CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G5 SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: Employee Timesheet Tracker

Prepared By: Joji O. Matsuda

Date of Submission: 2/6/2026

Version: 0.1

1. Introduction

1.1. Purpose

A web-based application designed to manage employee time tracking and attendance records. The intended audience includes:

- System developers and programmers who will implement the React frontend, Spring Boot backend, and Kotlin mobile application
- Database administrators managing the MySQL database
- Project managers and stakeholders who need to understand system capabilities
- Quality assurance testers who will validate system functionality
- End users (administrators and employees) who will utilize the timesheet tracking features

1.2. Scope

The Employee Timesheet Tracker is a full-stack web and mobile application designed to streamline employee time management and attendance tracking across multiple platforms. The system enables:

Core Functionality:

- User authentication and authorization with role-based access control
- Adding and managing employee records
- Recording clock-in and clock-out times for employees
- Automatic calculation of hours worked per session
- Viewing comprehensive timesheet records with advanced search and filtering
- Generating employee work hour summaries and reports
- Real-time data synchronization across web and mobile platforms

System Boundaries:

- Frontend (Web): React-based single-page application with responsive design
- Frontend (Mobile): Kotlin-based native Android application
- Backend: Spring Boot RESTful API with JWT authentication
- Database: Supabase (PostgreSQL) for persistent data storage with real-time capabilities
- Authentication: Supabase Auth for secure user management
- Future Integration: Payroll systems, HR management platforms, and reporting tools

1.3. Definitions, Acronyms, and Abbreviations

- SRS: System Requirements Specification
- API: Application Programming Interface
- REST: Representational State Transfer
- JWT: JSON Web Token
- UI: User Interface
- SPA: Single Page Application
- CRUD: Create, Read, Update, Delete operations
- PostgreSQL: Open-source relational database management system
- React: JavaScript library for building user interfaces
- Spring Boot: Java-based framework for creating microservices and web applications
- Supabase: Open-source Firebase alternative with PostgreSQL database
- Kotlin: Modern programming language for Android development
- Clock In: Recording the start time of an employee's work session

- Clock Out: Recording the end time of an employee's work session
- Timesheet Record: A complete entry containing employee name, clock-in time, clock-out time, and hours worked
- Active Record: An employee who has clocked in but not yet clocked out

2. Overall Description

2.1. System Perspective

The Employee Timesheet Tracker is a distributed, multi-tier application that operates across web browsers, mobile devices, and cloud infrastructure. The system architecture consists of:

Presentation Layer:

- Web Application: React-based SPA with modern component architecture, state management (Redux/Context API), and responsive design
- Mobile Application: Native Android app built with Kotlin, following Material Design guidelines

Application Layer:

- Spring Boot Backend: RESTful API server handling business logic, authentication, authorization, and data validation
- Microservices Architecture: Modular services for user management, employee management, timesheet operations, and reporting

Data Layer:

- Supabase Database: PostgreSQL database providing ACID-compliant transactions, real-time subscriptions, and Row Level Security (RLS)
- Supabase Auth: Integrated authentication service with JWT token management
- Supabase Storage: For future document/file storage capabilities

Integration Points:

- RESTful API endpoints for client-server communication
- WebSocket connections for real-time updates
- Supabase Realtime for live data synchronization
- Future integrations with HR systems, payroll processors, and analytics platforms

Deployment Architecture:

- Frontend: Cloud hosting (Vercel, Netlify, or Render)
- Backend: Container-based deployment (Docker + Kubernetes or AWS ECS)
- Database: Supabase managed cloud infrastructure
- Mobile: Google Play Store distribution

2.2. User Classes and Characteristics

The system supports three distinct user classes with different permission levels:

1. Guest User (Unauthenticated)

Characteristics: No account, limited access
Technical Proficiency: Basic web/mobile navigation skills
Permissions:

- View login page

- Register for new account
- Password recovery (future)

Frequency of Use: One-time (registration) or occasional (login)

2. Authenticated Employee
Characteristics: Registered employee with standard access
Technical Proficiency: Moderate; comfortable with mobile/web apps
Permissions:

- Clock in/out for themselves only
- View their own timesheet records
- View their personal work hour summary
- Update their profile information
- Log out and manage sessions

Frequency of Use: Daily (2-4 times per day for clock operations)

3. Administrator/Manager
Characteristics: Supervisory role with full system access
Technical Proficiency: Advanced; understands reports and analytics
Permissions:

- All employee permissions
- Add, edit, and deactivate employee accounts
- Clock in/out any employee (override capability)
- View all timesheet records across organization
- Generate comprehensive reports and summaries
- Export data (CSV, PDF formats)
- Manage system settings and configurations

Frequency of Use: Daily for monitoring; weekly for reporting

2.3. Operating Environment

Client Platform (Web):

- Modern web browsers: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- JavaScript ES6+ enabled
- Minimum screen resolution: 320px width (mobile-first responsive design)
- Operating Systems: Windows 10/11, macOS 11+, Linux (Ubuntu 20.04+), ChromeOS

Client Platform (Mobile):

- Android 8.0 (API level 26) or higher
- Minimum 2GB RAM
- 50MB available storage space
- Internet connectivity (Wi-Fi or cellular data)

Server Requirements:

- Backend: Spring Boot 3.x running on JVM 17+
- Container: Docker 20.x or higher
- Orchestration: Kubernetes 1.24+ (for production scaling)
- Operating System: Linux (Ubuntu Server 22.04 LTS recommended)

Database Requirements:

- Supabase cloud infrastructure (managed PostgreSQL 15+)
- Automatic backups and point-in-time recovery

- Connection pooling for high concurrency
- SSL/TLS encrypted connections

Network Requirements:
- HTTPS/TLS 1.2+ for all communications
- RESTful API over HTTP/2
- WebSocket support for real-time features
- Minimum 1 Mbps internet connection recommended
- Supports offline mode with sync (future enhancement)

2.4. Assumptions and Dependencies

Assumptions:
- Users have valid email addresses for registration
- Device system time is accurate for proper timestamp recording
- Users have stable internet connectivity for real-time operations
- Administrators have authority to manage employee records
- Employees will clock in/out honestly (honor system until biometric integration)
- Organization operates in a single timezone (multi-timezone support is future enhancement)

Dependencies:
Frontend (React) Dependencies:
- React 18.x for component rendering
- React Router for navigation
- Axios or Fetch API for HTTP requests
- Redux Toolkit or React Context API for state management
- Material-UI or Tailwind CSS for styling
- React Query for server state management and caching
- Date-fns or Day.js for date/time manipulation

Backend (Spring Boot) Dependencies:
- Spring Boot 3.x framework
- Spring Security for authentication and authorization
- Spring Data JPA for database operations
- JWT libraries (jjwt) for token generation/validation
- Hibernate ORM for entity management
- Lombok for reducing boilerplate code
- Spring Validation for input validation
- Jackson for JSON serialization/deserialization

Mobile (Kotlin) Dependencies:
- Kotlin 1.9.x
- Android Jetpack components (ViewModel, LiveData, Navigation)
- Retrofit for API communication
- Room Database for local caching (offline support)
- Coroutines for asynchronous operations
- Hilt for dependency injection
- Material Design Components

Database (Supabase) Dependencies:

- Supabase client libraries for authentication and real-time
- PostgreSQL 15+ features
- Row Level Security (RLS) policies for data protection
- PostgREST for automatic API generation

External Services:
- Email service (SendGrid, AWS SES) for notifications
- Cloud storage for future file uploads
- Analytics service (Google Analytics, Mixpanel) for usage tracking

3. System Features and Functional Requirements

3.1. Feature 1: User Authentication and Authorization

Description:
The system shall provide secure user authentication with role-based access control, supporting registration, login, logout, and session management across web and mobile platforms.
Functional Requirements:
- FR-1.1: The system shall allow new users to register with username, email, and password
- FR-1.2: The system shall validate email format and password strength (minimum 8 characters, 1 uppercase, 1 number, 1 special character)
- FR-1.3: The system shall hash passwords using bcrypt or Argon2 before storage
- FR-1.4: The system shall prevent duplicate usernames and emails during registration
- FR-1.5: The system shall authenticate users with valid credentials and issue JWT tokens
- FR-1.6: The system shall track failed login attempts and lock accounts after 5 consecutive failures
- FR-1.7: The system shall maintain active user sessions with automatic token refresh
- FR-1.8: The system shall expire sessions after 30 minutes of inactivity
- FR-1.9: The system shall allow users to logout and invalidate their session tokens
- FR-1.10: The system shall redirect unauthenticated users to login page when accessing protected routes
- FR-1.11: The system shall support "Remember Me" functionality for extended sessions
- FR-1.12: The system shall update last login timestamp in database upon successful authentication

3.2. Feature 2: Employee Management

Description:
The system shall provide comprehensive employee record management with CRUD operations, accessible based on user roles and permissions.
Functional Requirements:
- FR-2.1: The system shall allow administrators to create new employee records with name, email, role, and status
- FR-2.2: The system shall validate that employee names are not empty and emails are unique
- FR-2.3: The system shall prevent duplicate employee records based on email address

- FR-2.4: The system shall store employee records in Supabase database with created_by_user_id foreign key
- FR-2.5: The system shall allow administrators to edit employee information (name, email, role)
- FR-2.6: The system shall allow administrators to deactivate employees (soft delete) rather than permanent deletion
- FR-2.7: The system shall automatically populate employee dropdowns with active employees only
- FR-2.8: The system shall display employee list in alphabetical order by name
- FR-2.9: The system shall prevent deletion of employees with existing timesheet records
- FR-2.10: The system shall sync employee data in real-time across all connected clients using Supabase Realtime

