University of Pittsburgh

School of Computing and Information

Multimedia Software Engineering

Recreational Facility Daycare Management System
Final Report

Date: April 26, 2023

Jacob Hoffman

Advisors:

Shi-Kuo Chang, PhD

1. Introduction

a. Purpose

The Together Old and Young (TOY) Project seeks to facilitate learning and exchange between children and older adults through sharing experiences. Intergenerational learning aims to educate both sides of the exchange in a mutually beneficial way. The Recreational Facility Daycare Management System (DMS) offers a digital platform to any recreational facility offering activities and spaces that empower intergenerational learning. The system provides functionality for managing accepted kids and mentors (approved by the facility staff, i.e. administrators) with a system that is agnostic to the supplemented facility.

b. Scope

The Recreational Facility DMS will provide the following functionalities:

1. The system will allow an Admin User to manage Mentors and Kids, which includes approval of Mentor/Kid applications and banning/unbanning an existing Mentor/Kid.
2. The system will allow a Regular User to submit an Application for registration of their Kid. The User will be able to insert meaningful data with their Application, which is stored in the system database. The User may submit more than one Application for their Kid.
3. The system will allow a Regular User to submit an Application for registration to become a Mentor. The User will be able to insert meaningful data with their Application, which is stored in the system database. The User can only submit one Application to become a Mentor.
4. The system will allow a Regular User (who has already successfully registered their Kid) to submit an unscheduled Appointment for their Kid to attend.
5. The system will allow a Regular User (who has already successfully registered as a Mentor) to Submit a scheduled Appointment by accepting an unscheduled Appointment to attend.
6. The system will allow an Admin User to check-in the Kid and Mentor for a specified Appointment, which will cause the Appointment to become “active”.
7. The system will allow an Admin User to check-out the Kid and Mentor for a specified Appointment, which will cause the Appointment to conclude (become “inactive”).

c. List of Terms/Acronyms

TOY - Together Old and Young

DMS - Recreational Facility Daycare Management System

User - Someone that is interacting with the system (particularly from the UI).

Regular User - A User who does not have administrative permissions.

Regular User With Kid - A User who has successfully registered their Kid in the system.

Regular User As Mentor - A User who has successfully registered as a Mentor in the system.

Admin User - A User who has administrative permissions.

Application - A "document" submitted by a Regular User for registration of their child, or for registering to become a Mentor. The Application is approved or rejected by an Admin User.

Kid - A Regular User's registered child in the system as a Kid who will attend Appointments.

- A Kid initializes the Appointment in an unscheduled state. The Appointment needs to be accepted by a Mentor.

Mentor - A Regular User registered in the system as a Mentor who will attend Appointments.

- A Mentor transitions an unscheduled Appointment into a scheduled Appointment.

Appointment - A scheduled or unscheduled event associated with a date, time, and location (implicitly the facility location). A scheduled Appointment enables an Admin User to check-in and check-out the Appointment's associated Kid and Mentor.

2. Functional Description

a. Deployment

The system will consist of a Next.js application server and a PostgreSQL server. The Next.js application will provide an implementation of a React-based UI and simple back-end (Node.js Express server) for the DMS to be built on. The system servers may be containerized and launched locally with Docker, and in the future this container stack may be deployed using a system like Kubernetes for handling container orchestration and automating the deployment, scaling, and management of the container stack. Then, this system could be hosted with any popular SaaS platform (e.g. AWS, Azure, GCP). The deployment diagram displayed in *Figure 1* details the system architecture, and specifically the various components the system is composed of.

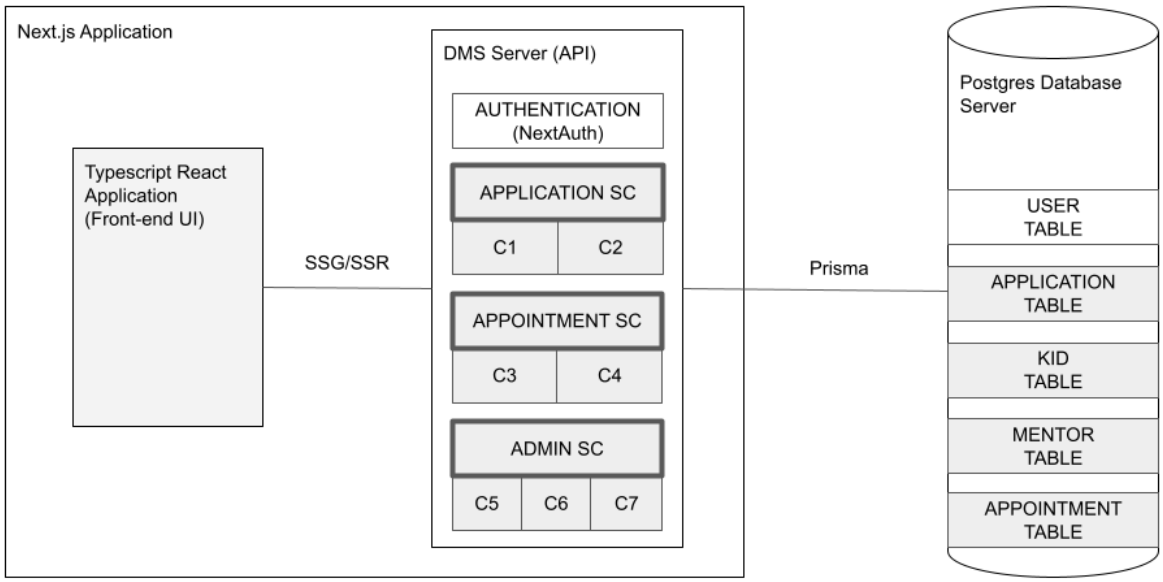


Figure 1: Deployment diagram for the DMS.

b. Components

User Authentication - Handles the user login and authentication of their credentials. The User Authentication component will determine whether the User is a Regular User or an Admin User. Currently, the component is planned to be simulated or primarily handled using NextAuth.js.

Submit Kid Application - Manages collecting and posting the required Application data associated with a Regular User's Kid.

Submit Mentor Application - Manages collecting and posting the required Application data associated with a Regular User that applies to be a Mentor.

Submit Unscheduled Appointment For Kid - Manages the collecting and posting of the required Appointment data for a Regular User that has successfully registered their Kid in the system.

Change Unscheduled Appointment To Scheduled Appointment - Handles the transition of an Appointment from unscheduled to scheduled whenever an Appointment is selected to attend by a Regular User that has successfully registered as a Mentor in the system.

Submit Kid - Handles the approval and posting of a Kid into the system by an Admin User.

Submit Mentor - Handles the approval and posting of a Mentor into the system by an Admin User.

Check In Kid/Mentor - Changes the state of a Scheduled Appointment to active whenever prompted by an Admin User.
Check Out Kid/Mentor - Changes the state of a Scheduled Appointment to completed whenever prompted by an Admin User.
Application Supercomponent - Acts as a supercomponent for handling various types of applications that can be submitted into the system. The possible enumeration of Application submissions are handled by the supercomponent. For example, the supercomponent can handle the correct case of allowing an Application to be submitted by a Regular User who has not successfully registered as a Mentor in the system yet, and also the supercomponent can properly handle the incorrect case of a Regular User who has attempted to submit another Application (even though they have already successfully registered as a Mentor in the system).
Appointment Supercomponent - Acts as a supercomponent for handling the possible interactions and consequential state changes that take place with existing Appointments in the system. The possible enumeration of system states that could occur for Appointments are handled by the supercomponent. For example, the supercomponent can handle the correct case of a successfully registered Mentor accepting an Appointment, and also the supercomponent could properly handle the incorrect case of a Regular User who has not successfully registered as a Mentor attempting to accept an Appointment.
Admin Supercomponent - Acts as a supercomponent for handling the various logical functionalities that are provided to the Admin User.

The components that handle the DMS logic are represented in the petri-net diagram displayed in *Figure 2*, which primarily showcases the control functionalities of the DMS. Additionally, the petri-diagram can be used to understand the various components involved in the system design. In *Figure 3*, the mapping of these components from the petri-diagram is displayed in the form of an I-card and C-card. Through further analysis, the defined components may be used to realize supercomponents. The supercomponents that are defined in the DMS are highlighted in *Figure 4* and *Figure 5*.

