CheckedOff Technical Documentation

Group 1

Members:

Nathan Diaz

Cody Hinz

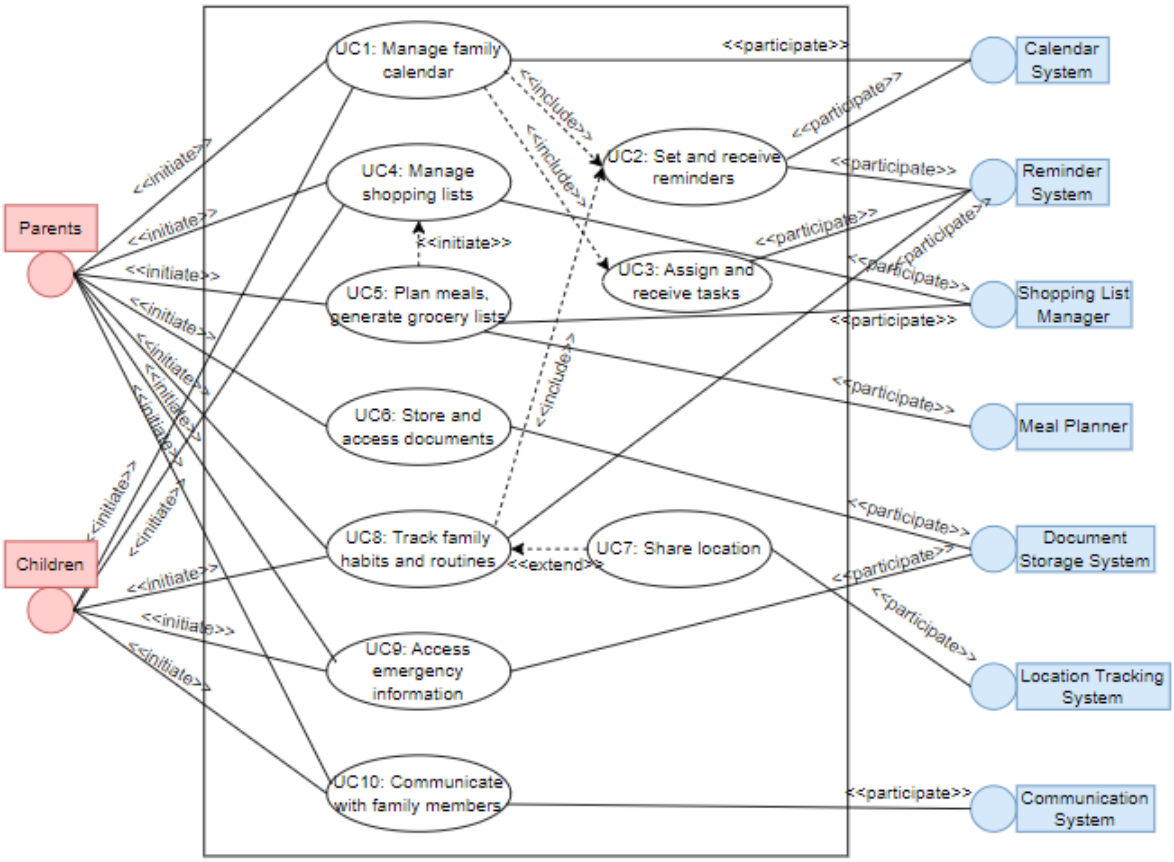
Gabriel Htet Aung Hlaing

Thiroth Khannara

Sethouday Prum

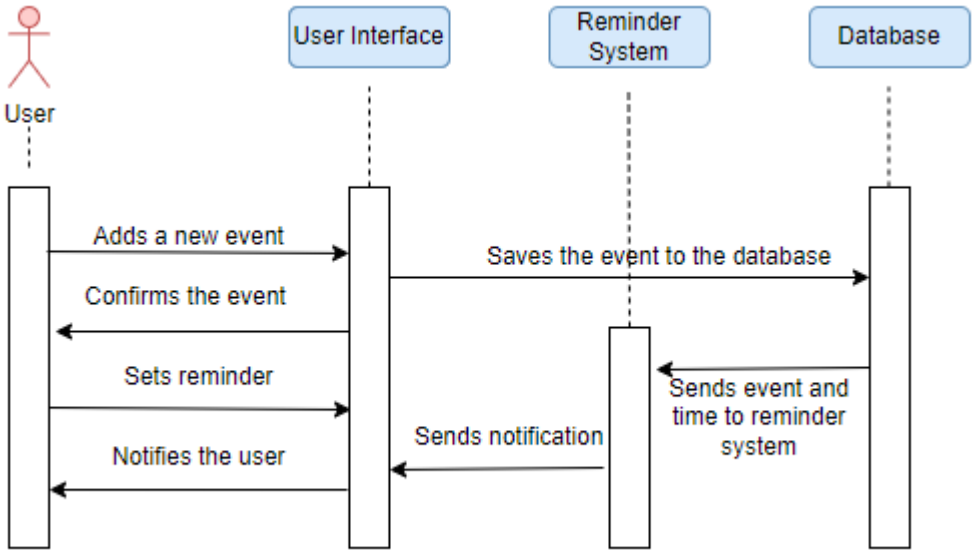
Quan Zhao

Use Case Diagram

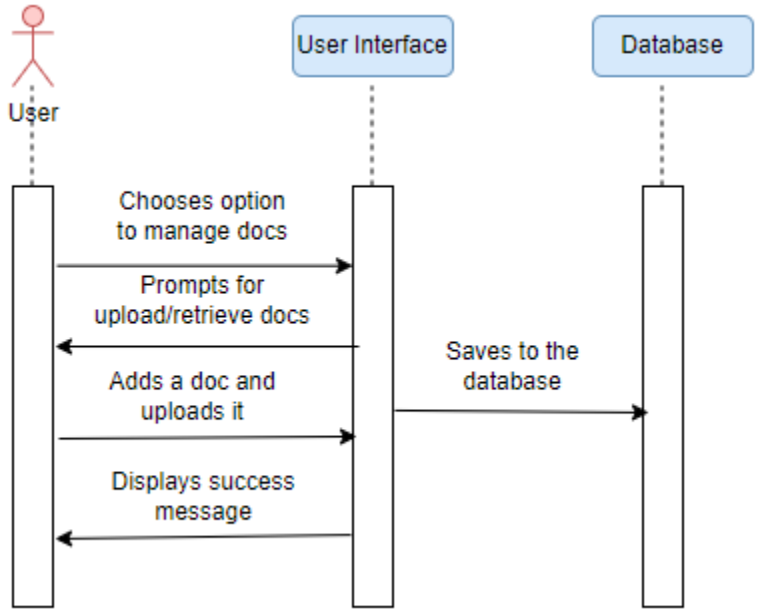


System Sequence Diagrams

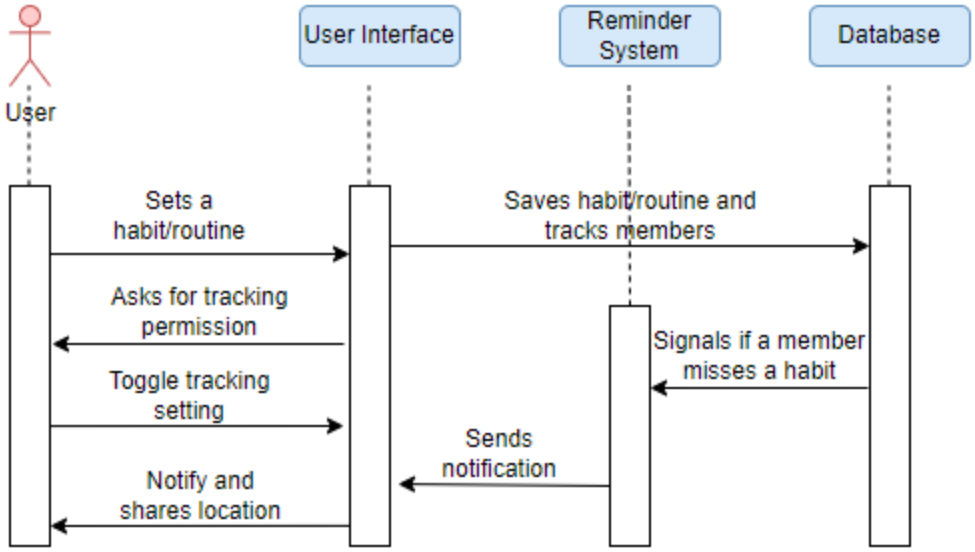