3.3. Feature 3: Clock In/Clock Out Operations

Description:
The system shall enable precise recording of employee work sessions through clock-in and clock-out functionality with automatic time calculation and validation.
Functional Requirements:

- FR-3.1: The system shall require employee selection before enabling clock-in button
- FR-3.2: The system shall record server timestamp when clock-in button is clicked to prevent client-side manipulation
- FR-3.3: The system shall create an active record in database with status 'clocked_in' and null clock_out_time
- FR-3.4: The system shall prevent employees from clocking in multiple times without clocking out
- FR-3.5: The system shall display alert if employee attempts to clock in while already clocked in
- FR-3.6: The system shall enable clock-out button only for employees with active clock-in records
- FR-3.7: The system shall record server timestamp when clock-out button is clicked
- FR-3.8: The system shall calculate hours worked using formula: (clock_out_time - clock_in_time)/3600 seconds
- FR-3.9: The system shall update record status to 'completed' upon successful clock-out
- FR-3.10: The system shall display hours worked rounded to two decimal places
- FR-3.11: The system shall format timestamps in user's local timezone (MM/DD/YYYY HH:MM:SS AM/PM)
- FR-3.12: The system shall validate that clock-out time is after clock-in time
- FR-3.13: The system shall notify users upon successful clock-in/out operations
- FR-3.14: The system shall support manual clock-in/out by administrators with reason field

3.4. Feature 4: Timesheet Records Management

Description:
The system shall provide comprehensive viewing, searching, filtering, and exporting of timesheet records with role-based data access.
Functional Requirements:

- FR-4.1: The system shall display timesheet records in paginated table with columns: Employee, Clock In, Clock Out, Hours Worked, Status

- FR-4.2: The system shall load records with pagination (25 records per page by default)
- FR-4.3: The system shall provide search functionality filtering by employee name (case-insensitive)
- FR-4.4: The system shall update search results in real-time as user types (debounced by 300ms)
- FR-4.5: The system shall allow filtering by date range (start date to end date)
- FR-4.6: The system shall allow filtering by status (clocked_in, completed, all)
- FR-4.7: The system shall display only employee's own records for standard users
- FR-4.8: The system shall display all organization records for administrators
- FR-4.9: The system shall sort records by clock-in time (most recent first) by default
- FR-4.10: The system shall allow sorting by any column (employee name, clock in, clock out, hours)
- FR-4.11: The system shall export filtered records to CSV format
- FR-4.12: The system shall export filtered records to PDF format (future)
- FR-4.13: The system shall cache records on mobile for offline viewing
- FR-4.14: The system shall sync new records automatically using Supabase Realtime subscriptions

3.5. Feature 5: Employee Management

Description:
The system shall generate aggregated summaries and reports showing total hours worked per employee with various time period groupings.
Functional Requirements:

- FR-5.1: The system shall calculate total hours worked for each employee across all completed records
- FR-5.2: The system shall display summary grouped by employee name
- FR-5.3: The system shall allow summary filtering by date range (daily, weekly, monthly, custom)
- FR-5.4: The system shall update summaries automatically when new timesheet records are added
- FR-5.5: The system shall display total hours rounded to two decimal places
- FR-5.6: The system shall format summary as: "Employee Name: XX.XX hrs"
- FR-5.7: The system shall show zero hours for employees with no completed records in selected period
- FR-5.8: The system shall calculate average hours per day for each employee
- FR-5.9: The system shall identify employees exceeding configured weekly hour limits
- FR-5.10: The system shall generate visual charts (bar/line graphs) for hour trends
- FR-5.11: The system shall export summary reports to PDF with company branding
- FR-5.12: The system shall send automated email reports to administrators (scheduled weekly/monthly)

3.6. Feature 6: Employee Management

Description:
The system shall provide modern, responsive, and accessible user interfaces optimized for web browsers and Android mobile devices.
Functional Requirements:

- FR-6.1: The web application shall display header with logo, user info, and logout button

- FR-6.2: The web application shall organize content in three responsive columns (desktop ≥1024px)
- FR-6.3: The web application shall stack sections vertically on tablet and mobile (<1024px)
- FR-6.4: The system shall use consistent color scheme throughout application (primary: #007bff, background: #f4f4f4)
- FR-6.5: The system shall provide hover effects and visual feedback for all interactive elements
- FR-6.6: The system shall disable buttons with visual indication when action is not applicable
- FR-6.7: The system shall display loading spinners during API requests
- FR-6.8: The system shall show toast notifications for success/error messages
- FR-6.9: The mobile app shall follow Material Design guidelines for Android
- FR-6.10: The system shall maintain readable font sizes (minimum 16px/14sp) across all devices
- FR-6.11: The system shall support light/dark mode themes (future)
- FR-6.12: The system shall meet WCAG 2.1 Level AA accessibility standards

4. Non-Functional Requirements

4.1. Performance Requirements

- NFR-1.1: The React web application shall load initial page within 3 seconds on 4G connection
- NFR-1.2: The Spring Boot API shall respond to requests within 200 milliseconds for 95% of requests
- NFR-1.3: The database shall support at least 100 concurrent users without performance degradation
- NFR-1.4: The system shall handle 10,000+ timesheet records with pagination and indexing
- NFR-1.5: Search and filtering operations shall complete within 500 milliseconds
- NFR-1.6: Real-time updates shall propagate to all clients within 1 second
- NFR-1.7: The mobile app shall launch within 2 seconds on mid-range Android devices

4.2. Security Requirements

- NFR-2.1: All communications shall use HTTPS/TLS 1.3 encryption
- NFR-2.2: Passwords shall be hashed using bcrypt with salt rounds of 12 or Argon2
- NFR-2.3: JWT tokens shall expire after 1 hour (access token) and 7 days (refresh token)
- NFR-2.4: The system shall implement Row Level Security (RLS) in Supabase database
- NFR-2.5: API endpoints shall validate and sanitize all inputs to prevent SQL injection and XSS attacks
- NFR-2.6: The system shall implement CORS policies restricting API access to authorized domains
- NFR-2.7: The system shall log all authentication events and failed access attempts
- NFR-2.8: Sensitive data shall be masked in logs (passwords, tokens)
- NFR-2.9: The system shall implement rate limiting (100 requests/minute per user)
- NFR-2.10: Database connections shall use connection pooling with encrypted credentials

4.3. Usability Requirements

- NFR-3.1: New users shall complete registration within 2 minutes without assistance
- NFR-3.2: Employees shall clock in/out within 3 clicks from dashboard
- NFR-3.3: Error messages shall be clear, actionable, and user-friendly (no technical jargon)
- NFR-3.4: The system shall provide consistent navigation patterns across all pages
- NFR-3.5: Form validation shall provide real-time feedback as users type
- NFR-3.6: The system shall remember user preferences (theme, language, default filters)
- NFR-3.7: Help tooltips shall be available for complex features

4.4. Reliability Requirements

- NFR-4.1: The system shall have 99.5% uptime during business hours (6 AM - 10 PM)
- NFR-4.2: Database backups shall occur automatically every 6 hours with 30-day retention
- NFR-4.3: The system shall recover from failures within 5 minutes using health checks and auto-restart
- NFR-4.4: Data integrity shall be maintained through ACID-compliant transactions
- NFR-4.5: The system shall validate all calculations server-side to prevent client manipulation

4.5. Scalability Requirements

- NFR-5.1: The architecture shall support horizontal scaling to handle 10,000+ users
- NFR-5.2: The database shall support read replicas for load distribution
- NFR-5.3: The Spring Boot backend shall support containerization and Kubernetes orchestration
- NFR-5.4: The system shall implement caching (Redis) for frequently accessed data

