VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Arnab Dhar

KINDERGARTEN ACTIVITY TRACKING WEB APP

Technology and Communication

2025

ACKNOWLEDGEMENTS

I would like to convey my deepest appreciation to Dr. Ghodrat Moghadampour, who guided me, supported me, and motivated me during my thesis work through his supervision that I could not have done without. His critical feedback and clear guidance were essential in improving the quality of this research. Without his support, I might have struggled during the development and writing process.

I am especially thankful to my older sister, Snigdha Dhar, and my brother-in-law for their constant support and for helping me manage my time and responsibilities.

Lastly, I would also like to express my deep gratitude to my parents in regard to their unconditional love, patience, and sacrifices. Without their backing, my path has been my foundation, thus making this attainment a possibility.

Arnab Dhar

7 January 2026, Vaasa

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Information Technology

ABSTRACT

Author	Arnab Dhar
Title	Kindergarten Activity Tracking Web App
Year	2025
Language	English
Pages	85
Name of Supervisor	Ghodrat Moghadampour

Many small Finnish kindergartens still use paper forms, spreadsheets, and email to manage attendance and daily communication, which can scatter records and reduce efficiency. This thesis presents a simple web application designed to support daily tasks in early childhood education and care.

The system includes three user roles administrator or manager, caregiver or nurse, and parent, each with features for managing children, groups, attendance, daily notes, and absence requests. It also allows comparison of planned and actual care hours to support planning and follow-up.

Role-based and functional testing was done to make sure the system works well for all users and that access rules are correct. In the future, the system could be improved by adding features like notifications and more detailed reports to make daily work and communication even easier.

Keywords daycare management system, attendance tracking, role-based access control, React and PHP REST API, MySQL database

CONTENTS

FIGURES

TABLES and CODE SNIPPET

ABBREVIATIONS

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DBMS	Database Management System
PHP	Hypertext Preprocessor
ECEC	Early Childhood Education and Care
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVP	Minimum Viable Product
RBAC	Role-Based Access Control
REST	Representational State Transfer
SPA	Single-Page Application
SQL	Structured Query Language
UL	User Interface
VPS	Virtual Private Server

1 INTRODUCTION

Digital tools are increasingly being used to support day-to-day activities in education, such as early childhood education and care. Kindergarten handles numerous data points daily, including attendance, scheduled and actual hours, employee placement, and communication with parents. Paper forms, email, or spreadsheets to manage this information will easily result in fragmentation and make it more challenging to maintain.

1.1 Background and Motivations of the Topic

The Finnish childhood education and care (ECEC) is built on the principles of equality, child-centered learning, and close cooperation between domestic and kindergarten environments(Finnish National Agency for Education (EDUFI/OPH) — National Core Curriculum for ECEC 2022 (PDF), n.d.). To promote these goals, kindergartens will need clear information on children's attendance, the hours during which children are to receive care, and an effective communication system between staff and parents. These activities are still performed manually in small kindergartens and with smaller private providers, either using paper forms, spreadsheets, or email. Emails often lead to duplicate work, poor records, and difficulties matching scheduled times with actual attendance.

Present digital interventions in the Finnish ECEC market are largely focused on large organizations. They usually require large budgets, complex integrations, or specific technical skills that smaller providers will hardly have. As a result, there is a clear need for lightweight, user-friendly web applications that can run on a single VPS (Virtual Private Server) or shared hosting, while still offering secure role-based access and basic forecasting support. The current thesis project aims to satisfy this need by designing and developing a web application tailored to the

needs of small Finnish kindergartens(*Legislation - Opetus- Ja Kulttuuriministeriö*, n.d.).

1.2 Objectives and Scope

This thesis is primarily aimed at creating a functional web application that will help kindergartens perform daily routine tasks more easily and reliably. The paper-based will focus on main areas like tracking attendance and working hours, supporting different user roles, and maintaining a technical solution that is not overly complex for small organizations to adopt.

The application allows caregivers to record children's arrival and departure times, enabling them to compare projected and actual arrival times and scheduled monthly hours. Each administrator, caregiver, and parent has a different interface that shows only the information relevant to their role. The system will be built with React on the front-end and a PHP REST API on the back-end, with MySQL for data storage, JWT-based authentication, and a basic security setup.

The thesis topics include designing and implementing a full-stack solution that includes the front-end, back-end, and database, as well as the security component. Further advanced functionalities, such as mobile applications or external integrations, are assigned to the prospective category.

1.3 Short description of the topic

The thesis is a design and development of a kindergarten activity tracking web application that is aimed at assisting in the daily activities of Finnish early childhood education and care (ECEC). A large number of small kindergartens continue to use manual or disjointed methods for tracking attendance and communication, which are both time-

consuming and unsynchronized. The suggested solution provides an easy-to-use role-based system that grants administrators, caregivers, and parents the ability to record and view attendance, compare planned and actual hours, and facilitate daily information flow. The application is a single-page web-based solution built with React and TypeScript, backed by a PHP REST API and a MySQL database, and secured with JWT-based authentication and RBAC.

This thesis outlines the design and implementation of a web application intended to support the daily operations of Finnish kindergartens through a web-based system. The application supports three user roles: administrator, Caretaker, and Parent. Key features include secure, role-based access control, straightforward deployment on a single server, and user-friendly forecasting tools that enable comparisons of expected versus actual arrivals and planned versus realized weekly hours. The system is mainly a frontend built with React and TypeScript, a RESTful PHP API backend, and a MySQL database. Security is enforced through JWT authentication and established best practices.

2 RELEVANT TECHNOLOGIES

This section presents an overview of the principal technologies used in the Finnish kindergarten activity-tracking web app. It achieves the following system as a modern web solution: a web app with a React front end and TypeScript on the client side, and a REST server written in PHP with a MySQL back-end database. Together, these features provide a design that is resilient yet easy to run on a single virtual private server (VPS) or in a shared hosting configuration, which can be considered a solution to the limitations of smaller Finnish kindergartens with limited technical capacity.

React is built on the client side to create a dynamic, responsive user interface that supports three roles (admin, nurse, parent). TypeScript introduces static typing, enabling code to be better and more maintainable as the application grows.

The backend is coded in PHP (Hypertext Preprocessor), a widely supported programming language. It is well supported by low-cost hosting providers, making it the right solution for the project's deployment goals. The server provides a set of RESTful HTTP (Hypertext Transfer Protocol) endpoints that allow transferring JSON data to the front end. The user's data, such as information about children, attendance, scheduled hours, and daily activity, daily notes, is continuously saved in MySQL, which uses a relational format that guarantees transaction integrity and supports querying.

The system uses stateless authentication, which is implemented using JSON Web Tokens (JWTs) to protect the system, and role-based access control (RBAC) defines the rights of administrators, caregivers, and parents. Password hashing uses bcrypt, and additional security features, including cross-origin resource sharing (CORS) settings, HTTPS encryption, and an audit log, are included to maintain the integrity of

sensitive data and track important actions. All these measures, combined, lead to a lightweight platform that is comprehensive, supporting functions such as attendance tracking, communication, and forecasting expected and actual arrivals, as well as the number of hours per week(*Legislation - Opetus- Ja Kulttuuriministeriö*, n.d.).

2.1 Web Application Architecture

The application follows a typical three-tier web architecture:

- Presentation layer (frontend) in the browser
- Application layer (backend) running on the server
- Data layer (database) storing persistent information

The user interface is written in React and communicates with the PHP-based backend using RESTful HTTP APIs that use JSON as message payloads. Each request is handled by the backend, which implements business rules and access control as well as any read or write operation against a MySQL database.

Such a clear separation of responsibilities results in a system that is simpler to understand, maintain, and expand. The user interface can be optimized without requiring any changes to the backend logic, and the backend can optimize itself for performance, security, or database schema.

2.2 Frontend: React and TypeScript

The user interface is built with React, a popular JavaScript library for creating component-based web applications. The React element model meets the needs of this project by allowing separation of views and components into admin, nurse, and parent areas.

Shared React libraries handle page routing and state management. To make the code more reliable and easier to work with, the frontend is written in TypeScript instead of plain JavaScript. TypeScript adds static typing, so many mistakes are caught already at compile time, and the data structures become clearer. In this project, these types are beneficial for clearly defining user roles, attendance records, forecasting results, and the exact shape of the data exchanged between the frontend and the PHP backend(*Describing the UI – React*, n.d.; *Handbook - The TypeScript Handbook*, n.d.).

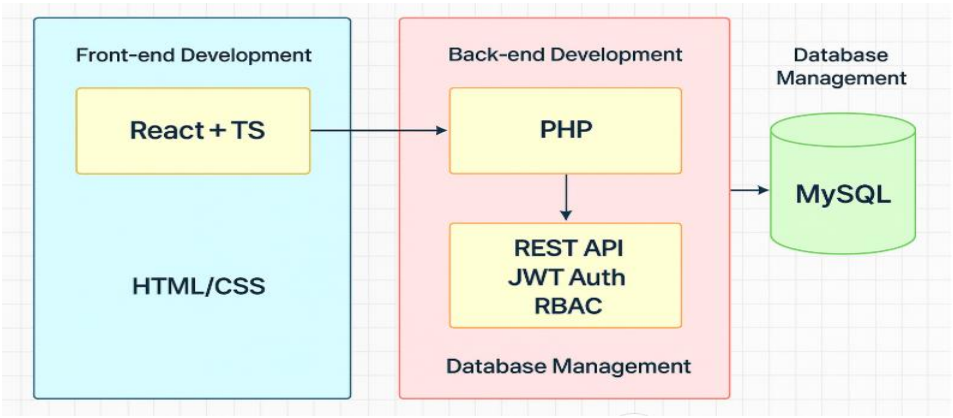

Figure 1. Technology stack: React and TypeScript(*Describing the UI – React*, n.d.; *Handbook - The TypeScript Handbook*, n.d.)

2.3 Backend: PHP and REST APIs

The backend uses PHP and runs on standard web servers like Apache or Nginx with PHP-FPM. PHP was chosen because it is supported by shared hosting and VPS platforms, aligning with the project's aim of making deployment easy and affordable for small organizations.

The backend provides REST API endpoints for tasks such as authentication, user and child management, attendance recording, and prediction. Each endpoint accepts JSON, handles

validation and business logic, connects to a MySQL database, and returns JSON responses. This stateless, resource-based setup works well with the current single-page app and makes it easy to add new clients, such as mobile apps, in the future(MySQL 8.0 Reference Manual, n.d.; PHP: FastCGI Process Manager (FPM) - Manual, n.d.).

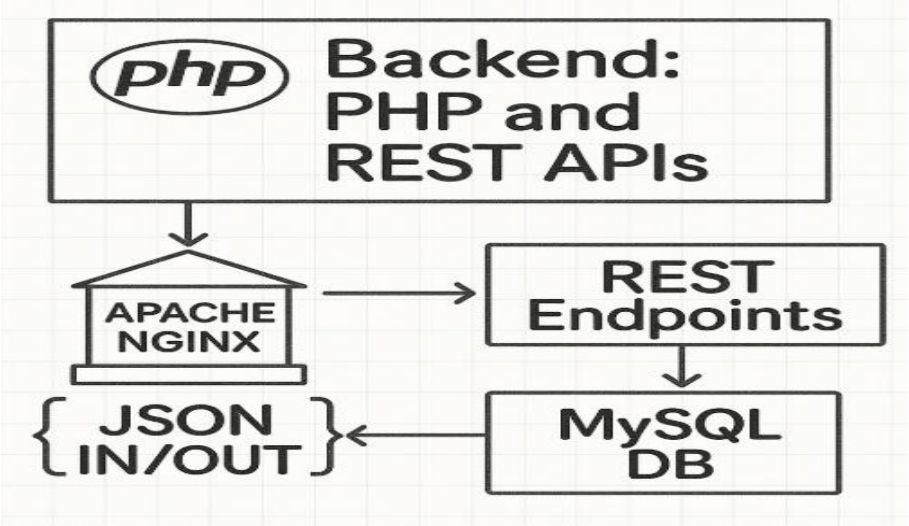

Figure 2. Technology stack: PHP backend and RESTful APIs(*PHP: PHP Manual - Manual*, n.d.)

2.4 Database: MySQL

MySQL is an open-source relational database management system (RDBMS)(*MySQL 8.0 Reference Manual*, n.d.). That follows the traditional data storage paradigm. Data is organized in tables with clearly defined columns and relationships and is read using Structured Query Language (SQL), supporting expressive queries and join operations(*17.2 InnoDB and the ACID Model*, n.d.). In turn, MySQL is more appropriate for areas with a stable, well-defined data model, e.g., user accounts, children's records, attendance logs, and forecast tables, within this kindergarten system.

MySQL has a significant advantage of the fact that it supports ACID-compliant transactions as well as ACID-compliant integrity constraints, such as primary keys, foreign keys, and unique indexes(*ISO/IEC 9075-1:2008*, n.d.). These mechanisms also make sure the data is consistent and prevent problems like multiple records of attendance or parent child relationships. MySQL is easily deployable on a single virtual private server (VPS) or shared computing setup, and it can be replicated and backed up, thus enabling scalability on demand. MySQL, therefore, will be a practical and developed solution to the given thesis project, where reliability and stated relational integrity are the main concerns, as opposed to excessive horizontal scalability.

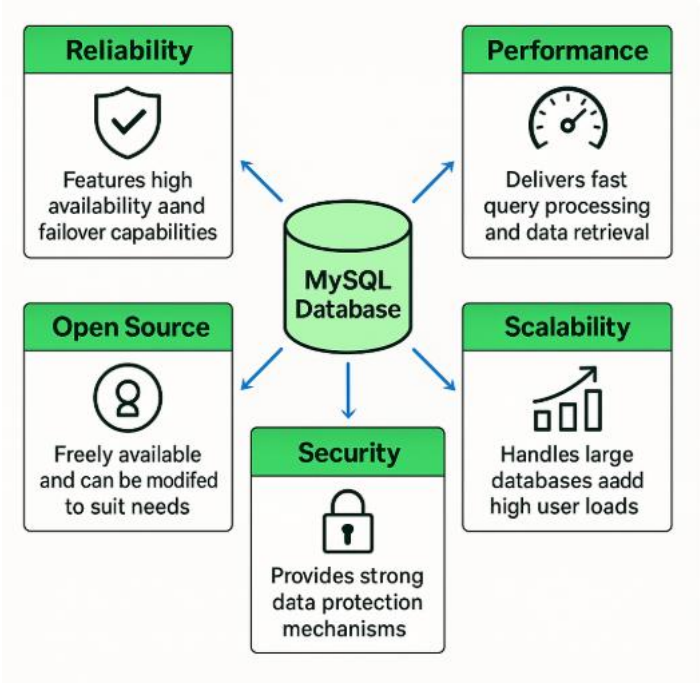

Figure 3. Technology stack: MySQL database(*MySQL Community Edition*, n.d.)

2.5 Authentication and Security Technologies

The frontend and backend use JSON Web Tokens (JWT) and the Bearer authentication scheme. Upon successful authentication, the backend

generates a cryptographically signed token containing the user identifier and role. Later frontend requests use this token in the Authorization header, and the backend checks the token before granting access to secured endpoints. This session's less authentication system makes the session management process easy and in line with the RESTful APIs concepts(Auth0, n.d.).

User passwords are protected by bcrypt hashing, a computationally expensive algorithm that includes built-in salting. This strategy significantly complicates the process of password recovery by the enemies, even in cases where the underlying database has been breached.

Role-based access control (RBAC) is used on the backend to restrict administrators, caregivers, and parents to the operations and data they are authorized to access according to their assigned roles. Essential transactions, i.e., authentications and changes to children's or attendance records, are also logged in an audit log, providing detailed traceability.

To secure the transferred data, the application will be served over HTTPS, which encrypts communication between the browser and the server(Diffie et al., 1976). Cross-Origin Resource Sharing (CORS) is also configured to allow only trusted frontend origins to prevent unauthorized cross-site requests.

2.6 Hosting and Deployment Environment

The system has been designed to run on one virtual private server (VPS) or in the shared hosting setup, which are both realistic and economical options for small Finnish kindergartens or any other private provider. In this setup, the web server, PHP runtime, and MySQL database reside on the same host, making them easy to configure and maintain.

Deployment is typically carried out with the PHP source code and the compiled frontend artifacts copied to the target server, the web server configured, and the MySQL database initialized with the necessary schema. Backups can be created through database dumps and file-level backups of the application code and application configuration files, which maintains operational overhead small but allows a modern and secure web application to support the daily routine of kindergarten work.

3 APPLICATION DESCRIPTION

The Kindergarten Web Application is a web-based framework designed to streamline day-to-day administrative processes and communication among administrators in Finnish kindergartens. The primary functionalities will include attendance tracking, control of planned care hours, and information sharing between staff and parents. The architecture of the system incorporates three distinct categories of roles, such as Administrator, Caregiver, and Parent, each having a specific interface and specific permissions to allow or prevent the disclosure of unwarranted information.

Along with attendance monitoring for elementary students, there are also predictive analytics and post-hoc validation of the duration of children's custody. Caregivers enroll accurate arrival and departure times; the platform compares these observations with pre-set itineraries, which expose anomalies in the expected and recorded entries and differences in projected and actual hours per week at the individual and cohort levels. This analytical tool reports human-resource adjudication and ensures that stipulated service hour commitments are observed.

The solution technically appears as a modern web application consisting of a React/TypeScript front-end, a PHP-based RESTful API, and a MySQL persistence store. Authentication is handled with JWT bearer tokens, and security is enhanced by cryptographic password hashing (bcrypt), role-based access control (RBAC), cross-origin resource sharing (CORS), TLS-enforced HTTPS, and detailed audit logs of privileged requests. Its architecture is explicitly lightweight and can be deployed on a single virtual private server or a shared hosting platform, and thus implementation and data-back-up become feasible for small kindergartens with limited technical strengths.

A use-case diagram is a pictorial representation of the actors that interact with the system and allowed operations. There are three main roles identified in this application: Administrator, Caregiver and Parent. Although the roles of the participants differ in the number of tasks assigned to them, there are some procedural functions that all participants have access to, including, most of all, authentication (login) and session end (logout).

3.1 Quality Function Deployment

Quality Function Deployment (QFD) is an organized approach to user requirements that converts them into clear technical requirements and development priorities. It can be used to measure the features of the system against the expectations of the stakeholders and inform decision-making about the design.

It is also possible to add a simple Priority legend (5=Critical, 4=High, 3=Medium) as a short line under the table.

Table 1. Quality Function Deployment

Reference	Description	Priority	Note
FR1	User authentication for all roles	5	Protects children's data
FR2	Role-based access control (admin/caregiver/parent)	5	Limits access by role
FR3	Record child check-in/check-out times	5	Core system function
FR4	Caregiver view of daily attendance	4	Supports daily operations
FR5	Planned vs. realized hours reports	4	Help with planning and monitoring
FR6	Parent absence reporting	3	Improves communication
FR7	Admin management of children/groups/users (CRUD)	4	Keeps data accurate
FR8	Audit log for critical actions	3	Adds accountability
NFR1	Privacy-focused data handling (GDPR-aligned)	5	High-sensitivity domain
NFR2	Secure password storage (bcrypt)	5	Reduces breach impact
NFR3	HTTPS enforced in production	5	Encrypts data in transit
NFR4	Usable for non-technical staff/parents	4	Drives adoption
NFR5	Good performance on modest hosting	3	Can be optimized later
NFR6	Reliable during working hours	3	Reduces disruptions
NFR7	Maintainable code and clear API	3	Eases future updates

3.2 Use-case Diagram

The three roles in the UML diagram are represented as actors (stick figures), with the Finnish kindergarten web application system connected to them. Every action the user can take is depicted as an oval (use case), and the connection lines between the actors and the ovals represent the allowable interactions. This representation is a clear, high-level description of the main functions before delving into technical details.

