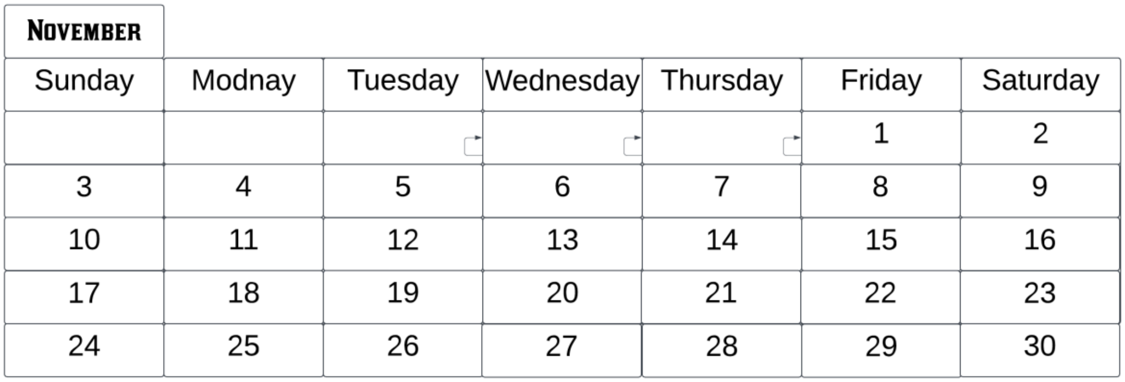
PocketClass

Step 1 : In Instructor interface open calendar icon



NOVEMBER

Sunday	Modnay	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Time slot	S 10	M 11	T 12	W 13	T 14	F 15	S 16
12AM	🟩	🟩	🟥	🟥	🟩	🟥	🟥
1AM	🟩	🟥	🟥	🟩	🟥	🟩	🟥
2AM	🟥	🟩	🟥	🟥	🟥	🟥	🟩
3AM	🟥	🟥	🟥	🟥	🟥	🟩	🟥

//When I tap on 11th date it will show

🟩 →Available

🟥 →Booked

//When u tap on any block

Upcoming Bookings

Student name	Date	Time slot	Status
Raghav	11/03/2025	6AM-9AM	Confirmed
Sanjay	10/11/2024	10AM-12PM	Confirmed
Chetan	20/12/2024	8PM-10PM	Not Confirmed

PocketClass

Step 1: Calendar section

Calendar icon

THU
7

Step 2: Tap on Calendar icon

November

Sun	Mon	Tues	Wed	Thur	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Dec

Jan

Feb

--
--
--

Step 3: Tap to particular date

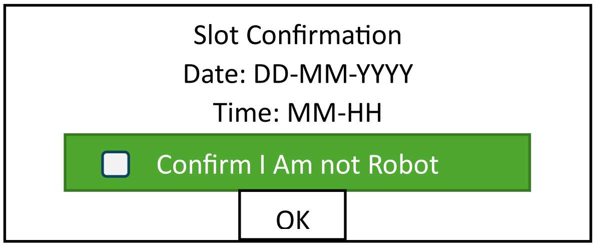
Time slot	Fri-29th
6AM	🟥
7AM	🟥
8AM	🟩
9AM	🟩
10AM	🟩
- - -	

🟥 →Not Available

🟩 →Available

Step 4: Select the available slot

Pop up the confirmation



Slot Confirmation

Date: DD-MM-YYYY

Time: MM-HH

PocketClass

1) Firestore Data schema

```
/users
   /{userId}  ---> User profile (Student or Instructor)
      - userId
      - name
      - email
      - role (Student/Instructor)

/instructors
   /{instructorId} ---> Instructor details
      - instructorId
      - name
      - availableSlots (array of time slots)
      - bookings (array of booking references)

/bookings
   /{bookingId} ---> Booking details
      - bookingId
      - instructorId
      - studentId
      - timeSlot
      - status (Pending/Confirmed)
```

B. Detailed Fields Breakdown

Users Collection:

Each document contains user details, including userId, name, email, and role.

The role field helps distinguish between students and instructors.

Instructors Collection:

Each instructor document contains:

instructorId: Unique identifier for each instructor.

name: Instructor’s name.

availableSlots: An array that holds the available time slots (each slot is an object with date and time).

bookings: An array storing references to confirmed bookings. This helps in keeping track of which slots are taken.

Bookings Collection:

Each document stores a booking record with:

bookingId: Unique ID for each booking.

instructorId: Reference to the instructor.

studentId: Reference to the student making the booking.

timeSlot: Date and time of the booked slot.

status: This could be either Pending (if the booking is yet to be confirmed) or Confirmed (once the booking is finalized).

2. Preventing Double-Booking and Handling Conflicts

1) Locking Mechanism:

When a student tries to book a slot, a **transaction** is used in Firebase to check whether the slot is still available. If the slot is already booked (checked from the `bookings` collection), the booking will fail.

2) Atomic Transactions
Use Firebase **transactions** to handle reading and writing data for booking. The transaction ensures that the availability of a time slot is checked and updated in one atomic operation, preventing a race condition where two students might book the same slot at the same time.

Real-Time Updates for Availability
Real-Time Data: Firebase allows real-time updates. When an instructor updates their availability or when a booking is made, the availability can be updated instantly across the system, reducing the chances of double-booking.

Live Updates: A student’s booking interface should listen for real-time updates on the availability of the instructor's slots. This way, students see the most current data and can only book available slots.

Steps in Booking Logic (Node.js + Firebase):

1)Fetching Availability:

Use Firebase Firestore’s get() method to fetch available slots for the instructor in real-time.

2)Making a Booking:

When a student selects a slot:

Use Firebase transaction to check if the selected time slot is still available.

