A logo for a university

Description automatically generated

**Requirement Analysis**

**SmartSchedule**

**Scheduling Application for Small and Medium-Sized Businesses**

**Prepared by**

| Harry Zhao | 100338161 | xingrui.zhao@student.kpu.ca |
| --- | --- | --- |
| An Vu | 100377387 | an.vu@student.kpu.ca |
|  |  |  |

### **Introduction**

SmartSchedule is a scheduling and reservation application designed specifically for small and medium-sized businesses. It offers a streamlined solution for managing employee schedules while also providing an interface for customers booking appointments or reservations. With features like shift creation, time-off management, and real-time notifications, SmartSchedule enhances internal communication and operational efficiency. The platform is accessible through both desktop and mobile devices, making it suitable for various industries such as retail, hospitality, healthcare, and logistics.

### **Target User Groups and Usage**

* Managers:

Managers use SmartSchedule to create and adjust employee schedules, approve or deny shift swaps, and handle time-off requests. They also manage customer reservations, assign tasks based on business needs, and monitor employee availability. The app’s integration with payroll systems allows managers to track hours worked and export data for payroll processing, simplifying administrative tasks.

* Employees:

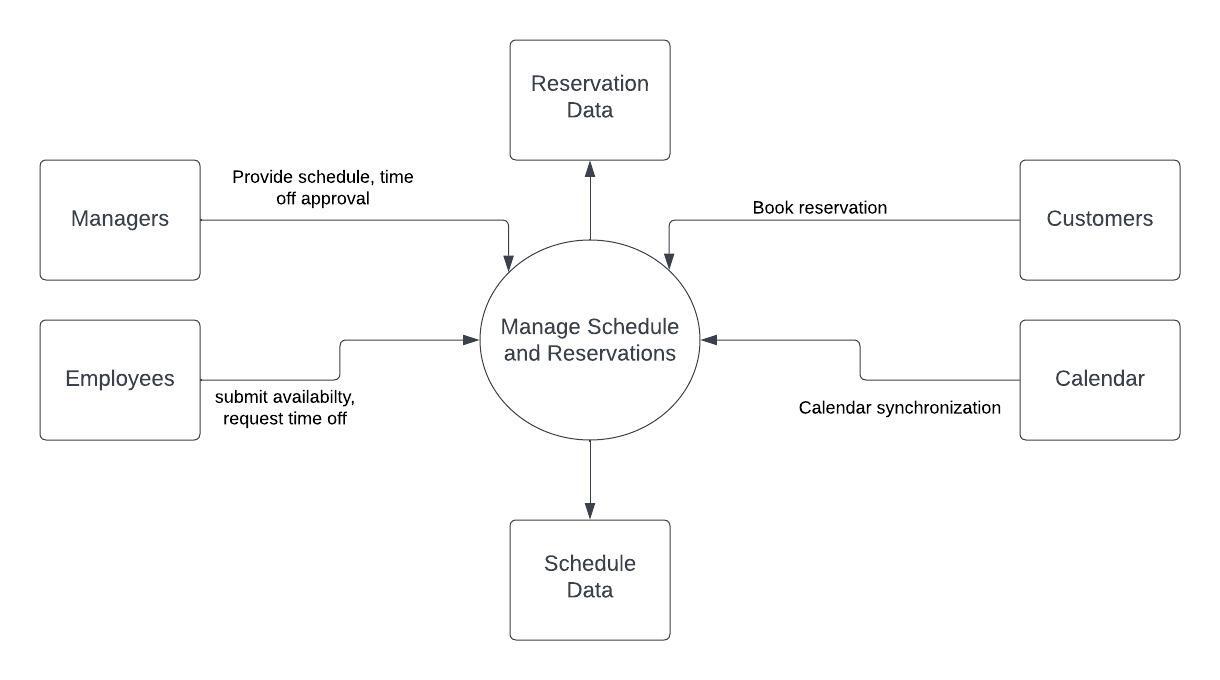
Employees use SmartSchedule to view their assigned shifts, submit availability, request time off, and swap shifts with their colleagues. They receive real-time notifications for upcoming shifts, schedule changes, and approved or denied requests. Employees primarily access the app through mobile devices for convenience, using it to stay updated and maintain clear communication with their managers.