Below is a more detailed description of what each role can do.

3.2.1 Admin

The admin role corresponds to the kindergarten manager or system administrator. This role has the broadest access rights and is responsible for configuring the system, maintaining data quality, and monitoring day-to-day operation.

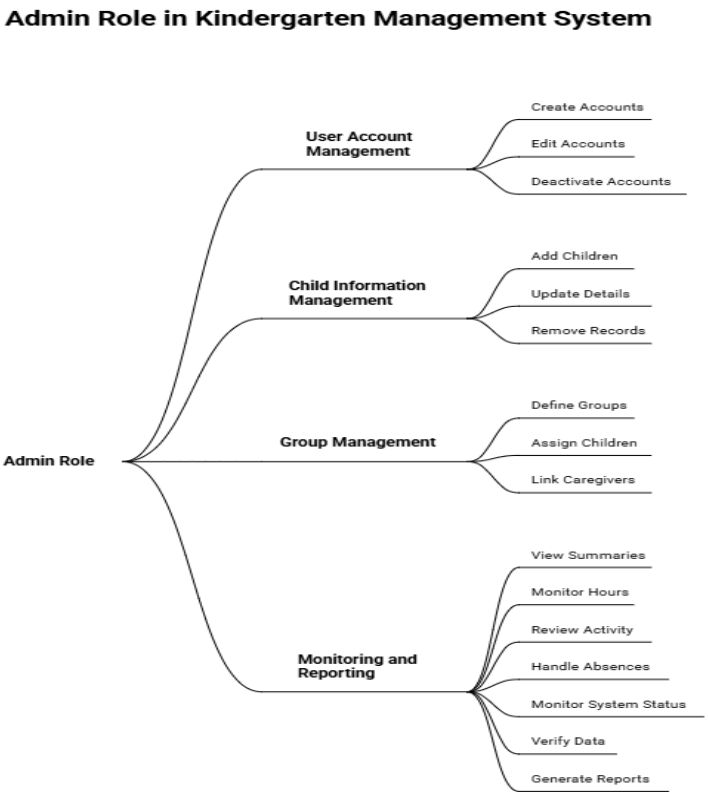

Figure 4. Admin Use-case Diagram (generated using AI)

Typical use cases for the admin include:

- Log in and log out – Access the system using an email and password and securely end the session after use.
- Manage user accounts: create, edit, and deactivate accounts for administrators, caregivers (nurses), and parents.
- Manage children's information – Add new children, update their personal and contact details, and remove records when a child leaves the kindergarten.
- Create and manage groups – Define children's groups (e.g., by age), assign children to groups, and link caregivers to the appropriate groups.

- View summaries and monitor hours – Open dashboard views that display attendance summaries and compare planned versus realized hours at the level of individual children or groups.
- Review daily activity and updates – Check the latest comments and notes recorded by caregivers (such as sleep, meals, or activities) to ensure that information is complete and up to date.
- Handle absence requests – Review, approve, or reject absence requests from parents, including long-term absences due to holidays or illness.
- Monitor system status – Use the main dashboard to quickly check key indicators such as the total number of groups, nurses, active parents, children, and today’s attendance.
- Verify data consistency – Regularly inspect attendance records, group assignments, and children's information to detect and correct inconsistencies or missing data.
- Generate reports – Produce summaries (for example, monthly attendance or total hours per child/group) for internal follow-up, invoicing, or reporting to authorities.

The admin actor is linked to various use cases in the use-case diagram, indicating its key role in setting up and managing the system.

3.2.2 Caregiver (nurse)

A Caregiver is an employee who works directly with the children in one or more groups. The caregiver interface focuses on daily routines, children's information, and attendance management rather than on system configuration. On the dashboard, caregivers immediately see the number of assigned groups, assigned children, today’s attendance, recent updates, and a quick action to take attendance.

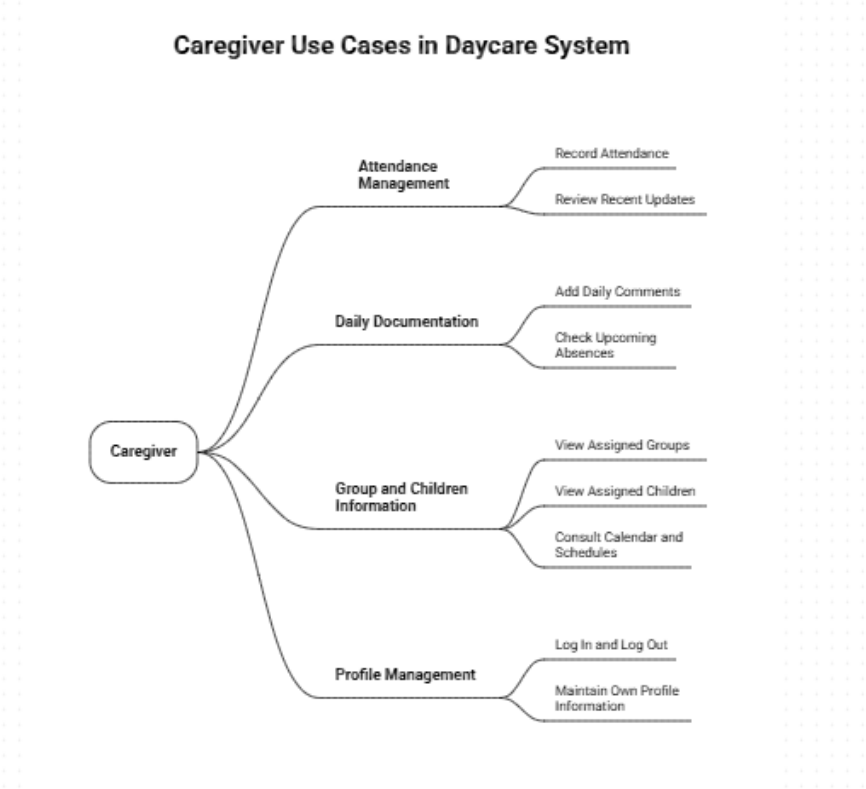

Figure 5. Caregivers use case diagrams for the daycare system (generated using AI)

Typical use cases for the Caregiver include:

- Log in and log out – Access the system securely using their personal account and end the session after use.
- View assigned groups – See the group or groups they are responsible for, including basic information such as group name and date.
- View assigned children – Display a list of children in their groups, including age and current status, and quickly navigate to detailed children information.
- Record attendance – Use the Take Attendance action to mark arrival and departure times for children during the day.
- Add daily comments – Enter short notes or comments for each child (e.g., sleep, meals, mood, activities, or special events), which can later be read by parents and administrators.

- Review recent updates – Check the latest comments and time records for children in their groups to ensure that information is complete and accurate.
- Check upcoming absences – View upcoming requested absences for the children in their groups so they can plan activities and staffing accordingly.
- Consult calendar and schedules – Open the calendar view to see planned daycare days, holidays, and other relevant events for their groups.
- Maintain own profile information – Update their personal details (e.g., contact information) via the My Information and Profile sections when necessary.

In the use-case diagram, the Caregiver actor is primarily associated with operational use cases related to attendance, daily documentation, and group and children information. This reflects the caregiver's role in recording and checking day-to-day data about the children rather than configuring the overall system.

3.2.3 Parent

The Parent (or guardian) uses the system to follow the children's daycare activities and to communicate absences to the kindergarten. The parental interface is intentionally simple and focused on child-specific information, so parents can easily check attendance, planned hours, and messages from caregivers. On the Parent Dashboard, parents see each of their children, including age, group, and monthly stay hours, as well as upcoming absences and recent updates.

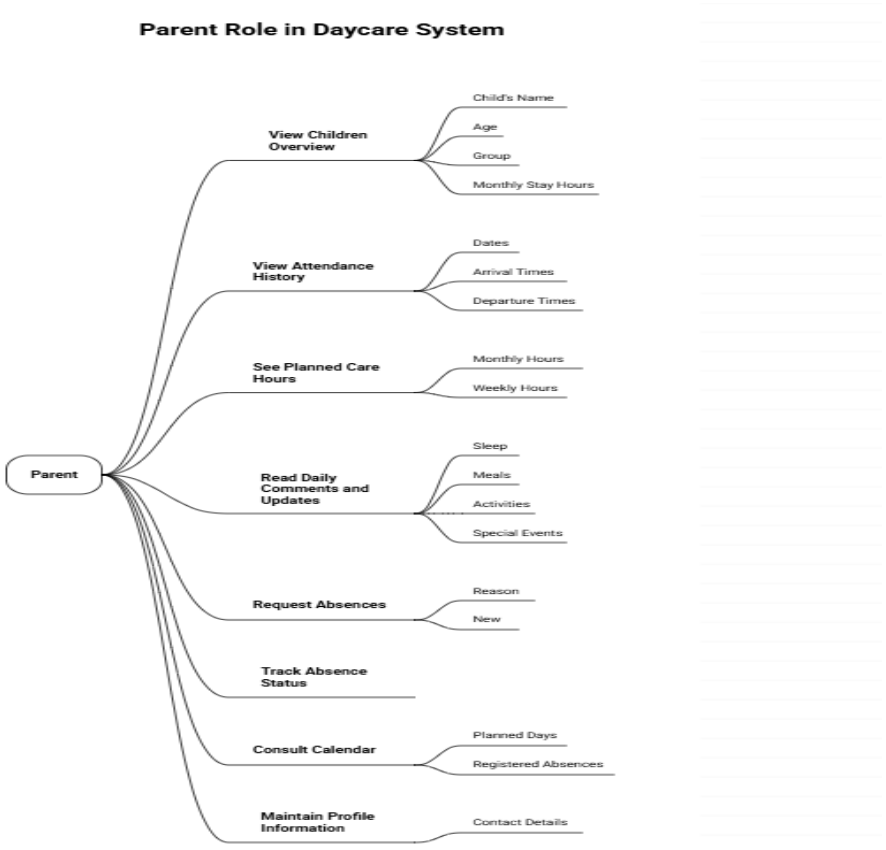

Figure 6. Parents use case diagrams for the daycare system (generated using AI)

Typical use cases for the Parent include:

- Log in and log out – Sign in securely to view information related only to their own children or children and end the session after use.
- View children overview – See basic information, including the child's name, age, assigned group, and agreed monthly stay hours.
- View attendance history – Browse past attendance records, including dates, arrival times, and departure times.
- See planned care hours – Check the agreed care hours (for example, monthly or weekly stay hours) to see how daycare usage compares with the plan.

- Read daily comments and updates – Read notes written by caregivers about the children's day, such as sleep, meals, activities, or special events, as they appear in the Recent Updates section.
- Request absences – Submit absence requests directly from the dashboard for selected children, including the reason and the planned dates.
- Track absence status – Review the list of upcoming absences and see whether each request is pending, approved, or rejected.
- Consult calendar – Use the calendar view to see planned daycare days, registered absences, and other relevant events.
- Maintain own profile information – Update their own contact details through the Profile section when necessary.

In the use-case diagram, the Parent actor is mainly associated with viewing and reading use cases, complemented by the ability to request absences. This reflects the limited but focused access of the parent role, which is restricted to information and actions concerning their own children.

3.3 Sequence Diagram

The sequence diagrams present the time-ordered interactions between the user (actor) and the main system components: the React front-end, the PHP REST API, and the MySQL database. While use-case diagrams describe what functionality the system provides, each sequence diagram explains how one specific use case is executed step by step.

In accordance with UML notation, a separate sequence diagram is provided for each selected use case. Because the application contains many use cases, this thesis focuses on the three most important use cases for each role (admin, caregiver, and parent). The diagrams also reflect the system's security and validation logic, including JWT-based

authentication and role-based access control (RBAC), before any database operation is performed.

3.3.1 Admin use-cases

The admin sequence diagram summarizes the main management tasks: handling user accounts, children, groups, and caregivers, assignments, report generation, and absence request approval or rejection. Each flow starts from an action in the admin interface and continues as an authenticated JWT-based API request. The backend authorizes the request, reads or updates the MySQL database, and returns the result to the interface as updated tables, status messages, or reports.

- Manage user accounts (CRUD):
  This sequence diagram describes how the admin creates, views, edits, and deletes user accounts. The backend verifies the admin role and executes the required database operation, returning a success status and/or updated user list.

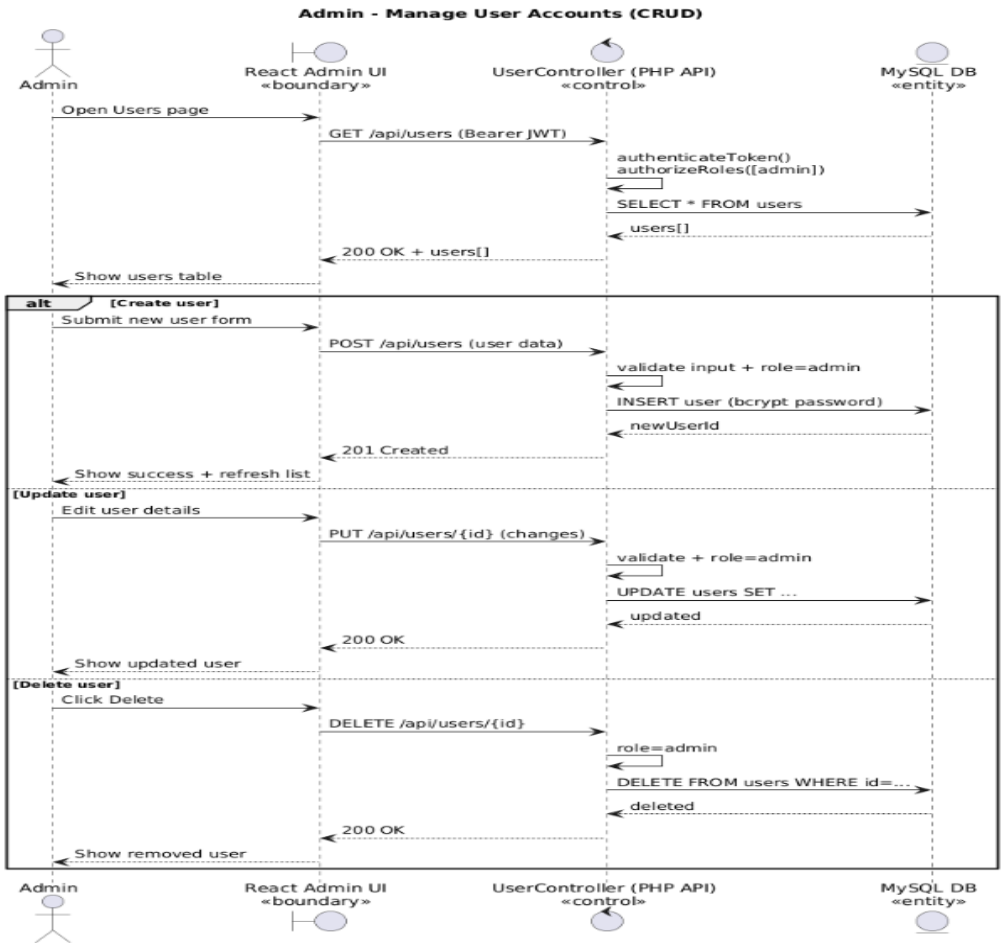

Figure 7. UML sequence diagram for Admin: Manage User Accounts (CRUD)

- Manage children's information (CRUD)

  This sequence diagram shows how the admin maintains children's profiles (create, view, update, delete). The API validates access, updates the database, and the UI refreshes the children list accordingly.

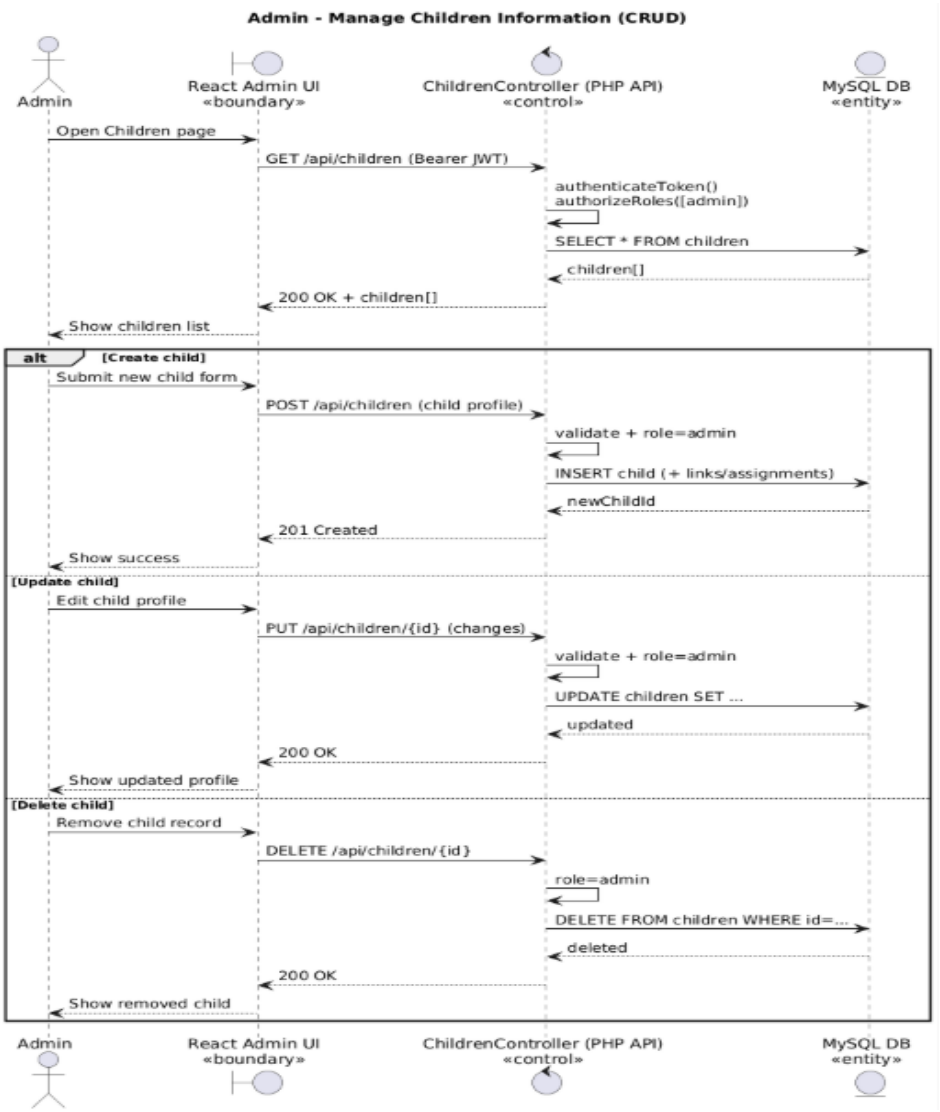

Figure 8. UML sequence diagram for Admin: Manage Children Information (CRUD)

- Handle absence requests (approve/reject)

  This sequence diagram explains how the admin reviews submitted absence requests and updates their status to be approved or rejected. The system validates authorization, updates the request status in MySQL, and returns the updated status to the admin dashboard.

