GSoC Student Application & Proposal

OpenMRS GSOC Program 2022

Student Name	**Anjula Samarasinghe**
OpenMRS Talk profile link (e.g. you can find this here)	https://talk.openmrs.org/u/**anjisvj**
GitHub profile link	https://github.com/**anjula-sack**
Country	Sri Lanka
Preferred method of contact & details (e.g. phone, email, IRC, etc.)	Email: anjulashanaka@gmail.com

Table of contents

1. Required Sections

1.1. About Me

1. Tell us a little about yourself. Who are you? What are you studying?

   I'm Anjula Samarasinghe, a first-year physical science ICT undergraduate from the University of Sri Jayawardenepura, Sri Lanka. I am contributing to the OpenMRS community as a developer and working as a Software Engineer at Sustainable Education Foundation.

2. Why are you the right person for this project?

   I've been an openMRS contributor for almost two years now and have gained, though not perfect, a fair bit of knowledge on the openMRS data structures, specifically on cohort builder and how the underlying metadata and workflows work especially after doing research for this specific project

   Apart from that I've worked in QA squad and DHIS2 squad in openMRS as a contributor

   Further I'm very much familiar with the technologies associated with this project as I've done two internships at promiseQ [1] and Rootcode Labs [2], both of them which involved development React Js with TypeScript (link to the projects/experience segment)

   I'm taking part as a software engineer in the core team of one of the most prominent Sri Lankan open source organizations Sustainable Education Foundation (SEF) [2], so I'm pretty familiar with the open-source culture as well.

   [1] https://www.promiseq.com/
   [2] https://rootcodelabs.com/
   [3] http://sefglobal.org

3. Describe in detail your software development experience by various technologies. Include all technologies you have used for development projects.

promiseQ web app - promiseQ
promiseQ utilizes machine learning recognition and human verification to provide accurate and fast decisions on alarm detections
[VueJs, TypeScript, Firebase]
https://www.promiseq.com/

promiseQ home page - promiseQ
promiseQ utilizes machine learning recognition and human verification to provide accurate and fast decisions on alarm detections
[NextJs, Material UI]
https://www.promiseq.com/

Expert Republic - Rootcode Labs
Expert Republic connects professionals with customers through 1-1 online video consultations. I've been working on implementing a dashboard since 2021.
[TypeScript, JavaScript, NextJS, Tailwind CSS]
https://expertrepublic.com/

Expert Republic Enterprise - Rootcode Labs
Expert Republic Enterprise is the enterprise version of the Expert Republic and I was working on implementing the website to showcase the product.
[TypeScript, JavaScript, NextJS, Tailwind CSS]
https://enterprise.expertrepublic.com/

ScholarX - Sustainable Education Foundation (SEF)
ScholarX is a mentoring platform we built to connect world-renowned mentors with students.
[TypeScript, ReactJS, Ant Design]
https://github.com/sef-global/scholarx-frontend

Academix - Sustainable Education Foundation (SEF)
Academix is an archive of knowledge resources that helps Sri Lankan students improve their knowledge in different areas.
[TypeScript, ReactJS, Ant Design]
https://github.com/sef-global/academix-frontend

Aphelia - Rootcode Labs
Aphelia is an AI-based resume parser API that can seamlessly integrate with your recruitment application. I was working on implementing the website to showcase the Aphelia AI.
[ReactJS]
https://aphelia.ai/

Entgra.io App Manager
Entgra is an open-source company that focuses on IoT, MDM, EMM and Factory Floor Solutions. I've contributed to Entgra in 2020.
[1] http://entgra.io
[2] My Contributions to Entgra IoT server
{ReactJs, Ant design}

Project GIC - Rootcode Labs
The project “Development of Land Resources Information Management System (LRIMS) for Afghanistan” is a part of the collaboration for Strengthening Afghanistan Institution’s capacity for monitoring and analyzing of agriculture production systems and development of Land Resources Information Management System (LRIMS) and National Agro-Ecological Zoning (NAEZ) project.

I was working on migrating the old outdated UI to a new one by adding additional features.
[HTML, Bootstrap JavaScript, jQuery, Django]
https://lrimsfaoaf.ait.ac.th/

Youtube Video Gallery - Sustainable Education Foundation (SEF)
Created a simple youtube video gallery to showcase SEF youtube videos.
[Spring Boot, Javascript, Ajax, Jquery]
https://sefglobal.org
https://github.com/sef-global/sef-data-holder

Website of Sustainable Education Foundation (SEF)
Sustainable Education Foundation (SEF) is a volunteer-driven organization. I led the website development of SEF which is an open-source project
[HTML , CSS, JS, JQuery, MustacheJS, Travis CI]
http://sefglobal.org
https://github.com/sef-global/sef-site

My Blog Posts: https://medium.com/@anjulashanaka
Knowledge Sharing Sessions: Youtube

4. List any previous experience working with open source projects other than OpenMRS. (This experience is not a requirement.)

   Sustainable Education Foundation
   Sustainable Education Foundation (SEF) is a volunteer-driven organization. I've been contributing to SEF since 2019. Currently, I’m working as a Senior Software Engineer at SEF. SEF runs projects which help to improve the quality of Sri Lanka’s Education.
   [1] http://sefglobal.org/
   [2] My Contributions to SEF Github organization

   Entgra.io
   Entgra is an open-source company that focuses on IoT, MDM, EMM and Factory Floor Solutions. I've contributed to Entgra in 2020.
   [1] http://entgra.io
   [2] My Contributions to Entgra IoT server

5. Provide links to any projects created by you, or other source code examples.

   My Github account:
   https://github.com/anjula-sack
   My Gitlab account:
   https://gitlab.com/anjula-sack

1.2. GSOC Background

6. Do you have any other commitments during the program? (Include any and all holidays, vacations, travel, exams, classes, research projects, other work, job offers, etc.)

   No

7. Have you participated in Google Summer of Code before? If yes, please mention the year, respected Organization, and your final result (Pass/Fail). *(Note: This will not affect your current application, it just helps the reviewers to have this background.)*
   Yes,
   2021
   OpenMRS
   Didn't get selected

8. Are you planning to be a Mentor for a Google Summer of Code project this year (2022) in any other organizations? If yes, which Organization(s)? *(Note: You can't be a student and mentor in the same year.)*
   No

9. Are you planning to apply to multiple Organizations this year? If yes, which Organization is your first preference if you get selected for both Organizations? *(Note: This does not need to be OpenMRS; please mention your preferred selection to help the reviewers.)*

   No

1.3. OpenMRS Experience

10. Describe your interactions with our community so far. Include dates of developer forums you have attended, and include any IRC nicknames used when visiting our channel previously.

