Nutritionist Lab Helper

Request

Hello Claude. I want you to put your senior software engineering hat on. Below I will add a description of a small web app project. I need you to write the full software code for the project described.

The code should be neatly organized into functional modules and separated by concerns. It should be clear what file each piece of code belongs to. Provide a diagram including all the project files and their organization. Create artifacts for each file.

Strive to keep files small by segregating coherent functionality to services. Make good use of Design Patterns where necessary and their use improves code readability.

For every vendor dependency used, provide clear instructions at the end of the solution, on how to install and configure the dependency. Use the most up-to-date vendor documentation.

Optimize for code readability over brevity and performance.

General Description

The client requests a web-app that ingest tabular data that charts patients lab results. The data should be displayed in a table that is delightful to use. Features that make the table easy to use include patient filtering by patient name and date range. Additionally, it should be possible to annotate any cell in the table.

Product Description

App layout

The app consists of two primary functions. First, it should allow the user to create a dictionary of snippets, the Snippets Dictionary, that will be used when writing notes. Second, it should display a page optimized for viewing patient lab data and recording observations from that data using regular sentences as well as snippets from the Snippets Dictionary.

A header should display two links one to the Labs and another to the Snippets Dictionary. The App should land on the Labs page by default.

The app should consist of a patient list on the left panel.

In the center the app should display the Work Product component. It consists of the Patient Filter Bar, a Table component displaying patient lab data and a Notes component that displays old patient notes and allows for the writing of new notes. The table should be filtered to the provided patients in the patient Patient Filter Bar. The table should span the entire vertical length of the window. A toggle button should reveal the notes, in the notes component, for the selected patient. The notes should include a button to expand the notes window to the entire vertical length of the window. A toggle button should reveal the table, again, from the top of the page.

Patient list

The patient list should display a list of links with the name of the patient. Clicking on a patient list should load a) the patient lab data in the table component and b) the patient notes in the notes component.

The list should be multi-selectable using a checkbox next to the patient name. Clicking on a checkbox automatically adds the patient to the data filter. The list of selected patients should be stored in a global in-memory store, the Selected Patients store.

Patient groups

The user should be able to group 2 or more patients into patient groups. Groups should be displayed at the top of the list.

A button on the top of the Patient panel to create patient groups should be displayed. On clicking the button an embedded view in the panel should slide from the top and provide an input to label the selected group as well as a submit button. This saves the selected patients to a new group.

Clicking on a group should have the effect of programmatically selecting all the group members in the Patient list. Clicking on a group should first deselect all other members before selecting the group members.

Patient groups should have an X icon on the side of the group label in the list to delete the group. Groups deleted should be marked as deleted.

Deleting patients

A button next to the group button at the top of the panel should enable a delete action. If the delete button is pressed, patients checked in the list should be marked as deleted.

Patient data

The patient data is populated from the ingestion of tabulated lab data. Data comes in the form of a CSV document. When a file is uploaded new patients should be stored. The unique identifier of a patient is the full patient name.

Patient Filter Bar

When a patient group or multiple patients are selected in the Patient list the Patient Filter Bar will display these patients as pills. It should show 5 patients at once and provide an ellipsis button that reveals a menu to display the remaining selected patients. The menu allows for multi-select using checkboxes.

The patients that populate the filter bar should come from the Selected Patients store.

Clicking on a pill or selections in the overflow menu should toggle the patient from the table filter. To capture the state of the filter a property in each patient should capture whether this patient should be included in the filter. The default state of the selected pills when a patient is selected or a group is selected should be to be included in the filter.

An ‘x’ icon should be included in each pill or next to a selection in the overflow panel to remove the selected patient from the Patient Filter bar.

Table component

The table component will tabulate uploaded lab data.

The table should include a date range filter. The default filter is to the last lab. Aside from a specific date range toggle the date filter should include relative date choices like:

- Last 2 labs
- Last 3 labs
- Last 7 days
- Last 14 days

The table should be allowed to overflow horizontally. Cells outside the width of the window should be reachable with a scrollbar.

Cell annotations

Every cell in the table should include a notes text area, the annotations component. The text area should allow for an unlimited number of characters. Only the first 200 characters should be shown. The entire note should be revealed on hover. A pin button should appear on the upper right corner of a note to allow the user to pin the note so it does not disappear when the mouse moves out.

The cell annotations component should consume the Notes Tokenizer component to enable Snippets searching and insert.

The Notes component

The notes component will consist of a large text area input allowing an unlimited number of characters. The component should load all notes for the selected patient. The input should be scrolled to the bottom of the loaded notes with two new-lines inserted to allow for the writing of a new note.

All input should be persisted after a brief delay of 10s. The timer should run from the time of the last stroke.

Undo Service

The notes component should implement a limited undo and redo operations. These should be handled in a separate service. The service should take all data, it should also bind to the change event of the input to capture any change in data. On a 10s delay after the last stroke it should save the entire document. It should save the last 10 versions.

Notes Tokenizer Component

The Notes Tokenizer component consists of three parts.

Snippets Search service

One, is a Snippets Search service that receives input characters. When a double curly brace input is detected, the tokenizer should interpret the following characters as input used in a Snippets Dictionary lookup. The service will run a fuzzy search of the dictionary entries. The Snippets Search service should trigger an event whenever the search results change. The event should provide the entries and results matching the search term.

The search service should also handle computed results. Computed results are dynamic values that trigger an algorithm. For example, a `[date]` entry triggers a result with today's date.