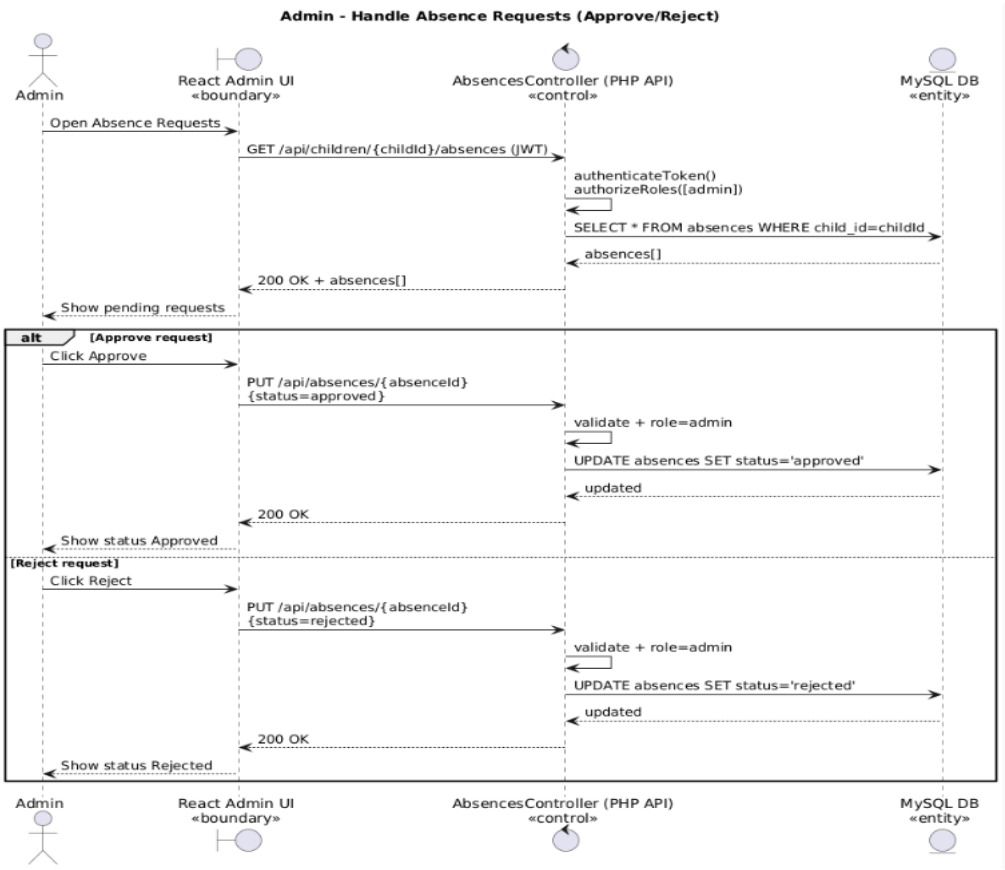

Figure 9. UML sequence diagram for Admin: Process Absence Requests (Approve/Reject)

3.3.2 Caregiver use-cases

Regular kindergarten activities are based on the caregiver role. The use cases of the particular caregivers depict the daily routine of the workflow, including the processes of retrieving the assigned lists of children, recording the attendance, and making the reports about the daily observations. In both cases, the caregiver places an interaction through the user interface, which, in turn, sends a Web JSON Secured API request. The backend then authenticates the caregiver authorizations (such as group-child allocations), executes prescribed

validation regulations, executes updates in a MySQL database, and it responds to refresh the user interface.

- Record attendance (check-in/check-out)

  This sequence diagram describes how a caregiver records attendance information for a child. The backend validates the caregiver's assignment and enforces basic rules (for example, check-out time cannot be earlier than check-in time) before saving the record.

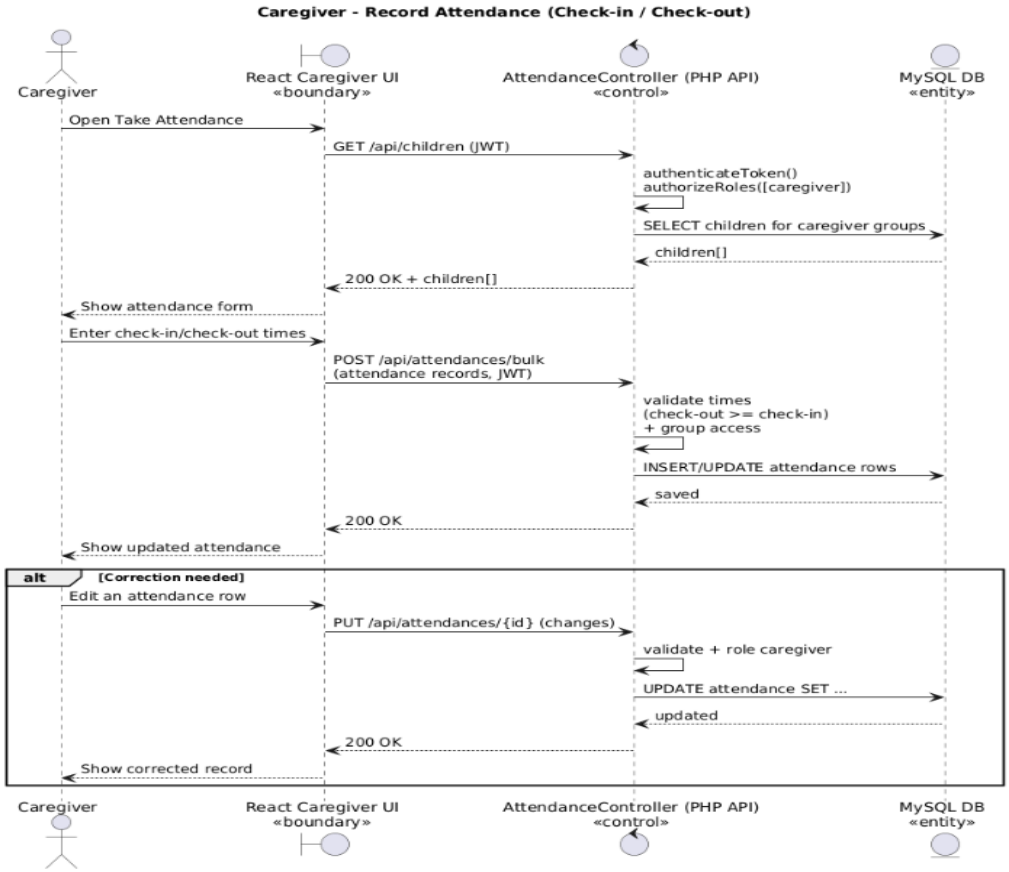

Figure 10. UML sequence diagram for Caregiver: Record Attendance (Check-in/Check-out)

- Add daily comments/notes

  This sequence diagram shows how a caregiver adds a daily note for a child. The system verifies permission, stores the notes in the

database, and returns the created entry to display in the child's update's view.

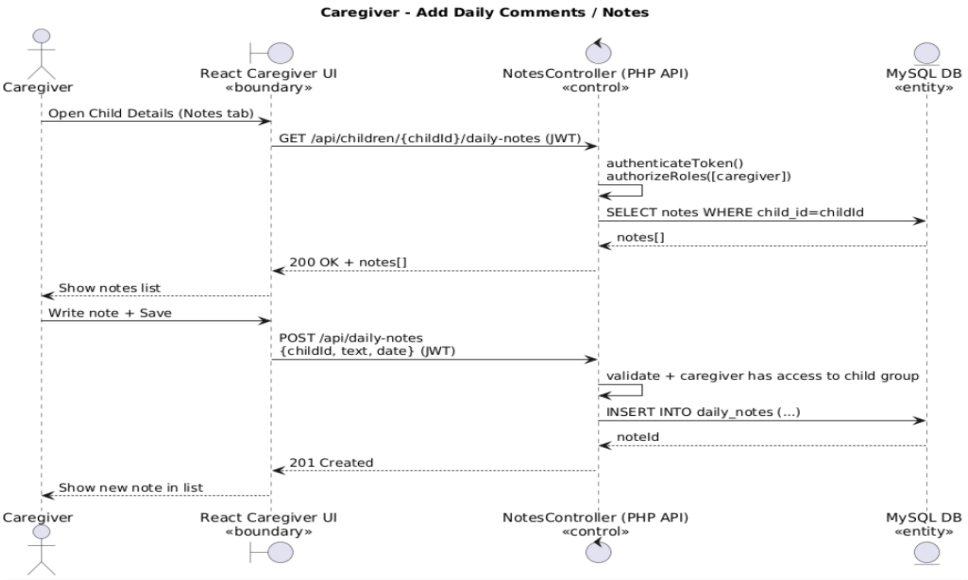

Figure 11. UML sequence diagram for Caregiver: Add Daily Comments/Notes

- View assigned children/group list

  This sequence diagram describes how a caregiver retrieves and views the list of assigned groups and children. The API verifies identity and returns only the children linked to that caregiver.

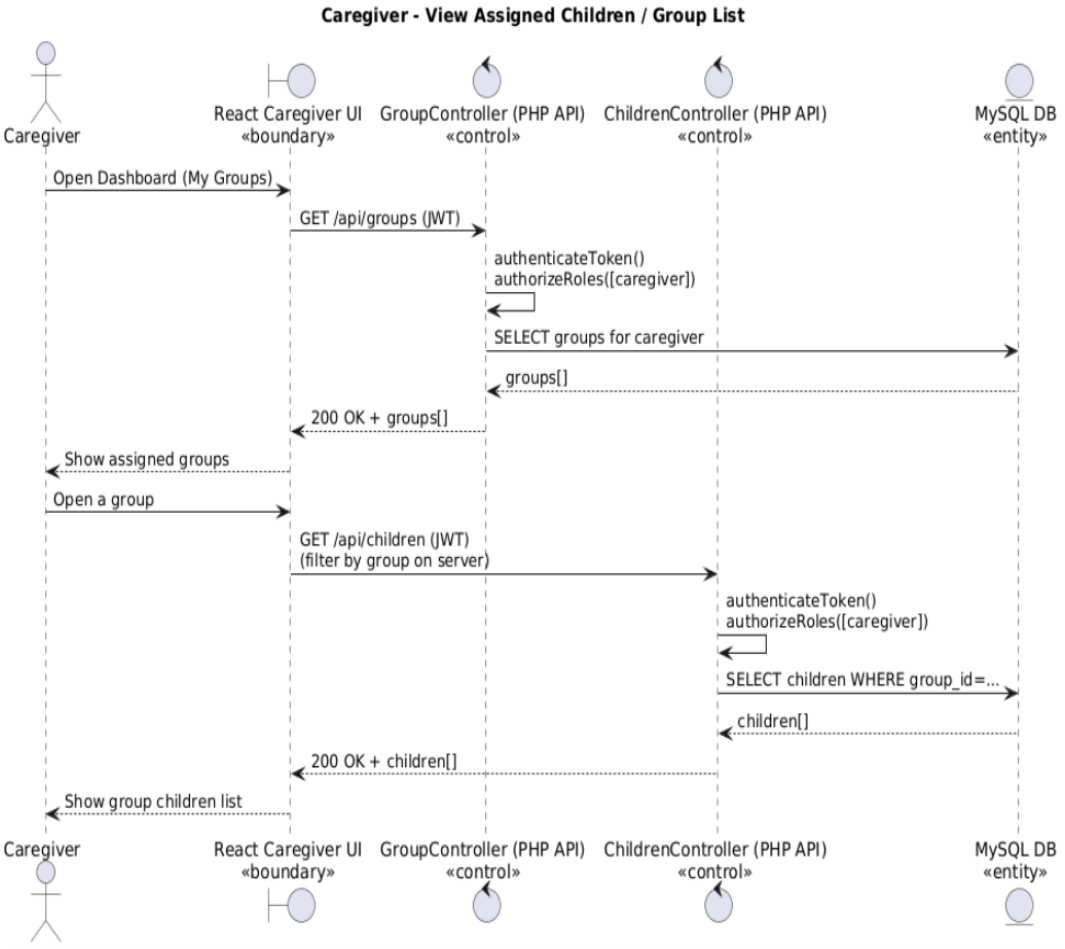

Figure 12. UML Sequence diagram: Caregiver views assigned children by group

3.3.3 Parent use-cases

The parent sequence chart outlines the modalities of interaction, which parents have in accessing the information of their child, which involves viewing attendance, daily notes, submitting an absence request, and tracking the permission status. Every request is sent with the parent JSON Web Token (JWT); the backend system then verifies the authentication credentials, as well as the parent-child relationship, before carrying out the data retrieval or modification procedures in a

MySQL database. The obtained data is then displayed in the parent dashboard and calendar interfaces.

- Request absence

  This sequence diagram explains how a parent submits an absence request for their child. The backend verifies the relationship and stores the request with an initial status (e.g., pending).

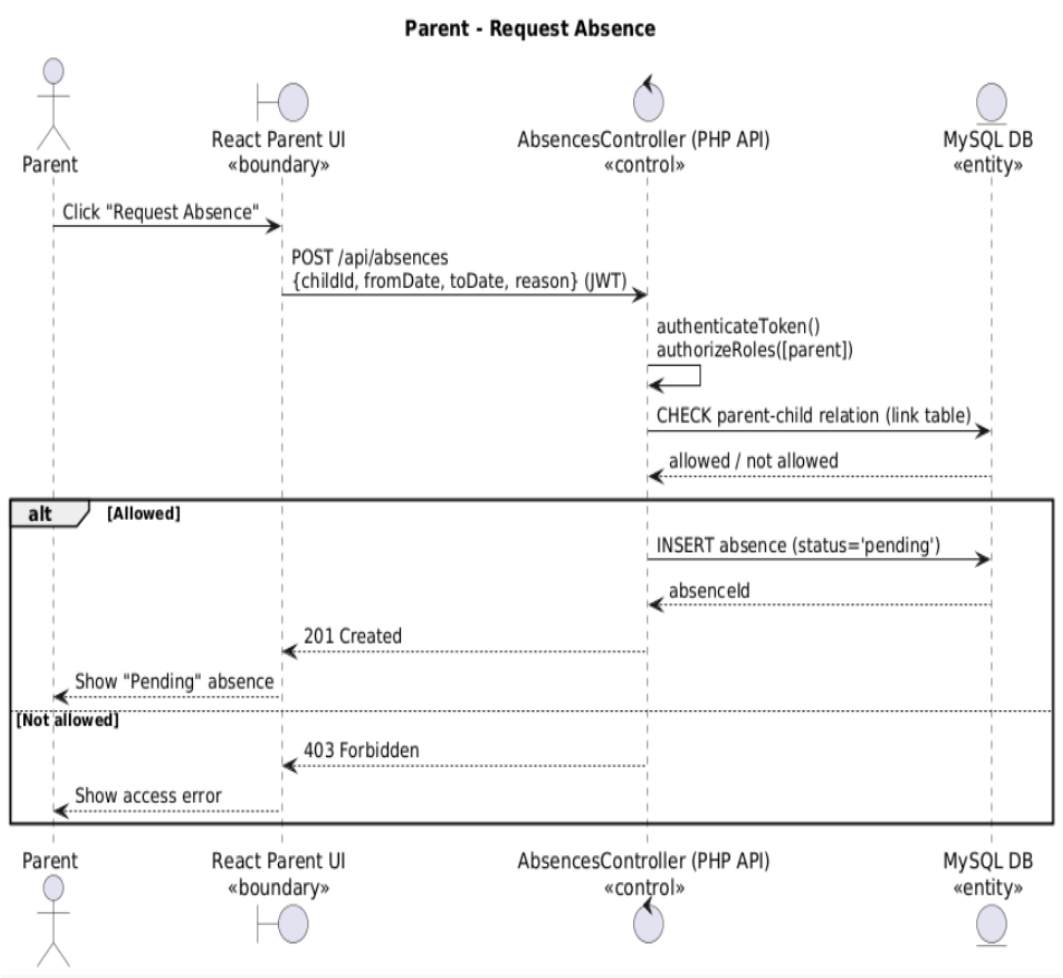

Figure 13. UML sequence diagram for Parent: Submit Absence Request

- View attendance history

  This sequence diagram shows how a parent retrieves historical attendance data for their child. Only authorized parent–child relationships are allowed to access attendance records.

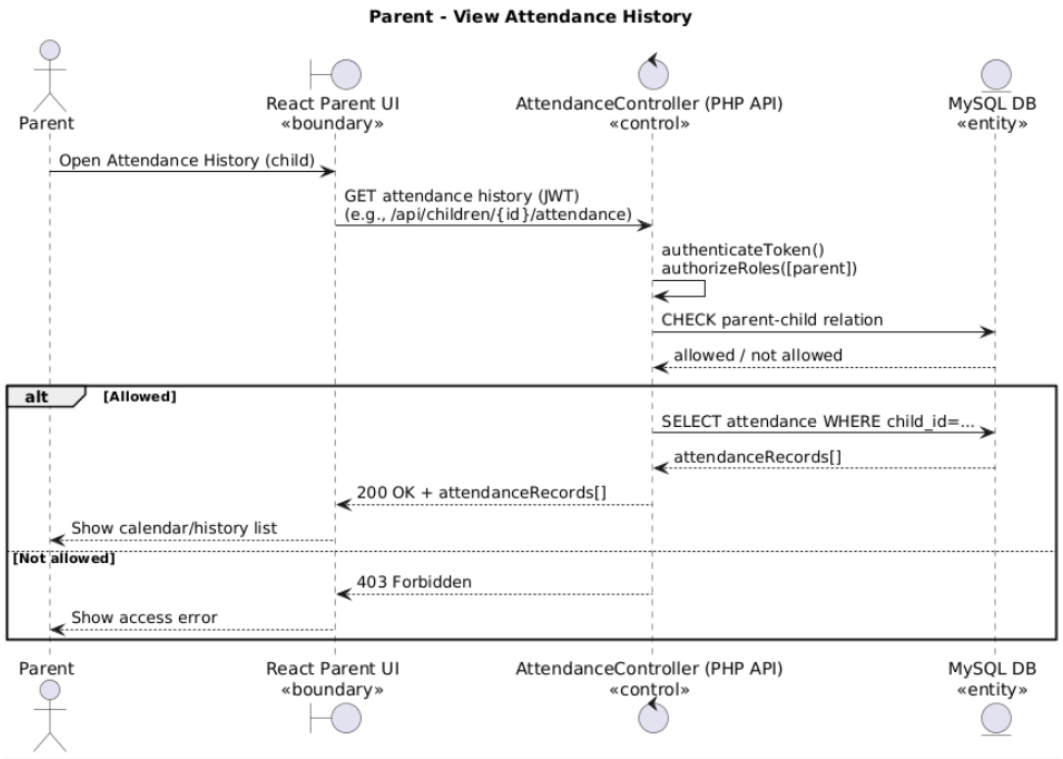

Figure 14. UML sequence diagram for Parent: View Attendance History

- Read daily comments/updates

  This sequence diagram describes how a parent views daily notes or comments created by caregivers for their child, after relationship verification and authorization.