    I joined the community back in July 2020. Mainly I've been engaged with the community via OpenMRS Talk. Also, I've contributed to the codebase by sending PRs and by reviewing. I've earned badges from the OpenMRS Talk like '/dev/1', 'Smart Developer', 'Friendly User', etc. I have attended weekly QA Squad, COVID squad calls and MFE

    I was able to present the QA Squad in the OpenMRS Conference 2021on QA Automation 3.x

11. **Code Contributions:** Please include all relevant issue numbers, pull requests, commit links, etc in the table below.
    *Note: You must have made **at least two coding contributions to OpenMRS before submitting** your application. If you don't include this information, your proposal will **not** be reviewed. We recommend achieving the /dev/1 stage as you become familiar with OpenMRS. It's not necessary for your pull requests to be merged; we just need to see that you've made some effort to learn the basics about OpenMRS development.*

Issue Number	**PR Link**	**Status** (e.g. In Progress, Pending Review, Merged)
CB-217	https://github.com/openmrs/openmrs-owa-cohortbuilder/pull/193	Ready for Review
RATEST-304	https://github.com/openmrs/openmrs-contrib-qaframework/pull/326	Merged
MF-869	https://github.com/openmrs/openmrs-contrib-qaframework/pull/169	Merged
MF-871	https://github.com/openmrs/openmrs-contrib-qaframework/pull/166	Merged
RATEST-246	https://github.com/openmrs/openmrs-contrib-qaframework/pull/143	Merged
DCM-5	https://github.com/openmrs/openmrs-module-dhisconnector/pull/5	Merged
DCM-10	https://github.com/openmrs/open	Merged

	mrs-module-dhisconnector/pull/10	
DCM-15	https://github.com/openmrs/openmrs-module-dhisconnector/pull/15	Merged
DCM-24	https://github.com/openmrs/openmrs-module-dhisconnector/pull/24	Merged
DCM-29	https://github.com/openmrs/openmrs-module-dhisconnector/pull/29	Merged
	TOTAL	**10**

12. **PR Reviews:** In the table below, please include all relevant issues where you have reviewed pull requests.
*Note: You must have made **at least three pull request reviews BEFORE submitting** your application. If you don't include this information, your proposal will **not** be reviewed.*

Project	**PR Link**	Issues/suggestions/improvements you mentioned in the review
OpenMRS DHIS Connector Module	https://github.com/openmrs/openmrs-module-dhisconnector/pull/26	*Suggested to use jQuery to handle div visibility* *Encouraged to move event listeners to separated functions* *Requested to change variable names* *Asked to remove hardcoded values*
OpenMRS ESM Patient Chart	https://github.com/openmrs/openmrs-esm-patient-chart/pull/217	*Suggested to use a proper variable naming convention for booleans*
OpenMRS ESM Primary Navigation	https://github.com/openmrs/openmrs-esm-primary-navigation/pull/63	*Suggested to use proper CSS class naming conventions*
OpenMRS ESM Patient Management	https://github.com/openmrs/openmrs-esm-patient-management/pull/97	*Removing extra white spaces and adding the EOL*

TOTAL	**04**

2. Project Proposal

2.1. Introduction

This proposal provides an overview of the GSoC project Redo Legacy UI Cohort Builder. It begins by describing the motivation behind selecting the project. Then the existing solutions are described followed by the problems of the existing solutions. Thereafter it describes the functional and non-functional requirements of the project. After describing the design of the proposed solution implementation will be presented. Finally, the proposal will be completed with the project timeline.

2.1.1. Project background

OpenMRS is a Java-based, web-based electronic medical record system. At the heart of OpenMRS is a concept dictionary. This dictionary, much like a typical dictionary, defines all of the unique concepts (both questions and answers) used throughout the system. Using combinations of questions and answers, we can define observations (observable data) as well as forms that gather multiple observations within a single encounter. With all this data OpenMRS needed to create groups of patients with different parameters. That's when the Cohort Builder too came to the rescue. To understand how this works first we need to understand the following,

1. Cohorts
2. Cohort Builder

1. Cohorts

A "Cohort" defines a group of patients. The need to define groups of patients comes up frequently in clinical care. Grouping of patients can be used during treatment (e.g., an HIV support group), for research (e.g., look at 'all males age 18-30 with diagnosis X' for shared risk factors), in population health (e.g., defining households in an outreach program), and many other scenarios. To date, OpenMRS has had a simple model for defining cohorts (the cohort – i.e., group of patients – has a name and description, patients are added to a cohort by adding an entry to the cohort_member table (there's a corresponding Java model for Cohort in openmrs-core). In the Reporting Module, cohorts were extended to include "evaluated cohorts," which are cohorts defined by a set of criteria – i.e., a list of patients determined by comparing patients to a set of criteria instead of a manually-managed, static list of patients.

2. Cohort Builder

The Cohort Builder is a tool in OpenMRS 1.0 in the Reporting Compatibility module (included with most OpenMRS installations) that lets the user perform ad-hoc queries for patients with defined characteristics, and combines multiple queries into more complex ones.

- It makes it easy to search for, and run saved report definitions or queries, for a given group of patients.
- It provides a fluid transition between the search, result tabulation and the result view.
- It presents data in a well structured and readable format
- The cohort-builder presents the data in a very good format.

There are seven ways of creating cohorts using different parameters.

1. Searching by Observations
2. Searching by demographics
3. Searching by encounters
4. Searching by program enrollments
5. Searching by Drug Order
6. Searching by SQL Queries
7. Combining searches

1. Searching by Observations
This gives the user the option to search via concepts or observations existing in the system.

- Concepts - individual data points collected from a population of patients. They include both questions and answers ex: Q. Does the patient value have a whole blood sample? A. whole blood sample
- Observations - anything actively measured or observed during an encounter ex: patient's weight, age or blood pressure

2. Searching by demographics
Searching by demographics gives the user the option to search using the demographics such as gender, age and also the ability to search based on a person's attributes

3. Searching by encounters
An encounter is a single, specific interaction between the patient and a provider.

Search by encounters searches for patients with a specific type of activity or encounter. For example, check-in, check out, transfer, admission etc.

4. Searching by program enrollments
Search by program enrollments allows the user to search for patients enrolled in a particular program or patients who have a particular status.

5. Searching by Drug Order
Searching by Drug Order allows the user to search for patients who taking specific drugs or patients who stopped or changed a drug

6. Searching by SQL Queries
WIth the search by SQL Queries feature the user can use custom SQL queries to build the cohorts.