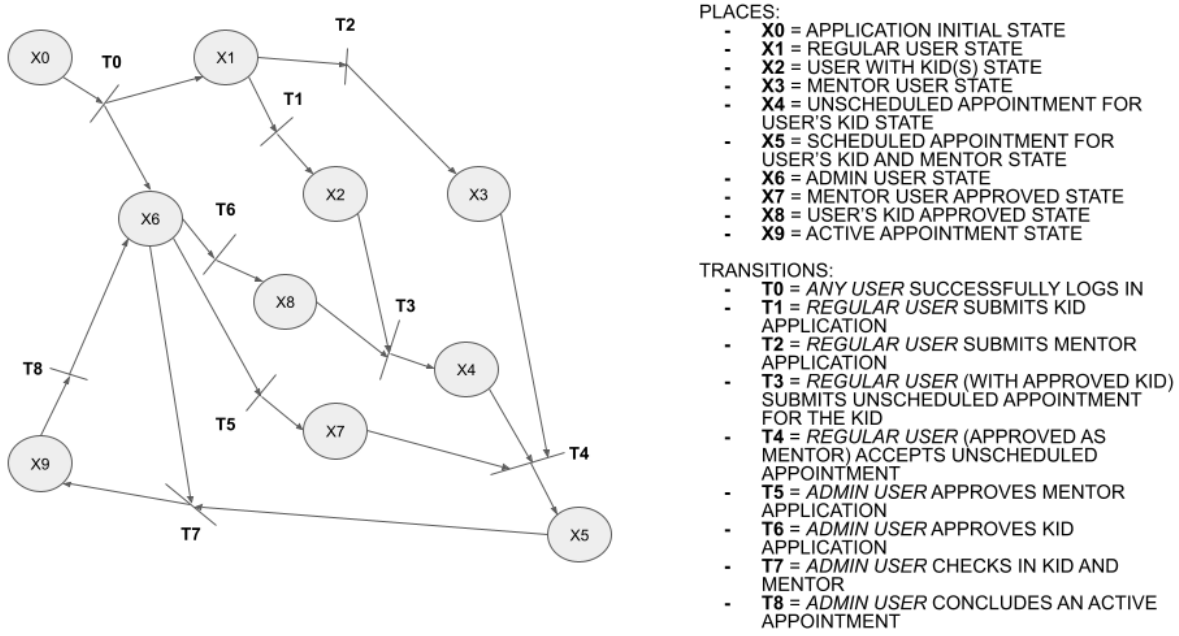


Figure 2: A petri-net diagram representing the logical system and control functionalities of the DMS.

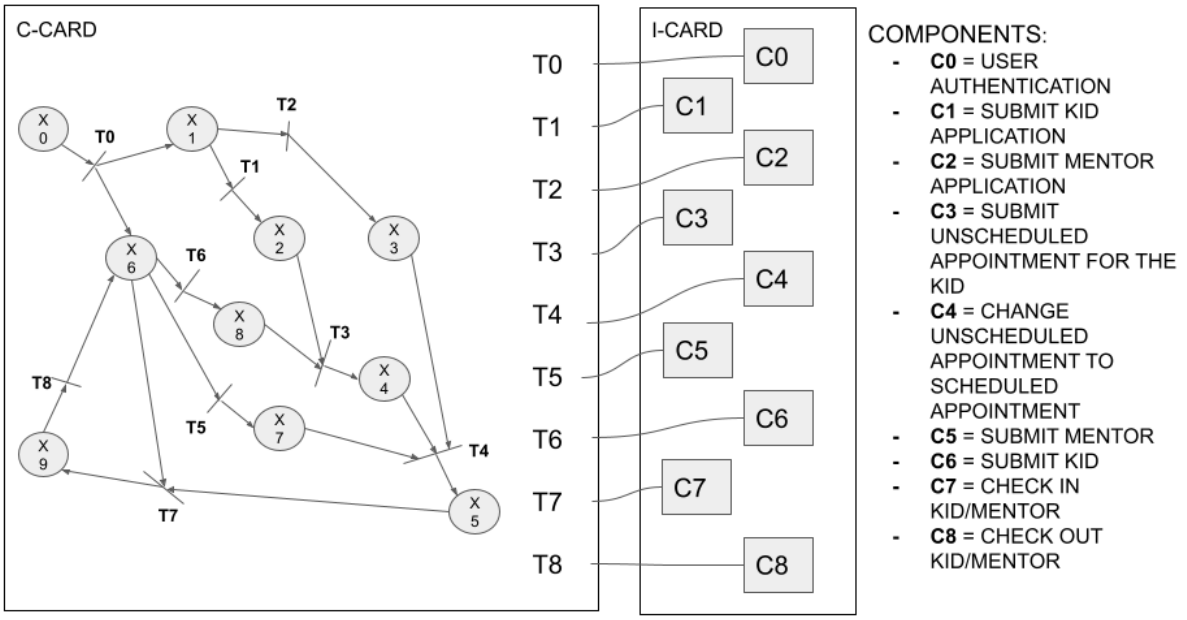


Figure 3: The translation of transitions in the petri-diagram representation of the system into specified components via C-card and I-card.

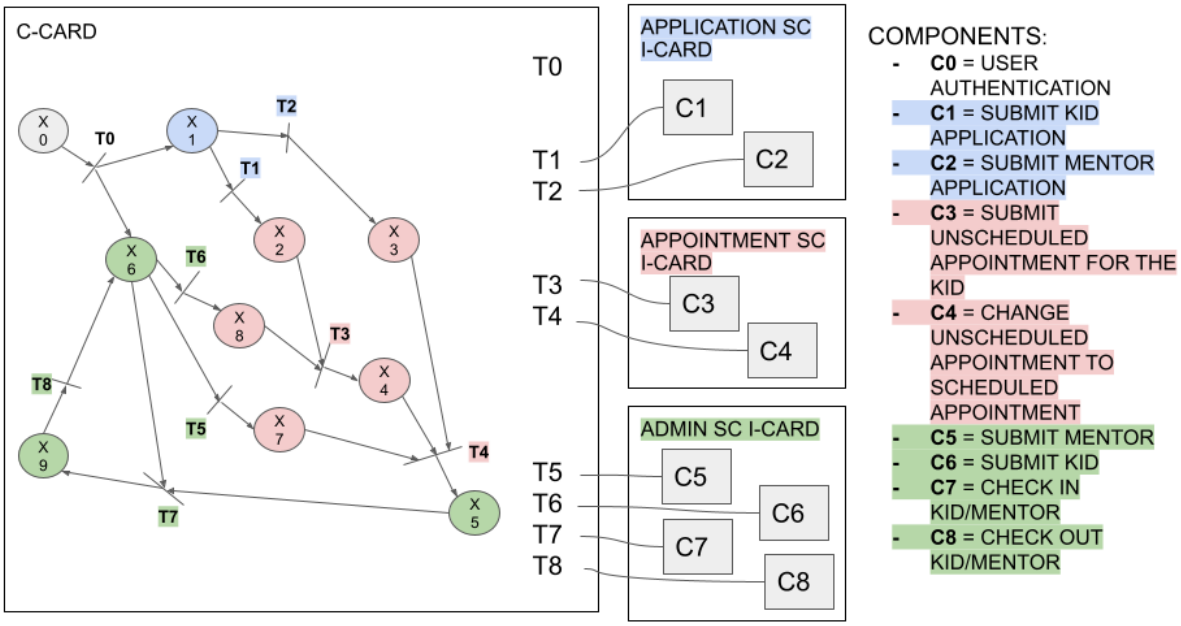


Figure 4: Highlighting the defined supercomponents based on the specified components in the DMS.

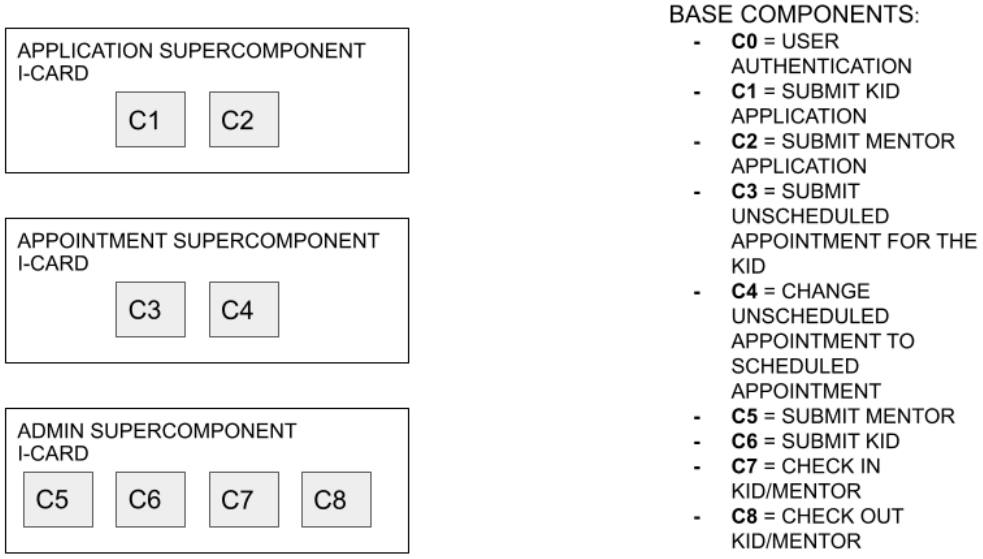


Figure 5: The isolated supercomponents in the DMS represented as I-cards.

c. Data Model

The system will enforce a data model for the database server. This will organize the system's collected data in a meaningful way. To perform queries, the Next.js application will utilize Prisma ORM, which will interface with the selected Postgresql database and handle any necessary queries for the DMS Server API. *Figure 6* presents an entity-relationship diagram, which serves as a visual representation of the data model that will be enforced for the database server. Finally, the textual schemas for this data model are displayed in *Figure 7*.