4.6. Maintainability Requirements

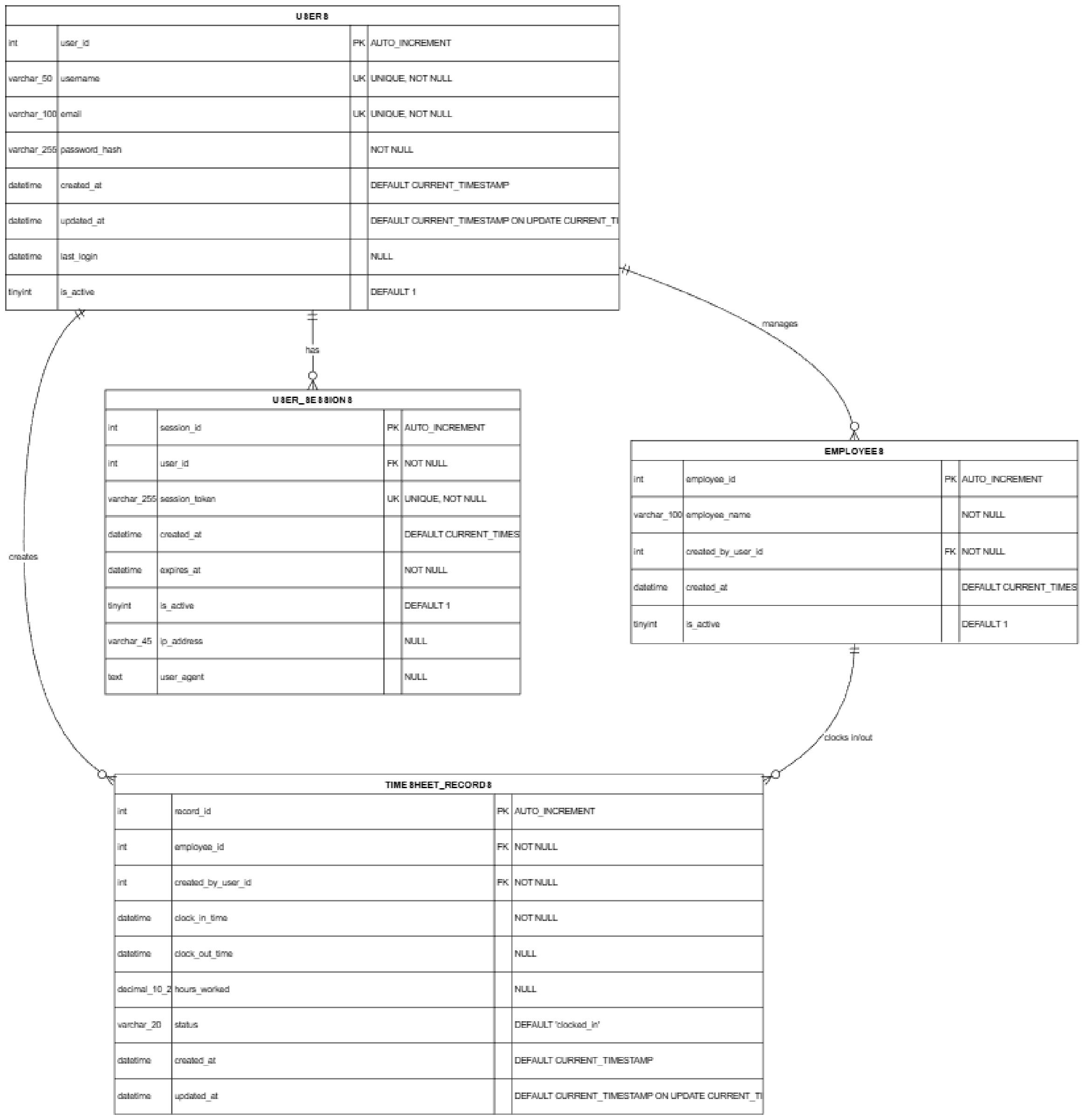
- NFR-6.1: Source code shall follow language-specific style guides (Airbnb for React, Kotlin official style guide)
- NFR-6.2: Code shall maintain minimum 80% unit test coverage
- NFR-6.3: API documentation shall be auto-generated using Swagger/OpenAPI
- NFR-6.4: The system shall use semantic versioning for releases
- NFR-6.5: CI/CD pipelines shall automate testing, building, and deployment

4.7. Compatibility Requirements

- NFR-7.1: Web application shall function on Chrome, Firefox, Safari, Edge (latest 2 versions)
- NFR-7.2: Mobile application shall support Android 8.0+ (covering 95% of active devices)
- NFR-7.3: The system shall be compatible with screen readers and assistive technologies
- NFR-7.4: API versioning shall maintain backward compatibility for at least 2 major versions

5. System Models (Diagrams)

5.1. ERD



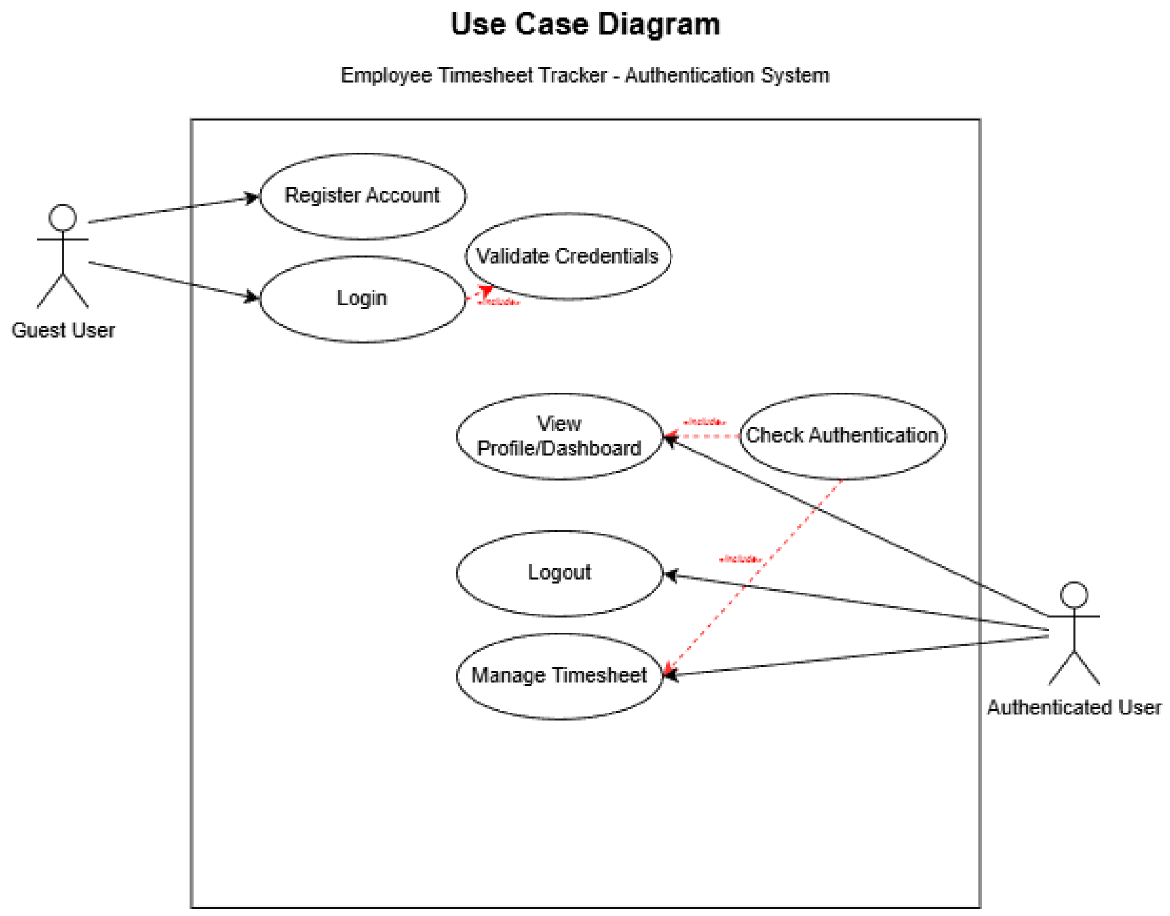
USERS			
int	user_id	PK	AUTO_INCREMENT
varchar_50	username	UK	UNIQUE, NOT NULL
varchar_100	email	UK	UNIQUE, NOT NULL
varchar_255	password_hash		NOT NULL
datetime	created_at		DEFAULT CURRENT_TIMESTAMP
datetime	updated_at		DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TI
datetime	last_login		NULL
tinyint	is_active		DEFAULT 1

USER_SESSIONS			
int	session_id	PK	AUTO_INCREMENT
int	user_id	FK	NOT NULL
varchar_255	session_token	UK	UNIQUE, NOT NULL
datetime	created_at		DEFAULT CURRENT_TIMES
datetime	expires_at		NOT NULL
tinyint	is_active		DEFAULT 1
varchar_45	ip_address		NULL
text	user_agent		NULL

EMPLOYEES			
int	employee_id	PK	AUTO_INCREMENT
varchar_100	employee_name		NOT NULL
int	created_by_user_id	FK	NOT NULL
datetime	created_at		DEFAULT CURRENT_TIMES
tinyint	is_active		DEFAULT 1

