 Team

Source Code: The source code for the website can be found in the GitHub repository PFW-Music/smcWebsite. This is a Next.js project bootstrapped with create-next-app.

The following keys are required in the .env file:

```
NEXT_PUBLIC_AIRTABLE_KEY="..."
NEXT_PUBLIC_AIRTABLE_BASE_ID="..."
```

### **Getting Started**

First, run the development server:

Type the following into the terminal:

```
npm run dev
# or
yarn dev
# or
pnpm dev
```

Open `http://localhost:3000` with your browser to see the result.

You can start editing the page by modifying `pages/index.js`. The page auto-updates as you edit the file.

API routes can be accessed on `http://localhost:3000/api/hello`. This endpoint can be edited in `pages/api/hello.js`.

The `pages/api` directory is mapped to `/api/*`. Files in this directory are treated as API routes instead of React pages.

This project uses `next/font` to automatically optimize and load Inter, a custom Google Font.

## Learn More

To learn more about Next.js, take a look at the following resources:

- Next.js Documentation - learn about Next.js features and API.

- Learn Next.js - an interactive Next.js tutorial.

- You can check out the Next.js GitHub repository - your feedback and contributions are welcome!

## Deploy on Vercel

The easiest way to deploy your Next.js app is to use the Vercel Platform from the creators of Next.js.

Check out our Next.js deployment documentation for more details.

## Notes:

The `.env` file is not on GitHub to prevent the leakage of user data. For more information on creating this file, please refer to this article. If changing the API Key (for example, to test on a copy of the Airtable base), it must be changed in the `.env` file in the root of the project directory. If this change needs to be deployed, the environment variable on Vercel must also be changed.



## **Part 8 - Code Design and Standards:**

1. Documentation
  - a. Comments - pieces of code that are longer than 5 lines and have functionality that is not self-explanatory should have a comment above explaining the purpose and logic of the code.
2. Programming Practices
  - a. Naming- variables, constants, functions, classes, and components should be given descriptive names that adequately describe their purpose and aid in code readability
3. Design
  - a. Code Reuse - codebase should be containerized into classes, components, and functions for the purpose of reusing code. This practice helps reduce bugs and keep the codebase adaptive to change.
  - b. Encapsulation - wherever beneficial, code should be separated into classes and components with proper use of data hiding and modularization.
4. Cohesion and Coupling
  - a. Cohesion - Components and methods should be closely related and focus on a single task or responsibility.
  - b. Coupling - Components and modules should be independent of one another. Code is modular and easy to update without affecting the codebase.
5. Design Patterns
  - a. Component-Based Architecture - Codebase should be centered around components that can be used repeatedly. Breaking down User Interface components should lead to a manageable codebase.
  - b. Render Props - Components should provide render functions to their children to allow for better code reuse.
  - c. Context API - Data should be shared between components without needing to pass data through other component levels.

## **Part 9 - Member Contributions:**

Team Member	Component 1: Requirements	Component 2: Design Models	Component 3: Documentation	Component 4: Implementation	Component 5: Testing	Component 6: Presentation
Daniel	x	x	x	x	x	x
Priyank	x	x	x	x	x	x
Ivie	x	x	x	x		x
Celeste	x	x	x	x		x
Andre	x	x	x	x	x	x

## **Part 10 - References:**

- <https://vercel.com>
- <https://tailwindcss.com>
- <https://mui.com>
- <https://nextui.org>
- <https://react.dev/reference/react>
- <https://airtable.com>
- <https://github.com>
- <https://developer.mozilla.org>