7. Combining searches
After the user has done several searches, the Composition tab allows the user to combine them using Boolean algebra. The user can use AND, OR, NOT, or parentheses to build complex combinations of the other searches in the user's history. Refer to users' previous searches using the number next to them in the Search History section.

2.1.2. Existing solutions

The cohort builder has been a tool in OpenMRS for years. Currently, there are two existing implementations of the cohort builder.

1. Cohort-builder Legacy UI
   The legacy user interface for OpenMRS Platform 2. x is chiefly comprised of administrative functions and the patient dashboard. This cohort builder has the above-mentioned functionalities. Legacy UI uses JSP on the UI and DWR to communicate with the backend.

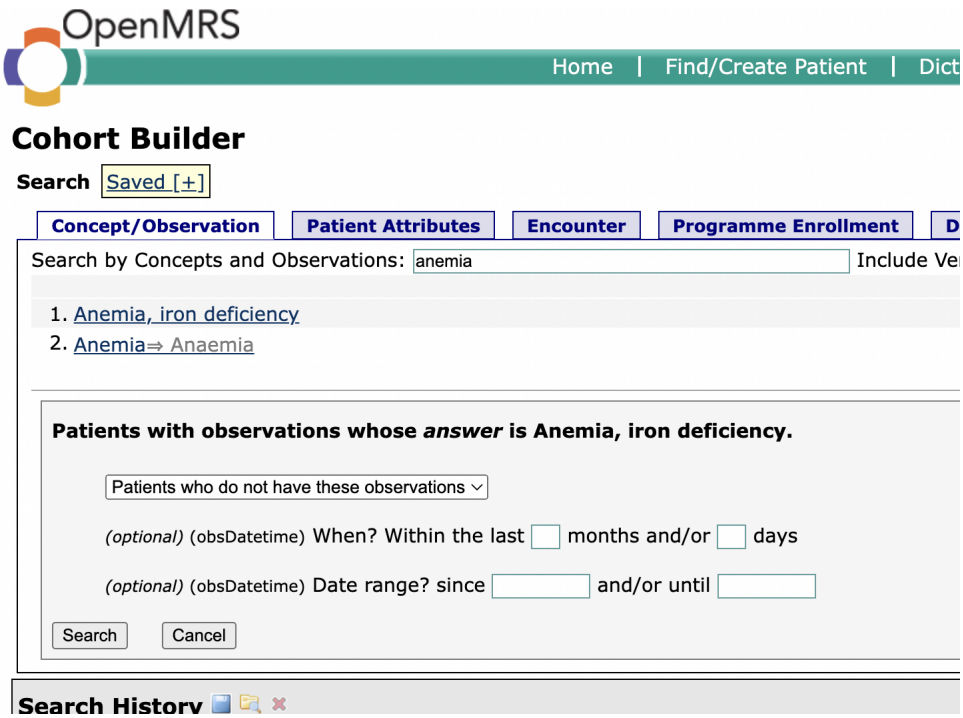


Figure 2.1.2.1 A screenshot of the legacy UI

2. Cohort-builder Open Web App

   The cohort-builder open web app is an OpenMRS tool used to generate reports as per the example on an ad hoc basis. This means that it builds a cohort of patients, based on the similarity of their characteristics.
   This doesn't offer the same functionalities that legacy UI offers like Searching by program enrollments Searching by Drug Order and Searching by SQL Queries.

   Cohort-builder OWA is built using ReactJS with JSX, Browsersync, Webpack, twitter-bootstrap, HTML and Javascript.

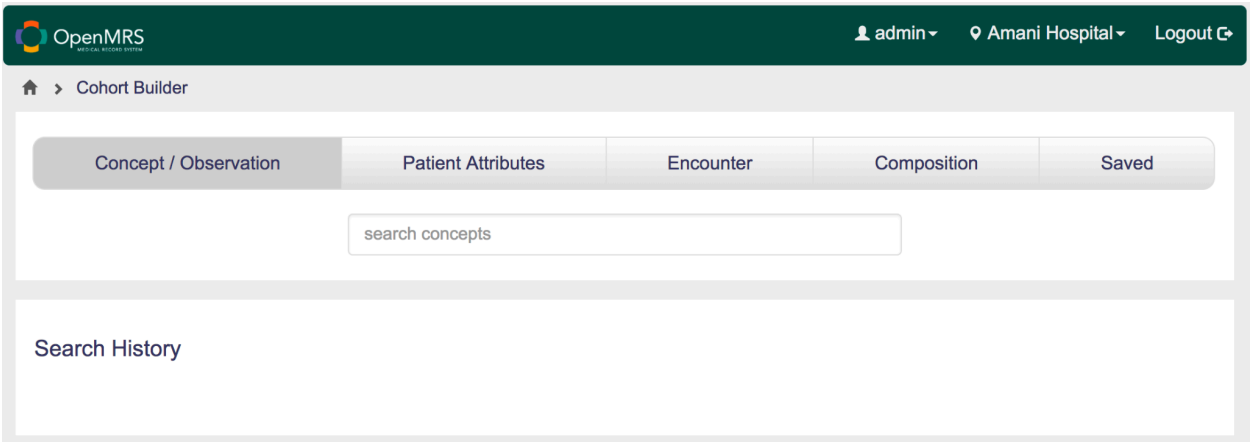


Figure 2.1.2.2 A screenshot of the OWA Cohort Builder UI

2.1.3. Problems with the current solutions

The main problem with the current solutions is they are outdated. Both solutions are implemented for previous OpenMRS versions(1.0 and 2.0). Since OpenMRS is moving from legacy server-rendered pages to React using Carbon Design within a micro frontend framework we need to help OpenMRS implementers to make this transition.

Mobile responsiveness is also a problem in these solutions. Another problem is Cohort builder OWA doesn't offer all the functions even though it uses relatively new technologies.

2.1.4. Functional Requirements

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks.

The Moscow method is a prioritization technique used in management, business analysis, project management, and software development to reach a common understanding with stakeholders on the importance they place on the delivery of each requirement

Priority Level	Description
Must have	Mandatory and critical requirements
Should have	Important, but not critical requirements. Better to have.
Could have	Nice to have. No impact on not having this requirement.
Will not have	Not required at the moment.

The new microfrontend of the module should definitely provide all the functionalities that the OpenMRS 1.0 legacy module had. There can be some additional features as well. The following table shows the requirements of the new microfrontend and their priorities according to the Moscow prioritization.