* Customers:

Customers interact with the app through a separate reservation interface, where they can view available time slots, book appointments, and receive automated confirmations and reminders. Their interaction is focused solely on making and managing their reservations, with options to cancel or modify bookings as needed.

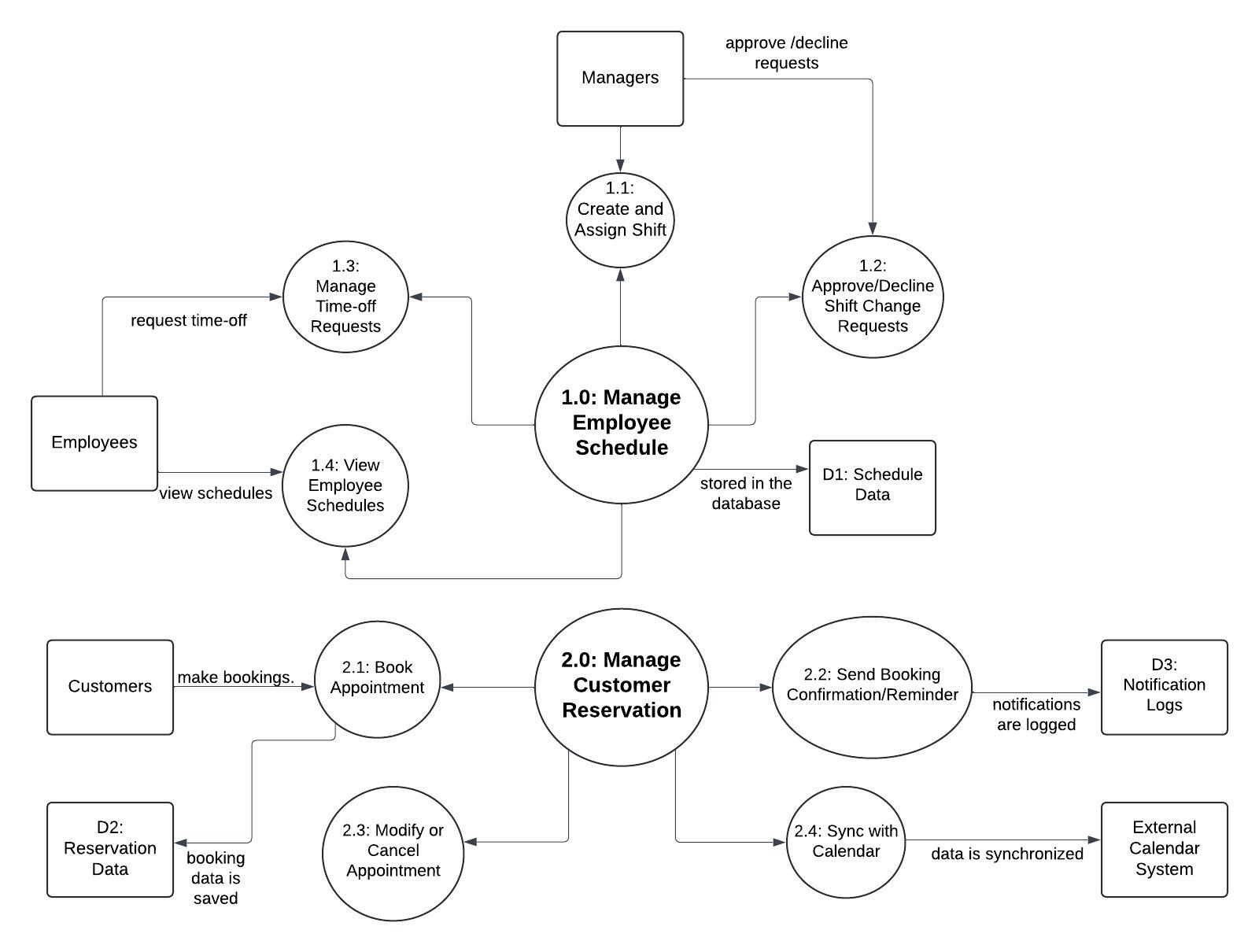
Each group has distinct interactions with the software, managers use it for oversight and scheduling, employees use it to manage personal work schedules, and customers use it for reservation purposes.