If available, create a new document in the bookings collection.

Update the instructor’s availableSlots and mark the slot as booked.

Send a confirmation response to the student.

3)Handling Conflicts:

If two students attempt to book the same slot simultaneously, Firebase transactions will ensure that only one booking succeeds.

Notify the second student with a message like "The slot has already been booked, please select another time."

PocketClass

1. Architectural Choices

High-Level Architecture Overview:

Frontend:

React.js or a similar modern frontend framework for the instructor and student interfaces.

Users interact with the system through the calendar and booking features, interacting with Firebase for real-time updates.

Integration with backend endpoints for booking, availability management, and notifications.

Backend:

Node.js + Express: Handles API requests from both students and instructors.

Firebase:

Firestore for storing data (instructors' available time slots, bookings, user profiles).

Firebase Authentication for managing user authentication.

Firebase Cloud Functions for triggering events (e.g., sending notifications, updating availability).

Key Challenges and Solutions:

Real-Time Updates: Firebase's real-time data sync ensures that both students and instructors can see up-to-date availability.

Handling High Traffic: Use Firestore's built-in scalability and rate-limiting to prevent traffic spikes from affecting performance.

Ensuring Data Consistency: Firebase transactions handle critical operations, ensuring data consistency for bookings and availability.

2. Database Schema and Endpoints

Here's a **high-level view** of the Firebase database schema and necessary API endpoints.

```
/users
   /{userId}  ---> User profile (Student or Instructor)
      - userId
      - name
      - email
      - role (Student/Instructor)
      - profileImageURL


/instructors
```

PocketClass

/{instructorId} ---> Instructor details

- instructorId
- name
- availableSlots (array of time slots)
- bookings (array of booking references)

/bookings

/{bookingId} ---> Booking details

- bookingId
- instructorId
- studentId
- timeSlot
- status (Pending/Confirmed)

API Endpoints:

1)GET /instructors/{instructorId}/availability

Fetch available time slots for a specific instructor.

Query Firestore’s availableSlots for the given instructor.

2)POST /bookings

Create a new booking.

Takes instructorId, studentId, and timeSlot as input.

Check if the time slot is available (transaction).

If available, create a booking in the bookings collection and update the instructor’s available slots.

3)GET /bookings/{bookingId}

Fetch booking details (useful for both students and instructors to track bookings).

4)PUT /instructors/{instructorId}/availability

Update instructor’s availability (add, edit, or remove time slots).

This modifies the availableSlots array in the Firestore document.

5)POST /users/register

Register a new user (student or instructor).

Creates a profile in the users collection.

Uses Firebase Authentication for account creation and login.

3. User Experience Considerations

To ensure a **smooth and effective** user experience for both students and instructors, we will focus on simplifying interactions, streamlining the booking flow, and incorporating useful reminders.

Streamlining the Booking Form:

1. **Instructor Interface**:
    - **Intuitive Calendar**: A clean and interactive calendar to set, edit, or delete availability. Instructors can use simple drag-and-drop or button-based interactions for setting and managing availability.
    - **Time Slot Management**: Instructors can easily select multiple time slots and set them as available or unavailable. The UI will clearly display these slots with different colors (e.g., green for available, red for booked).
2. **Student Interface**:
    - **Real-Time Slot Viewing**: Students will only see available slots for instructors. If a slot becomes booked during their session, it will immediately reflect in the UI.
    - **One-Click Booking**: Once a student selects a time, a **single confirmation button** initiates the booking. A pop-up will confirm the booking and provide a quick view of the booked slot.
    - **Personalized Profile**: Students can view their past bookings and upcoming sessions in a **simple list** format, making it easy to track all their interactions.

Notifications and Reminders:

- **Booking Confirmation**: After successfully booking a slot, both the instructor and student receive an instant **email** or **push notification** with details about the session.
- **Reminders**: Both parties will receive a reminder 24 hours before the session starts. A second reminder will be sent 1 hour before the session begins.
- **Availability Changes**: If an instructor updates their availability, students who were interested in those slots should be notified, providing them with an opportunity to rebook if needed.

Innovative Enhancements:

- **Suggested Times**: If a selected time is unavailable, the system could suggest alternative slots based on both the student's and instructor's available hours.
- **AI-Powered Recommendations**: Implement AI/ML to suggest optimal times based on previous booking patterns and user behavior.

4. Conflict Handling Considerations

Calendars and booking systems are prone to conflicts, especially when multiple users interact with the same resources (in this case, time slots). Here's how we can address **conflict detection** and **resolution** effectively:

A. Locking Mechanisms (Optimistic Concurrency)

- **Firebase Transactions**: Use **Firebase transactions** to ensure atomic operations. When a student tries to book a slot, the backend first checks whether the slot is available. If it is, the booking goes through. If not, the system will reject the booking and suggest alternative slots.
- **Soft Locking**: Once a student selects a time, the system temporarily locks that slot for a short time (e.g., 3 minutes) to allow them to complete the booking process without the risk of another student booking it simultaneously.
- **Lock Timeout**: If the student doesn't complete the booking within the timeout period, the slot is unlocked and made available to others.

B. Live Availability Updates

- **Real-Time Updates**: Both the instructor and students can see live availability. If another student books the slot while another student is in the process of booking it, the UI will update instantly to show the slot as unavailable.
- **Event-driven Updates**: Whenever a booking is made, a **Cloud Function** can notify all clients of the status update, refreshing the UI in real-time without the need to refresh the page.

C. Retry Mechanisms

- If a booking attempt fails due to a conflict, offer the student a **retry option** or present alternative available slots.
- **Queueing**: In cases of high demand, implement a simple queuing system where students can join a waitlist for their preferred time slots. If someone cancels, the system notifies the next person in the queue.