#	Requirement	Priority Level
FR-01	Should be able to search by Concepts/Observations	Must Have
FR-02	Should be able to search by demographics	Must Have
FR-03	Should be able to search by encounters	Must Have
FR-04	Should be able to search by program enrollments	Must Have
FR-05	Should be able to search by drug order	Must Have
FR-06	Should be able to search by SQL Queries	Must Have
FR-07	Should be able to combine searches	Must Have
FR-08	Should be able to schedule run saved cohorts	Must Have
FR-09	Should be able to see the search history with necessary actions	Must Have
FR-10	Should be able to search for saved search and cohort definitions	Must Have

2.1.5. Non Functional Requirements

User experience

User experience is important because it tries to fulfil the user's needs. It aims to provide positive experiences that keep a user loyal to the product.

Responsiveness
The importance of responsive web design is that it offers an optimized browsing experience.

Localization
Localization is ensuring that each page makes sense in a linguistic and cultural context for each country.

2.1.6. Chapter Summary

In this chapter the project background was described. Then the existing solutions were described followed by the problems of the existing solutions. In the end, the functional and non-functional requirements of the project were described. In the next chapter design of the proposed solution will be discussed.

2.2. Design

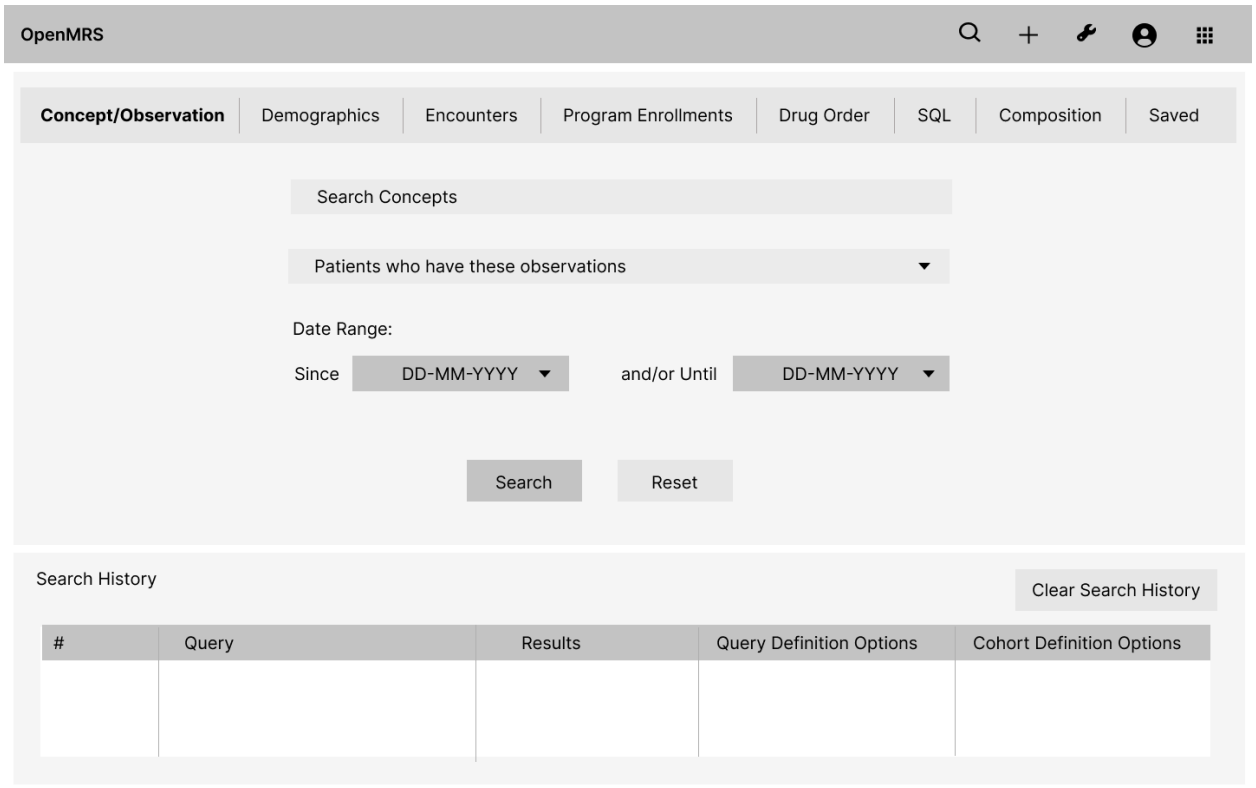
In this chapter the design of the proposed solution will be presented. Based on the above feature the proposed solution will have 10 main views to serve the main functionalities. Each view will be presented with the relevant wireframes.

1. Searching by Observations
2. Searching by Demographics
3. Searching by Encounters
4. Searching by Program Enrollments
5. Searching by Drug Order
6. Searching by SQL Queries
7. Composition
8. Search History
9. Saved Definitions Search
10. Cohort Results

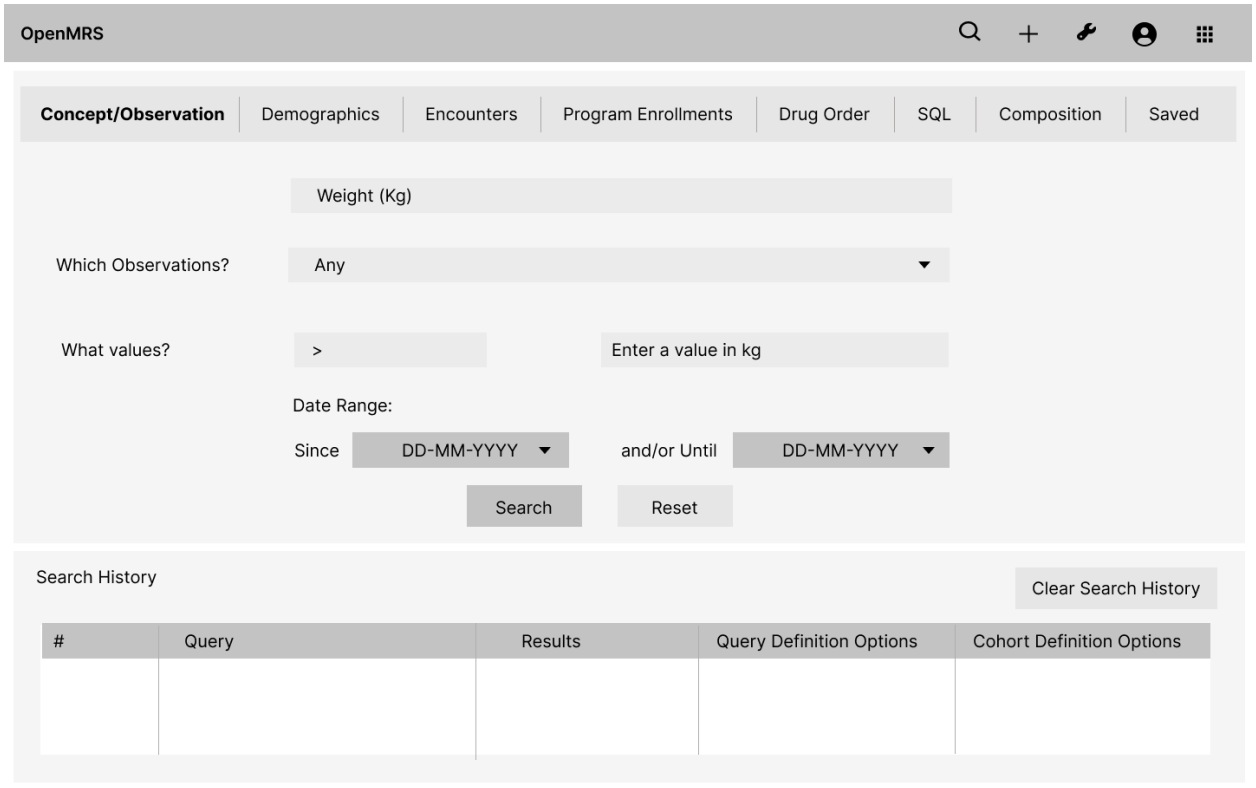
Since a cohort query returns a list of patients matching the specified criteria outputs of the above components will be a list of patients

