Proposal:

Project: For our final project, we propose to design and implement a functional command-line calendar persistent data structure where modification of it produces new schedules in Haskell. The system will allow users to create, store, view, and manage events. The calendar will allow users to define events, recurring tasks, and scheduling rules, then use functional algorithms to detect conflicts and generate solutions to problems / optimized schedules.

Milestones:

- Easy: Create the custom event type which will include: title, date, time, location, reminders, etc
    - Create the custom calendar type where it will be able to insert and delete events, organize events, report conflicts,
        - Define core data types for schedules, events and time intervals
        - Detect time conflicts between events
        - Print daily or weekly calendar views in terminal

- Medium:
    - Support recurring events (weekly, monthly patterns)
    - Maybe if possible be able to store events/ log maybe have a databank of past events on disk,
    - Come medium hard functions will be able to filter and show events, store the events,
    - Export schedules using JSON or some text format?

- Hard
    - Constraint-based scheduling that can fit tasks into available time slots
    - Be able to recommend reschedulings to avoid conflicts
    - Build parser for a simple scheduling “domain specific language” - make commands inputted simple / human-readable
    - Be able to print/ view the entire calendar for a week/ month if possible

Resources:

- Algebraic data types which we have learned in class
- Be able to filter events by lifting functions (learned in class)
- IO functions we have learned in class to print the calendar and be able to respond to user requests
- We may need to use the state monad maybe to help with events and whether they have been altered, are new, etc I would need to learn more about how state.
- Parser combinator resources online
- DSL and constraint solver are stretch goals, but will look into advanced functional patterns for them

- We will need the map function if we need to lift sorting functions for example from sorting by date maybe to sorting by the entire event.