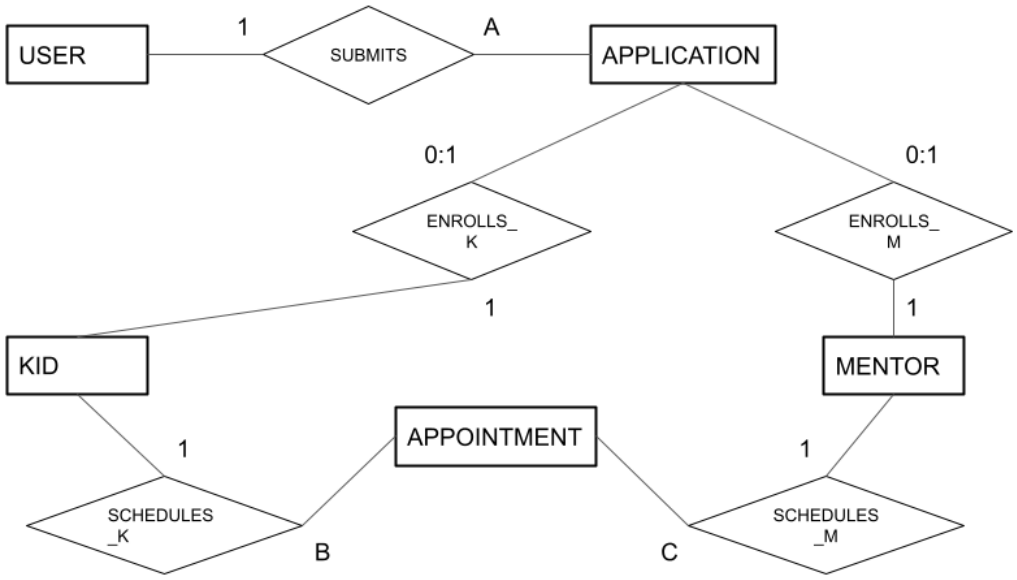


Figure 6: An entity-relationship diagram that represents the data model to be used in the DMS.

- **ENTITIES:**
  - **USER**: id, name, *email*
  - **APPLICATION**: app_id, user_id, application_type, applicant_name, applicant_age, applicant_photo, applicant_file, isApproved
  - **KID**: app_id (kid_id)
  - **MENTOR**: app_id (mentor_id)
  - **APPOINTMENT**: appt_id, app_id, schedule_timestamp, start_timestamp, end_timestamp, isActive

- **RELATIONSHIPS:**
  - **SUBMITS<USER, APPLICATION> 1:A, PARTIAL/PARTIAL,**
  - **ENROLLS_K<APPLICATION, KID> 0/1:1, PARTIAL/PARTIAL,**
  - **ENROLLS_M<APPLICATION, MENTOR> 0/1:1, PARTIAL/PARTIAL,**
  - **SCHEDULES_K<KID, APPOINTMENT> 1:B, PARTIAL/PARTIAL,**
  - **SCHEDULES_M<MENTOR, APPOINTMENT> 1:C, PARTIAL/PARTIAL,**

Figure 7: A textual diagram displaying the data model's schemas that will be used in the DMS.

d. Results

Database Implementation:

a. The relational database was implemented as described in section 2. For simplicity, the current prototype application is locally hosted using Node, and it connects remotely to a Postgresql database hosted using Supabase. The data model was implemented using Prisma client and Next.js.
b. See Appendix f.1 for an illustration of the Prisma models declared in the source code.

API Implementation:

a. As described in section 2, an API was implemented using Next.js API routes (an extension on Express.js). Static Site Generation (SSG) is used to call on the back-end only at build time of the web application. Then, the data collected is used (along with the React Javascript) at build-time to statically generate the HTML and CSS required for the application. Then, the application is loaded significantly faster than if it were to hydrate on the client-side once the Javascript has been rendered. On the client-side, any time there is an API hit (POST requests) the client will properly handle this request to the application's back-end, and the application will incrementally regenerate the corresponding static HTML/CSS files. Please refer to the Next.js for more information about SSG.
b. See Appendix f.2 for an illustration of the Next API Routes.

Scenarios:

1. Regular User Login
    a. Click "Sign In Using Email Address" and then follow the sign-in process (this is currently mocked using Email Magic Links with Next-Auth).
    b. After successfully logging in, the Regular User has been redirected to the Regular User Dashboard.
    c. See Appendix f.3 for an illustrated scenario walkthrough.
    d. Handled by Auth component.

2. Admin User Login
    a. Click “Sign In Using Email Address” and then follow the sign-in process (this is currently mocked using Email Magic Links with Next-Auth, and there is an environment variable in the app that specifies a single email address used for the admin account).
    b. After successfully logging in, the Admin User has been redirected to the Admin User dashboard.
    c. See Appendix f.4 for an illustrated scenario walkthrough.
    d. Handled by Auth component.
3. Regular User Submits Kid Application(s)
    a. As a Regular User viewing the Regular user dashboard, click the “Register Kid” button.
    b. Once the “Insert Application Information” popup has been displayed, insert the required information.
    c. Once the popup has closed, the Regular User will be able to view their list of Applications on the Regular User dashboard.
    d. See Appendix f.5 for an illustrated scenario walkthrough.
    e. Handled by Application supercomponent.
4. Regular User Submits Mentor Application
    a. As a Regular User viewing the Regular User dashboard, click the “Register Mentor” button.
    b. Once the “Insert Application Information” popup has been displayed, insert the required information. Then, click the “Submit” button.
    c. Once the popup has closed, the Regular User will be able to view their list of Applications on the Regular User dashboard. Notice that the “Register Mentor” button has become hidden because the Regular User may only submit one Mentor application.
    d. See Appendix f.6 for an illustrated scenario walkthrough.
    e. Handled by Application supercomponent.

5. Admin User Toggles Application Status Between Approved/Denied (Insert/Remove Mentor Or Kid From System)
    a. Once a Regular User has submitted any type of Application, an Admin User will be presented with all of the available applications in the Admin User dashboard. To accept a specific Application, the Admin User may click the “Accept App” button.
    b. After the Admin user clicks the “Accept App” popup button, the specific Appointment will change state and color (from red=denied to green=approved), which indicates a successful update to approved. Then, the Admin User may click the “Deny App” button to toggle back.
    c. See Appendix f.7 for an illustrated scenario walkthrough.
    d. Handled by Admin supercomponent.
6. Regular User With Kid Submits Unscheduled Appointment
    a. After an Admin User has approved a Regular User’s Kid Application, the Regular User will be presented with their approved Application state, and a “Request Appointment” button for the Kid that was approved. The Regular user may click this button to begin submitting an Unscheduled Appointment.
    b. Once the “Insert Appointment Information” popup has been displayed, insert the required information. Then, click the “Submit” button.
    c. Once the popup has closed, the Regular User will be able to view their list of Unscheduled Appointments on the Regular User dashboard.
    d. See Appendix f.8 for an illustrated scenario walkthrough.
    e. Handled by Appointment supercomponent.
7. Regular user As Mentor Accepts Unscheduled Appointment (Update Appointment Status To Scheduled)
    a. After a Regular User with Kid submits an Unscheduled Appointment, a different Regular User will be presented a list of all Unscheduled Appointments (barring any for their own Kid(s) in the system). The Regular User may click the “Accept” button for any of them.

    b. Once the “Accept” button has been clicked, the Unscheduled Appointment will be moved to the Scheduled Appointment(s) list and presented on the Regular user dashboard for all of the relevant Regular Users.
    c. See Appendix f.9 for an illustrated scenario walkthrough.
    d. Handled by Appointment supercomponent.
8. Admin User Starts Scheduled Appointment (Update Appointment Status To Active)
    a. Once there are any available Scheduled Appointments in the system, a list of all Scheduled Appointments is displayed to an Admin User. The Admin User may select the “Start” button for a specific Appointment to begin.
    b. Once the “Start” button has been clicked, the Scheduled Appointment state will be updated and color changed (to yellow), which will be present for all of the relevant Users.
    c. See Appendix f.10 for an illustrated scenario walkthrough.
    d. Handled by Admin supercomponent.
9. Admin User Concludes Scheduled Appointment (Update Appointment Status to Inactive)
    a. Once there are any available Scheduled Appointments that are also active in the system, an Admin User will be able to view them in the list of Scheduled Appointments. The Admin User may select the “end” button for a specific Appointment to conclude.
    b. See Appendix f.11 for an illustrated scenario walkthrough.
    c. Handled by Admin supercomponent.