1. **Data Flow Diagram (DFD):**
2. **Level 0:**

****

**Figure 1**: Level 0 DFD

1. **Level 1 DFD:**

****

**Figure 2.0:** Level 1 DFD

1. **Functional Requirements:**
2. **User Authentication and Authorization**

* Users can create accounts and log in with secure credentials.
* Different user roles (e.g., admin, manager, employee, customer) with different levels of access.

1. **Employee Schedule Management**

* Create, edit, and assign employee shifts.
* Allow employees to view their schedules in real-time.
* Ability for employees to request shift changes or swaps.
* Managers can approve or decline shift change requests.

1. **Time-off Management**

* Employees can request time off (vacation, sick leave, etc.).
* Managers can approve or decline time-off requests.

1. **Customer Reservation System**

* Customers can book appointments or reservations through a user-friendly interface.
* Automated confirmations and reminders for bookings via email or SMS.
* Customers can cancel or modify reservations, subject to company policies.

1. **Notifications**

* Notify employees of schedule updates, shift changes, or cancellations.
* Notify managers of pending requests (time off, shift swaps).
* Send reminders for upcoming shifts or bookings.
* Notify customers of appointment confirmations, reminders, or changes.

1. **Shift Templates and Recurring Schedules**

* Create and save shift templates for regular schedules.
* Set up recurring shifts (e.g., weekly or monthly patterns).

1. **Availability Management**

* Employees can submit their availability preferences.
* Managers can view availability when creating schedules.

1. **Schedule Conflict Resolution**

* Automated detection of scheduling conflicts (e.g., double-booking or overlapping shifts).

1. **Accessibility**

* The platform is available on both desktop and mobile devices
* Ensure real-time synchronization between devices.

1. **User-friendly Interface**

* Intuitive dashboard for managers to easily view and adjust schedules.
* Easy-to-navigate booking system for customers.
* Simple mobile-friendly design for on-the-go access.

### **V. Non-Functional Requirements**

1. Response Time: The system should respond to user inputs (e.g., shift creation, reservation bookings, or time-off requests) within 3 seconds on average.
2. Scalability: The application should handle up to 300 concurrent users without a decrease in performance.
3. Data Processing: Employee scheduling data and customer reservations should be processed and saved in under 2 seconds per transaction.
4. Uptime: The system must maintain 99.5% uptime, ensuring continuous access for employees, managers, and customers.
5. Fault Tolerance: The system should have failover mechanisms in place to handle unexpected crashes or outages, such as database replication and backup servers.
6. Backup and Recovery: The system should automatically back up data every 24 hours, and full recovery must be possible within 2 hours of a failure.
7. Authentication: All user groups must authenticate using unique credentials (username and password) or Single Sign-On (SSO) if integrated with third-party identity providers.
8. Data Encryption: Employee and customer data must be encrypted both in transit and at rest.
9. Role-Based Access Control: Different user groups (managers, employees, customers) should have role-based permissions to ensure only authorized users can access sensitive information and functions.
10. Compliance: The system must comply with any industry-specific regulations relevant to the client.
11. User Interface Design: The application should have an intuitive UI that allows a new user to perform basic functions (e.g., view schedule, book a reservation) without requiring training.
12. Accessibility: The system must comply with WCAG 3.0 standards to support users with disabilities, including features like keyboard navigation and screen reader compatibility.
13. Multi-Language Support: The application should offer multilingual support for at least English, French, and Chinese to accommodate a broader user base.
14. Code Documentation: All source code should be well-documented to ensure ease of maintenance and future updates.
15. Modularity: The system should be built using a modular architecture to allow for easy addition of new features (e.g., expanding to advanced scheduling algorithms or more customer management features).
16. Version Control: A versioning system must be in place to manage updates and rollbacks.
17. Third-Party Integrations: The system should support integration with third-party applications such as payroll systems, customer relationship management (CRM) tools, and calendar applications using standard APIs.
18. Data Export/Import: The system must allow data export in CSV format and import of external data (e.g., existing employee schedules) to facilitate system adoption.
19. Horizontal and Vertical Scalability: The system should be designed to scale horizontally (adding more servers) and vertically (increasing server capacity) to accommodate increased data loads as the user base grows.
20. Dynamic Resource Allocation: Utilize cloud-based resources that can be dynamically allocated based on traffic demands.
21. Browser Compatibility: The system should support the latest versions of Chrome, Firefox, Safari, and Edge.
22. Mobile and Desktop Compatibility: The application should be fully functional on Android and iOS devices as well as Windows and macOS desktop systems.