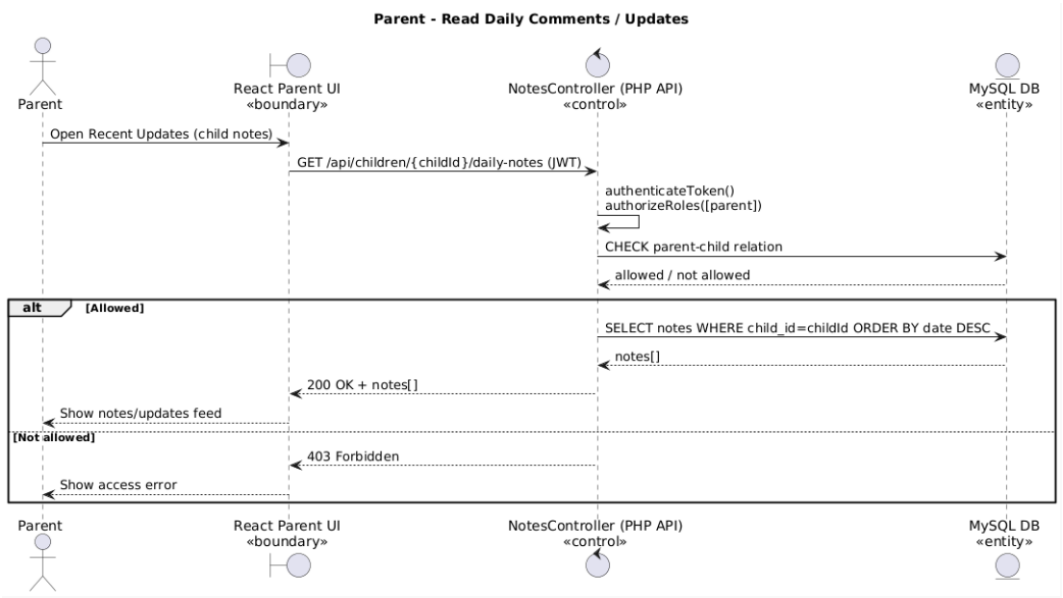

Figure 15. UML sequence diagram for Parent: View Daily Comments/Updates

3.3.4 Alternative flows and validation

The sequence diagrams also include common alternative outcomes. If authentication fails (missing or invalid token), the API returns an unauthorized response, and the user is required to log in again. If the user is authenticated but not permitted to access a resource (wrong role or not linked to the relevant child/group), the API returns a forbidden response. If submitted input is invalid, the API returns a validation error, and no database update is performed. These alternative paths ensure that the diagrams reflect real system behavior and demonstrate that security checks are always performed before any data is modified.

Overall, the sequence diagrams provide a clear view of how information flows through the system and confirm that each use-case is executed in the correct order, with authentication, authorization, and validation enforced before any changes are written to the database.

3.4 Domain Model Class Diagram

The domain model class diagram depicts the core entities of the childcare system and their relationships. It focuses on the main data concepts handled by the application, such as User (admin or manager, caregiver or nurse, parent), Child, Group, Attendance, Daily Notes/Comments, Absence Requests, and planned hours (Schedules) where applicable. The purpose of the diagram is to describe the system at a conceptual level and to show key cardinalities (e.g., one group has many children, and one child has many attendance records).

This diagram is not intended to list every implementation-level class in the source code (for example controllers, services, repositories/DAO, middleware, validation helpers, or UI components). Those technical components are described in the implementation and API sections. The domain model therefore serves as a bridge between requirements and the database design, while the MySQL schema (Figure 13) shows the physical table structure used in implementation.

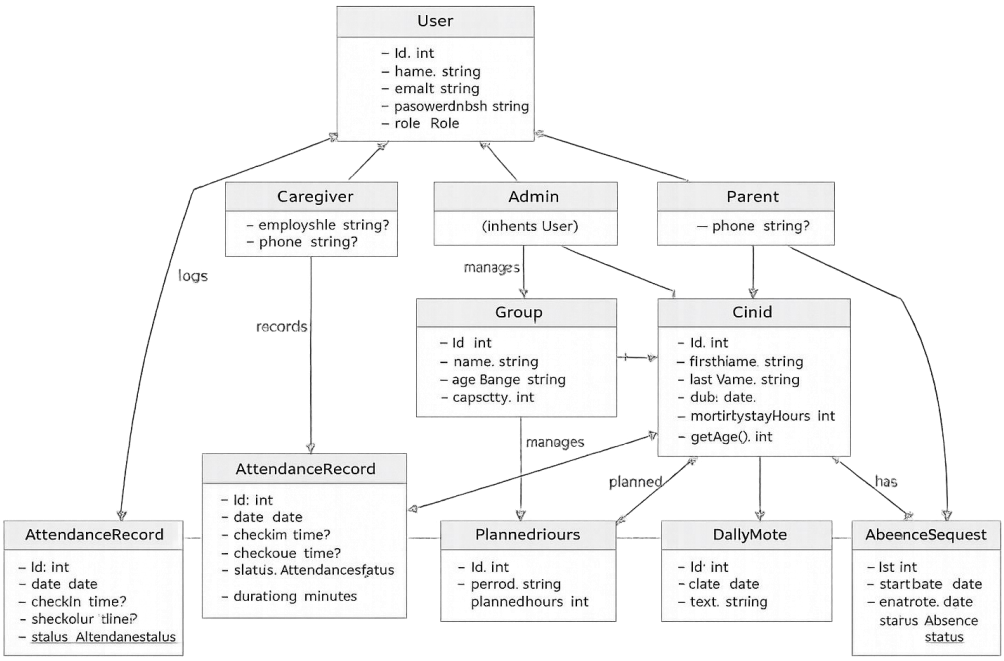

Figure 16. Kindergarten data model (entities and relationships)

3.5 Architectural Diagram

The Daycare Child web application has a separate three-layer architecture: a client built with React and TypeScript, a PHP API server with a POC-based backend, and a MySQL database. The stratification defines presentation, business logic, and data storage levels, making maintenance, security, and deployment easier in small kindergartens.

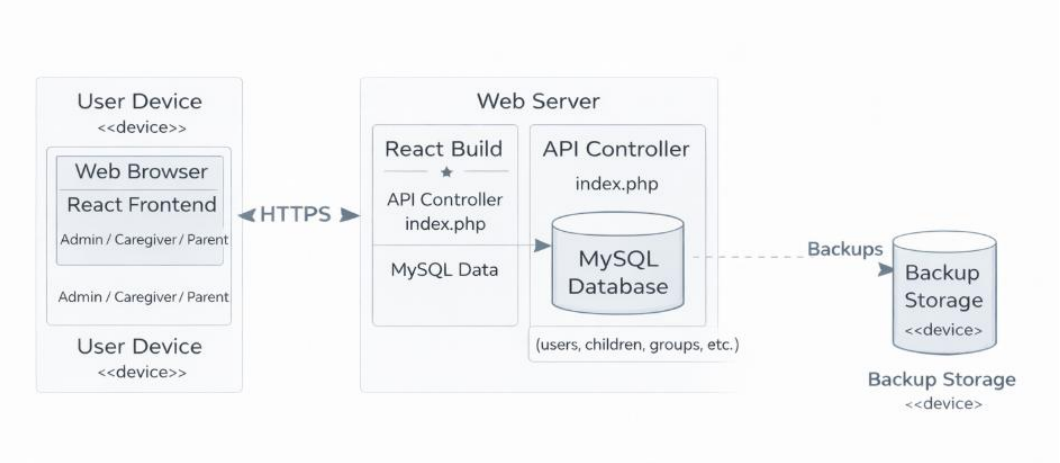

Figure 17. UML Deployment Diagram of the Daycare System

React (TypeScript) client-side:

The frontend is a single-page application through which the users interact with role-specific interfaces. It makes pages, screens, valid input, and communicates with the backend using JSON over HTTP. Three different views are provided with the system:

- Manager or Admin View: charged with the administration of users, groups, children, attendance, and absence requests.
- Caretaker or Nurse View: allows viewing of assigned groups and children, checking in and out, and adding daily notes.
- Parent View: allows the child's details, attendance logs, notes, and absence requests to be sent.

The backend handles authorization, validation, and role-based access control. It reveals RESTful interfaces of fundamental functionalities, including users, children, groups, attendance, absences, and daily notes. There is also the use of Secure requests with JSON Web Tokens (JWT) and passwords with bcrypt hashing.(*PHP*, n.d.)

4 DATABASE AND GUI DESIGN

In this chapter the architectural design of the database and graphical interface (GUI) of the Finnish kindergarten web application is outlined. The goal was to keep the data model and the user interface as simple and comprehensible as needs to be to continue with daily use in a kindergarten situation. The database design is made to hold the most crucial data such as users, children, groups, attendance, and intended hours in a structured and always reliable form. In line with this, the GUI architecture will be designed in such a manner that it provides user-friendly and intuitive screens that are specific to the three key user identities, namely: Administrator, Caregiver and Parent.

4.1 Design of the database

The database is implemented with MySQL and is based on a small set of core tables. The design follows standard relational principles: each main concept has its own table, and relationships are handled with foreign keys and link tables where needed.

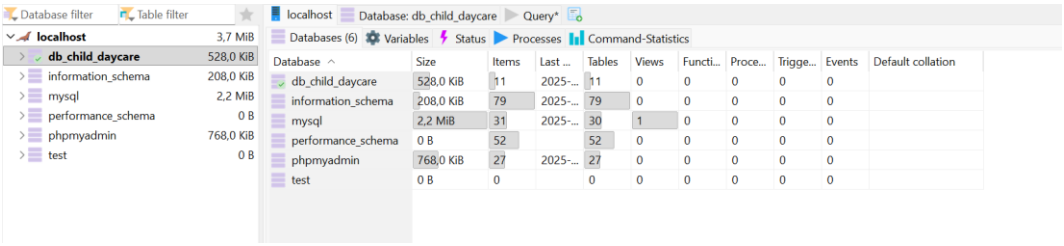

Figure 18. MySQL Database Overview

Each table is designed with a clear purpose and has fields that match its real-world role in the kindergarten. The core tables are:

- Users table:
  This table stores everyone who uses the system: admins, caregivers, and parents.
  It includes fields such as user ID, first name, last name, email,

encrypted password, role, and timestamps.
The role field (admin/Nurse/parent) is critical because it tells the system what each user is allowed to see and do.

- Children's table:
  This table represents each child in the kindergarten.
  It stores the children's ID, first and last name, date of birth, and a reference to the group they belong to.
  Each child is also linked to one or more parents, and to their attendance and planned hours.
- Parent-Child Table:
- This is a link table between parents and children.
  It contains a parent (pointing to a user with role = parent) and a children_id (pointing to a child).
  This design supports real family situations where one parent can have several children, and a child can have more than one guardian.
- Groups table:
  This table stores the different children groups in the kindergarten, such as Daisies or Owls.
  It includes a group ID and a group name, and optionally a short description.
  Each group is linked to many children, and, if needed, caregivers can be connected to groups through an extra link table.
- Attendance Records table:
  This table keeps track of children's daily attendance.
  It stores attendance ID, children ID, date, arrival time, departure time, caregiver ID (who recorded it), and an optional comment.
  These records are used to show daily presence, build attendance history, and calculate how many hours a children has actually been in care.
- Planned Hours table:
  This table stores the agreed care hours for each child.

It includes an ID, children's ID, and information about the date or week, together with planned start/end times or total planned hours.
By comparing this table with the Attendance Records, the system can display planned vs. realized hour**s** for children and groups.

- Auditor table:
  This table records important actions for security and transparency.
  It stores an ID, the user ID of the person who performed the action, the type of action (for example, login, update children, change attendance), optional details, and a timestamp.
  This makes it possible to see who changed what and when, which is important when handling sensitive children's data.

4.2 GUI Design

The user interface is designed to be calm, simple, and easy to use for managers, caretakers, and parents. The application follows a dashboard layout with a clear top navigation bar and a central content area that changes based on the selected role and page. Common elements—tables, buttons, and forms—are implemented as reusable React components so that the look and behavior stay consistent.

Each role sees only the tools it needs. Managers get data-oriented views for children, groups, and caretakers; caretakers see a daily list of children with quick actions to record arrival/leaving times and add comments; parents see a compact children's profile with attendance history and notes. The GUI is responsive, so the same interface works on desktops, tablets, and phones, with large tap targets and simple validation messages to support fast everyday use.

4.2.1 Admin interface

The admin interface is designed for the person who is responsible for "running the system in the kindergarten. The idea is that an admin should be able to quickly manage users, children, and groups, and get a basic overview of what is happening in the kindergarten.

The top bar shows the Children Daycare logo and system name, a friendly greeting like " Welcome, Dhar!", and the manager's profile picture, so it's always clear who is logged in.

Key statistics at a glance

- Total Groups: show how many groups are currently set up in the daycare.
- Total Nurses – displays the total number of nurses/staff in the system.
- Total Active Parents – indicates how many parents are actively using the platform.
- Total Children – shows the total number of children registered.
- Today's Attendance – highlights how many children are checked in for the current day.

Recent activity:

A Recent Updates panel summarizes the latest changes or announcements in the system. If no new actions have been recorded, the panel clearly indicates that there are no recent updates, ensuring the dashboard remains easy to interpret.

Absence management:

An Upcoming Requested Absences section lists children with planned absences, including the reason, dates, and duration. The manager can quickly respond to each request with Approve and Reject buttons directly from the dashboard.

Overall, this admin dashboard gives the daycare manager a clear, friendly overview of what's happening today and what needs attention, all in one place.

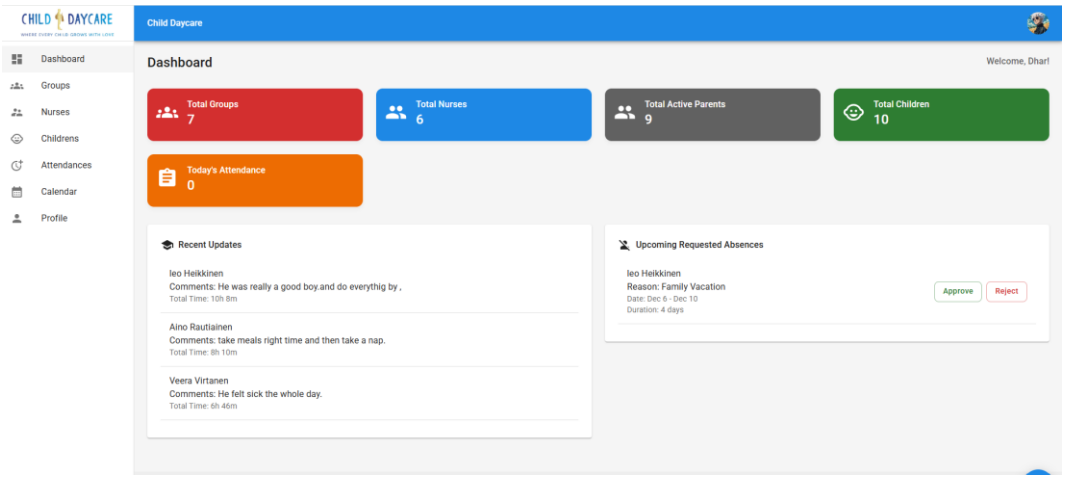

Figure 19. Admin Dashboard

4.2.2 Caregiver interface

This dashboard is designed around the caregiver's everyday work with their group of children and keeps everything necessary on a single screen:

At the top, the caregiver is greeted by name, along with three clear summary cards showing Assigned Groups, Assigned Children, and Today's Attendance. This lets them instantly see how many children and groups they are responsible for and how many have already been marked present.

The My Groups panel lists the caregiver's current group (for example, Happy Hippos) with a quick link to View All Groups for more detailed group management. The My Children panel shows the children assigned to the caregiver along with their ages, plus a View All Children button for a more complete overview.

Recent Updates keep the caregiver informed about any new information related to children, groups, or the daycare. Upcoming Requested

Absences summarize absence requests connected to the caregiver's children, so they know who to expect on upcoming days.

At the bottom, a Quick Actions area highlights the main task of the day: a large Take Attendance button that opens the attendance view, where arrivals and departures can be recorded quickly.

The overall layout focuses on speed and clarity, helping caregivers move through busy mornings and afternoons without unnecessary clicks or distractions.

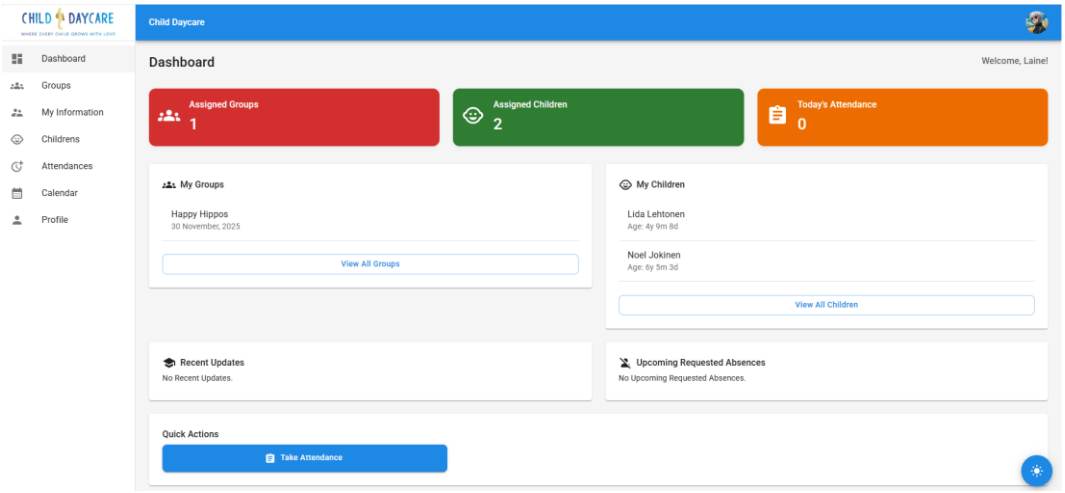

Figure 20. Caretaker (Nurse) Dashboard

4.2.3 Parent interface

This parent dashboard offers a clear, friendly overview of a family's daycare activity.

- Personalized welcome:
  The top of the page greets the logged-in Parent by name, making it easy to see which account is currently active.
- Children's cards:
  Each child appears in a dedicated card showing their name, age, and group. The card also displays the children's monthly stay hours, so parents can quickly see how much care time is being

used. A prominent "Request Absence" button on each card lets parents submit an absence for that specific child in just one click.

- Recent Updates panel:
  A simple summary area shows any new updates regarding the children or the daycare. When there's nothing new, it clearly states that there are no recent updates, keeping the interface clean and uncluttered.
- Upcoming Absences section:
  A detailed list shows all upcoming absence requests with the children's names, reasons, dates, and durations. Each request is marked with a clear status badge, such as Approved or Rejected, so parents instantly know the decision.

The overall layout focuses on what parents care about most: their children's basic details, how often they attend, and the status of any absence requests—all presented in a clean, easy-to-read design.

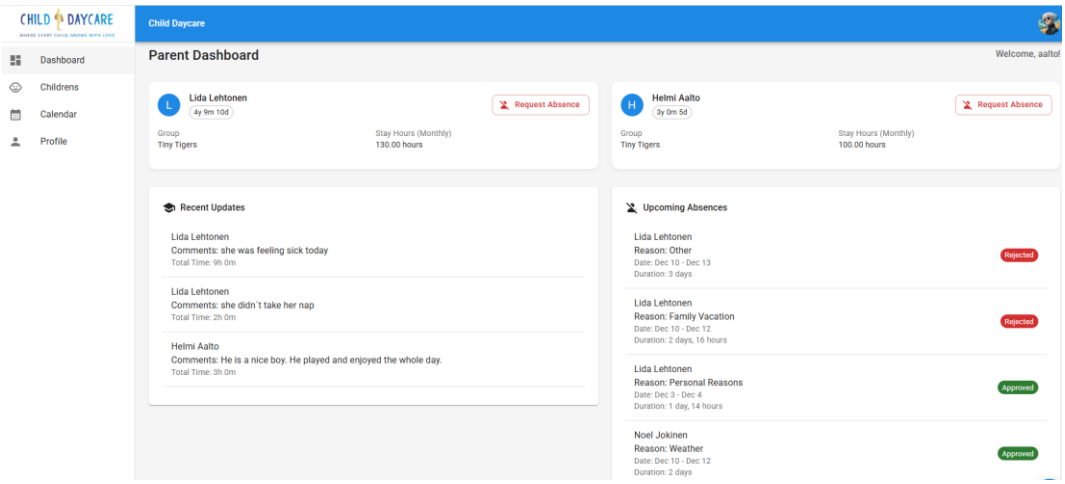

Figure 21. Parent Dashboard.

