Participation 1: Singleton Pattern in Real Systems

Colby P. Frison
Emily R. Locklear
James Totah
Grant P. Parkman
Antonio M. Natusch Zarco

Group H

System Description

- ➔ Web application
- ➔ Firebase for backend
- ➔ Client-side components
  - ◆ User interactions
  - ◆ Initializes Firebase app instance
- ➔ Server-side components
  - ◆ Firebase Cloud Functions
  - ◆ Operations
  - ◆ Initializes Firebase Admin SDK

```
2 import { useState, useEffect, useCallback } from 'react';      'useCallback' is declared but its value is r
3 import { useRouter } from 'next/navigation';
4 import { setUserType, getUserType } from '@/lib/auth';
5
6 export default function SelectRolePage() {
7   const router = useRouter();
8   const [userType, setUserTypeState] = useState<'student' | 'professor' | null>(null);
9
10  useEffect(() => {
11    // Check if user type is already selected
12    const currentUserType = getUserType();
13    if (currentUserType === 'professor') {
14      router.push('/professor');
15    } else if (currentUserType === 'student') {
16      router.push('/student');
17    }
18  }, [router]);
19
20  const handleContinue = () => {
21    if (!userType) return;
22
23    // Store the user type
24    setUserType(userType);
25
26    // Redirect based on user type
27    if (userType === 'professor') {
28      router.push('/professor');
29    } else {
30      router.push('/student');
31    }
32  };
```

Problem

- ➔ Initializes a new Firebase app instance for each client-side user session.
- ➔ Every user creates their own connection to the database.
- ➔ Leads to multiple simultaneous connections.
- ➔ Causes performance bottlenecks and resource exhaustion.

Problem (code) pt. 1

firebase.ts

```ts
let app: FirebaseApp;
let db: Firestore;
let analytics: any;

try {
  console.log("Initializing Firebase...");

  // Check if Firebase is already initialized
  if (getApps().length === 0) {
    app = initializeApp(firebaseConfig);
    console.log("Firebase initialized successfully");
  } else {
    app = getApps()[0];
    console.log("Using existing Firebase app");
  }

  db = getFirestore(app);

  // Initialize analytics if we're in the browser
  if (typeof window !== 'undefined') {
    try {
      analytics = getAnalytics(app);
      console.log("Firebase analytics initialized");
    } catch (analyticsError) {
      console.warn("Analytics initialization failed:", analyticsError);
    }
  }
```

// Exporting the “singleton” instance

```ts
export { db };
```

classSession.ts

```
19  import { db } from './firebase';
```

classSession.ts

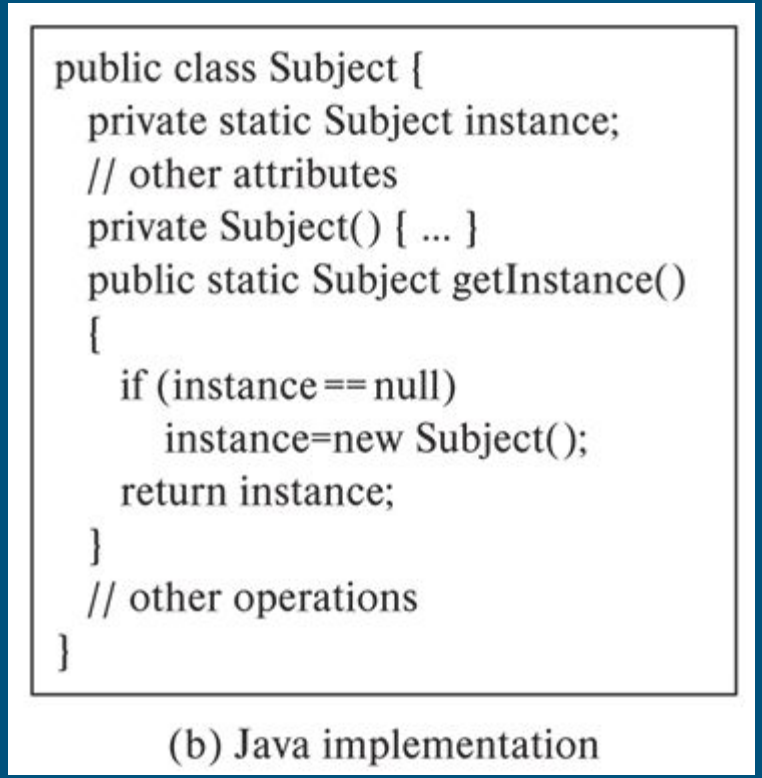
```
30 export const createClassSession = async (
29   className: string,
28   professorId: string
27 ): Promise<{ sessionId: string, sessionCode: string }> => {
26   if (!className || !professorId) {
25     console.error("Missing parameters for createClassSession");
24     throw new Error("Class name and professor ID are required");
23   }
22
21   try {
20     console.log(`Creating class session for ${className} with professor ${professorId}`);
19
18     // Generate a random session code for this session
17     const sessionCode = generateSessionCode();
16
15     // Create timestamp for tracking
14     const currentTime = Date.now();
13
12     // Create the session object
11     const session: Omit<ClassSession, "id"> = {
10       code: className,          // Original class name
9        sessionCode: sessionCode, // New randomly generated code for this session
8        professorId,
7        status: 'active',         // Set status according to interface requirements
6        createdAt: currentTime,
5        lastActiveAt: currentTime,
4        lastActive: currentTime   // For maintenance functions
3      };
2
1      // Add to Firestore
104    const docRef = await addDoc(collection(db, CLASS_SESSIONS_COLLECTION), session);
```