**VI. Tech stack:**

**Front-End Development**

**React:** React’s component-based approach allows for the creation of reusable UI components, which simplifies development and maintenance.

**Mobile Development**

**React Native:** Allows for code sharing between the web and mobile platforms, reducing overall development time and ensuring consistent design and behavior.

**Back-End Development**

**Node.js:** Node.js is built on a non-blocking, event-driven architecture, making it ideal for handling asynchronous I/O operations, such as real-time updates and notifications in the scheduling app.

**Real-Time Communication**

**Socket.IO:** Socket.IO supports low-latency, bidirectional communication, making it ideal for real-time features such as shift updates, notifications, and reservations.

**Database**

**MySQL:** Most familiar by the team.

**VII. Explanation of developed features testing:**

1. **Basic Unit Testing:**
   * Test individual pieces of functionality like an API endpoint or a button click in isolation.
   * Use Jest for both frontend (React) and backend (Node.js).
   * Ensure small chunks of code, such as adding a new employee shift or making a booking, work as expected.

Example:

* + Test that when a manager creates a shift, it correctly saves in the system.
  + Test that clicking "Book Appointment" triggers the right function on the frontend.

1. **Simple Integration Testing:**
   * Test how the frontend and backend interact with each other.
   * Use Postman to manually test API requests and responses. I can test whether the backend correctly handles requests from the frontend without needing a complex setup.
   * Ensure that the frontend and backend work together properly without introducing errors when they interact.

Example:

* + Send a request from the frontend to book an appointment and check the backend's response to ensure data is correctly saved in the database.

1. **Manual Testing:**
   * Manually test the entire system by simulating user interactions on the frontend.
   * Use the browser's built-in Developer Tools (Chrome/Firefox) to test how the system works across different devices and screen sizes. Also, use manual testing to verify the flow of real-time features, like notifications.
   * Goal is to test core features (e.g., managing shifts, booking appointments) as a user would. This ensures that everything works well

Example:

* + Pretend to be an employee or customer using the system. Create a schedule, book an appointment, and see if it all behaves as expected.

Ensuring Continuous Integration (CI):

1. **Git-Based Workflow for Collaboration:**
   * Use GitHub to host code repository, ensuring that all changes are version-controlled. Whenever the team makes changes, create a pull request before merging into the main codebase.
   * Ensure that every piece of new code is reviewed and tested before it is merged into the main application, reducing the risk of bugs being introduced.
2. **Basic Automated Testing on Push:**
   * Set up GitHub Actions to automatically run basic unit tests and integration tests whenever someone pushes new code or creates a pull request.
   * GitHub Actions runs Jest tests every time code changes are pushed to the repository.
   * Automatically verify that nothing is broken whenever new code is pushed, without requiring manual intervention. It can be set up to notify if a test fails.

Example:

* + If someone pushes code that breaks the "book appointment" feature, GitHub Actions will detect the failure during the test run and notify me before the code is merged.

1. **Simplified Deployment:**
   * Use Heroku for backend and Netlify (or Vercel) for frontend. These platforms can automatically deploy the code whenever I push changes to the main branch.
   * Integrate Heroku or Netlify with GitHub repository so that new changes are automatically deployed to the staging or production environment.
   * Ensure that each time code is merged, it is automatically deployed to the live site without requiring manual steps.

Example:

* + When a pull request is merged into the main branch, the system is automatically deployed to Heroku (backend) and Netlify (frontend), making the new features live without needing manual steps.