5 IMPLEMENTATION

This chapter explains how the Children's Daycare system was actually built. Earlier chapters described what the application should do and which technologies were selected; here, the focus is on how those ideas were turned into a working web application for admins, caregivers, and parents.

The implementation is divided into a few main parts:

Frontend – a React + TypeScript single-page application running in the browser, providing the different dashboards and views for each user role.

Backend – a PHP REST API on the server that handles all core operations, such as managing users, children, groups, attendance, absences, and daily notes.

Database – a MySQL database (db_children_daycare) that stores structured data in tables like cd_users, cd_childrenren, cd_children_groups, cd_nurses, cd_attendance, cd_absences, and cd_daily_notes.

Moreover, the chapter outlines the real world application of the login and security measures, which includes user authentication through email and password, hashing of the credentials before being stored in the cd_users table, and role-based access control which limits functionality based on whether the user who is logged in is an administrator, caregiver or a parent. Finally, it describes how the frontend, API, and the database will be deployed on the same server at the same time, thus forming a full daycare management system.

5.1 Back-end and Front-end Setup

The project is split into two parts: a PHP back end (backend folder) and a React + TypeScript front end (frontend folder). This keeps the API and the user interface clearly separated and easy to maintain.

On the back end, PHP exposes REST API endpoints for users, children, groups, and attendance. The code uses a central config.php file to connect to the MySQL database. Authentication is handled with JWT bearer tokens, passwords are stored with bcrypt, and simple role-based checks (Admin, Caregiver, Parent) control access to each endpoint. Important actions are written to an audit log in the database.

The front end is a Vite-based React/TypeScript app. It is organized into pages, components, hooks, and stores. Main libraries from the package. JSON includes React Router for navigation, Redux Toolkit (with React-Redux and Redux-Persist) for state management, Axios for calling the PHP APIs, and Material-UI for building responsive forms and tables. This setup provides a modern, fast UI that talks to the secure PHP REST API and presents different views for the three user roles.

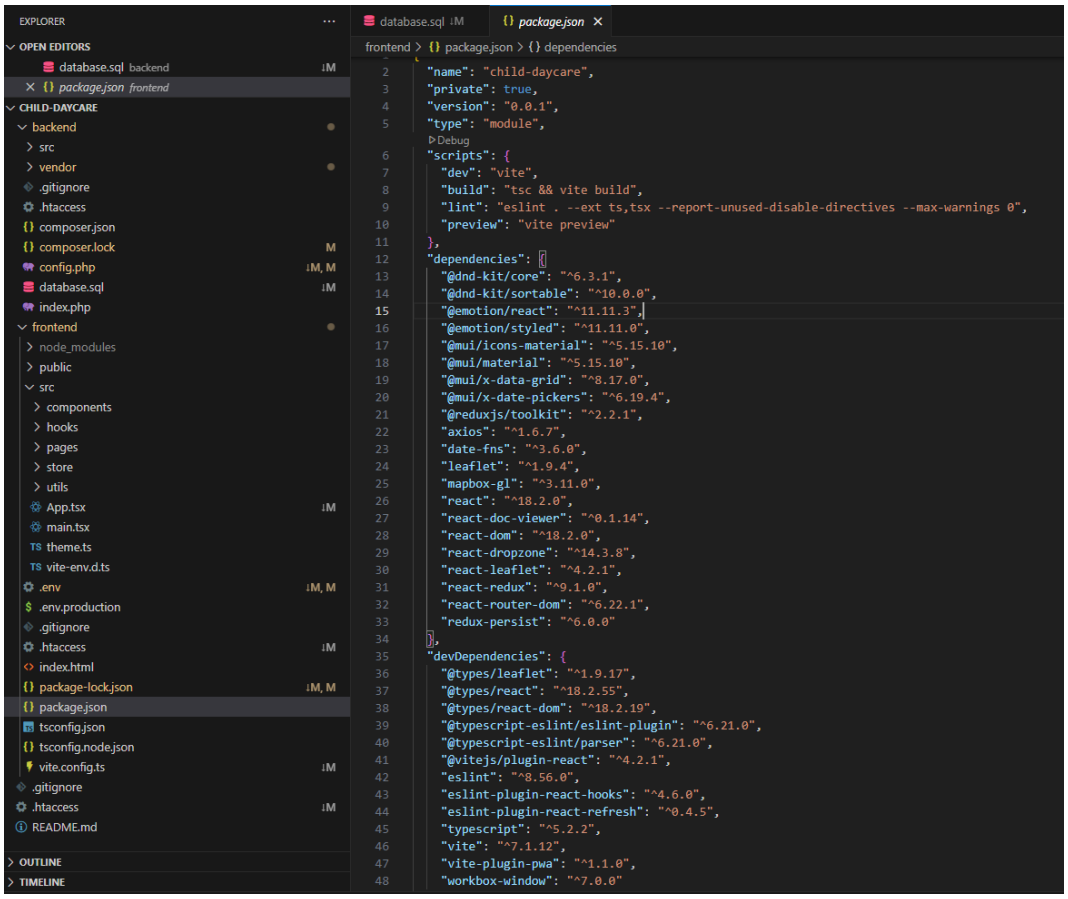

Figure 22. React frontend configuration (package. Json)

5.2 Version Control

Git was used to oversee the management of the project in order to document all changes in the source code. The backend (PHP + MySQL) and the frontend (React + TypeScript) directories are stored in one repository and allow the cloning and building of the entire system in the same location. The repository is stored on some remote host (e.g., GitHub or GitLab), which is an off-site backup and facilitates the transfer of the project between the machines.

In the development, the developer made short and specific commits with explanatory messages (e.g., add attendance API routes or connect frontend to login endpoint). New development and experimentation

were done on other branches and were integrated with the original branch when they became working and had been tested. This process made it easy to turn back mistakes in a sequence, compare versions, and record how the kindergarten system has evolved through the years.

5.3 Back-end version control

The PHP-based backend is located in the backend folder of the Git repository, and it is kept as a version-controlled entity, independent of the React frontend. This part of the project has all the API code, configuration templates, and SQL scripts. All major changes to the backend (adding a new endpoint, changing validation logic, or updating the database schema) were made through the use of Git with succinct descriptive notes (e.g., add attendance POST endpoint, hash passwords with bcrypt).

Dependencies with the composer in the vendor folder, env file, and other machine-specific artefacts are excluded through the use of .gitignore, so that only the actual source code and a template config.example.php file are in the repository. This setup allows easy cloning of the project, followed by running the composer install, and duplication of the backend environment. Besides, the commit history records the development of the API and database layer over the period of developing the kindergarten system.

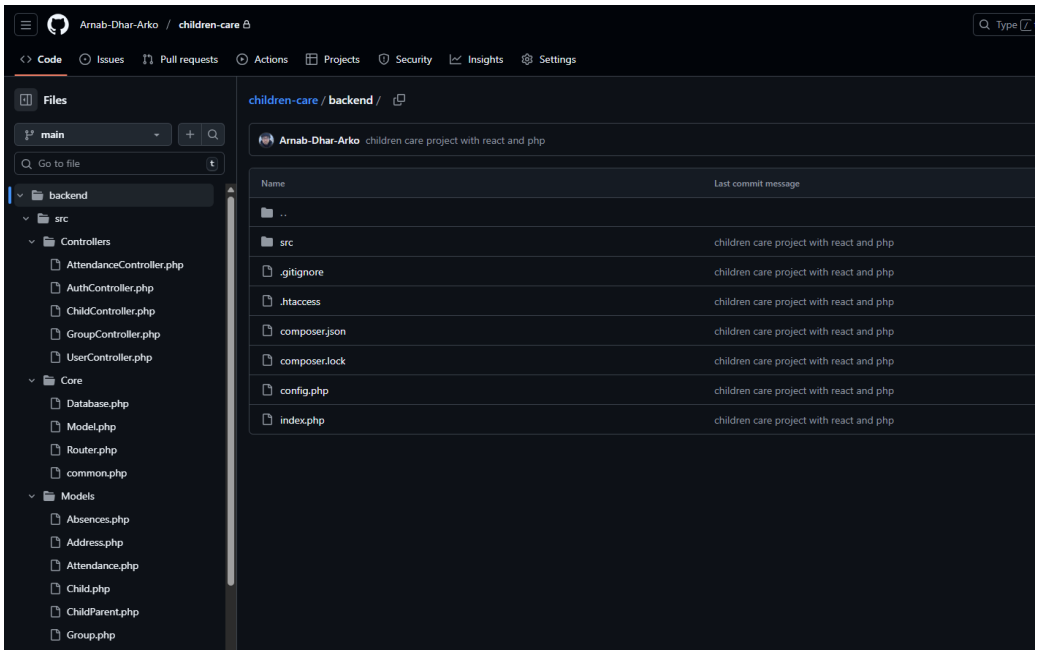

Figure 23. Git repository: backend source code

5.4 Front-end version control

The React + TypeScript front end is stored in the frontend folder of the same Git repository and is versioned independently from the PHP back end. All UI code, react components, pages, hooks, and state management files are committed here. I used Git to record each change, for example, when creating a new page, connecting a form to an API endpoint, or adjusting the layout, using clear commit messages such as *"add parent dashboard page"* or *"connect login form to auth API"*.

The node modules folder, Vite build output and environment files (such as .env) are excluded with. .gitignore, so the repository contains only the actual source code and configuration. This makes it easy to clone the project, run npm install, and reproduce the same front-end setup, while the commit history shows how the user interface of the kindergarten system evolved over time.

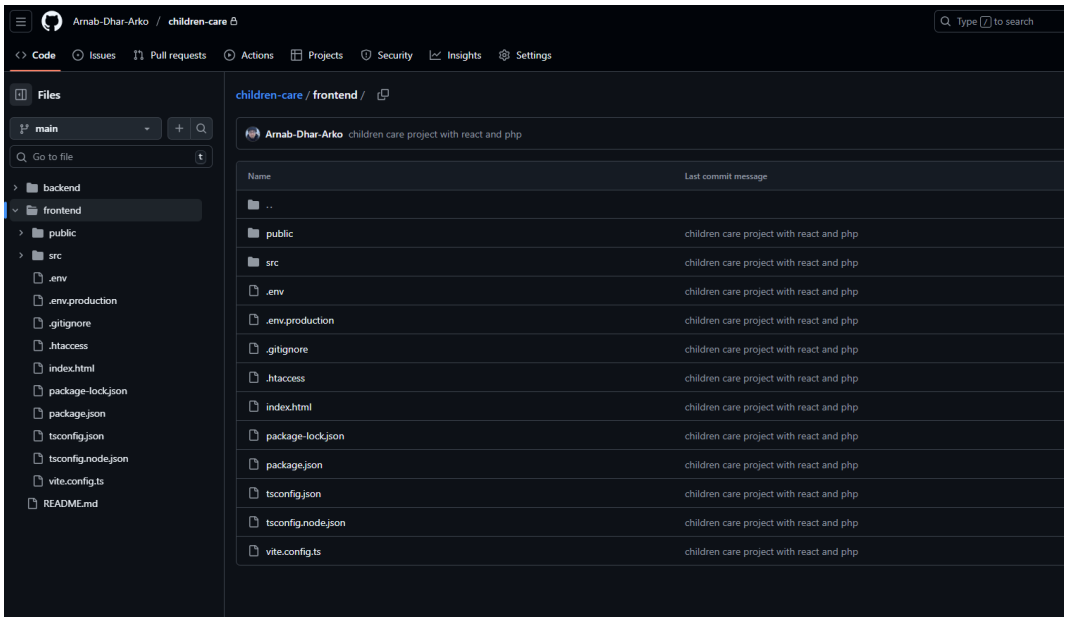

Figure 24. Git repository: frontend source code

5.5 Backend Structure

The backend of the kindergarten web application is implemented as a PHP-based REST API built on a MySQL database. The Children Daycare backend is a small PHP REST API that communicates with the db_children_daycare MySQL database and returns JSON to the React app.

A custom router (Core/Router.php) looks at the URL (e.g. /API/login, /API/children, /API/attendance) and sends the request to the right controller.

The controllers (AuthController, UserController, ChildrenController, GroupController, AttendanceController) contain the main daycare logic: logging in, managing users, children, groups, and attendance.

The models (such as User.php, Children.php, Group.php, Attendance, Absences.php, Notes.php) handle all database queries.

Shared core files like Database, Model.php, and common.php provide the database connection and helper functions.

Simple HTML templates in Views/emails are used for emails such as credentials, password resets, and registration verifications.

Passwords are hashed before saving, and each request checks the user’s role (admin, caregiver, parent), protecting the data while keeping it easy to maintain and extend.

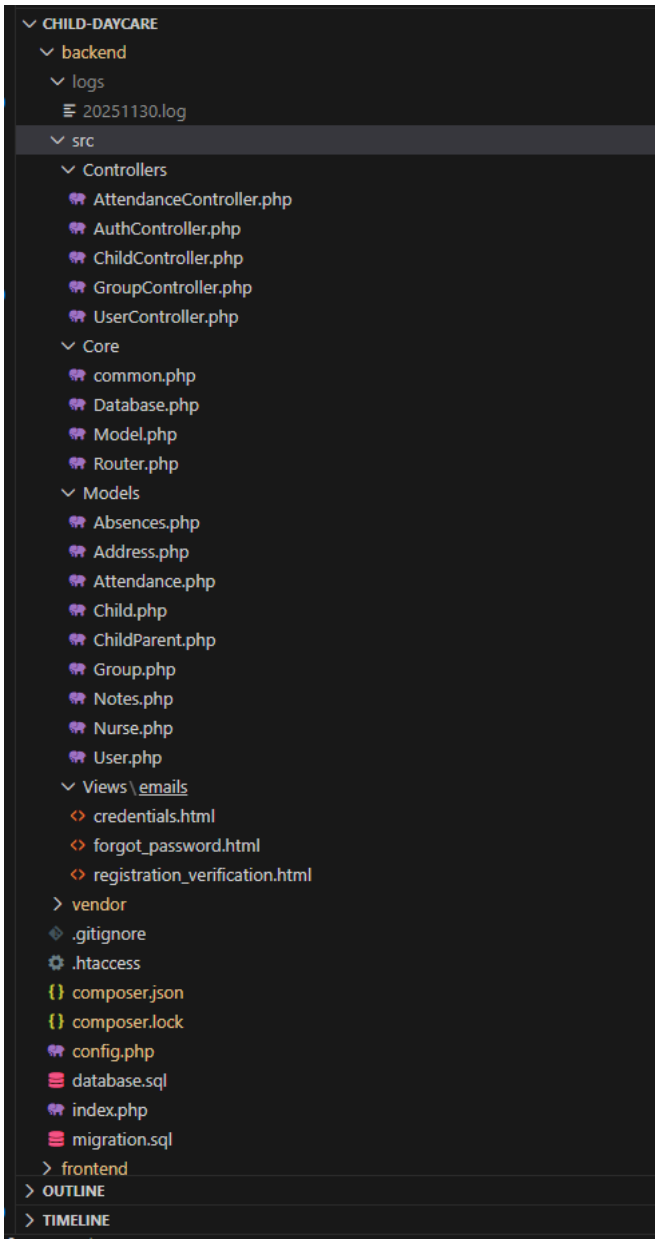

Figure 25. Backend project structure (directory layout)

5.5.1 Configurations and Environments

The backend structure is designed around a few central files that start up the application, start configuration parameters, and provide database connectivity. This kind of arrangement enables local development and deployment in a low-cost hosting environment.

The main entry point of the inbound traffic is the file index.php. It loads the necessary dependencies, starts the router and redirects every request to the necessary controller method. This approach ensures that the application start-up process is succinct and that distinction between routing and business logic is ensured.

```
<?php
require_once __DIR__ . "/config.php";
require_once __DIR__ . "/src/Core/Router.php";
require_once __DIR__ . "/src/Core/Database.php";
use Core\Router;
header("Content-Type: application/json; charset=UTF-8");
$router = new Router();
$router->registerRoutes();
$router->dispatch($_SERVER['REQUEST_METHOD'],
$_SERVER['REQUEST_URI']);
```

Code Snippet 1. Application entry point (index.php)

Environment-specific configurations, such as the database credentials and base path definitions, are contained in the environs config.php file. By centralizing these parameters, the hardcoding of sensitive parameters throughout the codebase is avoided, simplifying deployment.

```
define('DB_HOST','localhost');
define('DB_USER','root');
define('DB_PASS',''); // use env variables in production
define('DB_NAME','db_children_daycare');
define('JWT_SECRET','CHANGE_ME'); // never commit real secrets
define('JWT_EXPIRE',3600); // seconds
define('CORS_ORIGIN','http://localhost:5173');
```

Code Snippet 2. Application configuration variables (config.php)

The backend communicates with the frontend using RESTful endpoints. Requests and responses are structured in JSON to ensure consistent data exchange among modules, including authentication, users, children, groups, attendance, absences, and daily notes.

5.5.2 Database Connection and Data Access

MySQL is used as the persistent storage layer in the architecture. The database schema is defined in SQL scripts, e.g., database.sql and

migration.sql, where each real-life object, e.g., users, children, groups, attendance, absences, and so forth, is represented in its own table. The relational entities amongst these are conceived through the use of foreign keys and in case of the need, linking tables.

A special database class will be responsible for provisioning an object-oriented PDO connection, which will, in turn, provide a safe and stable substrate for all CRUD activities within the project.

```
class Database{
 private static ?PDO $conn=null;
 public static function getConnection(): PDO{
  if(!self::$conn){

$dsn="mysql:host=".DB_HOST.";dbname=".DB_NAME.";charset=utf8mb4";
   self::$conn=new
PDO($dsn,DB_USER,DB_PASS,[PDO::ATTR_ERRMODE=>PDO::ERRMODE_EXCEPTIO
N,PDO::ATTR_DEFAULT_FETCH_MODE=>PDO::FETCH_ASSOC]);
  }
  return self::$conn;
 }
}
```

Code Snippet 3. Database connection using PDO (Database.php)

5.5.3 Controllers and RESTful Endpoints

The backend uses controller classes found in the Controllers directory. Each controller checks requests, manages role-based permissions, and calls model methods to work with the database.

For example, the attendance controller records arrival and departure times and retrieves attendance history. This setup keeps the code modular and makes it easier to maintain as the system grows.

```
$currentUserId = $this->user->getUserIdFromToken();
if (!$this->user->isAdminOrNurse($currentUserId)) {
    http_response_code(403);
    return ['message' => 'Unauthorized'];
}
```

Code Snippet 4. Permission check in controller layer (authorization example)

Routing Layer

To connect endpoints to controllers, the system includes a routing mechanism in the Core layer. The router maps HTTP methods and URL patterns to specific controller actions. This ensures a clean separation between request paths and business logic

```
class Router{
 private array $routes=[];
 public function add(string $m,string $p,callable $h): void{$this-
>routes[]=['m'=>strtoupper($m),'p'=>$p,'h'=>$h];}
 private function match(string $m,string $path): ?array{
  foreach($this->routes as $r){
   if($r['m']!==$m) continue;
   $pat=preg_replace('/\{([^}]+)\}/','(?P<$1>[^/]+)',$r['p']);
   if(preg_match('#^'.$pat.'$#',$path,$mch)) return
[$r['h'],array_filter($mch,'is_string',ARRAY_FILTER_USE_KEY)];
  }
  return null;
 }
 public function dispatch(): void{
  $m=$_SERVER['REQUEST_METHOD'];
$p=parse_url($_SERVER['REQUEST_URI'],PHP_URL_PATH);
  $match=$this->match($m,$p);
  if(!$match){http_response_code(404); echo
json_encode(['error'=>'Not Found']); return;}
  [$h,$params]=$match; header("Content-Type: application/json;
charset=UTF-8");
  echo json_encode(call_user_func($h,$params));
 }
}
```

Code Snippet 5. API route mapping (router definition)

5.5.4 Authentication and Authorization

The system of authentication and authorization used by the backend of the Child Daycare is based on the use of the JSON Web Tokens (JWTs) and the role-based access control (RBAC). The validation of user credentials is against the table of a MySQL database named as cd-users, password-hashing is done through the use of the bcrypt. After successful authentication results in the issuance of a JWT which contains the user identifier and the role and must be presented in the Authorization header in future requests to secured endpoints.