This implementation creates a singleton *within each client browser session* because:

1. The code runs in the user's browser (client-side)
2. Each user loads their own copy of the application
3. Each browser gets its own Firebase instance
4. The `db` variable is a singleton only within each browser session

Solution

The Singleton Pattern!



```
public class Subject {
  private static Subject instance;
  // other attributes
  private Subject() { ... }
  public static Subject getInstance()
  {
    if (instance == null)
      instance=new Subject();
    return instance;
  }
  // other operations
}
```

(b) Java implementation

Why is the Singleton Pattern an ideal solution?

- ➔ Allows us to centralize database access
- ➔ Reduces the overhead of establishing multiple connections
- ➔ Ensures consistent access to the database

Specific Solution with Singleton

- Backend service
  - Maintain single connection to the database
  - Client requests
- Client-server communication
  - Clients communicate via WebSockets or RESTful APIs
- Singleton implementation
  - Singleton applied to backend
  - Check for existing connections and reuse them

Singleton Solution Proposal (code)

```
export class FirebaseService {
  private static instance: FirebaseService;
  private db: Firestore;

  // Private constructor prevents direct instantiation
  private constructor() {
    try {
      const app = initializeApp({
        credential: cert(serviceAccount as ServiceAccount),
      });

      this.db = getFirestore(app);
      console.log('Firebase Admin SDK initialized successfully');
    } catch (error) {
      console.error('Error initializing Firebase Admin SDK:', error);
      throw error;
    }
  }

  /**
   * Get the singleton instance of FirebaseService
   * If instance doesn't exist, creates it
   */
  public static getInstance(): FirebaseService {
    if (!FirebaseService.instance) {
      FirebaseService.instance = new FirebaseService();
    }
    return FirebaseService.instance;
  }
```

```
public async createClassSession(
  className: string,
  professorId: string,
): Promise<{ sessionId: string; sessionCode: string }> {
  if (!className || !professorId) {
    throw new Error('Class name and professor ID are required');
  }

  try {
    // Generate session code
    const sessionCode = this.generateSessionCode();
    const currentTime = Date.now();

    // Create session object
    const session = {
      code: className,
      sessionCode: sessionCode,
      professorId,
      status: 'active',
      createdAt: currentTime,
      lastActiveAt: currentTime,
      lastActive: currentTime,
    };

    // Add to Firestore
    const docRef = await this.db.collection('classSessions').add(session);
    return { sessionId: docRef.id, sessionCode };
```

Singleton Solution Proposal (code)

```ts
// Initialize Socket.io
const io = new Server(server, {
  cors: {
    origin: process.env.CLIENT_URL || 'http://localhost:3000',
    methods: ['GET', 'POST'],
  },
});

// Get the singleton instance of FirebaseService
const firebaseService = FirebaseService.getInstance();

// Socket.io connection handler
io.on('connection', (socket) => {
  console.log(`New client connected: ${socket.id}`);

  // Class Session Events
  socket.on('createClassSession', async (data, callback) => {
    try {
      const { className, professorId } = data;
      const result = await firebaseService.createClassSession(
        className,
        professorId,
      );
      callback({ success: true, data: result });
    } catch (error: any) {
      console.error('Error creating class session:', error);
      callback({ success: false, error: error.message });
    }
  });
```

NORMAL index.ts main 2 1

```ts
  /**
   * Create a class session
   */
  public createClassSession(
    className: string,
    professorId: string,
  ): Promise<{ sessionId: string; sessionCode: string }> {
    return new Promise((resolve, reject) => {
      this.socket.emit(
        'createClassSession',
        { className, professorId },
        (response) => {
          if (response.success) {
            resolve(response.data);
          } else {
            reject(new Error(response.error));
          }
        },
      );
    });
  }
```

NORMAL socketService.ts main

src/services/socketService.ts" line 70 of 91 --76%-- col 3

Singleton Solution Proposal (code)

```
25  const handleStartSession = async () => {
24    if (!className || !professorId) {
23      setError("Class name or professor ID is missing");
22      return;
21    }
20
19    try {
18      // Reset welcome message when starting a new session
17      resetWelcomeMessage();
16
15      setIsLoading(true);
14
13      // This is the key change - using socketService instead of direct function call
12      const result = await socketService.createClassSession(className, professorId);
11
10      setSessionId(result.sessionId);
 9      setSessionCode(result.sessionCode);
 8      setSessionActive(true);
 7      setSessionStartTime(Date.now());
 6      setLastActivity(Date.now());
 5
```

Benefits of Using Singleton

- Reduced load
  - Due to centralized database connection
- Improved Performance
  - No more overhead of establishing new connections
- Scalability
  - The backend can handle multiple client requests through one connection

The End.