2.2.1. Searching by observations

Searching by observations gives the user the option to search via concepts or observations existing in the system. This component has an input to search and a to search including verbose or without verbose.



2.2.1.1 Wireframe of the default view



2.2.1.2 Wireframe of search with an observation view

2.2.2. Searching by demographics

Searching by demographics is where the user can create cohorts using the demographics and patient attributes.

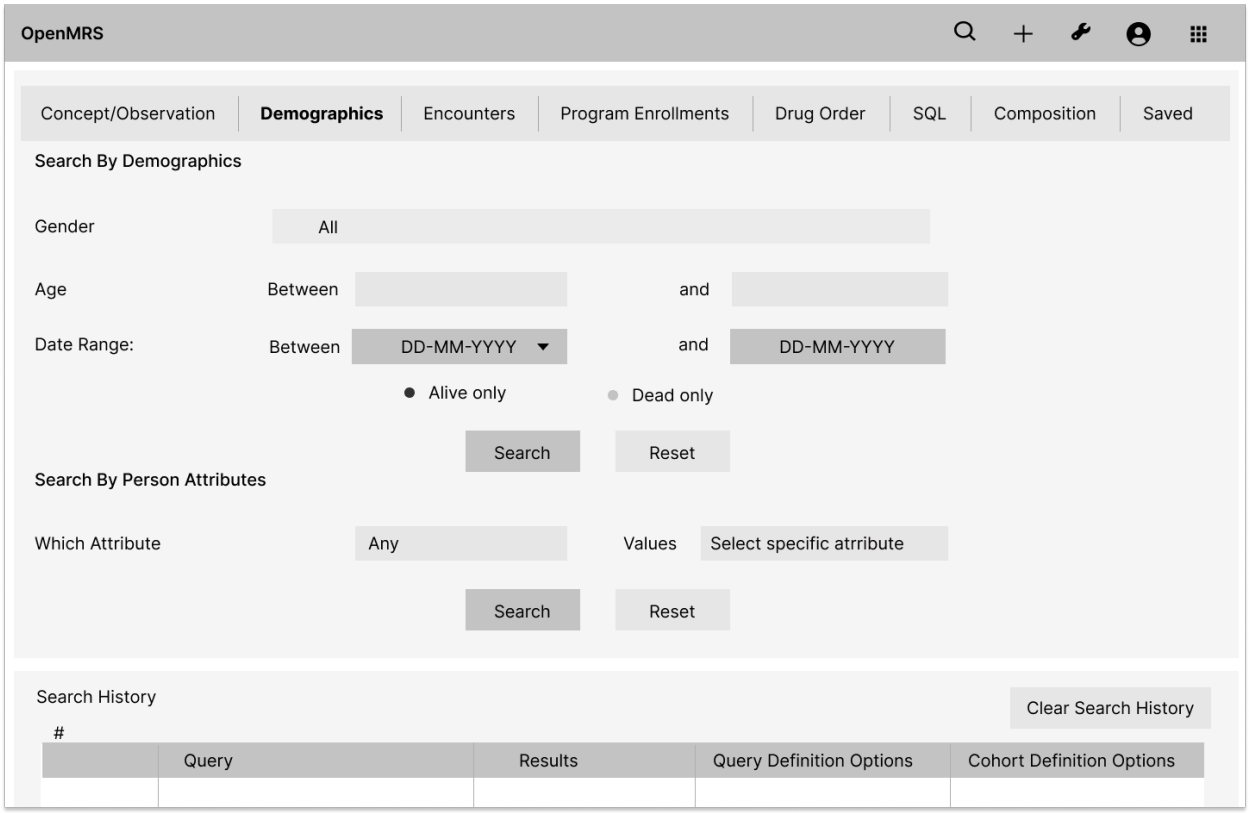
There are 5 demographic search options namely:

1. Gender - Search by gender option
2. Age - Filter patient search results by age range i.e. 0-14
3. Birthdate - Filter patient search results by birthdate range ie 20 Feb 1988 - 03 May 2006
4. Alive - Search only for patients who are alive.
5. Dead - Search for deceased patients.

There are 10 person attributes search options:

1. Birthplace filter by place of birth
2. Citizenship - Filter patient results by specifying their country of origin
3. Civil Status
4. Health Center - Filter by details of health centre attended by patients
5. Health District allows you to filter by similar health districts
6. Mother's Name - Filter by similar mother's name
7. Race - Filter results by the patient's racial background
8. Telephone number - Filter by similar phone numbers
9. Unknown patient
10. Test patient

This allows the user to search for patients with specific information that further describes them.



2.2.2.1. Wireframe of the search by demographics view

2.2.3. Searching by encounters

Search by encounters has two main ways to run the search with different parameters.

1. Search by Encounter
2. Search by Location

Search by Encounter:
Search by Encounter uses to search patients with a specific type of activity or encounter. For example, check-in, check-out, transfer, admission etc.

There are 5 search options namely: -

1. Of Type allows you to filter by the type of the encounter
2. The location allows you to filter by the location of the encounter
3. At least this many allows you to filter by the minimum encounter count
4. Up to this many allows you to filter by the maximum encounter type
5. From - To allows you to filter by the date range in which the encounter(s) occurred.