e. References

[1] CS2310 Multimedia Software Engineering, Shi-Kuo Chang, https://people.cs.pitt.edu/~chang/231/231syl.html

[2] Next-Auth Documentation, Lain Collins, https://next-auth.js.org/getting-started/introduction

[3] Next-Auth Email Provider and Magic Links, https://next-auth.js.org/providers/email

[3] Prisma CRUD Reference, Prisma Data, Inc https://www.prisma.io/docs/concepts/components/prisma-client/crud

[4] Next.js Documentation, Vercel, https://nextjs.org/docs/getting-started

[5] MUI Documentation, Material-UI SAS, https://mui.com/

[6] Emotion Documentation, https://emotion.sh/docs/introduction

[7] Use Supabase With NextJS (Remote Postgresql Server), https://supabase.com/docs/guides/getting-started/quickstarts/nextjs

[8] Project Source Code, Jacob Hoffman, https://github.com/Jacob-Hoff-man/cs2310-dms

f. Appendix

1. Declaration of Prisma Models (Database Schemas)

```
//Next Auth
model Account {
 id                String  @id @default(cuid())
 userId            String
 type              String
 provider          String
 providerAccountId String
 refresh_token     String?
 access_token      String?
 expires_at        Int?
 token_type        String?
 scope             String?
 id_token          String?
 session_state     String?
 user              User    @relation(fields: [userId], references: [id], onDelete: Cascade)

 @@unique([provider, providerAccountId])
}

model Session {
 id           String   @id @default(cuid())
 sessionToken String   @unique
 userId       String
 expires      DateTime
 user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)
}

model User {
 id            String    @id @default(cuid())
 name          String?
 email         String?   @unique
 emailVerified DateTime?
 image         String?
 accounts      Account[]
 sessions      Session[]
 applications         Application[]
 @@map(name: "users")
}

model VerificationToken {
 identifier String
 token      String   @unique
 expires    DateTime
 @@unique([identifier, token])
}

// Daycare Management System Models
model Application {
 id        String     @default(cuid()) @id
 title     String
 content   String?
 published Boolean @default(false)
 user    User?   @relation(fields: [userId], references: [id])
 userId  String?
 appType AppType @default(MENTOR)
 isApproved Boolean @default(false)
 kidName String?
 kid Kid?
 mentor Mentor?
 }
```

```
// Daycare Management System Models continued
enum AppType {
 KID
 MENTOR
}

model Kid {
 id String @default(cuid()) @id
 kidName String
 app Application? @relation(fields: [appId], references: [id], onDelete: Cascade)
 appId String @unique
 appointments Appointment[]
 @@map(name: "kids")
}

model Mentor {
 id String @default(cuid()) @id
 mentorName String
 app Application? @relation(fields: [appId], references: [id], onDelete: Cascade)
 appId String @unique
 appointments Appointment[]
 @@map(name: "mentors")
}

model Appointment {
 id String @default(cuid()) @id
 startTime DateTime @default(now()) @db.Timestamp()
 endTime DateTime? @db.Timestamp()
 isScheduled Boolean @default(false)
 isActive Boolean @default(false)
 kid Kid? @relation(fields: [kidId], references: [id])
 kidId String?
 mentor Mentor? @relation(fields: [mentorId], references: [id])
 mentorId String?
}
```

Source code:

https://github.com/Jacob-Hoff-man/cs2310-dms/blob/master/prisma/schema.prisma

2. Next Api Routes (Express.js)

- api/
    - app/
        - update/
            - *isApproved*
        - *add*
        - *delete*
    - appt/
        - update/
            - *isActive*
            - *mentor*
    - auth/
        - *[...nextauth]* (slug for authentication service to use)
    - kid/
        - *add*
        - *delete*
    - mentor/
        - *add*
        - *delete*

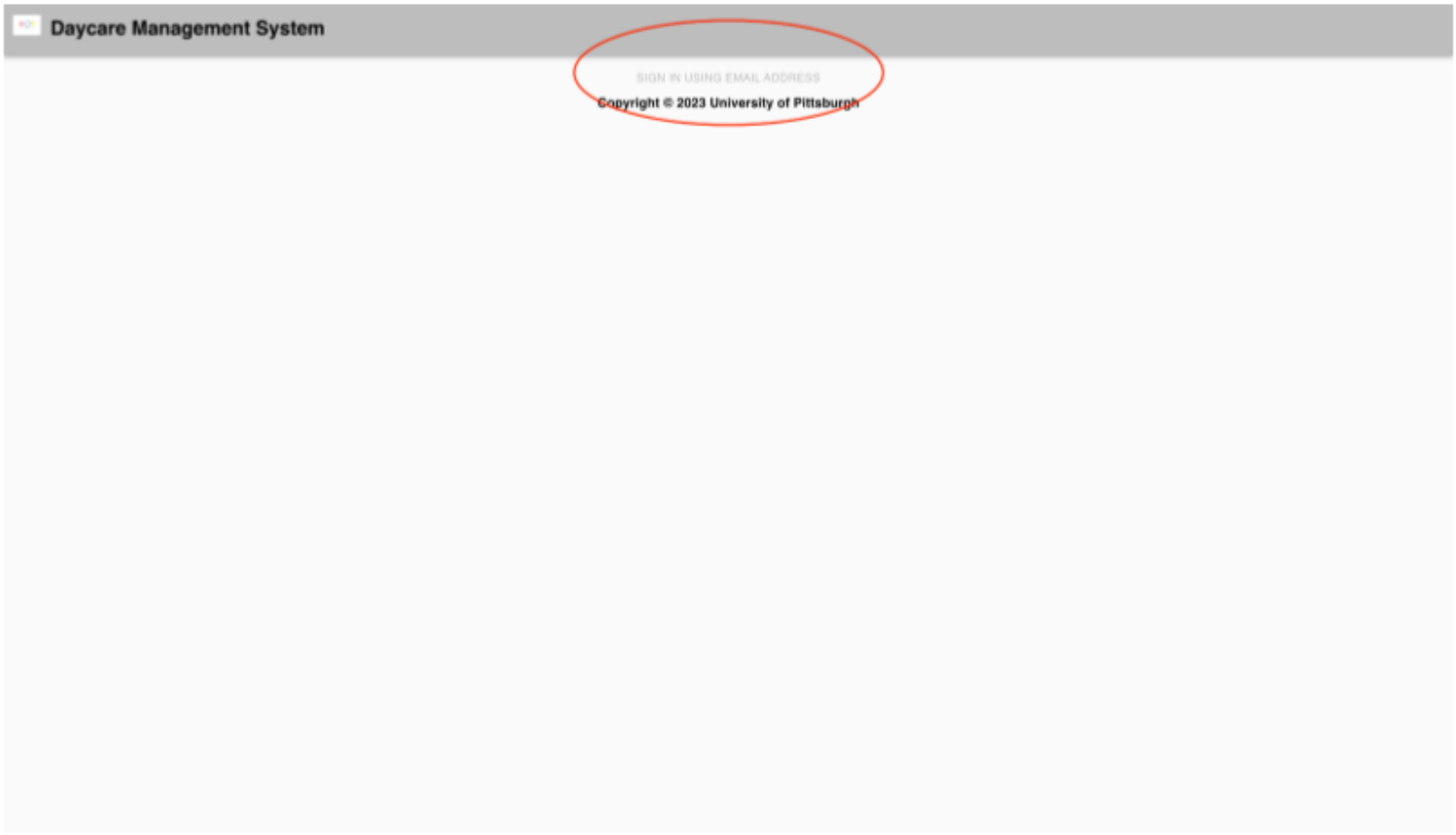
Source code:
https://github.com/Jacob-Hoff-man/cs2310-dms/tree/master/src/pages/api

3. Regular User Login

Prototype Detailed Scenarios

Scenario 1: Regular User Log In

- Click “Sign In Using Email Address” and then follow the sign-in process (this is currently mocked using Email Magic Links with NextAuth)