Graphical component

The second part of the Notes Tokenize component consists of a component that displays a result list with the top 5 results as a menu over the position of the text cursor. Clicking on the menu should delete the user's search query and input the selection at the beginning of the query. The component should listen for directional key strokes to allow for the results to be navigated using the keyboard. Hitting the enter key should place the snippet result in the input area.

Cursor positioning service

The third part is a service, the Cursor Positioning service, takes an input element and calculates the position of the cursor and returns the top and left position coordinates of the bottom of the cursor. This service is used to position the results menu.

Orchestration

This component takes the text area input element as a parameter. The input is used to a) determine where the cursor is by the Cursor Positioning service, b) to collect text to be passed to the Snippets Search and c) to display the menu at the text cursor.

Technical Spec

Tech stack

The app should be built for the web using typescript Next.js

A popular UI component framework should be used for the graphical aspects.

The Vercel postgres database should be used for storage. A prisma ORM should be used to communicate with the database.

In-memory stores should use the Context API.

Data design

Data should be stored in the following tables:

- User table. The user table stores the the name of a user, in a `[Name]` column and assigns an incrementing `[PatientId]` to each user.
- Patient groups table. Keyed by `[GroupId]` this table has a `[GroupName]` column to define a group.
- Patient Group Membership table. A lookup table associating a `[PatientId]` with a `[GroupId]`.
- Labs table. A table storing lab results for a patient. A `[PatientId]` foreign key for each new row of labs. The table includes a `[CollectedDate]` column and a `[Results]` column taking a json object with the collected lab results. The json object has the following shape:
  ```
  {
    column: string,
    value: string,
  ```

```
  annotations: string
}[]
```

- Notes table. Keyed by `[PatientId]` each row in the table includes a `[Date]` column and a `[Note]` column which should allow for unlimited space.

Ingesting data

As described in the Patient list section, lab data will be uploaded to update the patients lab. The headers in the uploaded data have the following columns:

```
HdrDateRange,SortNameField0,CollDateField1,ALBField2,NPNAHField3,KField4,CAField5,ADJCAField6,PHOSField7,ADJCPField8,PTHIField9,PTHINField10,HGBField11,FERRField12,SATField13,KTVField14,GLUField15,VD25AField16,NAField17,POWTField18,ALBCount2,NPNAHCount3,KCount4,CACount5,ADJCACount6,PHOSCount7,ADJCPCount8,PTHICount9,PTHINCount10,HGBCount11,FERRCount12,SATCount13,KTVCount14,GLUCount15,VD25ACount16,NACount17,POWTCount18,ALBAvg2,NPNAHAvg3,KAvg4,CAAvg5,ADJCAAvg6,PHOSAvg7,ADJCPAvg8,PTHIAvg9,PTHINAvg10,HGBAvg11,FERRAvg12,SATAvg13,KTVAvg14,GLUAvg15,VD25AAvg16,NAAvg17,POWTAvg18,ALBMedian2,NPNAHMedian3,KMedian4,CAMedian5,ADJCAMedian6,PHOSMedian7,ADJCPMedian8,PTHIMedian9,PTHINMedian10,HGBMedian11,FERRMedian12,SATMedian13,KTVMedian14,GLUMedian15,VD25AMedian16,NAMedian17,POWTMedian18,ALBStDev2,NPNAHStDev3,KStDev4,CAStDev5,ADJCAStDev6,PHOSStDev7,ADJCPStDev8,PTHIStDev9,PTHINStDev10,HGBStDev11,FERRStDev12,SATStDev13,KTVStDev14,GLUStDev15,VD25AStDev16,NAStDev17,POWTStDev18,ALBLow2,NPNAHLow3,KLow4,CALow5,ADJCALow6,PHOSLow7,ADJCPLow8,PTHILow9,PTHINLow10,HGBLow11,FERRLow12,SATLow13,KTVLow14,GLULow15,VD25ALow16,NALow17,POWTLow18,ALBHigh2,NPNAHHigh3,KHigh4,CAHigh5,ADJCAHigh6,PHOSHigh7,ADJCPHigh8,PTHIHigh9,PTHINHigh10,HGBHigh11,FERRHigh12,SATHigh13,KTVHigh14,GLUHigh15,VD25AHigh16,NAHigh17,POWTHigh18
```

Of these the only columns of interest are:

- `SortNameField0`: this is the `[Name]` in the Patients table

- `CollDateField1`: is the `[Date]` in the Labs table
- `ALBField2,NPNAHField3,KField4,CAField5,ADJCAField6,PHOSField7,ADJCPField8,PTHIField9,PTHINField10,HGBField11,FERRField12,SATField13,KTVField14,GLUField15,VD25AField16,NAField17,POWTField18`: These fields are the columns that define the lab results. They will be the `keys` in the `[Result]` JSON object. The word `Field` and the number that comes after it should be removed when using them as key entries in the `[Result]` JSON object.

All other columns can be ignored.

To ingest the labs the following steps should be followed:

1. Load the entire patient list in memory
2. Loop through each row. Take the name in the `SortNameField0` and lookup whether the patient exists. If it doesn't make a new entry in the Patients table and save the resulting `[PatientId]`.
3. Extract the `CollDateField1` and the result fields. Process the fields into the JSON `[Result]` object. Take the `[Date]` and the `[Results]` and insert them in the table under the `[PatientId]` stored in an earlier step