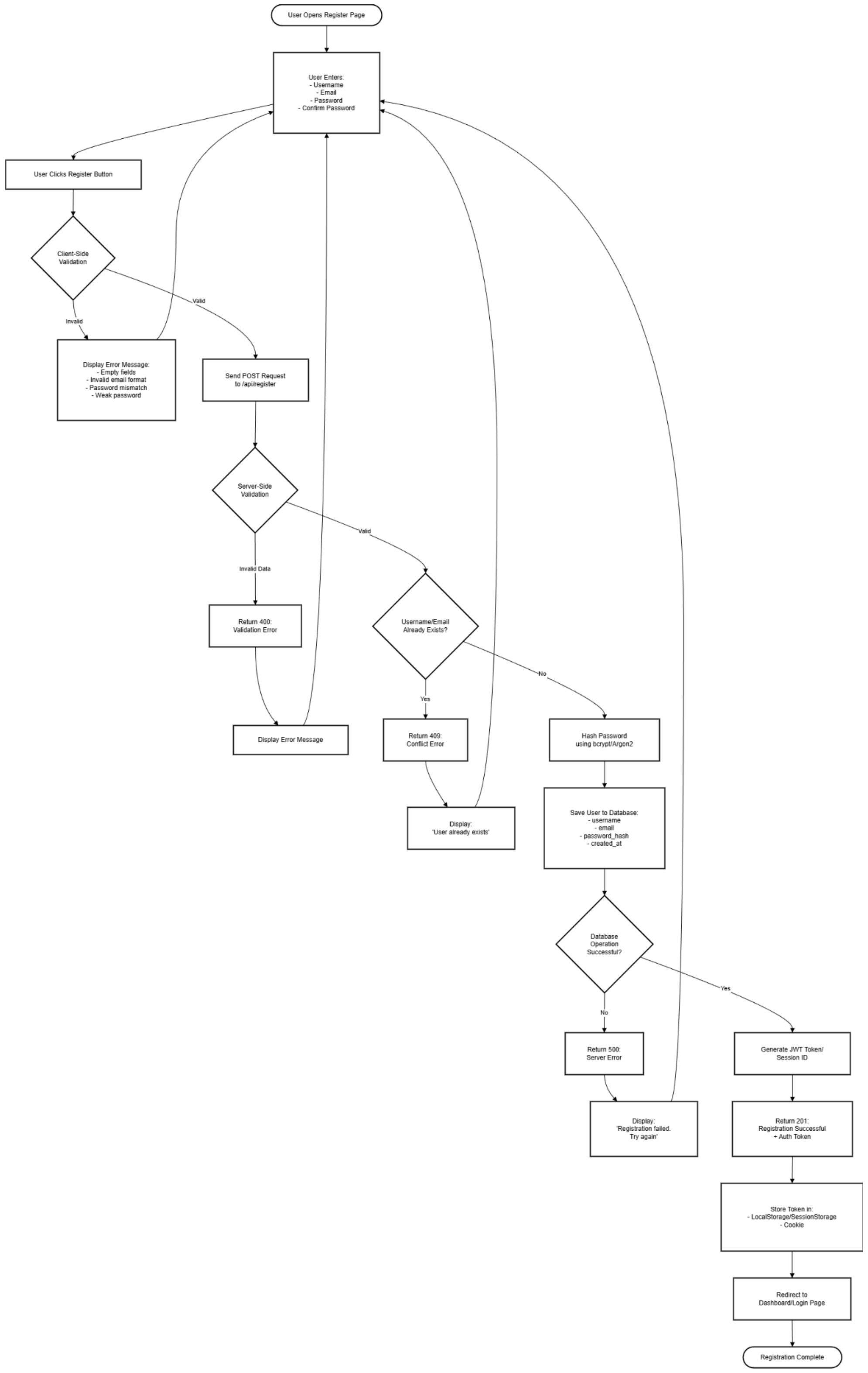
TIMESHEET_RECORDS			
int	record_id	PK	AUTO_INCREMENT
int	employee_id	FK	NOT NULL
int	created_by_user_id	FK	NOT NULL
datetime	clock_in_time		NOT NULL
datetime	clock_out_time		NULL
decimal_10_2	hours_worked		NULL
varchar_20	status		DEFAULT 'clocked_in'
datetime	created_at		DEFAULT CURRENT_TIMESTAMP
datetime	updated_at		DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TI

5.2. Use Case Diagram

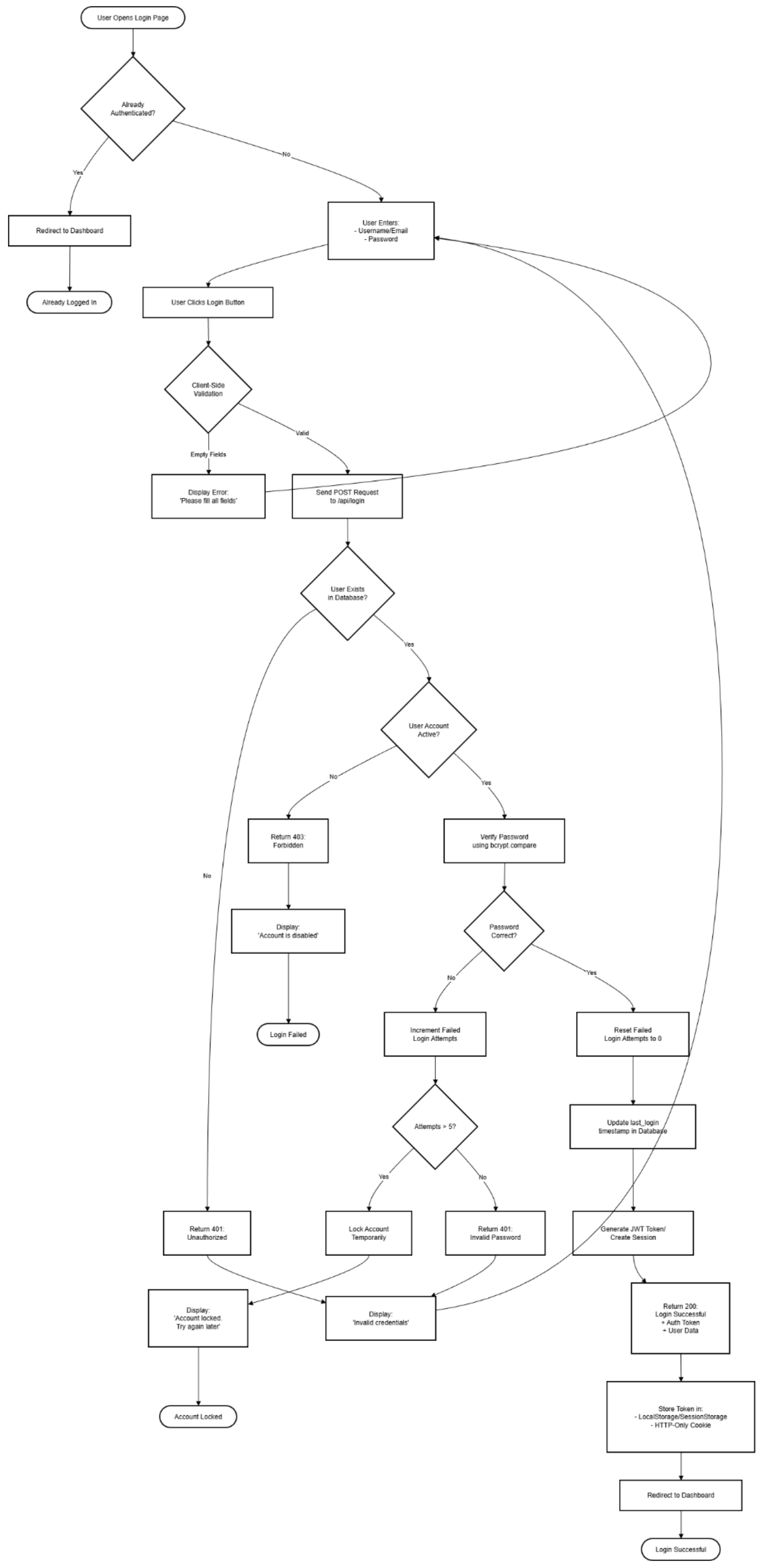


5.3. Activity Diagram

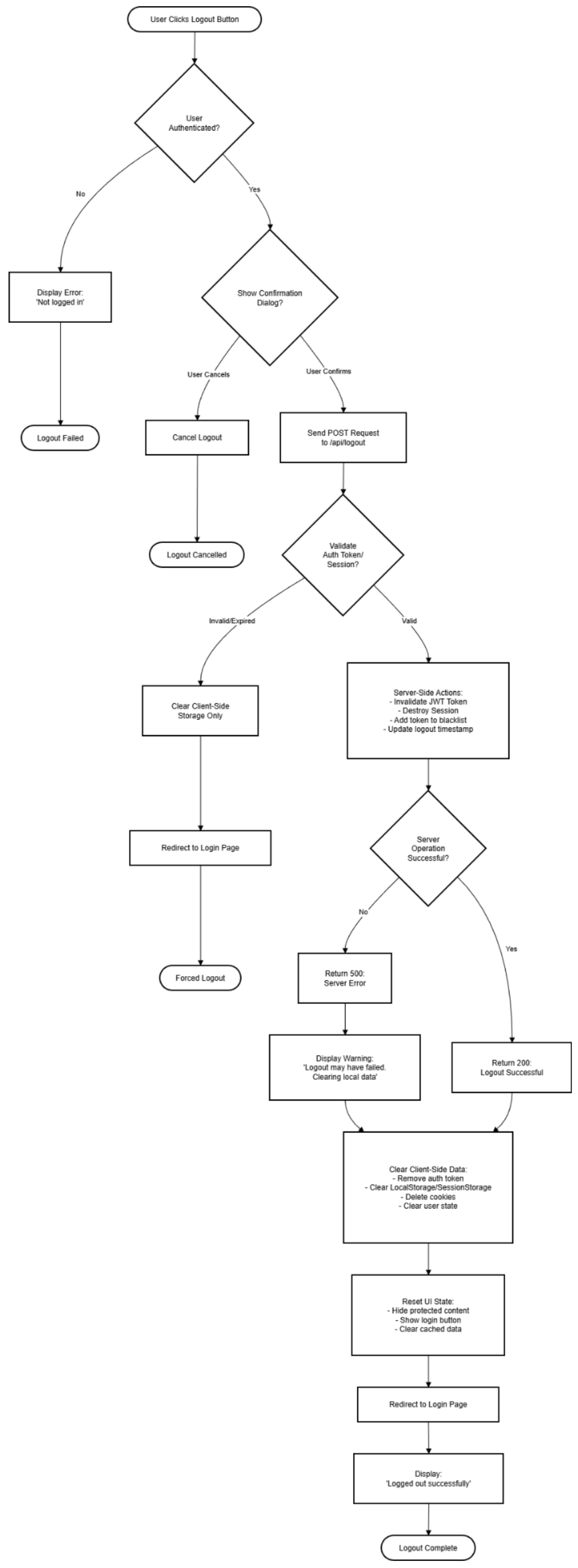
Register:


User Opens Register Page
User Enters:
- Username
- Email
- Password
- Confirm Password
User Clicks Register Button
Client-Side Validation
Invalid
Valid
Display Error Message:
- Empty fields
- Invalid email format
- Password mismatch
- Weak password
Send POST Request to /api/register
Server-Side Validation
Invalid Data
Valid
Return 400: Validation Error
Display Error Message
Username/Email Already Exists?
Yes
No
Return 409: Conflict Error
Display: 'User already exists'
Hash Password using bcrypt/Argon2
Save User to Database:
- username
- email
- password_hash
- created_at
Database Operation Successful?
No
Yes
Return 500: Server Error
Display: 'Registration failed. Try again'
Generate JWT Token/ Session ID
Return 201: Registration Successful + Auth Token
Store Token in:
- LocalStorage/SessionStorage
- Cookie
Redirect to Dashboard/Login Page
Registration Complete

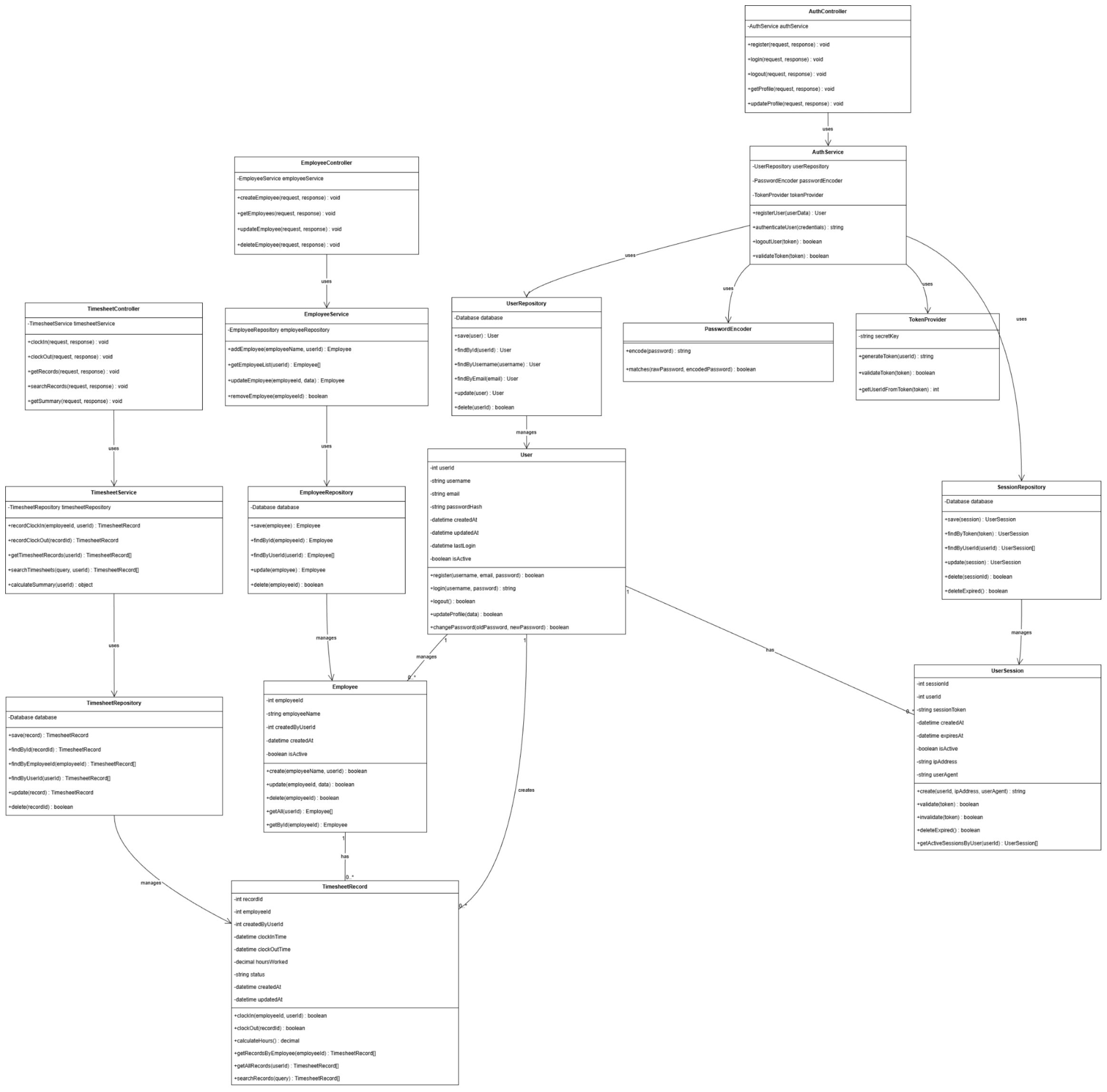
Login:


User Opens Login Page
Already Authenticated?
Yes
No
Redirect to Dashboard
Already Logged In
User Enters: - Username/Email - Password
User Clicks Login Button
Client-Side Validation
Empty Fields
Valid
Display Error: 'Please fill all fields'
Send POST Request to /api/login
User Exists in Database?
Yes
No
User Account Active?
No
Yes
Return 403: Forbidden
Verify Password using bcrypt.compare
Display: 'Account is disabled'
Password Correct?
No
Yes
Login Failed
Increment Failed Login Attempts
Reset Failed Login Attempts to 0
Attempts > 5?
Yes
No
Update last_login timestamp in Database
Return 401: Unauthorized
Lock Account Temporarily
Return 401: Invalid Password
Generate JWT Token/ Create Session
Display: 'Account locked. Try again later'
Display: 'Invalid credentials'
Return 200: Login Successful + Auth Token + User Data
Account Locked
Store Token in: - LocalStorage/SessionStorage - HTTP-Only Cookie
Redirect to Dashboard
Login Successful

Logout:

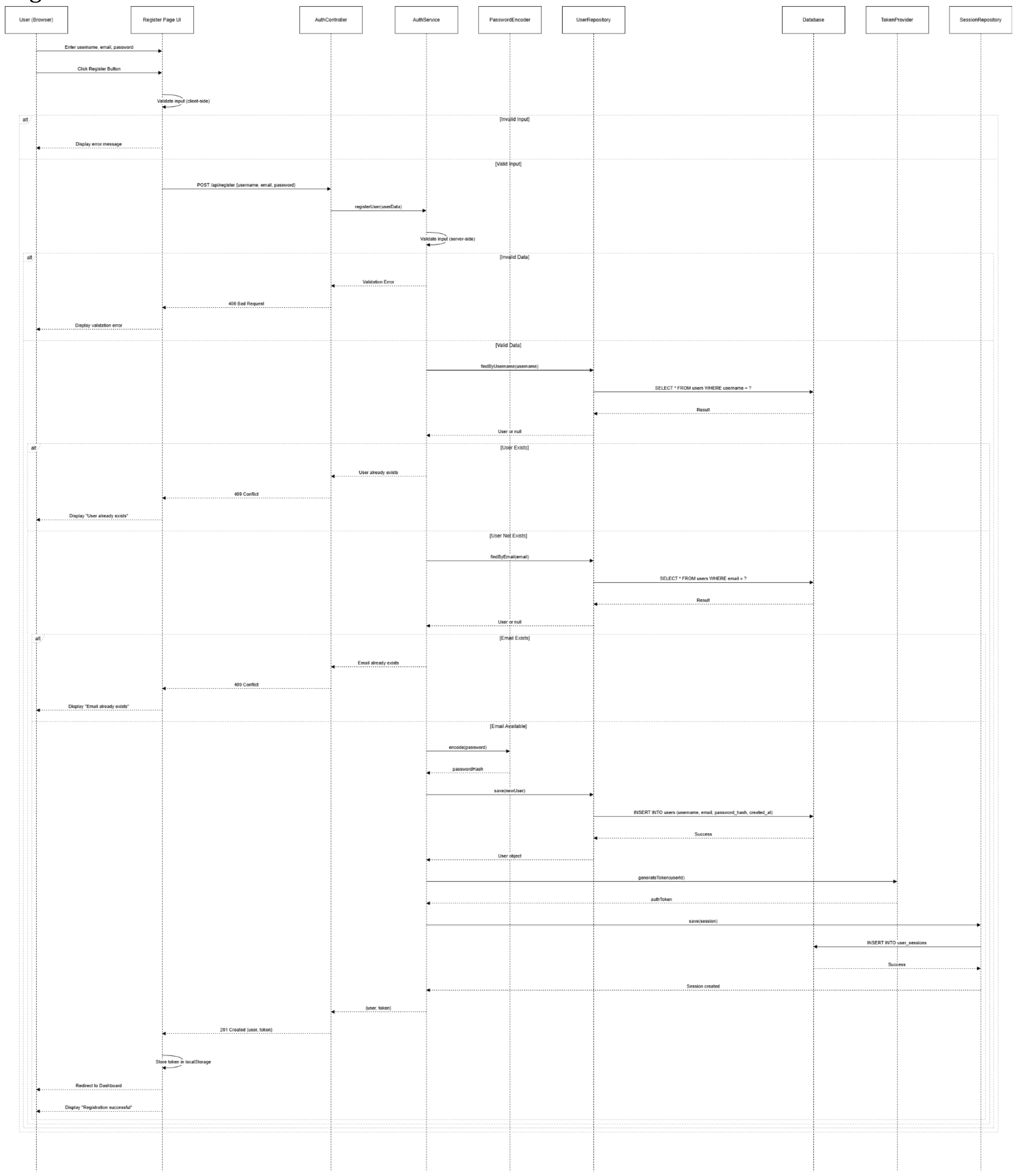

User Clicks Logout Button
User Authenticated?
No
Yes
Display Error: 'Not logged in'
Logout Failed
Show Confirmation Dialog?
User Cancels
User Confirms
Cancel Logout
Logout Cancelled
Send POST Request to /api/logout
Validate Auth Token/ Session?
Invalid/Expired
Valid
Clear Client-Side Storage Only
Redirect to Login Page
Forced Logout
Server-Side Actions:
- Invalidate JWT Token
- Destroy Session
- Add token to blacklist
- Update logout timestamp
Server Operation Successful?
No
Yes
Return 500: Server Error
Display Warning: 'Logout may have failed. Clearing local data'
Return 200: Logout Successful
Clear Client-Side Data:
- Remove auth token
- Clear LocalStorage/SessionStorage
- Delete cookies
- Clear user state
Reset UI State:
- Hide protected content
- Show login button
- Clear cached data
Redirect to Login Page
Display: 'Logged out successfully'
Logout Complete

5.4. Class Diagram

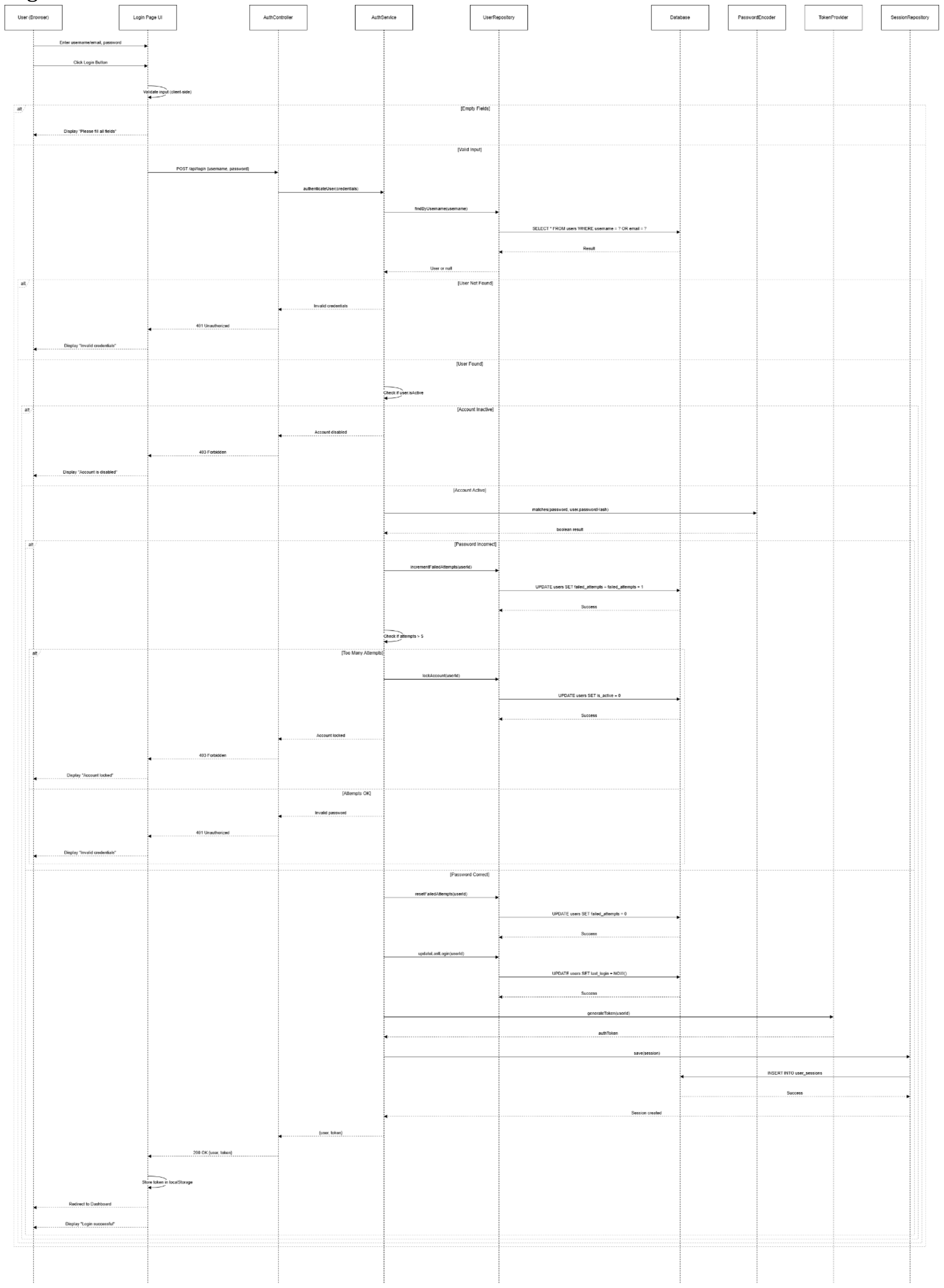

AuthController
-AuthService authService
+register(request, response) : void
+login(request, response) : void
+logout(request, response) : void
+getProfile(request, response) : void
+updateProfile(request, response) : void
uses
AuthService
-UserRepository userRepository
-PasswordEncoder passwordEncoder
-TokenProvider tokenProvider
+registerUser(userData) : User
+authenticateUser(credentials) : string
+logoutUser(token) : boolean
+validateToken(token) : boolean
uses
uses
uses
uses
EmployeeController
-EmployeeService employeeService
+createEmployee(request, response) : void
+getEmployees(request, response) : void
+updateEmployee(request, response) : void
+deleteEmployee(request, response) : void
uses
TimesheetController
-TimesheetService timesheetService
+clockIn(request, response) : void
+clockOut(request, response) : void
+getRecords(request, response) : void
+searchRecords(request, response) : void
+getSummary(request, response) : void
EmployeeService
-EmployeeRepository employeeRepository
+addEmployee(employeeName, userId) : Employee
+getEmployeeList(userId) : Employee[]
+updateEmployee(employeeId, data) : Employee
+removeEmployee(employeeId) : boolean
UserRepository
-Database database
+save(user) : User
+findById(userId) : User
+findByUsername(username) : User
+findByEmail(email) : User
+update(user) : User
+delete(userId) : boolean
PasswordEncoder
+encode(password) : string
+matches(rawPassword, encodedPassword) : boolean
TokenProvider
-string secretKey
+generateToken(userId) : string
+validateToken(token) : boolean
+getUserIdFromToken(token) : int
uses
uses
manages
User
-int userId
-string username
-string email
-string passwordHash
-datetime createdAt
-datetime updatedAt
-datetime lastLogin
-boolean isActive
+register(username, email, password) : boolean
+login(username, password) : string
+logout() : boolean
+updateProfile(data) : boolean
+changePassword(oldPassword, newPassword) : boolean
TimesheetService
-TimesheetRepository timesheetRepository
+recordClockIn(employeeId, userId) : TimesheetRecord
+recordClockOut(recordId) : TimesheetRecord
+getTimesheetRecords(userId) : TimesheetRecord[]
+searchTimesheets(query, userId) : TimesheetRecord[]
+calculateSummary(userId) : object
EmployeeRepository
-Database database
+save(employee) : Employee
+findById(employeeId) : Employee
+findByUserId(userId) : Employee[]
+update(employee) : Employee
+delete(employeeId) : boolean
SessionRepository
-Database database
+save(session) : UserSession
+findByToken(token) : UserSession
+findByUserId(userId) : UserSession[]
+update(session) : UserSession
+delete(sessionId) : boolean
+deleteExpired() : boolean
uses
manages
1
1
manages
0..*
1
has
0..*
manages
Employee
-int employeeId
-string employeeName
-int createdByUserId
-datetime createdAt
-boolean isActive
+create(employeeName, userId) : boolean
+update(employeeId, data) : boolean
+delete(employeeId) : boolean
+getAll(userId) : Employee[]
+getById(employeeId) : Employee
TimesheetRepository
-Database database
+save(record) : TimesheetRecord
+findById(recordId) : TimesheetRecord
+findByEmployeeId(employeeId) : TimesheetRecord[]
+findByUserId(userId) : TimesheetRecord[]
+update(record) : TimesheetRecord
+delete(recordId) : boolean
UserSession
-int sessionId
-int userId
-string sessionToken
-datetime createdAt
-datetime expiresAt
-boolean isActive
-string ipAddress
-string userAgent
+create(userId, ipAddress, userAgent) : string
+validate(token) : boolean
+invalidate(token) : boolean
+deleteExpired() : boolean
+getActiveSessionsByUser(userId) : UserSession[]
creates
1
has
0..*
manages
0..*
TimesheetRecord
-int recordId
-int employeeId
-int createdByUserId
-datetime clockInTime
-datetime clockOutTime
-decimal hoursWorked
-string status
-datetime createdAt
-datetime updatedAt
+clockIn(employeeId, userId) : boolean
+clockOut(recordId) : boolean
+calculateHours() : decimal
+getRecordsByEmployee(employeeId) : TimesheetRecord[]
+getAllRecords(userId) : TimesheetRecord[]
+searchRecords(query) : TimesheetRecord[]