Prototype Detailed Scenarios

Scenario 1: Regular User Log In

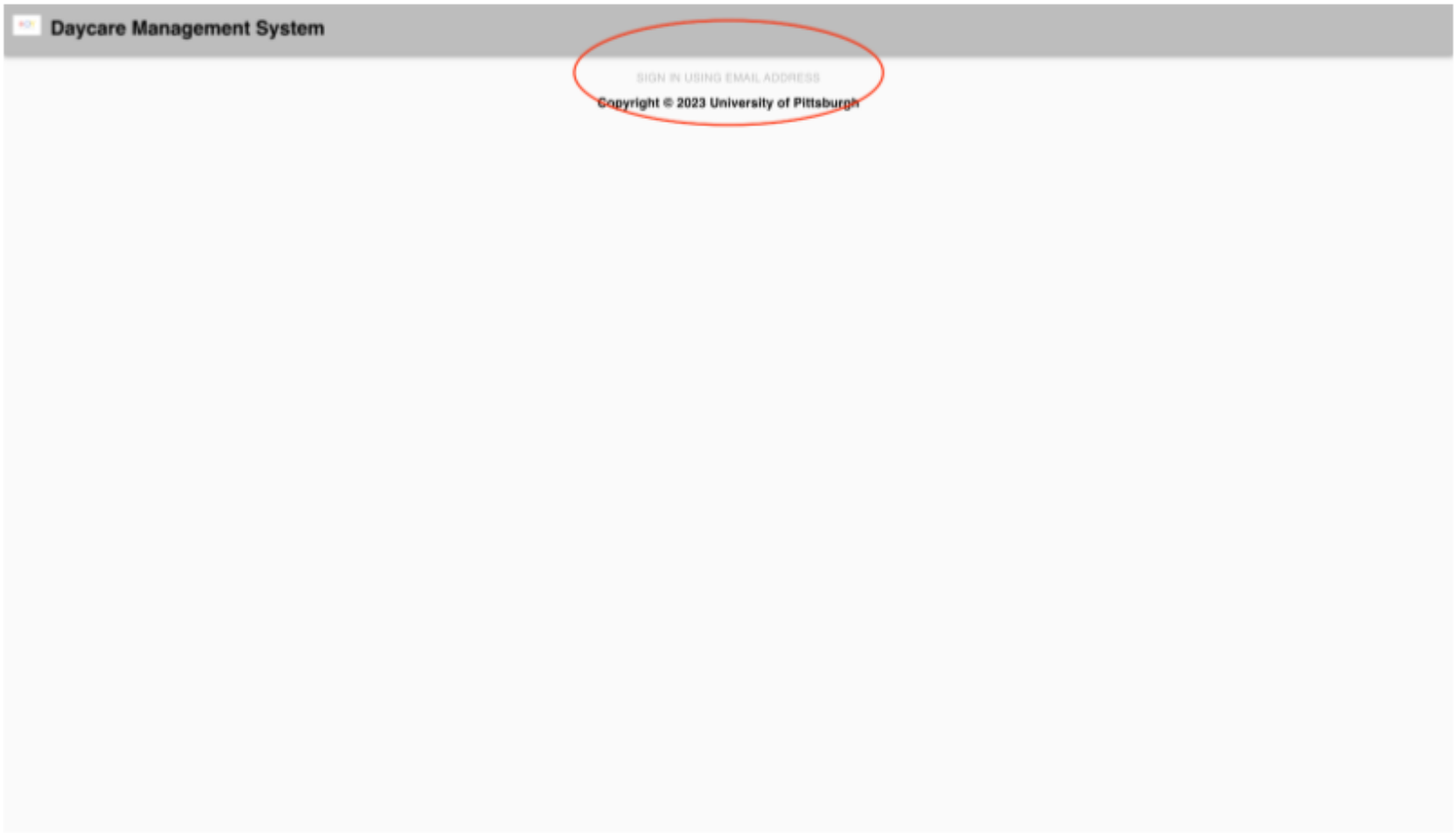
- After successfully logging in, the Regular User has been redirected to the Regular User Dashboard.

4. Admin User Login

Prototype Detailed Scenarios

Scenario 2: Admin User Log In

- Click “Sign In Using Email Address” and then follow the sign-in process (this is currently mocked using Email Magic Links with NextAuth)



Prototype Detailed Scenarios

Scenario 2: Admin User Log In

- After successfully logging in, the Admin User has been redirected to the Admin User dashboard.

5. Regular User Submits Kid Application(s)

Prototype Detailed Scenarios

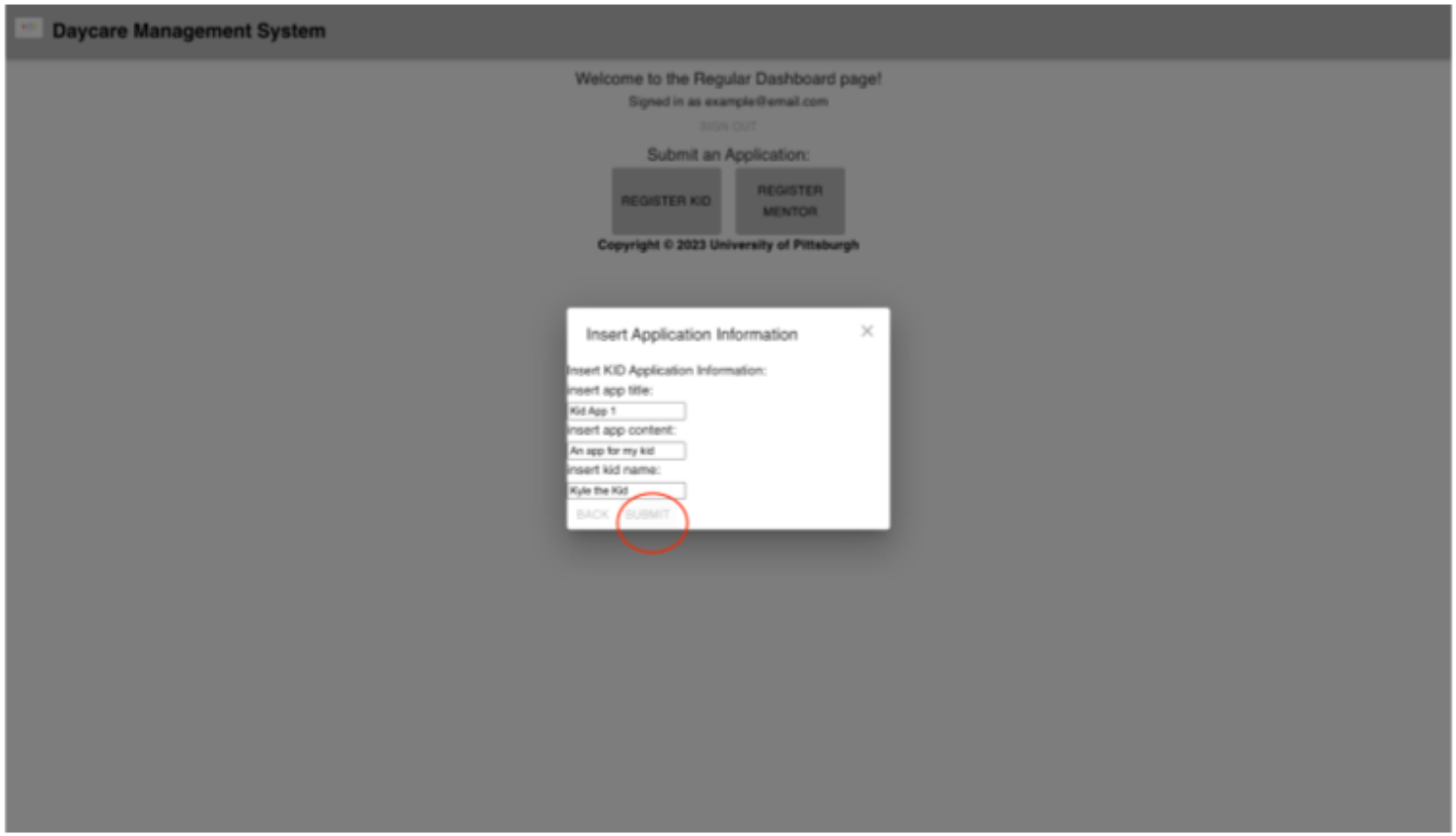
Scenario 3: Regular User Submits Kid Application(s)

- As a Regular User viewing the Regular User dashboard, Click “Register Kid” button.

Prototype Detailed Scenarios

Scenario 3: Regular User Submits Kid Application(s)

- Once the Insert App Info popup has displayed, insert the required information. Then, click the “Submit” button



Prototype Detailed Scenarios

Scenario 3: Regular User Submits Kid Application(s)

- Once the popup has closed, the Regular User will be able to view their list of Applications on the Regular User dashboard.