UC 1 & 2



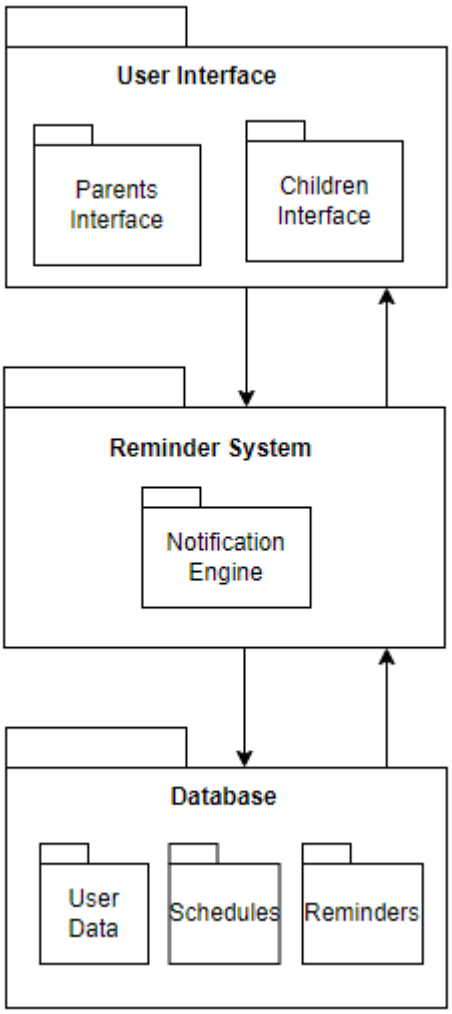
UC 6



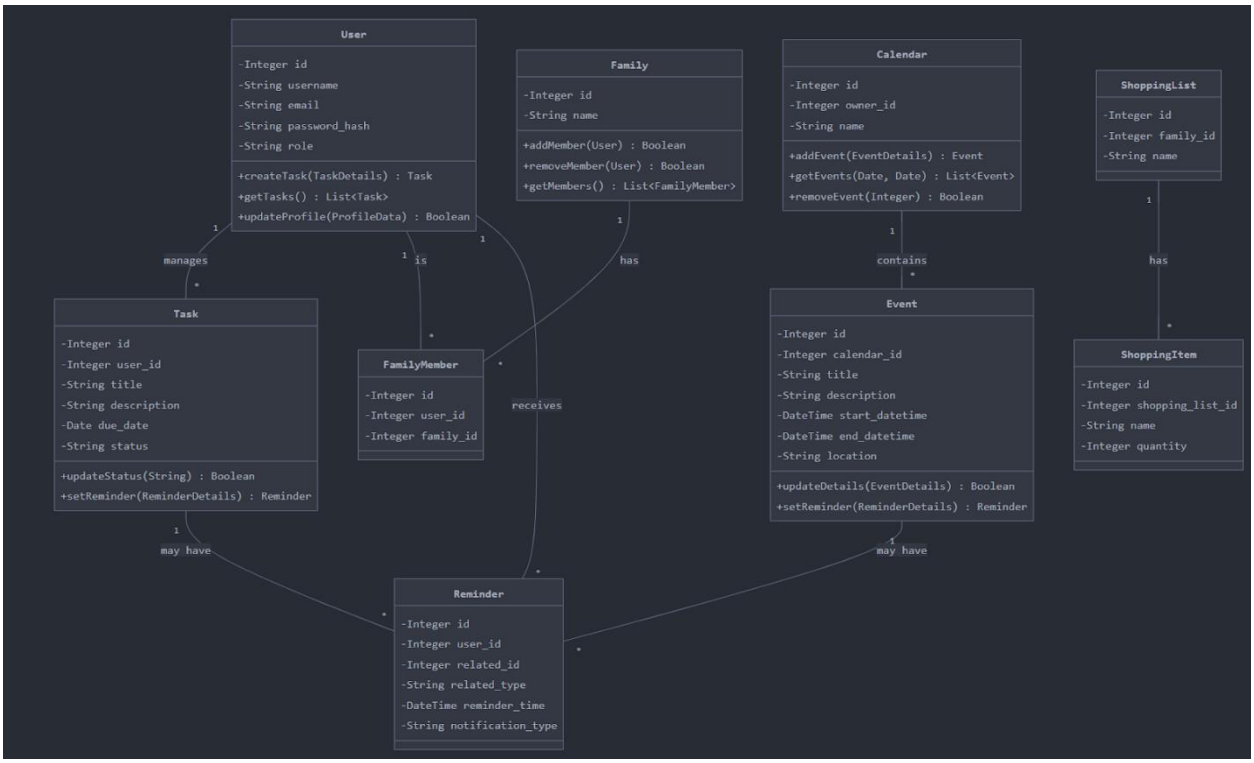
UC 8


User
User Interface
Reminder System
Database
Sets a habit/routine
Saves habit/routine and tracks members
Asks for tracking permission
Signals if a member misses a habit
Toggle tracking setting
Sends notification
Notify and shares location

UML Diagram


User Interface
Parents Interface
Children Interface
Reminder System
Notification Engine
Database
User Data
Schedules
Reminders

Class Diagrams and Interface Specifications



Project Architecture

The CheckedOff project is built using a modern React frontend architecture with a planned Node backend and PostgreSQL database.