5.5. Sequence Diagram

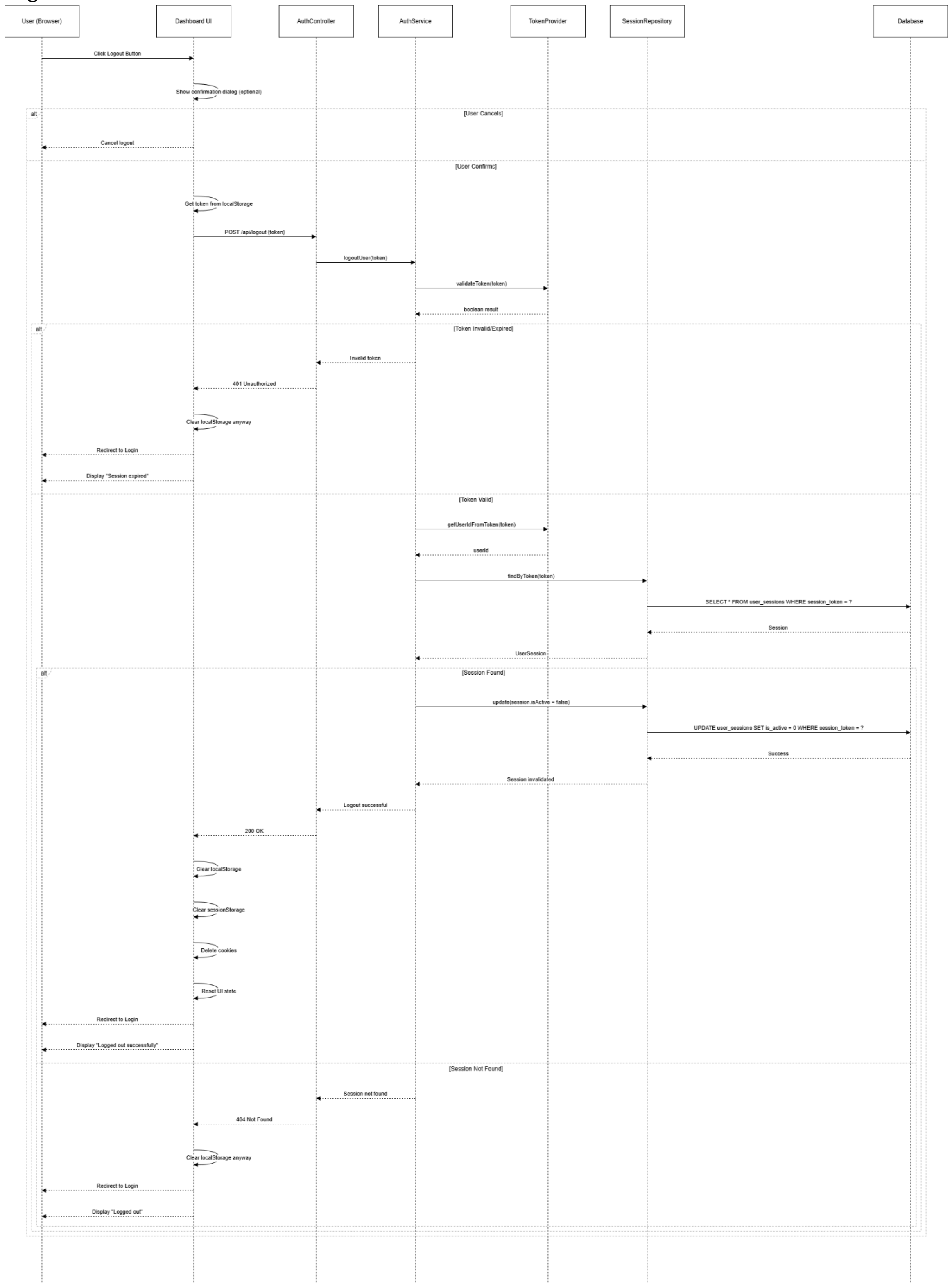
Register:



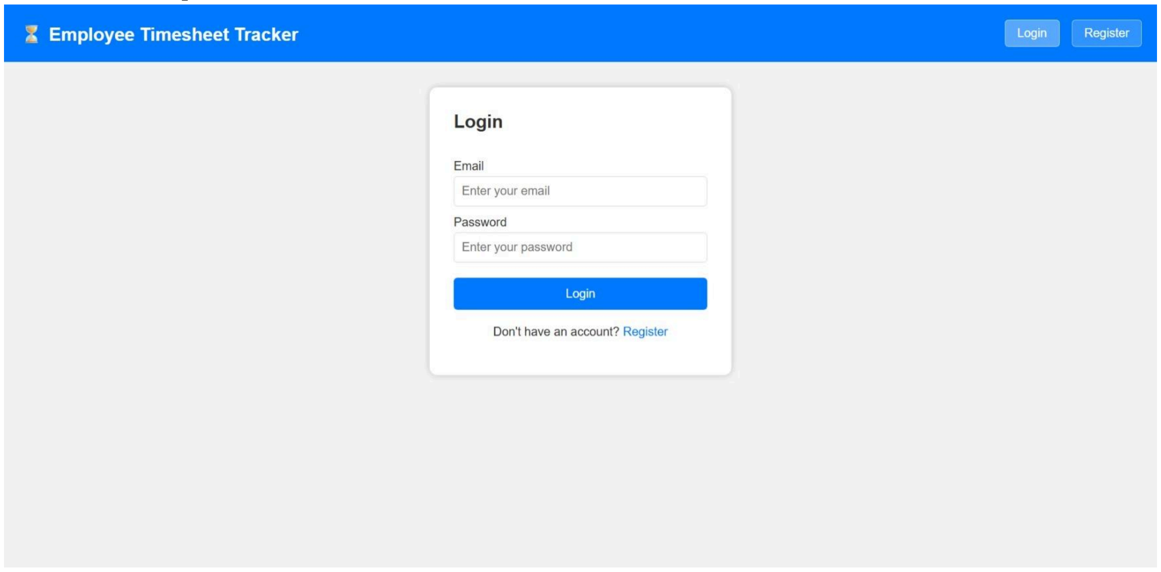
Login:

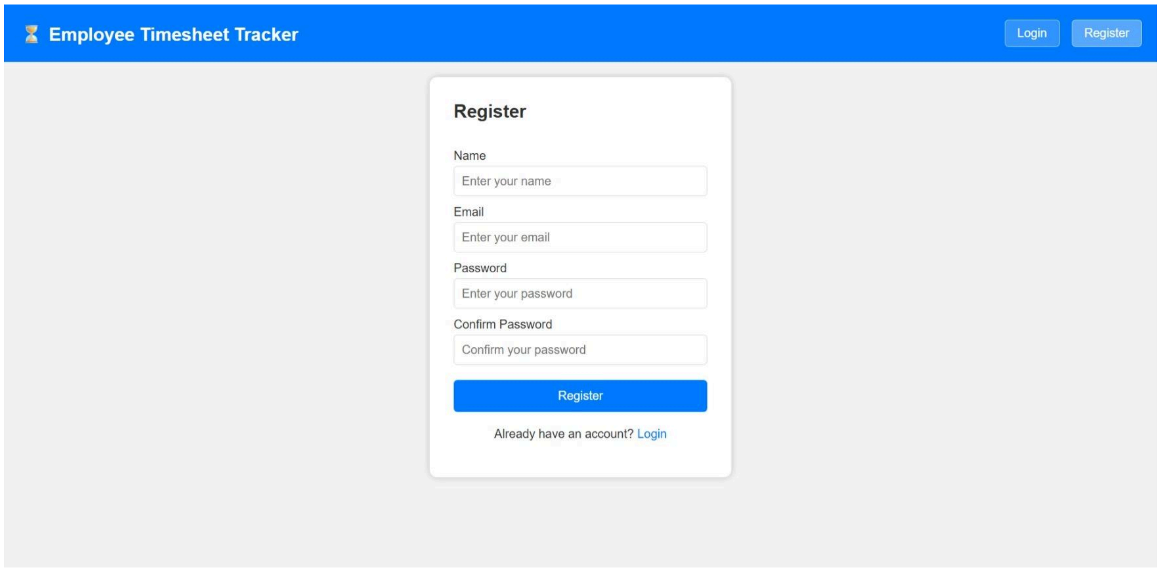

User (Browser)
Login Page UI
AuthController
AuthService
UserRepository
Database
PasswordEncoder
TokenProvider
SessionRepository
Enter username/email, password
Click Login Button
Validate input (client-side)
alt
[Empty Fields]
Display "Please fill all fields"
[Valid Input]
POST /api/login {username, password}
authenticateUser(credentials)
findByUsername(username)
SELECT * FROM users WHERE username = ? OR email = ?
Result
User or null
alt
[User Not Found]
Invalid credentials
401 Unauthorized
Display "Invalid credentials"
[User Found]
Check if user.isActive
alt
[Account Inactive]
Account disabled
403 Forbidden
Display "Account is disabled"
[Account Active]
matches(password, user.passwordHash)
boolean result
alt
[Password Incorrect]
incrementFailedAttempts(userId)
UPDATE users SET failed_attempts = failed_attempts + 1
Success
Check if attempts > 5
alt
[Too Many Attempts]
lockAccount(userId)
UPDATE users SET is_active = 0
Success
Account locked
403 Forbidden
Display "Account locked"
[Attempts OK]
Invalid password
401 Unauthorized
Display "Invalid credentials"
[Password Correct]
resetFailedAttempts(userId)
UPDATE users SET failed_attempts = 0
Success
updateLastLogin(userId)
UPDATE users SET last_login = NOW()
Success
generateToken(userId)
authToken
save(session)
INSERT INTO user_sessions
Success
Session created
{user, token}
200 OK {user, token}
Store token in localStorage
Redirect to Dashboard
Display "Login successful"