6. Regular User Submits Mentor Application

Prototype Detailed Scenarios

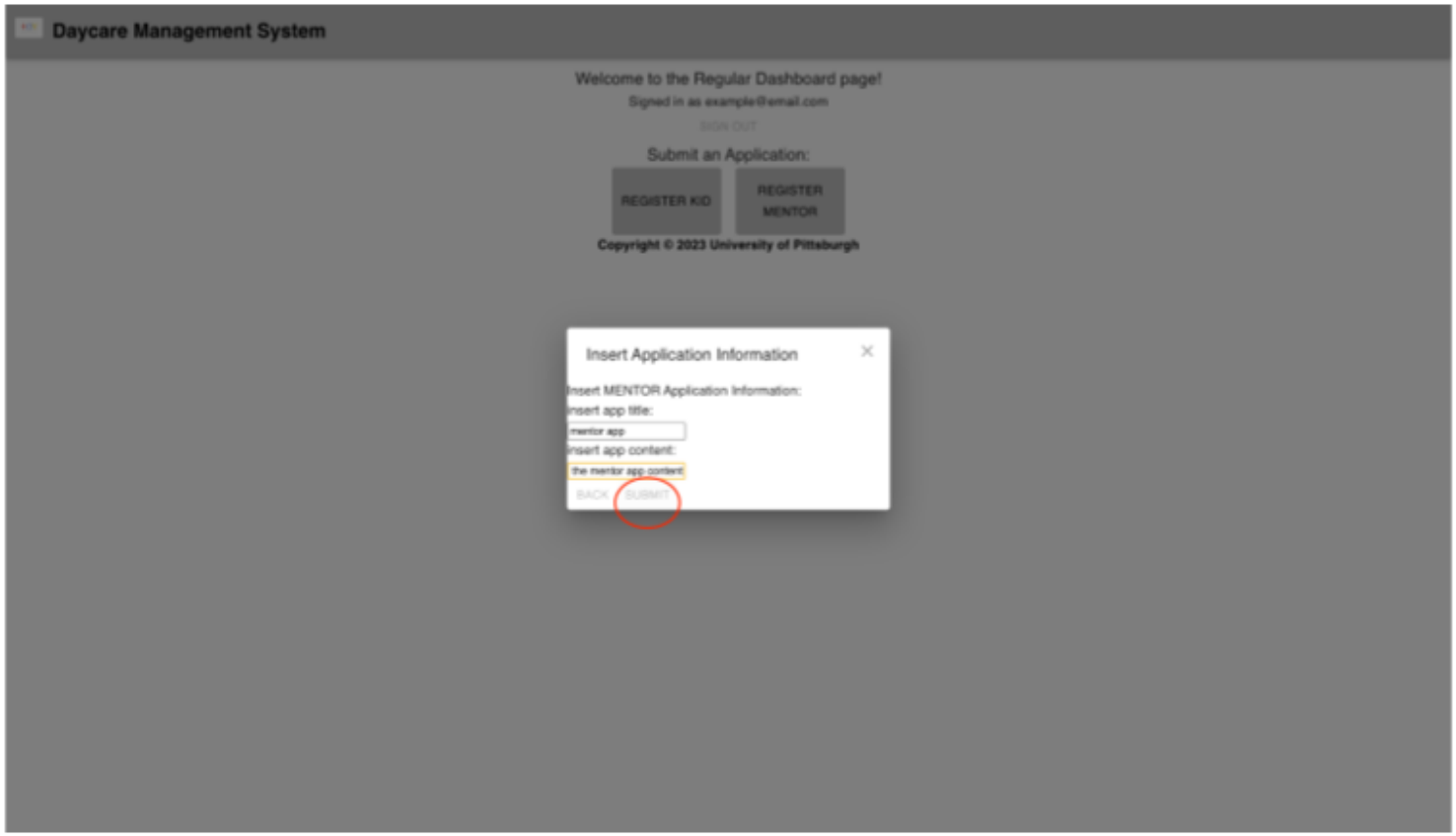
Scenario 4: Regular User Submits Mentor Application

- As a Regular User viewing the Regular User dashboard, Click “Register Mentor” button.

Prototype Detailed Scenarios

Scenario 4: Regular User Submits Mentor Application

- Once the Insert App Info popup has displayed, insert the required information. Then, click the “Submit” button



Prototype Detailed Scenarios

Scenario 4: Regular User Submits Mentor Application

- Once the popup has closed, the Regular User will be able to view their list of Applications on the Regular User dashboard. Notice that the "Register Mentor" button has become hidden because the Regular User may only submit one Mentor Application.

7. Admin User Toggles Application Status Between Approved/Denied

Prototype Detailed Scenarios

Scenario 5: Admin User Toggles Application Status Between Approved/Denied (Insert/Remove Mentor Or Kid From System)

- Once a Regular User has submitted any type of Application, an Admin User will be presented with all of the available Applications in the Admin User Dashboard. To accept a specific Application, the Admin User may click the “Accept App” button.

Prototype Detailed Scenarios

Scenario 5: Admin User Toggles Application Status Between Approved/Denied (Insert/Remove Mentor Or Kid From System)

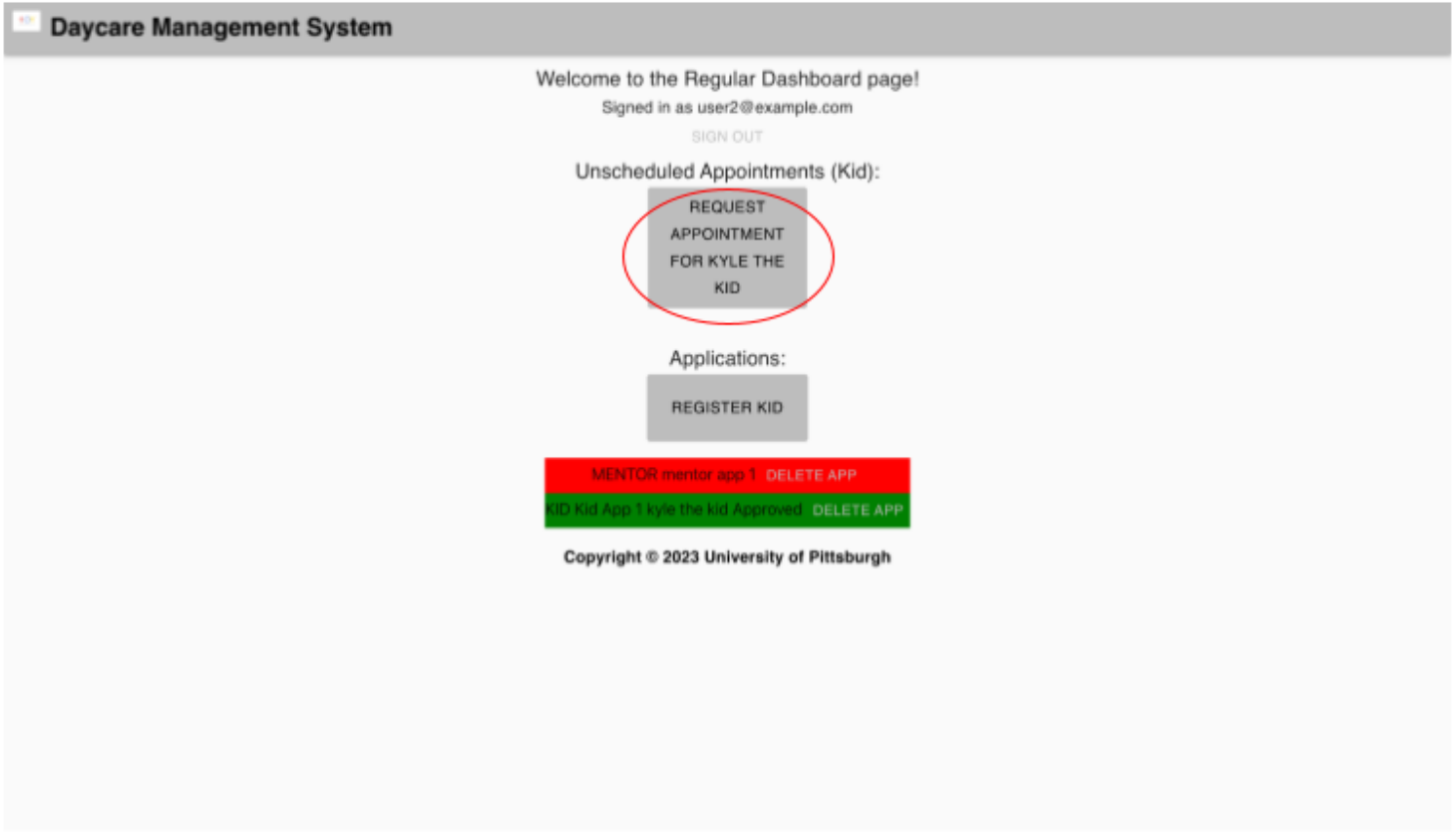
- After the Admin User clicks the “Accept App” button, the specific Appointment will change state and color (from red=denied to green=approved), which indicates a successful update to approved. The Admin User may now click the “Deny App” button to toggle back.