Frontend Structure (React)

Core Application Files

- `App.js`: The root component that handles routing and overall application structure
- `index.js`: Entry point that renders the React application
- `App.css`: Global styles for the application

Context Management

- `TaskContext.js`: Implements React Context for global state management of tasks
- Provides shared task data and management functions across components

Component Organization

1. **Layout Components**
   - `Header.js`: Navigation and branding
   - `Footer.js`: Login/signup links

2. **Feature Components**
   - `Main.js`: Dashboard/home page
   - `Calendar.js`: Calendar view and management
   - `Tasks.js`: Task management
   - `Shopping.js`: Shopping list management
   - `Recipes.js`: Recipe management
   - `Documents.js`: Document management

3. **Authentication Components**
   - `Login.js`: User login
   - `Signup.js`: New user registration

Database Schema (PostgreSQL)

The database is designed with several key tables:

```sql
- User: Stores user authentication and profile data
- Family: Manages family group information
- FamilyMember: Links users to families
- Calendar: Stores calendar information
- Event: Manages calendar events
- Task: Stores task information
- ShoppingList: Manages shopping lists
- ShoppingItem: Individual shopping items
- Reminder: Handles notifications and reminders
```

Component Deep Dive

Calendar Component
- Implements a full calendar view with month navigation
- Supports filtering by user
- Shows tasks and events on their respective dates
- Uses CSS Grid for layout
- Responsive design for different screen sizes

Tasks Component
- Manages task creation, display, and deletion
- Implements filtering by user
- Uses TaskContext for state management
- Features a form for adding new tasks

- Responsive grid layout for task cards

Shopping List Component
- Manages shopping items with categories
- Supports filtering by user, store, and category
- Features a modal for adding new items
- Implements item completion toggling
- Responsive design with clean UI

Recipe Management
- Complex form handling for recipe creation
- Ingredient management
- Category and difficulty filtering
- Modal-based new recipe creation
- Responsive recipe card display

Document Management
- Secure document upload and storage
- Category-based organization
- User-based filtering
- Preview and download functionality
- Access control management

State Management

1. **Global State (Context)**
   - Task management through TaskContext
   - User authentication state (planned)
   - Family group management (planned)

2. **Local State**

- Component-specific UI states
- Form management
- Filter selections
- Modal visibility

Styling Architecture

The project uses a modular CSS approach:
- Each component has its own CSS file
- Global styles in App.css
- CSS variables for consistent theming
- Responsive design breakpoints
- Mobile-first approach

Key CSS Features
```css
:root {
  --blueprimary: #007bff;
  --bluesecondary: #043e7b;
  --lightbackground: #f0f8ff;
  --white: #fff;
  --darkfont: #333;
  --hovercolor: #cfcfcf;
}
```
- Consistent color scheme
- Responsive breakpoints at 768px and 1024px
- Flexbox and Grid layouts
- Shadow and animation effects

Planned Backend Integration

The frontend is prepared for backend integration with:
- API endpoint structures
- Form submission handling
- Authentication flow
- Data validation
- Error handling

API Routes (Planned)
```
/api/auth/: Authentication endpoints
/api/tasks/: Task management
/api/calendar/: Calendar operations
/api/shopping/: Shopping list management
/api/recipes/: Recipe operations
/api/documents/: Document management
```

Security Considerations

1. **Frontend Security**
   - Form validation
   - Input sanitization
   - Secure routing
   - Protected routes (planned)

2. **Data Security**
   - PostgreSQL security best practices
   - User data isolation
   - Document access control
   - Encryption considerations

Performance Optimization

1. **React Optimizations**
   - Efficient state management
   - Memoization opportunities
   - Lazy loading for routes
   - Component code splitting

2. **Asset Optimization**
   - CSS modularization
   - Responsive images
   - Efficient bundling
   - Code minification

Future Enhancements

1. **Technical Improvements**
   - Real-time updates
   - Push notifications
   - Offline support
   - Mobile app version

2. **Feature Additions**
   - Family chat system
   - Calendar sharing
   - Advanced recipe features
   - Document collaboration

Testing Strategy

1. **Current Testing**
   - Basic component testing
   - Form validation testing
   - Route testing

2. **Planned Testing**
   - Unit tests for all components
   - Integration testing
   - End-to-end testing
   - Security testing