The manager, nurse and parent groups are subjected to role-based restrictions. Cross-Origin Resource Sharing (CORS) is also enabled via PHP that allows only requests with the trusted frontend origin in development. The role under discussion carries out the process of validation of entered e-mail and password with the database. In case the given password matches the stored bcrypt hash, the function creates and identifies access and refresh tokens.

```
class AuthController{
 private User $user;
 public function __construct(){ $this->user=new User(); }
 public function login(){
  $d=json_decode(file_get_contents('php://input'),true) ?? [];
  if(empty($d['email'])||empty($d['password'])){
http_response_code(400); return ['message'=>'Email and password
are required']; }
  $u=$this->user->findByEmail($d['email']);
  if(!$u||!$this->user-
>verifyPassword($d['password'],$u['password'])){
http_response_code(401); return ['message'=>'Invalid email or
password']; }
  unset($u['password']); return ['user'=>$u,'token'=>$this-
>generateToken($u)];
 }
 private function generateToken(array $u): string{

$pl=['sub'=>$u['id'],'role'=>$u['role'],'iat'=>time(),'exp'=>time(
)+JWT_EXPIRE];
  $h=base64_encode(json_encode(['typ'=>'JWT','alg'=>'HS256']));
  $b=base64_encode(json_encode($pl));
  $s=base64_encode(hash_hmac('sha256',"$h.$b",JWT_SECRET,true));
  return "$h.$b.$s";
 }
}
```

Code Snippet 6. User login and JWT generation (authentication)

This handler verifies the refresh token and provides a new access token without requiring the user to log in again. It improves usability while keeping access tokens short-lived.

```
<?php
function verifyJwt(string $token): ?array{
 $p=explode('.',$token);
 if(count($p)!==3) return null;
 [$h,$b,$sig]=$p;
 $calc=base64_encode(hash_hmac('sha256',"$h.$b",JWT_SECRET,true));
 if(!hash_equals($calc,$sig)) return null;
 $pl=json_decode(base64_decode($b),true) ?? [];
 if(($pl['exp'] ?? 0) < time()) return null;
 return $pl;
}
```

Code Snippet 7. JWT verification function (signature and expiry validation)

This middleware extracts the JWT from the Authorization header, verifies it, and sets the authenticated user context.

```
<?php
function authenticateToken(): array{
 $auth=getallheaders()['Authorization'] ?? '';
 if(!str_starts_with($auth,'Bearer ')){ http_response_code(401);
exit(json_encode(['message'=>'Access token required'])); }
 $pl=verifyJwt(substr($auth,7));
 if(!$pl){ http_response_code(403);
exit(json_encode(['message'=>'Invalid or expired token'])); }
 $GLOBALS['auth_user']=['id'=>$pl['sub'],'role'=>$pl['role']];
 return $GLOBALS['auth_user'];
}
```

Code Snippet 8. Authentication middleware using JWT (protecting endpoints)

This function ensures that only users with allowed roles can access a route. It prevents role abuse and limits access to sensitive operations.

```
<?php
function authorizeRoles(array $allowed): void{
 $u=$GLOBALS['auth_user'] ?? null;
 if(!$u||!in_array($u['role'],$allowed,true)){
http_response_code(403); exit(json_encode(['message'=>'Access
denied'])); }
}
```

Code Snippet 9. Role-based access control middleware (RBAC)

This example shows how authentication and role checks are combined on a real endpoint. It demonstrates the practical enforcement of security rules in the attendance module.

```
<?php
authenticateToken();
authorizeRoles(['admin', 'nurse']);
```

Code Snippet 10. Applying RBAC to a protected route (usage example)

This configuration allows cross-origin requests only from the trusted frontend URL. It enables safe frontend-backend communication during development.

```
<?php
header('Access-Control-Allow-Origin: ' . CORS_ORIGIN);
header('Access-Control-Allow-Methods: GET, POST, PUT, DELETE,
OPTIONS');
header('Access-Control-Allow-Headers: Content-Type,
Authorization');
if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(200);
    exit;
}
```

Code Snippet 11. CORS configuration for API access (security configuration)

5.6 Authentication, security, and deployment

Authentication, security, and deployment were key considerations in the implementation of the kindergarten web application, since the system handles sensitive data about children and families. User authentication is based on JSON Web Tokens (JWT): when a role based user logs in with their email and password, the backend verifies the credentials using bcrypt-hashed passwords stored in the database and, if successful, issues a signed JWT that contains the user ID and role (admin, caregiver, or Parent). This token is then sent with each subsequent request in the Authorization header, and the backend checks it on every protected endpoint before allowing access.

Role-based access control is enforced so that admins can manage users and groups, caregivers can only work with children in their own groups, and parents can only view information related to their own children. Security is further strengthened by using HTTPS to encrypt traffic between the browser and server, configuring CORS so that only the trusted frontend can call the API, and hash-storing all passwords with bcrypt rather than in plain text.

Important actions, such as logins and updates to children or attendance data, are written to an audit log to provide traceability. For deployment, the whole system is designed to run on a single VPS or shared hosting environment, with the web server serving both the built React frontend and the PHP API, and a MySQL database running on the same host or a small managed instance. This setup keeps deployment and maintenance relatively simple while still providing a secure and robust environment suitable for a small kindergarten.

5.7 Authentication and Authorization

Child Daycare backend authentication and authorization are provided using JSON Web Tokens (JWT) and role-based access control (RBAC)

scheme. Authentication is done by users supplying an email address and a password that are verified against bcrypt-hashed user credentials, stored in a MySQL database. On successful authentication the backend will issue an access token with a short-lived validity period and a refresh token with a long-lived validity period. Sensitive operations, as well as protected routes, are additionally limited by verifying the role of the authenticated user and require an access token that is valid at any time. Moreover, the PHP also has CORS headers that are set to permit requests by the trusted frontend origin only when in development.

5.8 RESTful API End Point

The back end exposes a set of RESTful API endpoints that the React frontend uses to communicate with the server. All routes are registered in the main PHP entry file with a custom Router class and mapped to the controllers AuthController, UserController, GroupController, ChildrenController, and AttendanceController. Each endpoint follows standard REST conventions (clear URL paths with HTTP methods GET, POST, PUT, DELETE) and returns JSON responses. Role-based access control ensures that only authorised roles (admin, caregiver, parent) can perform sensitive operations.

Table 2. Authentication Endpoints(Diffie et al., 1976)

Method	**Endpoint**	**Role Access**	**Content**
POST	/api/auth/login	All roles	Authenticate using existing user credentials.
POST	/api/auth/register	All roles (if enabled)	Register a new user account (subject to system rules).
POST	/api/auth/reset-password	All roles	Request or complete a password reset.

This table describes the API endpoints used to handle authentication. All roles can log in, register (depending on system configuration) and request a password reset. After successful login, the server issues a token that is used to access protected endpoints.

Table 3. User Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/users	Admin, Caregiver	Retrieve all user accounts.
GET	/api/users/{id}	Admin (any), Caregiver (self only), Parent (self only)	Retrieve a specific user by ID (restricted by role).
POST	/api/users	Admin	Create a new user account.
PUT	/api/users/bulk-update-group	Admin	Update the group assignment for multiple users.
PUT	/api/users/{id}	Admin	Update an existing user account.
DELETE	/api/users/{id}	Admin	Delete a user account.

This table lists the endpoints for managing user accounts. Managers can create, update, and delete users, while caretakers and parents are limited to viewing their own details through the single-user endpoint.

Table 4. Group Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/groups	Admin, Caregiver	List all children’s groups.
GET	/api/groups/{id}	Admin, Caregiver	Retrieve details for a specific group.
POST	/api/groups	Admin	Create a new children’s group.
PUT	/api/groups/{id}	Admin	Update an existing group.
DELETE	/api/groups/{id}	Admin	Delete a group.

This table lists the endpoints for children's groups. Caretakers can view groups, but only managers can create, change, or delete them.

Table 5. Children Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/children	Admin, Caregiver	List all children.
GET	/api/children/{id}	Admin, Caregiver, Parent (own child only)	Retrieve details for a specific child.
POST	/api/children	Admin	Create a new child record.
PUT	/api/children/bulk-update-group	Admin	Update group assignments for multiple children.
PUT	/api/children/bulk-update-nurse	Admin	Update nurse assignments for multiple children.
PUT	/api/children/{id}	Admin	Update an existing child record.
DELETE	/api/children/{id}	Admin	Delete a child record.

This table explains the endpoints for managing children. Managers can create and update records. Caretakers can view children's information, and parents can see details about their own children.

Table 6. Daily Notes Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/children/{id}/daily-notes	Admin, Caregiver, Parent (own child only)	List daily notes for a specific child.
POST	/api/daily-notes	Admin, Caregiver	Create a new daily note.
PUT	/api/daily-notes/{id}	Admin, Caregiver	Update an existing daily note.
DELETE	/api/daily-notes/{id}	Admin, Caregiver	Delete a daily note.

This table presents the endpoints used for daily notes between daycare and home. Caretakers and managers can create and update notes, while parents can read the notes linked to their children.

Table 7. Absence Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/children/{id}/absences	Admin, Caregiver, Parent (own child only)	List absences for a specific child.
POST	/api/absences	Parent, Caregiver, Admin	Create an absence request or record.
PUT	/api/absences/{id}	Admin, Caregiver	Update an absence (e.g., approve/reject or adjust).
DELETE	/api/absences/{id}	Admin, Caregiver	Delete an absence record.

This table shows the endpoints used for handling absences. Parents can create absence requests, while managers and caretakers can view, update, and remove absences as part of daily planning.

Table 8. Attendance Endpoints

Method	**Endpoint**	**Role Access**	**Content**
GET	/api/attendances	Admin, Caregiver	List attendance records (optionally filtered).
POST	/api/attendances/bulk	Admin, Caregiver	Create multiple attendance records in a single request.
PUT	/api/attendances/{id}	Admin, Caregiver	Update a specific attendance record.

This table summarizes the attendance endpoints, which managers and caretakers use to record arrivals and departures and to correct attendance information when necessary.

5.8.1 API Output (Postman)

Postman was used to show how the React frontend communicates with the PHP REST API and what the backend returns in JSON. The example in Figure X shows a GET request for fetching users using pagination parameters (page and limit) and the JSON response returned by the server. This is the same request–response format used by the application when loading data in the user interface.(*Send API Requests and Get Response Data in Postman | Postman Docs*, n.d.)

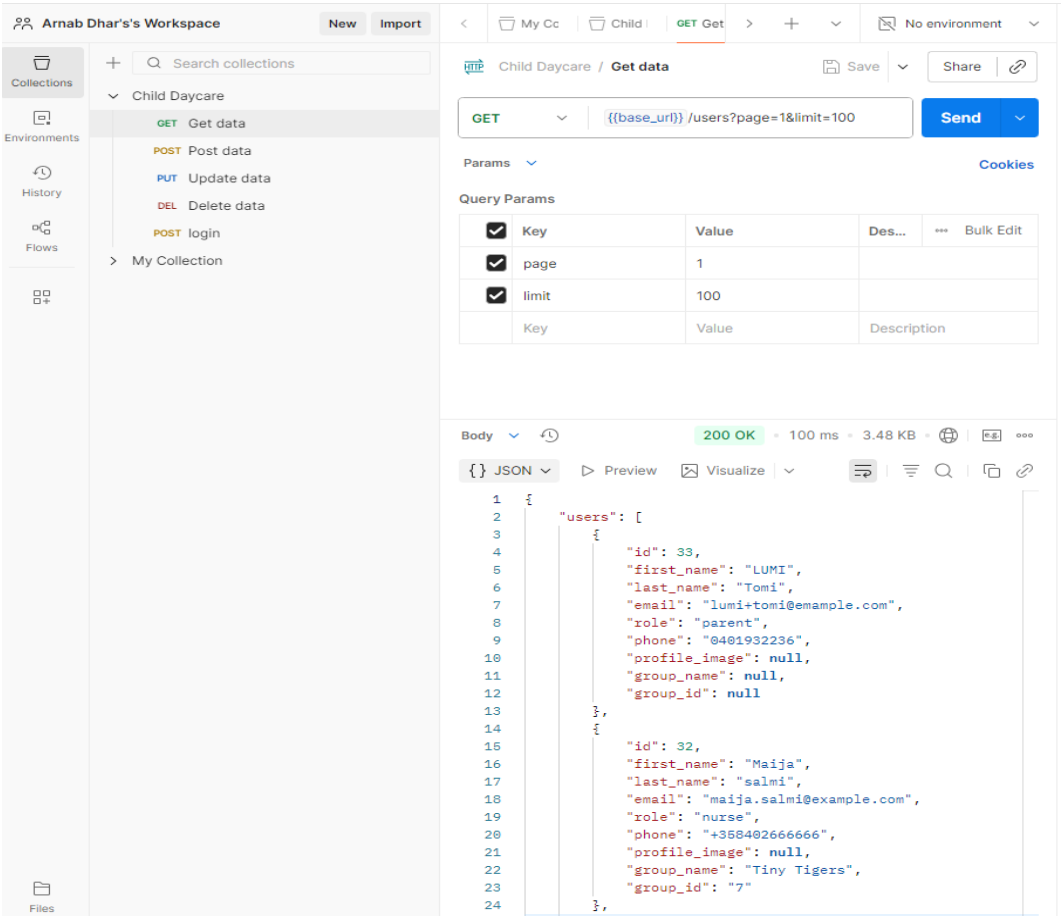

Figure 26. Postman request–response example (Send API Requests and Get Response Data in Postman | Postman Docs, n.d.)

5.8.2 Database uses MySQL

MySQL stores system tables for users, children, groups, nurses, absences, attendance, and daily records. The relational schema keeps data accurate and makes it easy to run queries, supporting daily operations and reporting(*MySQL HeatWave*, n.d.).

When users interact with the frontend, API calls are sent to the PHP server. The backend processes these requests, checks authentication, performs CRUD operations on MySQL, and sends JSON responses to update the user interface. This structure helps control access to sensitive child data and keeps records consistent for all users(Auth0, n.d.).

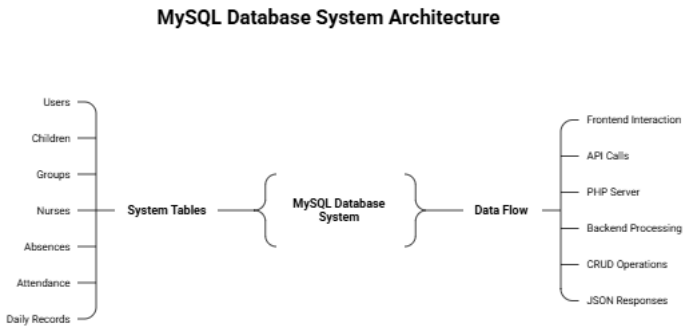

Figure 27. Architectural Diagram of the Daycare Child System

5.9 Front-end Structure

The Children Daycare client is a React + TypeScript single-page application built with Vite. It talks to the PHP REST API through a small API client and a custom use Api hook, sending and receiving JSON. The UI is role-based (manager, caregiver, parent) and focuses on keeping the main actions—dashboards, attendance, and absence handling—fast and easy to reach.

The core of the frontend lives in the src folder:

Components/ – reusable UI pieces, organized into subfolders such as common, dialogs, layout, map, and toolbars for things like forms, modals, page layouts, and top bars.

Pages/ – route-level views for each area of the system (dashboards, children's views, attendance screens, etc.).

hooks/ – shared logic for talking to the backend so that pages and components can load and save data with minimal code.

store/ – global state for the logged-in user, their role, and other shared data used across multiple pages.

utils/ – helper functions, small utilities, and shared theme/type definitions used throughout the app.

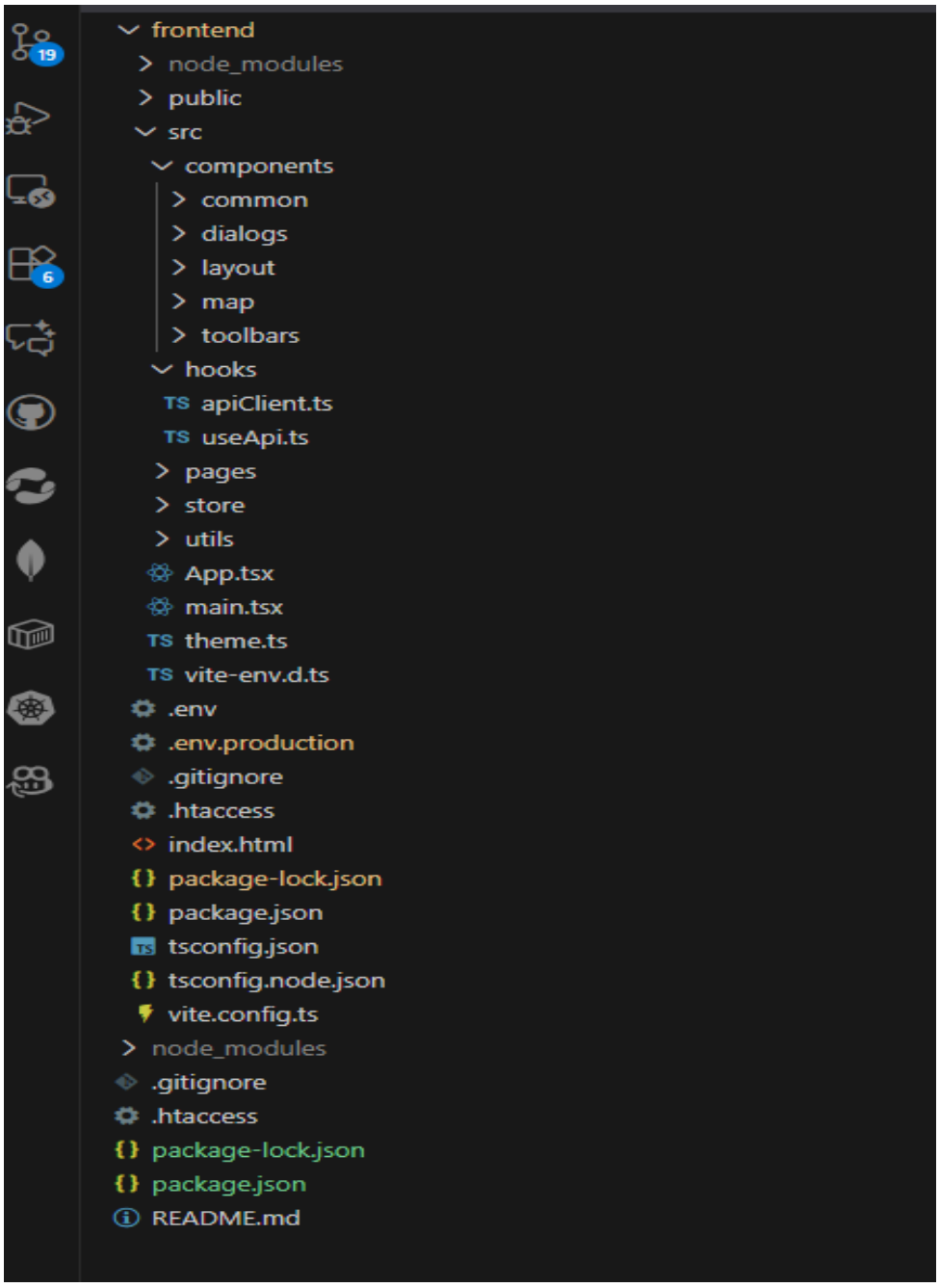

Figure 28. Frontend project structure (directory layout)

5.9.1 Authentication and Registration

Authentication and registration are implemented with React forms that send JSON requests to the PHP REST API. Client-side validation checks

required fields, email format, and password strength. After successful login, the API returns a JWT and role, which are stored in the global state and attached as a Bearer token to subsequent requests. Invalid credentials or expired tokens clear the session and redirect the user back to the login view.