8. Regular User With Kid Submits Unscheduled Appointment

Prototype Detailed Scenarios

Scenario 6: Regular User With Kid Submits Unscheduled Appointment

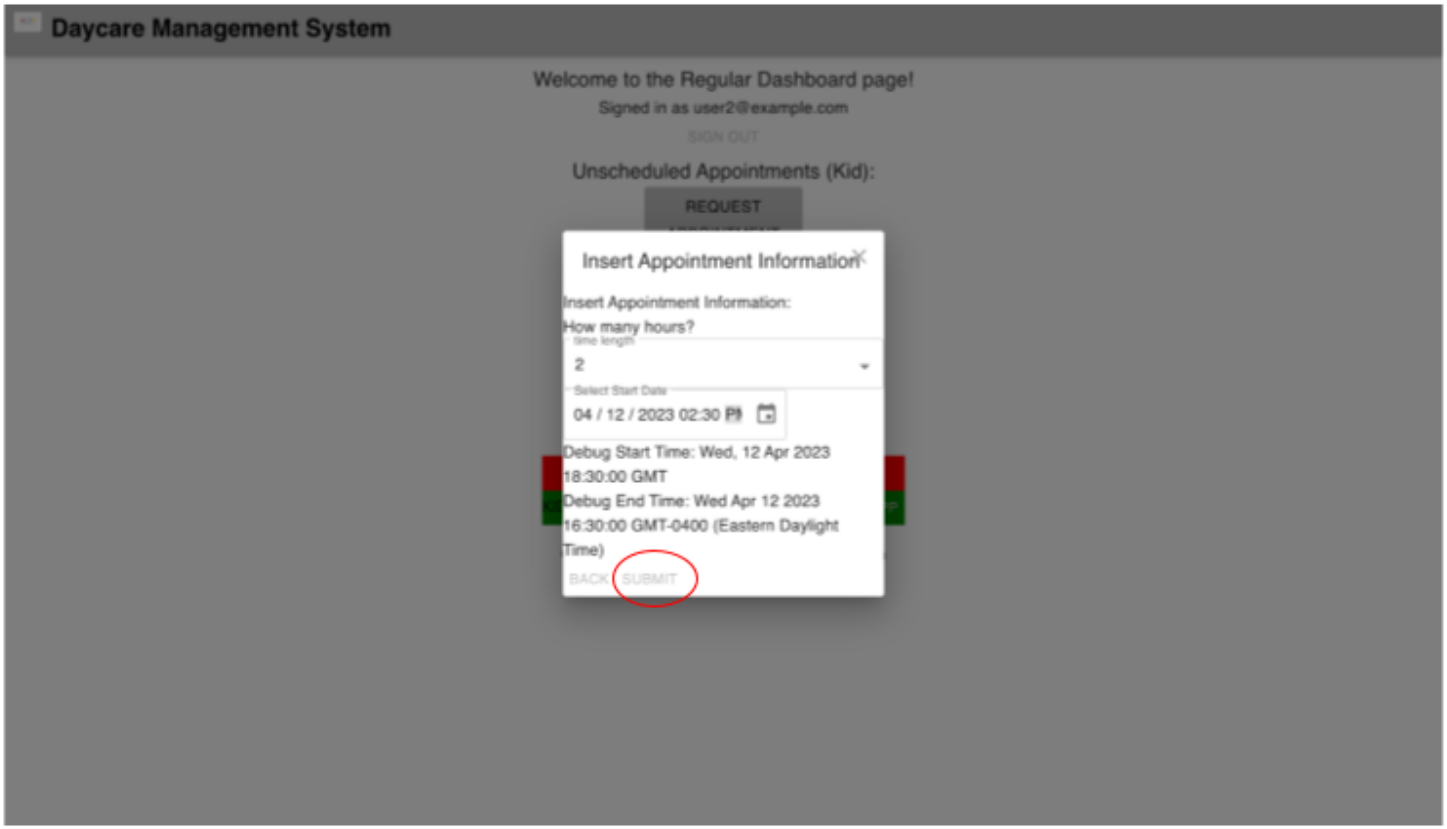
- After an Admin User has approved a Regular User's Kid Application, the Regular User will be presented with their approved Application state, and a "Request Appointment" button for the Kid that was approved. The Regular User will click this button.



Prototype Detailed Scenarios

Scenario 6: Regular User With Kid Submits Unscheduled Appointment

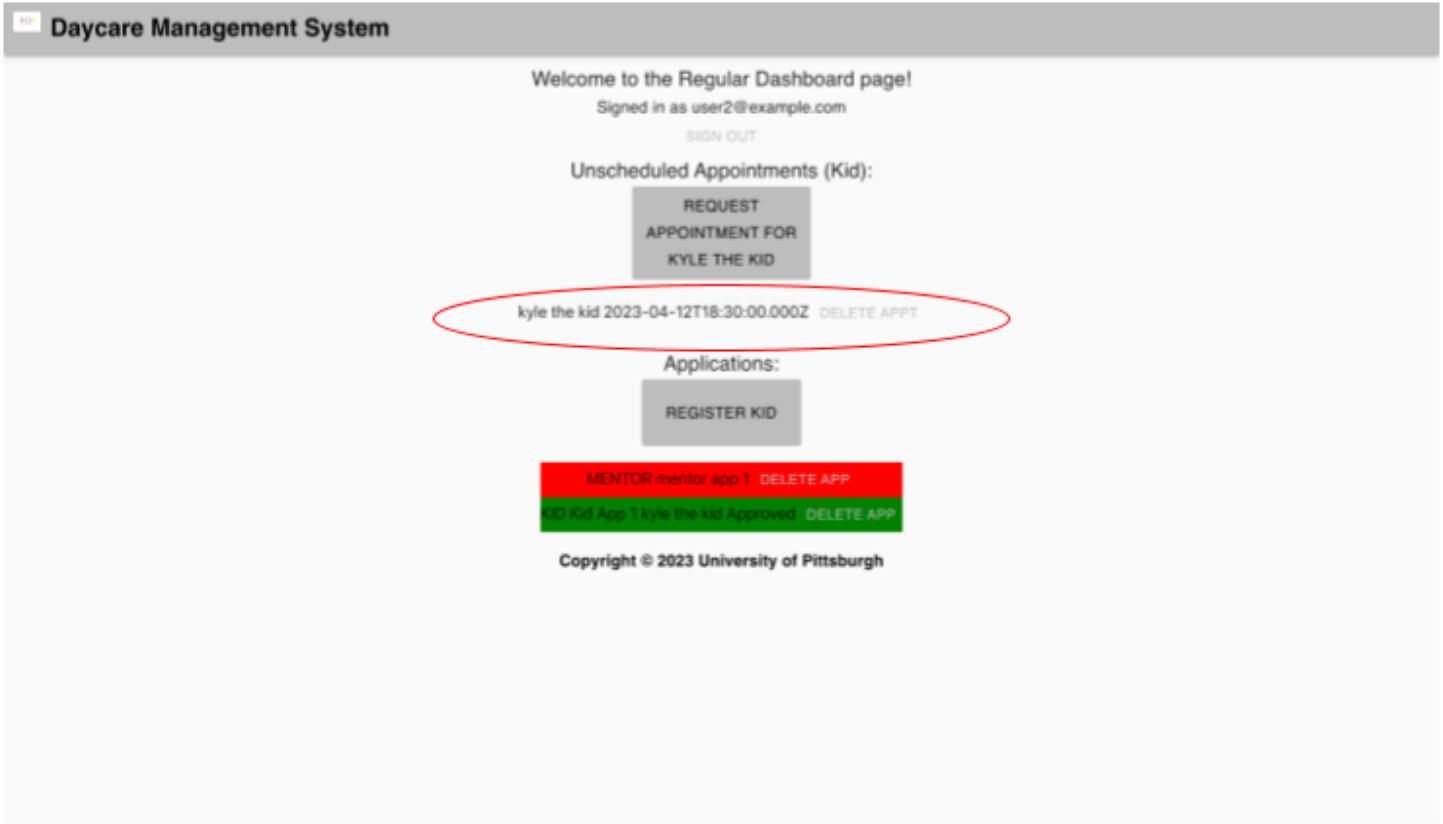
- Once the Insert Appointment Info popup has displayed, insert the required information. Then, click the “Submit” button



Prototype Detailed Scenarios

Scenario 6: Regular User With Kid Submits Unscheduled Appointment

- Once the popup has closed, the Regular User will be able to view their list of Unscheduled Appointments on the Regular User dashboard.

