Remix workshop

1. Introduction

- we have React/Express application that we want to rebuild in Remix to show major differences.

2. Setup

Follow Readme to setup the project.

3. Remix overview

- Remix is a full stack web framework
- Focused on web standards and modern web app UX
- Build around React Router (6) by people who developed React Router
- Server rendered with client hydration (optional progressive enhancement)
- Simple "mental model" - components, loaders, actions
- Simple but also powerful error handling
- Can also work completely without JS enabled on the client
- Most important - it's fun to use!

4. Adding events route

Branch 01

Todo

- [ ] add a route "events"
- [ ] add exported component `EventsIndexPage`
- [ ] update `index.tsx` (cleanup and link to `/events` instead of `/notes`)
- [ ] update `login.tsx` to navigate to `/events` instead of `/notes`

5. Adding events/index route

Branch 02

Todo

- [ ] add a route `events/index.tsx`, with component `EventIndexPage`
- [ ] update `events/index.tsx` add `loader` function that returns static data
- [ ] display data in the component using `useLoaderData`
- [ ] update `events.tsx` to render `Outlet`

6. DB setup

Branch 03

Todo

- [] update `prisma/schema.prisma` with model for `Event` and `Registration`
- [] run `npm run setup` to update DB
- [] update `prisma/seed.ts` to seed DB with some data for events
- [] run `npm run setup` again to update DB with new seed

7. Loading data from DB

Branch 04

Todo

- [ ] add async `models/events.server.ts` with function `getEventsWithRegistrationCount`
- [ ] update `events/index.tsx` to use `getEventsWithRegistrationCount` to load data
- [ ] update component to registrations correctly

8. Adding event details page

Branch 05

Todo

- [ ] add route `events/$eventId.tsx` with component `EventDetailsPage`
- [ ] update `events/index.tsx` to link to event details page
- [ ] add async function `getEventById` to `models/events.server.ts`
- [ ] update `events/$eventId.tsx` to use `getEventById` to load data

9. Adding registration form

Branch 06

Todo

- [] add `Form` with method `POST` to `events/$eventId.tsx`
- [] add `input` of type `text` for `name`
- [] add `button` of type `submit` with text `Register`
- [] add async `action` handler
- [] add Tailwind forms
  - `npm install @tailwindcss/forms`
  - add `@tailwindcss/forms` to `tailwind.config.js` (`plugins: [require("@tailwindcss/forms")]`)

10. Adding registration to DB

Branch 07

Todo

- [] add `models/registrations.server.ts` with async `createRegistration` function
- [] updated `action` handler in `events/$eventId.tsx` to use `createRegistration`
- [] use `request.formData` to read value from the form
- [] `redirect` back to `events` page

11. Adding form validation

Branch 08

Todo

- [] extend `action` handler to validate eventId and name. If one of them is missing return `json` with errors
- [] use `useActionData` hook to display error in the form

12. Displaying list of registrations

Branch 09

Todo

- [ ] extend `models/registration.server.ts` with `getEventRegistrations` function
- [ ] update `events/$eventId.tsx` to use `getEventRegistrations` to load data and return it together with `event` data
- [ ] display list or registrations(registered users) in the `events/$eventId.tsx` page
- [ ] extract list of registrations to separate component (`components/RegistraitonList.tsx`)

13. Adding boundaries

Branch 10

Todo

- [ ] navigate to page with non-existing event id - you will see 404 error, but no UI
- [ ] add `CatchBoundary` to `events/$eventId.tsx` to display error UI
- [ ] use `useCatch` to get caught error and display it in the UI
- [ ] throw new `Error`, if status is unknown
- [ ] add `ErrorBoundary` to `events/index.tsx` to handle thrown Errors in the UI