Authentication (Login)

The login view is implemented as a React form that sends the user's email and password to the PHP REST API. On success, the frontend stores the returned JWT and role and uses them to protect routes and attach the Bearer token to subsequent requests.

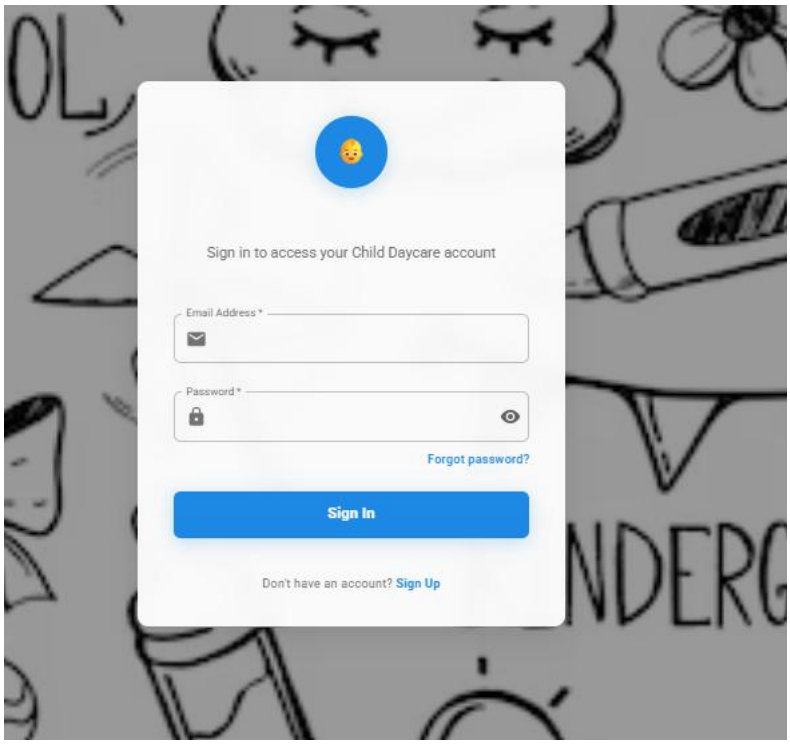

Figure 29. Login Form

Registration (Sign Up)

In this project, registration is handled by a two-step Signup view. In the first step, the user enters their personal information – first name, last name, and email address – in a clean, card-style form. After validation, the flow continues to the account setup step (where credentials and role

are defined) and finally sends the data to the backend to create a new daycare account.

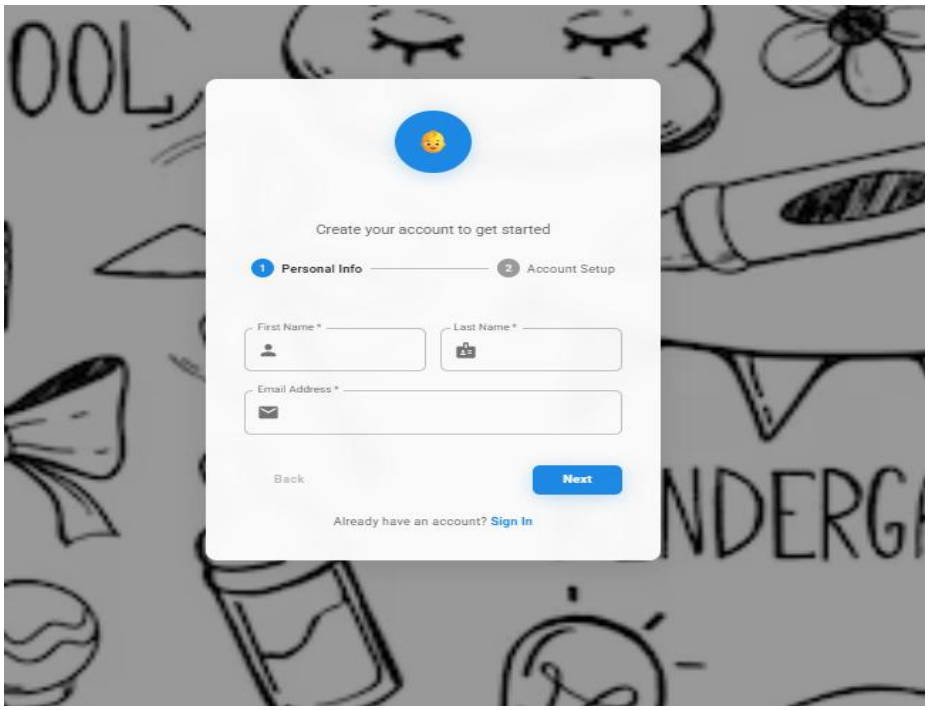

Figure 30. Registration Form

5.10 Application Demo

This section presents a practical walkthrough of the Children Daycare web application and its main features. The demo follows the system from the perspective of the three user roles: Manager, Caregiver, and Parent. For managers, it shows how the dashboard is used to oversee groups, children, nurses, attendance, and absence requests. The caregiver's view demonstrates daily work in the children's group, including taking attendance, checking assigned children, and viewing upcoming absences. The parent view highlights how families can see their children’s basic information, monthly hours, daily notes, and request absences online. Together, these role-based demos illustrate how the application supports everyday communication and coordination in the daycare environment.

5.10.1 Admin view

In the admin view, the application focuses on overall daycare management.

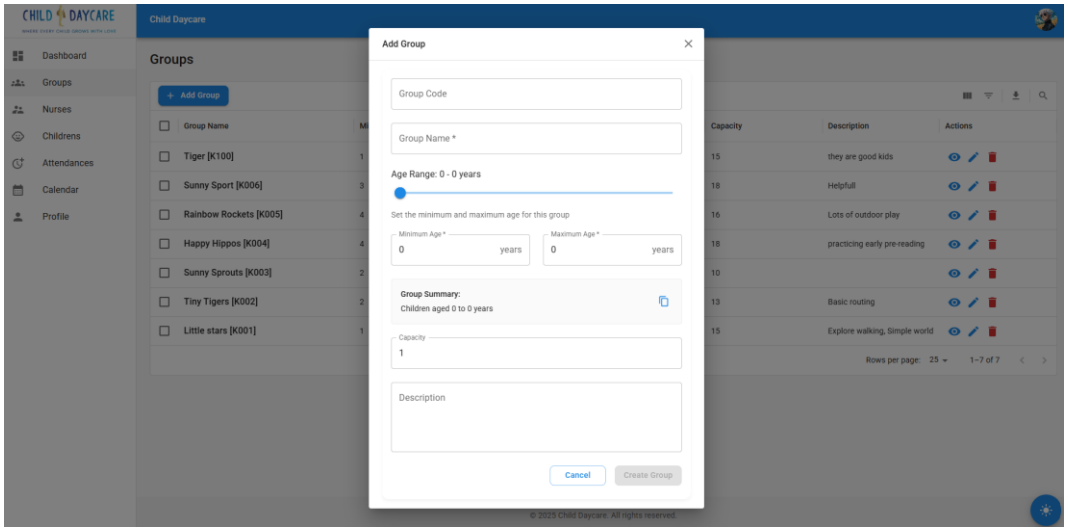

Figure 31. Create new group form

The admin Add Group dialog lets the manager define a new children group by setting its code, name, age range, capacity, and description directly from the groups list.

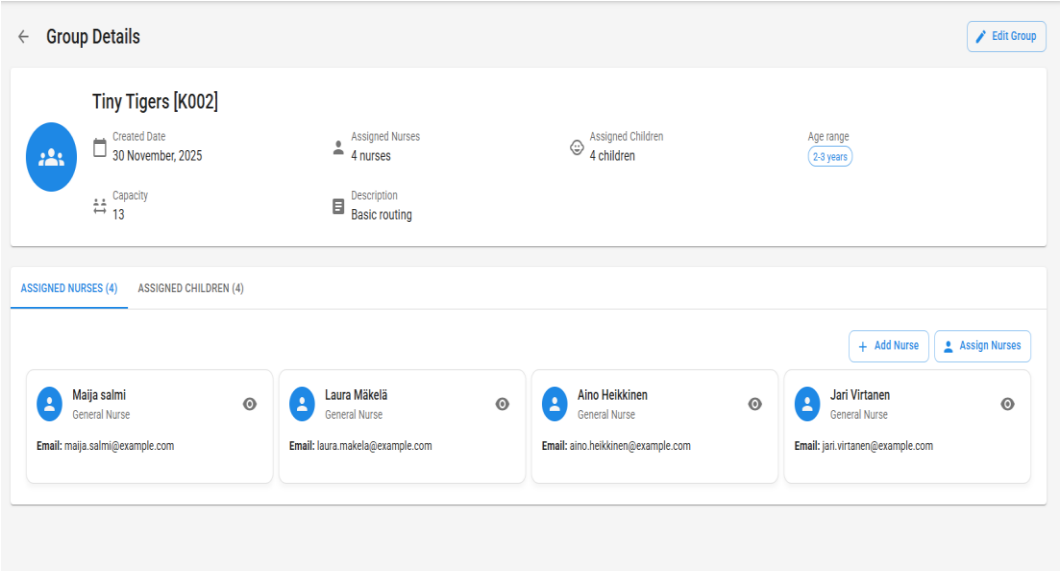

Figure 32. Group Details

The Group Details view shows all key information for a children's group (Tiny Tigers), including the creation date, capacity, assigned nurses and children, age range, and cards for each nurse, with quick actions to add or assign staff.

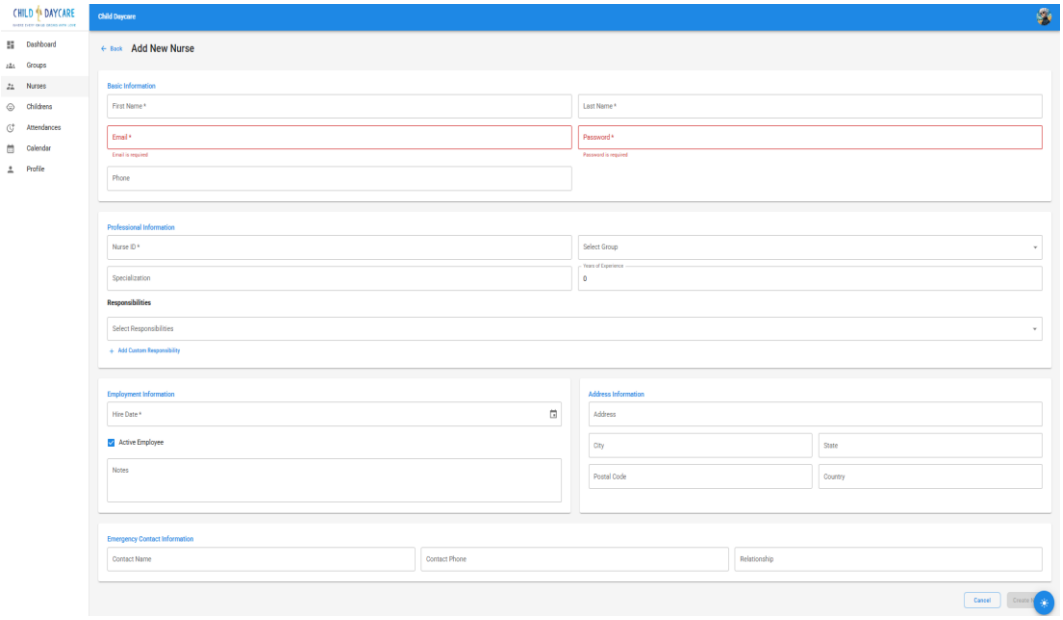

Figure 33. Add a new nurse

The admin Add New Nurse form lets the manager register a nurse in a single step, capturing personal details, professional information, responsibilities, employment data, address, and emergency contact.

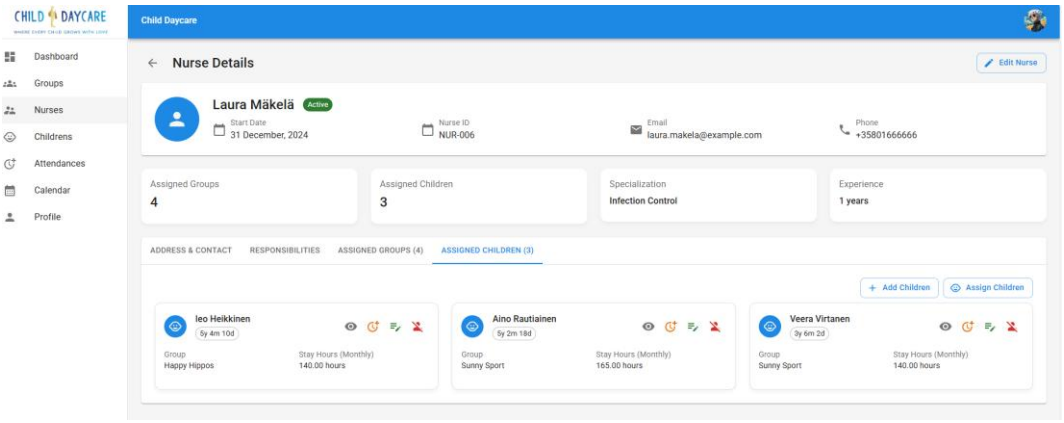

Figure 34. Nurse details page

The Nurse Details view on the admin side displays a nurse's full profile status, contact info, specialization, experience, assigned groups, and children on a single page. From here, the manager can review the nurse's workload and quickly add or reassign children as needed.

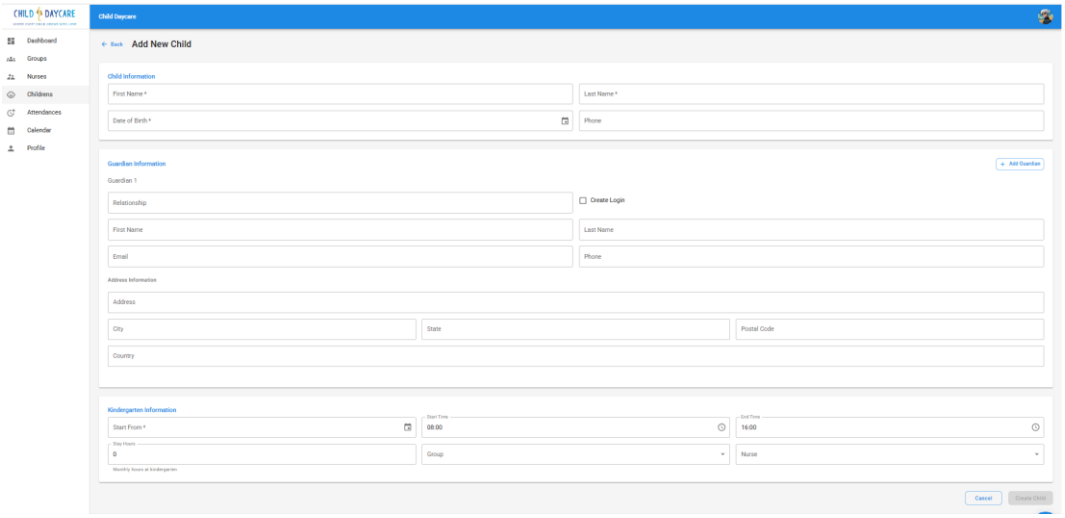

Figure 35. Create child profile form

The admin add New Children form lets the manager register a child, link guardians and their contact details, and define kindergarten information

such as start date, daily schedule, group, and assigned nurse in one place.

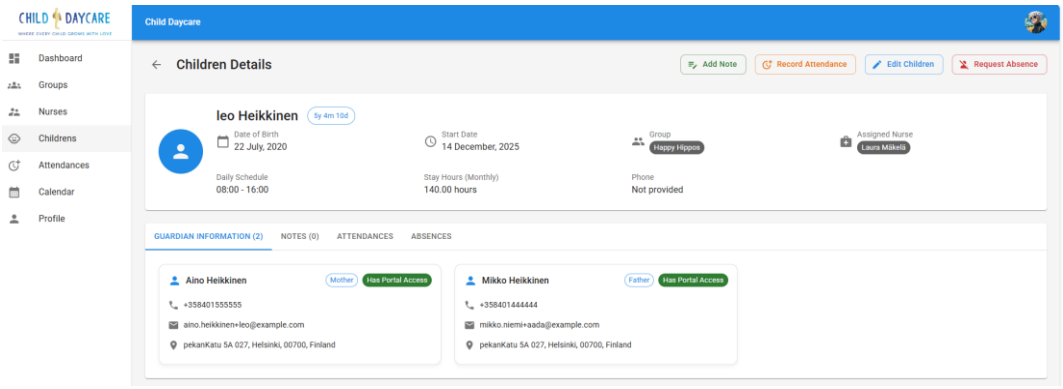

Figure 36. Child profile details page

The Children's Details view lets the admin see a child's full profile at a glance—basic info, daily schedule, stay hours, group, and assigned nurse—along with guardian contact details and tabs for notes, attendance, and absences, plus quick actions to add notes, record attendance, edit the children, or request an absence.

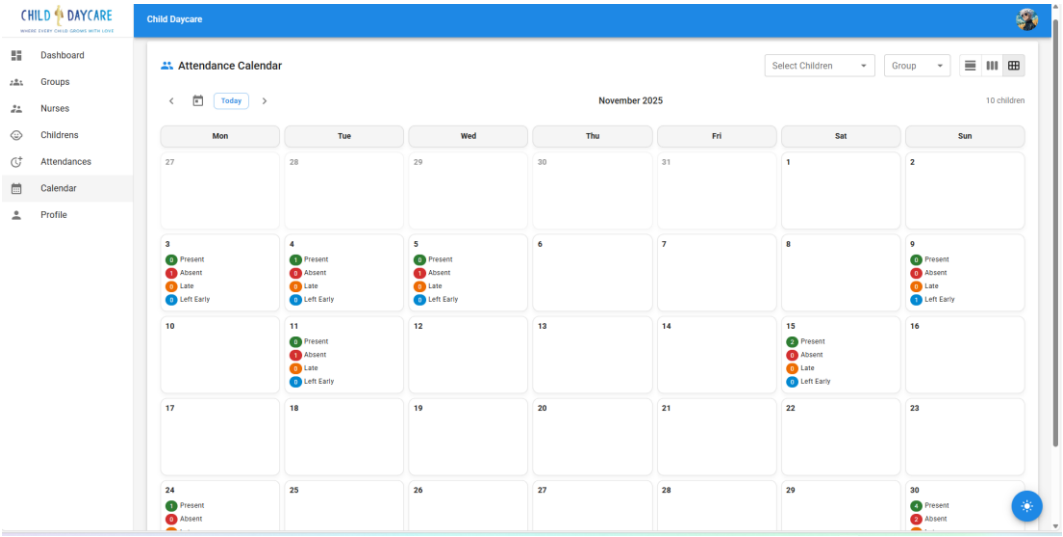

Figure 37. Admin Attendance Calendar

The Attendance Calendar view gives admins and caregivers a month-by-month overview of attendance, showing for each day how many children were present, absent, late, or left early. Filters for children or

groups, along with different layout options, make it easy to spot patterns and quickly review attendance history.