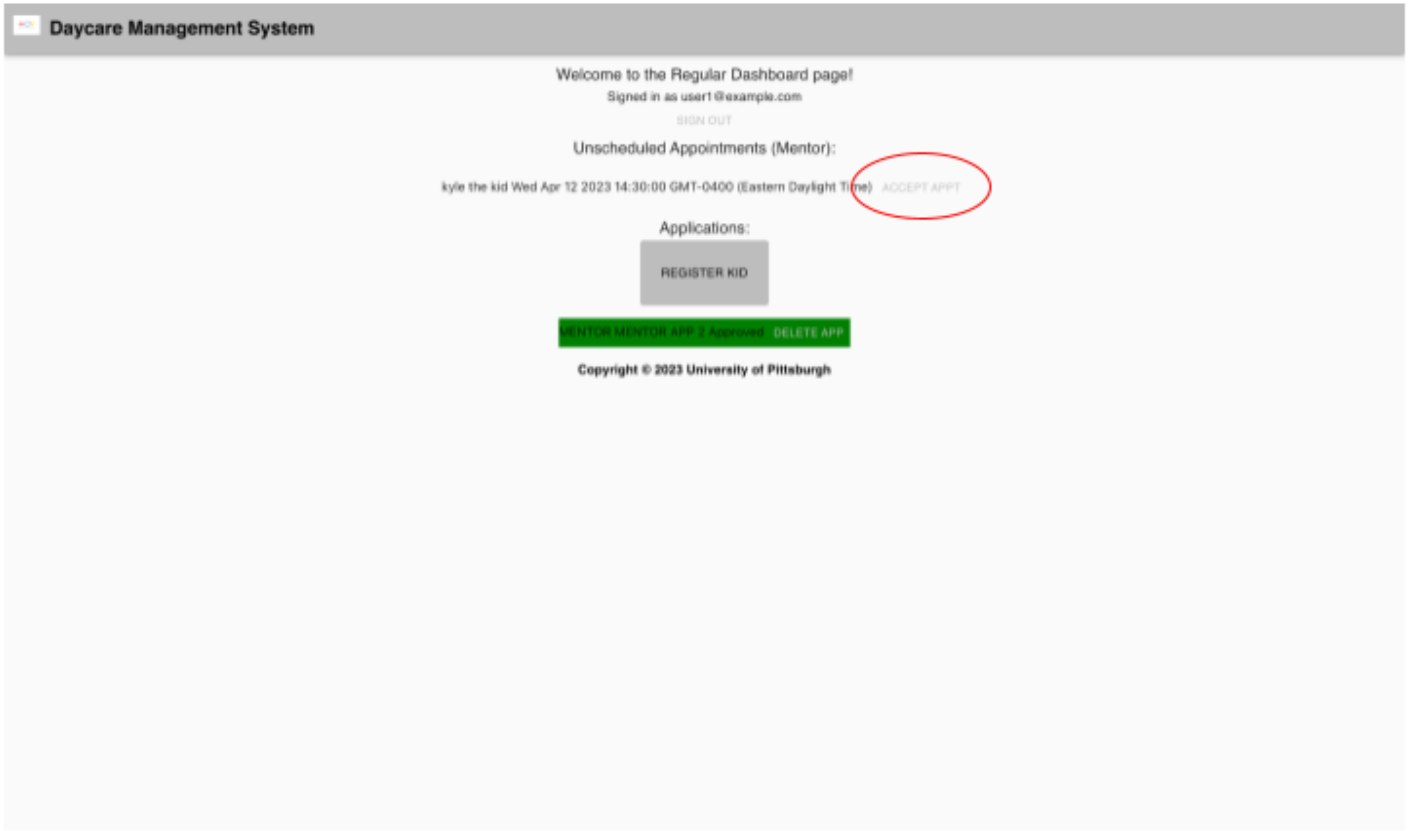


9. Regular User As Mentor Accepts Unscheduled Appointment

Prototype Detailed Scenarios

Scenario 7: Regular User As Mentor Accepts Unscheduled Appointment (Update Appointment Status To Scheduled)

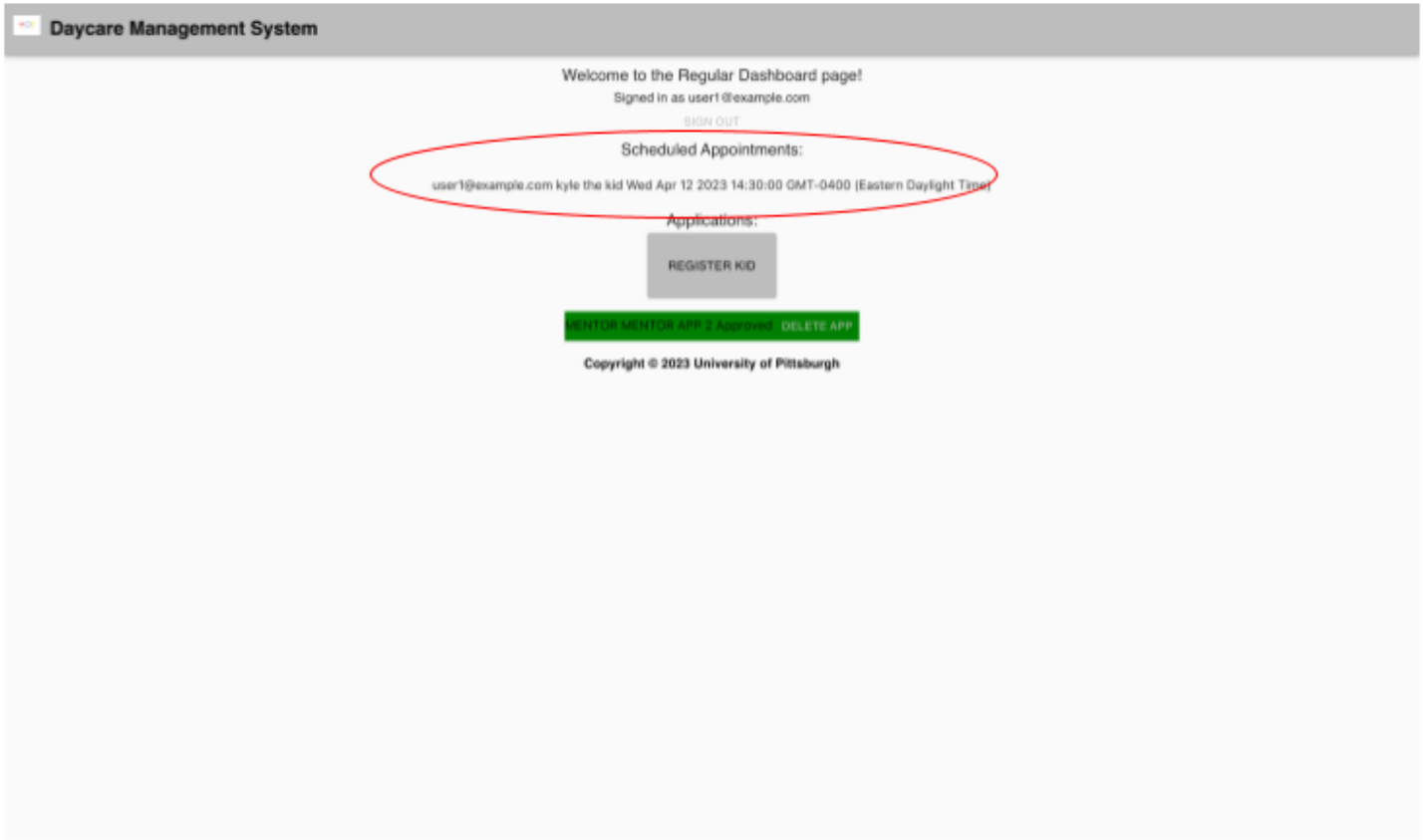
- After a Regular User with Kid submits an Unscheduled Appointment, a different Regular User will be presented a list of all Unscheduled Appointments (barring any for their own Kid(s) in the system). The Regular User may click the “Accept” button for any of them.



Prototype Detailed Scenarios

Scenario 7: Regular User As Mentor Accepts Unscheduled Appointment (Update Appointment Status To Scheduled)

- Once the “Accept” button has been clicked, the Unscheduled Appointment will be moved to the Scheduled Appointment(s) list and presented on the Regular User dashboard for all of the relevant Regular Users.



10. Admin User Starts Scheduled Appointment

Prototype Detailed Scenarios

Scenario 8: Admin User Starts Scheduled Appointment (Update Appointment Status To Active)

- Once there are any available Scheduled Appointments in the system, a list of all Scheduled Appointments is displayed to an Admin User. The Admin User may select the “Start” button for a specific Appointment to begin.

Prototype Detailed Scenarios

Scenario 8: Admin User Starts Scheduled Appointment (Update Appointment Status To Active)

- Once the “Start” button has been clicked, the Scheduled Appointment state will be updated and color changed (to yellow), which will be present for all of the relevant Users.

11. Admin User Concludes Scheduled Appointment

Prototype Detailed Scenarios

Scenario 9: Admin User Concludes Scheduled Appointment (Update Appointment Status To Inactive)

- Once there are any available Scheduled Appointments that are also active in the system, an Admin User will be able to view them in the list of Scheduled Appointments. The Admin User may select the “End” button for a specific Appointment to conclude.

Prototype Detailed Scenarios

Scenario 9: Admin User Concludes Scheduled Appointment (Update Appointment Status To Inactive)

- Once the “End” button has been clicked, the Scheduled Appointment state will be updated and color changed (to transparent), which will be present for all of the relevant Users.