Search by Location:

Search by location allows the user to search for patients by the encounter location. Ex: Amani hospital, inpatient ward, isolation ward etc.
Here the filters will be applied according to the following:-

1. Patients belonging to a group of encounters from a specific location.
2. According to the timeline of the encounter

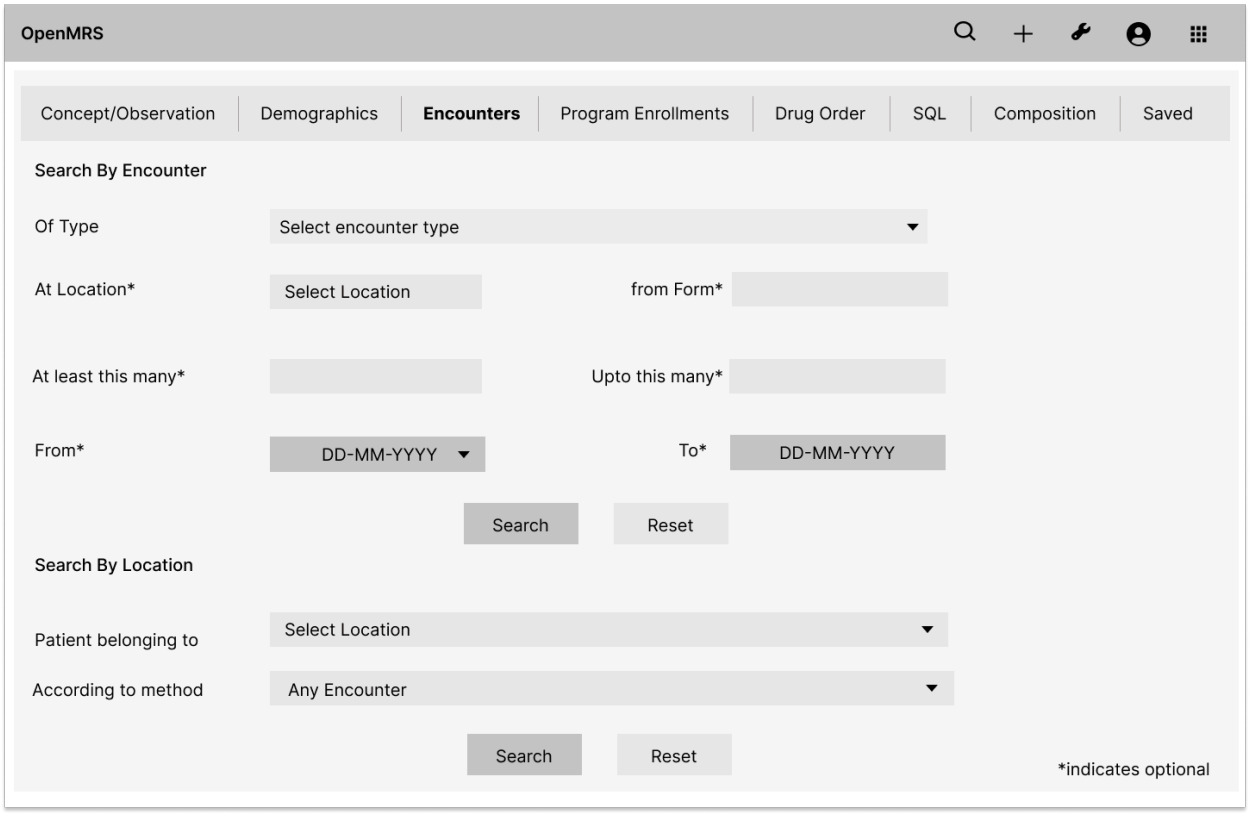


Figure 2.2.3.1. Wireframe of the search by Encounters

2.2.4. Searching by program enrollments

Searching by program enrollments has the following fields required by the user to perform the search.

1. Program:
2. Date Ranges for the following
    a. In The Program
    b. Enrolled in the program
    c. Completed the program

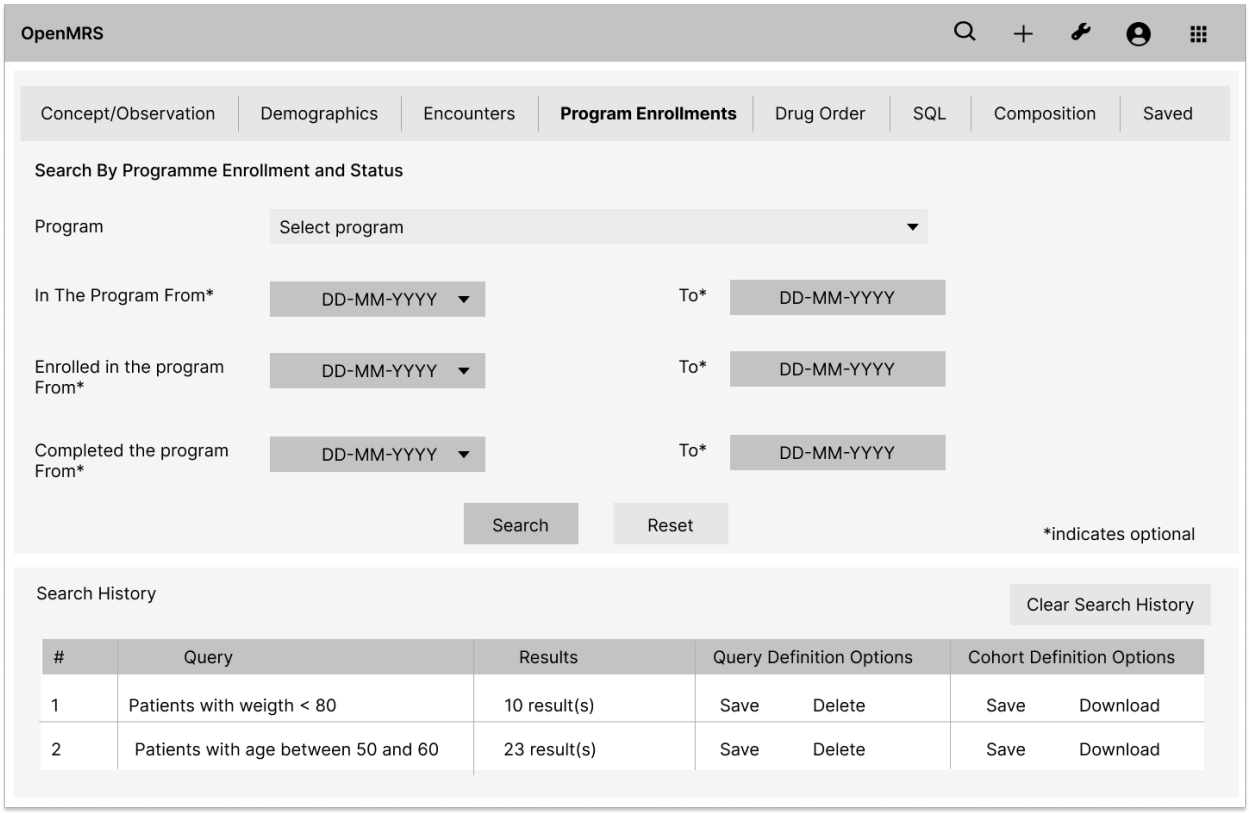


Figure 2.2.4.1. Wireframe of the search by program enrollments

2.2.5. Searching by Drug Order

This component has two search types:

1. Patients taking specific drugs
    a. Drugs
    b. When: Current drug regimen or All drug regiment
2. Patients who stopped or changed a drug
    a. Reason for stop/change
    b. Drugs
    c. When

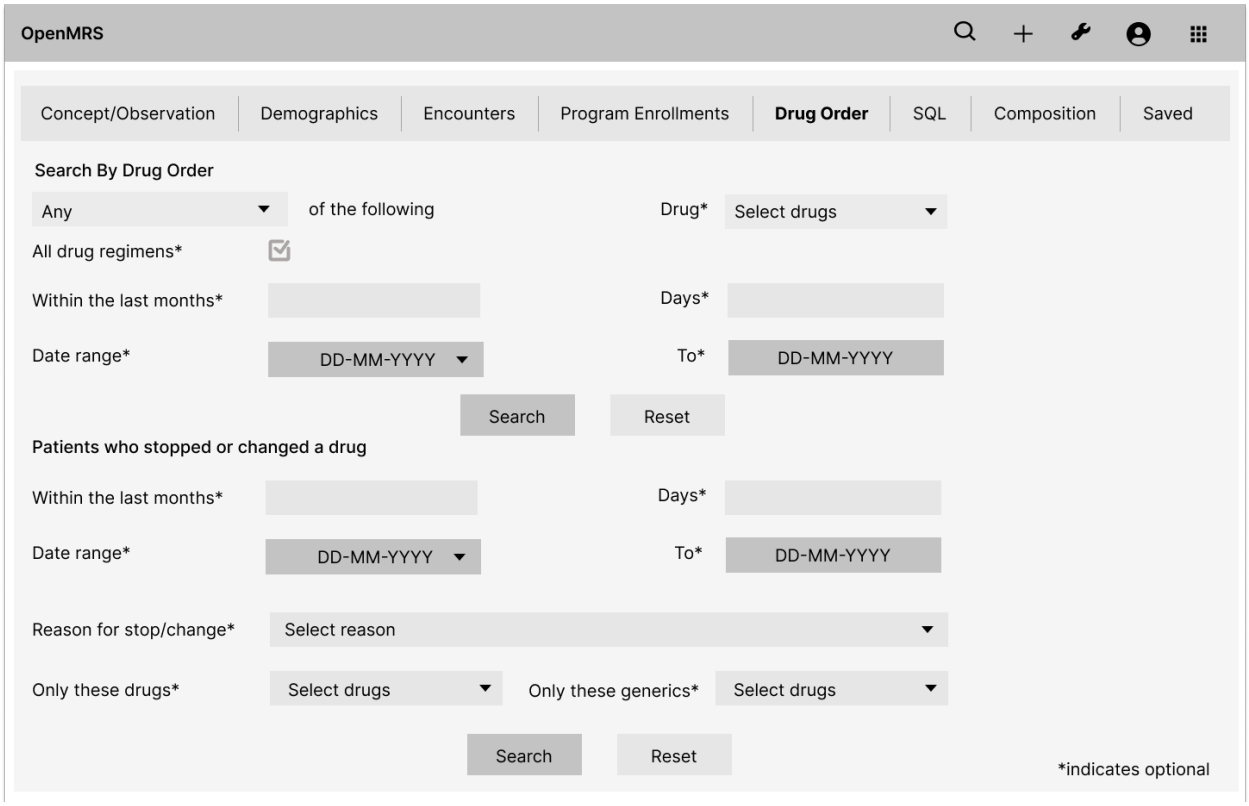


Figure 2.2.5.1. Wireframe of the search by drug order view

2.2.6. Searching by SQL Queries

Searching by SQL Queries has an input to run the custom SQL queries

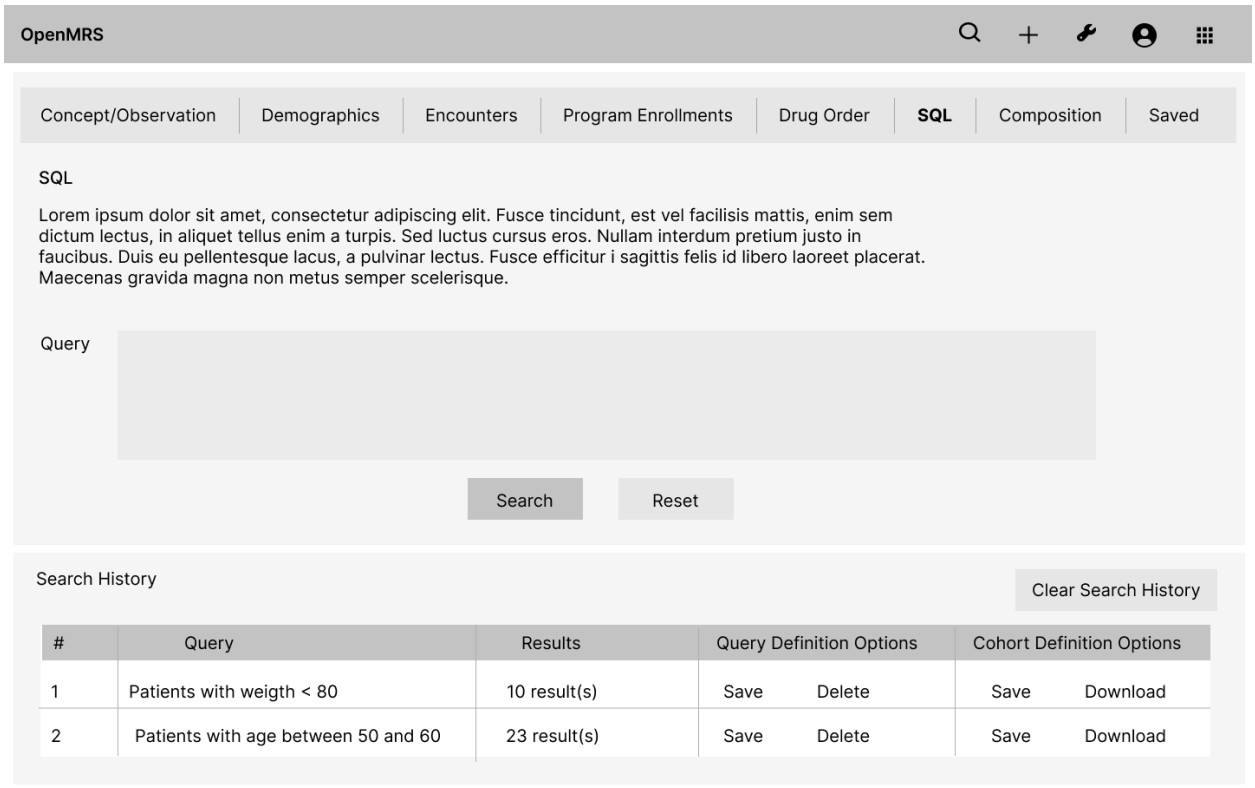


Figure 2.2.6.1. Wireframe of the search by SQL Queries view

2.2.7. Composition

To use the composition view, the user needs to have query results from the other queries in your search history.
These are the queries that will then be combined to yield new results.

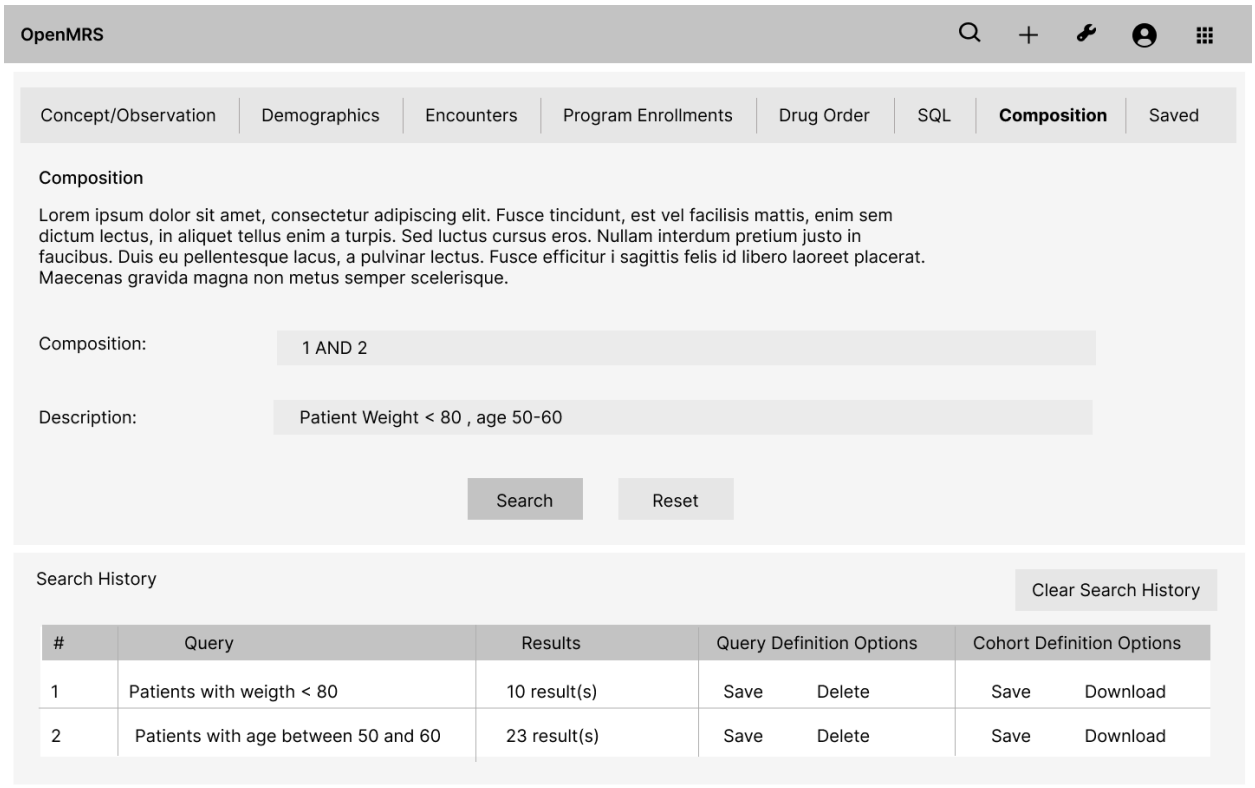


Figure 2.2.7.1. Wireframe of the composition view

2.2.8. Search History

This is where a history of executed searches are saved and displayed to the user, each search can be saved either as a cohort or as a definition. There are 4 fields under this component, namely