5.10.2 Caretaker

From the Caretaker's view, the application focuses on daily group work: they can quickly see their assigned groups and children, take attendance, record notes, and review upcoming absences. The interface is streamlined for busy mornings and afternoons so check-ins, check-outs, and small updates can be done with just a few clicks.

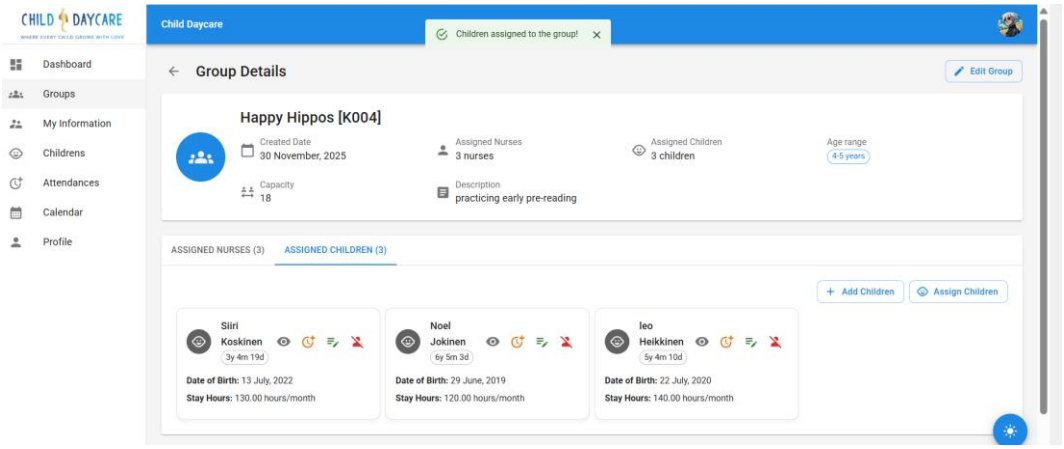

Figure 38. Group details page

The caretaker Group Details view summarizes a group's basic info and lists all assigned children with quick actions for viewing details, taking attendance, adding notes, or recording absences.

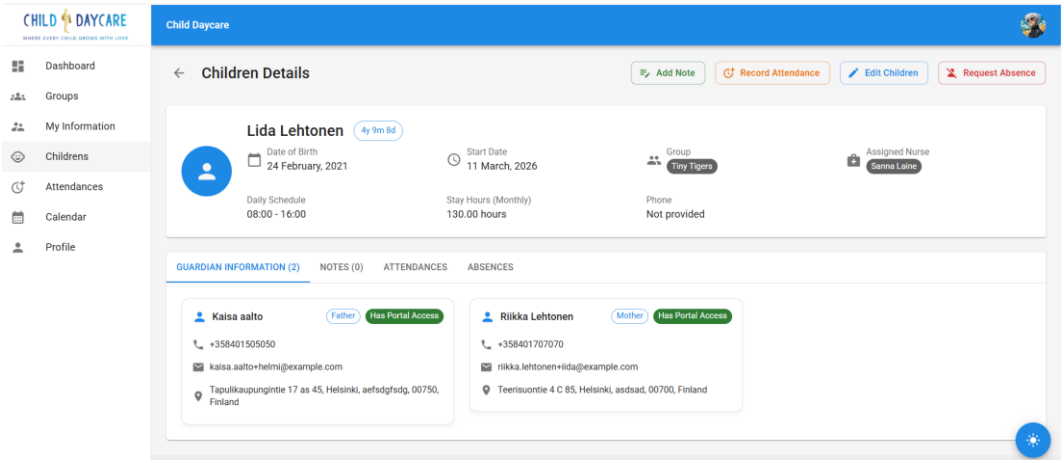

Figure 39. Caregiver Children's Details

The caregiver Children's Details view shows a child’s key information, guardians, group, and assigned nurse, with quick actions to add notes, record attendance, edit details, or request an absence.

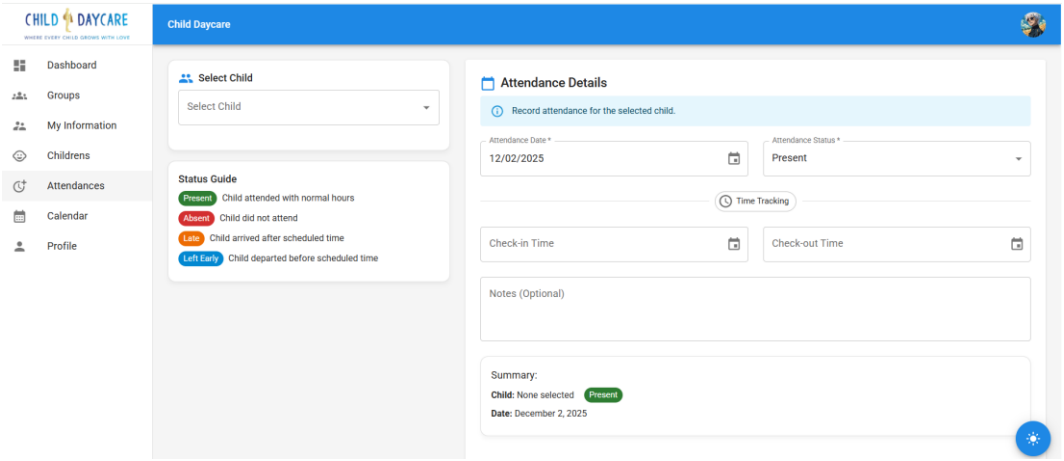

Figure 40. Caregiver Attendance Details

The caretaker Attendance Details view lets staff select a child, choose the date and status, record check-in/check-out times, and add notes, supported by a simple color-coded status guide.

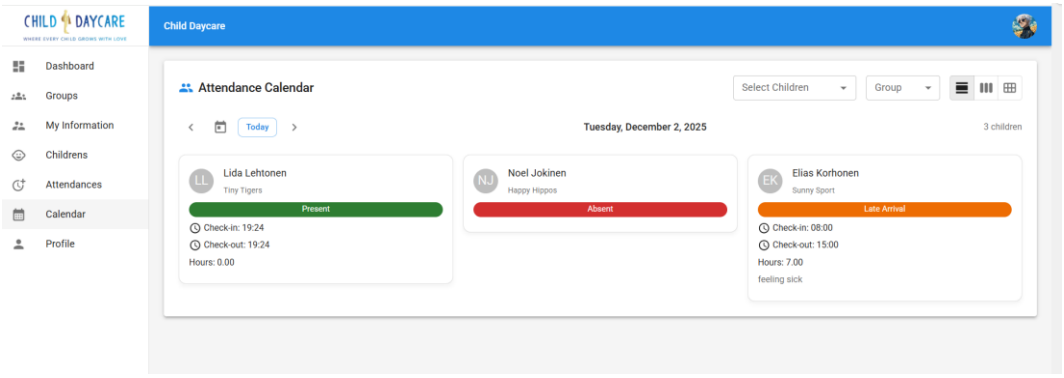

Figure 41. Caregiver Attendance Calendar

The caregiver Attendance Calendar (day view) shows each child's daily status card—present, absent, or late—with check-in/check-out times and hours, making it easy to review attendance at a glance.

5.10.3 Parents

For parents, the application provides a simple dashboard that shows each child's basic info, assigned group and nurse, monthly stay hours, daily notes, and upcoming absences. From the same view, parents can easily request new absences and stay up to date with what is happening in the daycare.

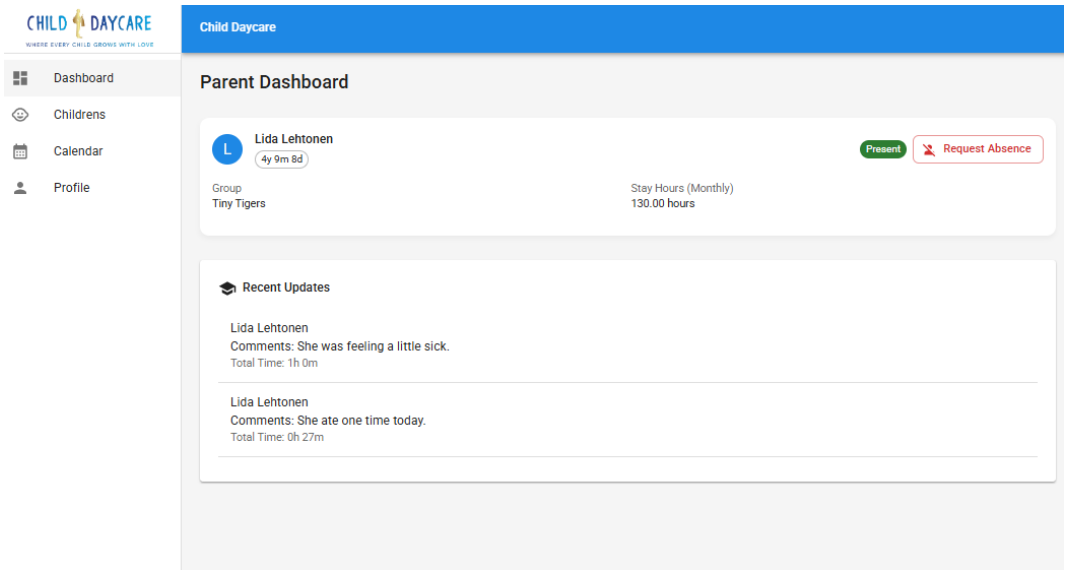

Figure 42. Recent updates page

The Parent Dashboard gives a quick overview of the children's status, group, and monthly hours, plus a recent update feed and a button to request an absence.

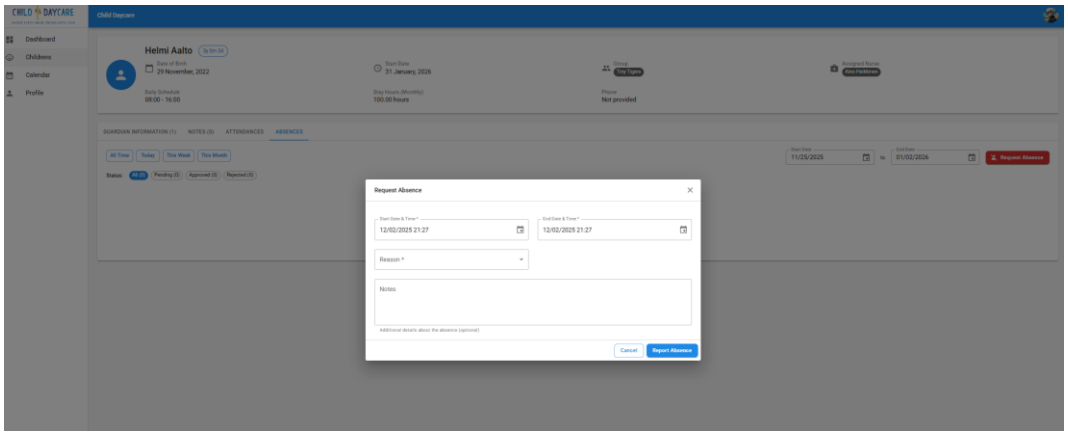

Figure 43. Absence request form (parent UI)

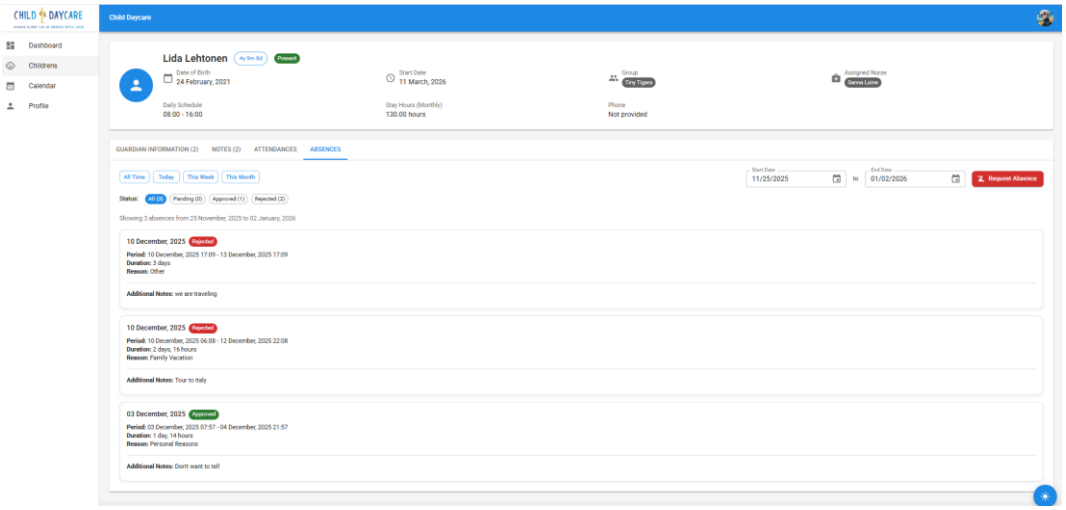

Figure 44. Children's absence request

The Parent Children Absences view lets families view their children’s full absence history, including status (approved/rejected), reasons, durations, date filters, and a button to request new absences.

6 TESTING

The Children Daycare application was tested in a local development environment where the PHP REST API, MySQL database, and React + Vite frontend all ran on localhost. I first focused on the backend and used tools like Postman and the browser network panel to call each API endpoint (auth, users, groups, children, daily notes, absences, attendance). For every route I tried, both valid and invalid payloads: missing required fields, wrong IDs, invalid dates or times, and incorrect status values. I also repeated the same calls with different roles (manager, Nurse, parent) to confirm that the role checks work correctly, for example, only managers can create or delete users and groups. In contrast, parents can only access data for their own children.

On the frontend, I used scenario-based manual testing instead of just clicking individual buttons. As a manager, I logged in, created groups, nurses, and children, assigned them to each other, and checked that the dashboards and detail views (group details, nurse details, children's details) reflected the changes immediately. As a caretaker, I opened the group and attendance views, checked in and checked out children, created daily notes and absences, and then verified that the attendance calendar and children's profiles showed the correct history.

During these flows, I deliberately tested edge cases: leaving form fields empty, entering obviously wrong values (e.g., end date before start date), double-clicking buttons, refreshing during loading, and navigating back and forth between pages. I checked that validation messages appear, that success and error notifications are clear, and that the numbers shown in summary cards (total groups, assigned children, stay hours, attendance counts) matched the underlying database records. Most issues found were small bugs—such as missing validation, incorrect role-based button visibility, or minor layout issues—which were fixed and re-tested until all core workflows ran reliably for each role.

7 SUMMARY

This thesis presents the design and implementation of a Children's Daycare web application to support the daily work of a Finnish kindergarten. The aim was to replace scattered paper-based routines with a single digital system for managing children, staff, attendance, and parent communication.

The application includes three roles: admin or manager, caregiver or nurse, and parent. Managers maintain core data (users, children, groups, timetables) and oversee attendance and absences. Caregivers record daily attendance and write short child notes. Parents access simplified portals to view profiles, attendance history, absence decisions, and daily notes.

The system is built with a PHP REST API, MySQL database, and a React + TypeScript single-page front end. Security is ensured through password hashing and role-based authorization. Functional and role-based testing confirmed reliable workflows and correct access control. The project delivers a working prototype and provides a foundation for future features such as notifications and more detailed reports.

8 CONCLUSIONS AND FUTURE WORK

The main objective of this thesis was to design and implement a lightweight web application that supports attendance tracking, daily documentation, and communication between staff and parents in small Finnish kindergartens, while ensuring secure handling of sensitive child data. This objective was achieved through the successful implementation of a role-based system that integrates a PHP REST API, a MySQL database, and a modern React and TypeScript frontend.

From a technical perspective, the project demonstrated that a modular backend architecture with clearly separated controllers, models, and routing can be effectively combined with a single-page frontend consuming JSON-based APIs. Testing showed that authentication, authorization, and core CRUD operations behave as expected, and that typical user flows—such as managers configuring groups, caregivers recording attendance, and parents viewing updates or submitting absence requests can be completed reliably. Some challenges were encountered, particularly in balancing interface simplicity with the needs of three distinct user roles, but iterative improvements resulted in a stable and usable minimum viable product.

Despite the successful implementation, the system also has limitations that point toward future development opportunities. The application has primarily been validated through manual and functional testing and has not yet been deployed or evaluated across multiple kindergarten environments. In addition, the current solution focuses on core workflows and does not yet include richer communication features such as real-time

messaging or in-application chat. Future enhancements could therefore include configurable notifications, more advanced reporting and analytics for managers, mobile or progressive web application support for caregivers, and multilingual configuration options. Performance and scalability could further be improved through database optimization, catching strategies, and background processing for reporting tasks. These developments would transform the current prototype into a more complete and scalable system suitable for broader real-world adoption.

9 REFERENCES

17.2 InnoDB and the ACID Model. (n.d.). Retrieved December 8, 2025, from https://dev.mysql.com/doc/refman/9.5/en/mysql-acid.html.

Auth0. (n.d.). JSON Web Token Introduction - jwt.io. Https://Www.Jwt.Io/Introduction#what-Is-Json-Web-Token.

Describing the UI – React. (n.d.). Retrieved December 8, 2025, from https://react.dev/learn/describing-the-ui.

Diffie, W., Diffie, W., & Hellman, M. E. (1976). New Directions in Cryptography. IEEE Transactions on Information Theory. https://doi.org/10.1109/TIT.1976.1055638

Finnish National Agency for Education (EDUFI/OPH) — National Core Curriculum for ECEC 2022 (PDF). (n.d.). In https://www.oph.fi/sites/default/files/documents/National%20core%20curriculum%20for%20ECEC%202022.pdf.

Handbook - The TypeScript Handbook. (n.d.). Retrieved December 8, 2025, from https://www.typescriptlang.org/docs/handbook/intro.html.

ISO/IEC 9075-1:2008. (n.d.). Retrieved December 8, 2025, from https://webstore.ansi.org/standards/incits/INCITSISOIEC90752008.

Legislation - Opetus- ja kulttuuriministeriö. (n.d.). Retrieved December 9,2025, from https://okm.fi/yleissivistava-koulutus-lainsaadanto.

MySQL 8.0 Reference Manual. (n.d.). Retrieved December 9, 2025, from https://dev.mysql.com/doc/refman/8.0/en/?source=post_page----------.

MySQL Community Edition. (n.d.). Retrieved December 9, 2025, from https://www.mysql.com/products/community/.

MySQL HeatWave. (n.d.). Https://Www.Mysql.Com/.

PHP. (n.d.). Https://Www.Php.Net/.

PHP: FastCGI Process Manager (FPM) - Manual. (n.d.). Retrieved December 10, 2025, from https://www.php.net/manual/en/install.fpm.php.

PHP: PHP Manual - Manual. (n.d.). Retrieved December 10, 2025, from https://www.php.net/manual/en/index.php.

Send API requests and get response data in Postman | Postman Docs. (n.d.). Retrieved December 10, 2025, from https://learning.postman.com/docs/tests-and-scripts/write-scripts/postman-sandbox-reference/overview/.