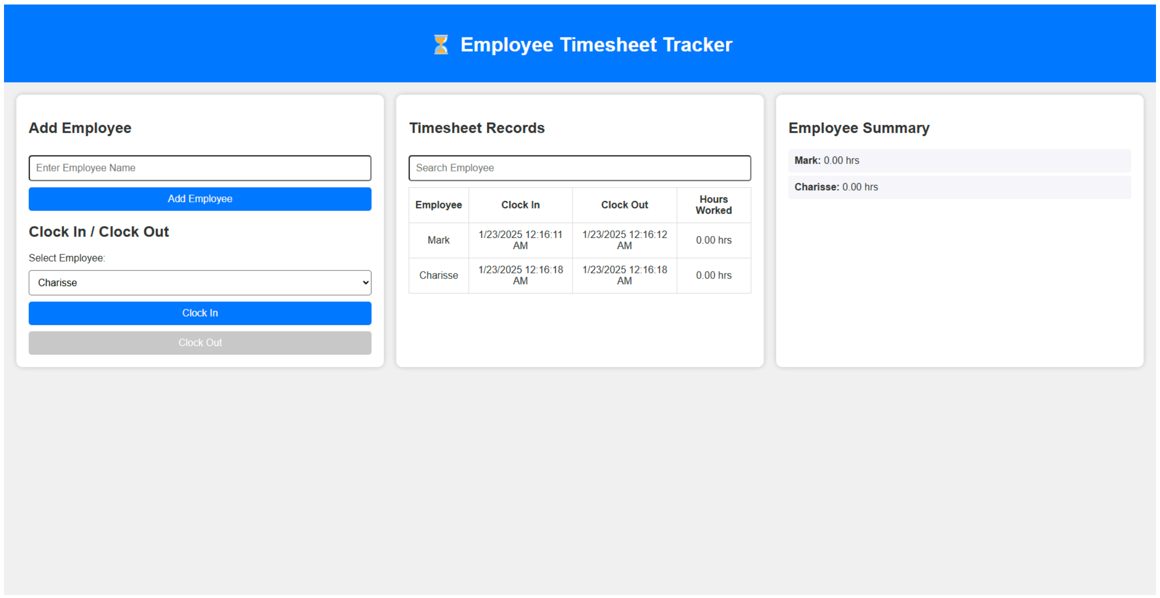
Logout:


User (Browser)
Dashboard UI
AuthController
AuthService
TokenProvider
SessionRepository
Database
Click Logout Button
Show confirmation dialog (optional)
alt
[User Cancels]
Cancel logout
[User Confirms]
Get token from localStorage
POST /api/logout (token)
logoutUser(token)
validateToken(token)
boolean result
alt
[Token Invalid/Expired]
Invalid token
401 Unauthorized
Clear localStorage anyway
Redirect to Login
Display "Session expired"
[Token Valid]
getUserIdFromToken(token)
userId
findByToken(token)
SELECT * FROM user_sessions WHERE session_token = ?
Session
UserSession
alt
[Session Found]
update(session.isActive = false)
UPDATE user_sessions SET is_active = 0 WHERE session_token = ?
Success
Session invalidated
Logout successful
200 OK
Clear localStorage
Clear sessionStorage
Delete cookies
Reset UI state
Redirect to Login
Display "Logged out successfully"
[Session Not Found]
Session not found
404 Not Found
Clear localStorage anyway
Redirect to Login
Display "Logged out"

Web UI Mockup:







⏳ Employee Timesheet Tracker

Add Employee

Enter Employee Name

Add Employee

Clock In / Clock Out

Select Employee:

Charisse

Clock In

Clock Out

Timesheet Records

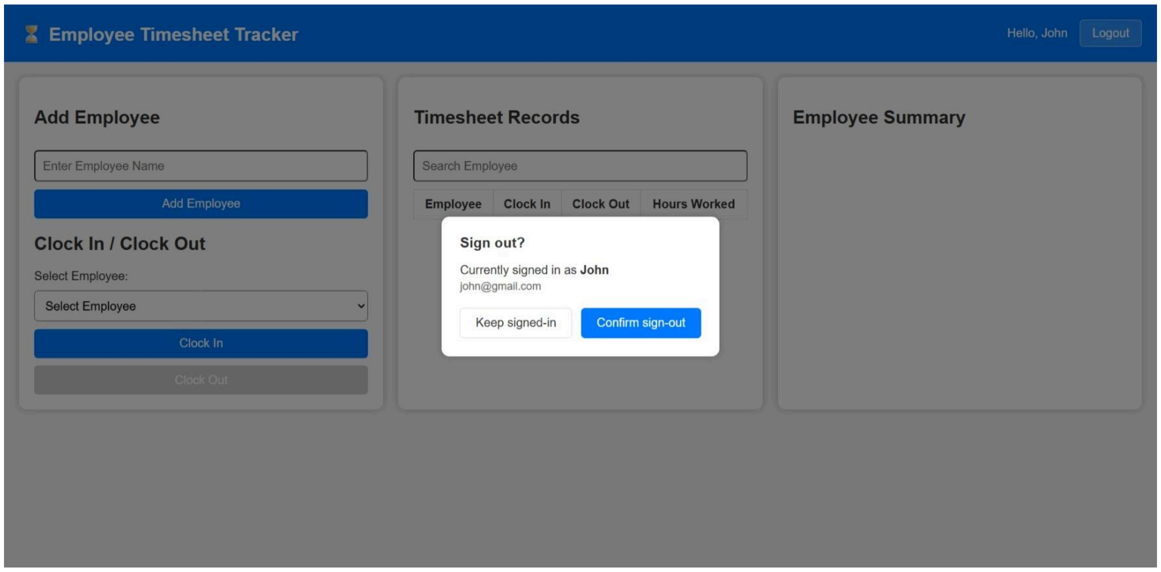
Search Employee

Employee	Clock In	Clock Out	Hours Worked
Mark	1/23/2025 12:16:11 AM	1/23/2025 12:16:12 AM	0.00 hrs
Charisse	1/23/2025 12:16:18 AM	1/23/2025 12:16:18 AM	0.00 hrs

Employee Summary

Mark: 0.00 hrs

Charisse: 0.00 hrs



⏳ Employee Timesheet Tracker

Hello, John Logout

Add Employee

Enter Employee Name

Add Employee

Clock In / Clock Out

Select Employee:

Select Employee

Clock In

Clock Out

Timesheet Records

Search Employee

Employee	Clock In	Clock Out	Hours Worked

Employee Summary

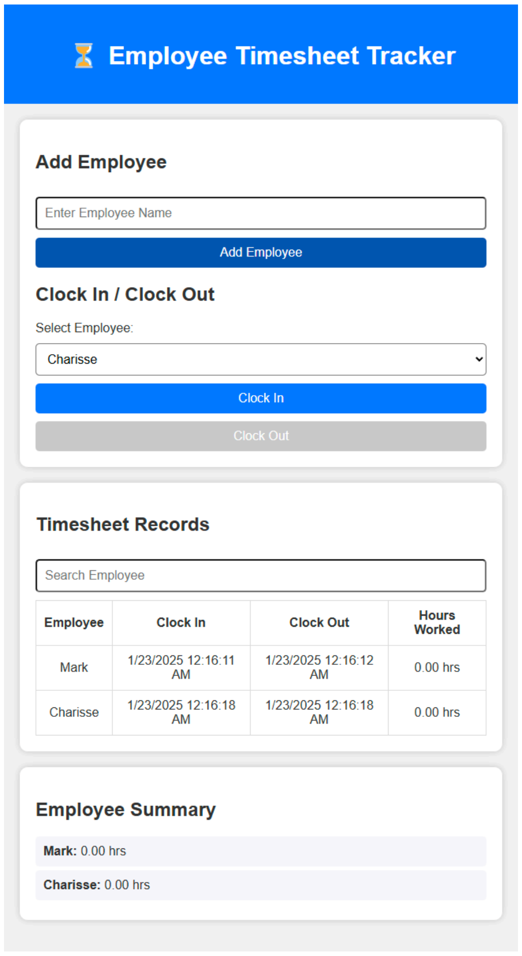
Sign out?

Currently signed in as **John**

john@gmail.com

Keep signed-in Confirm sign-out

Mobile UI Mockup:



⌛ Employee Timesheet Tracker

Add Employee

Clock In / Clock Out

Select Employee:

Charisse

Timesheet Records

Employee	Clock In	Clock Out	Hours Worked
Mark	1/23/2025 12:16:11 AM	1/23/2025 12:16:12 AM	0.00 hrs
Charisse	1/23/2025 12:16:18 AM	1/23/2025 12:16:18 AM	0.00 hrs

Employee Summary

Mark: 0.00 hrs

Charisse: 0.00 hrs