1. Query: This is where the name or description of the query is displayed
2. Query Definition Options
    a. Save: This saves a definition to the database
    b. Delete: This removes a definition from the search history
3. Results: This contains a list of all the patients contained in the search result
4. Query Definition Options
    a. Save: This saves a cohort to the database
    b. Delete: This removes a definition from the search history

This history is preserved until you choose to clear it or the web application is restarted.

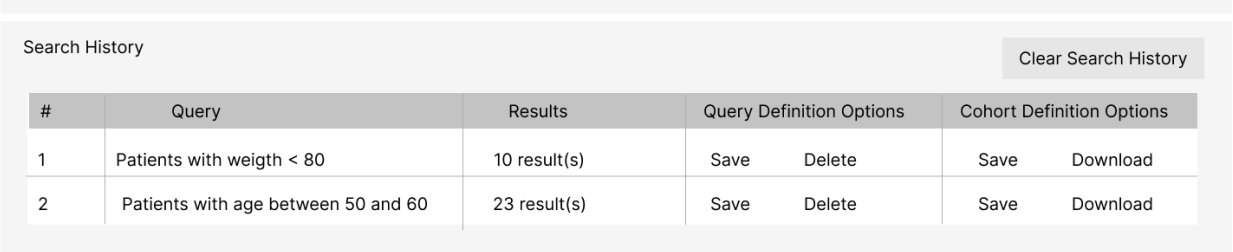


Figure 2.2.8.1. Wireframe of the search history view

2.2.9. Saved Definitions Search

Saved definitions search gives the user the option to search for a saved cohort or definition, it has two fields,

1. Search Saved Definitions allows the user to search for a saved definition by its name
2. Search Saved Cohort allows the user to search for a saved cohort by its name

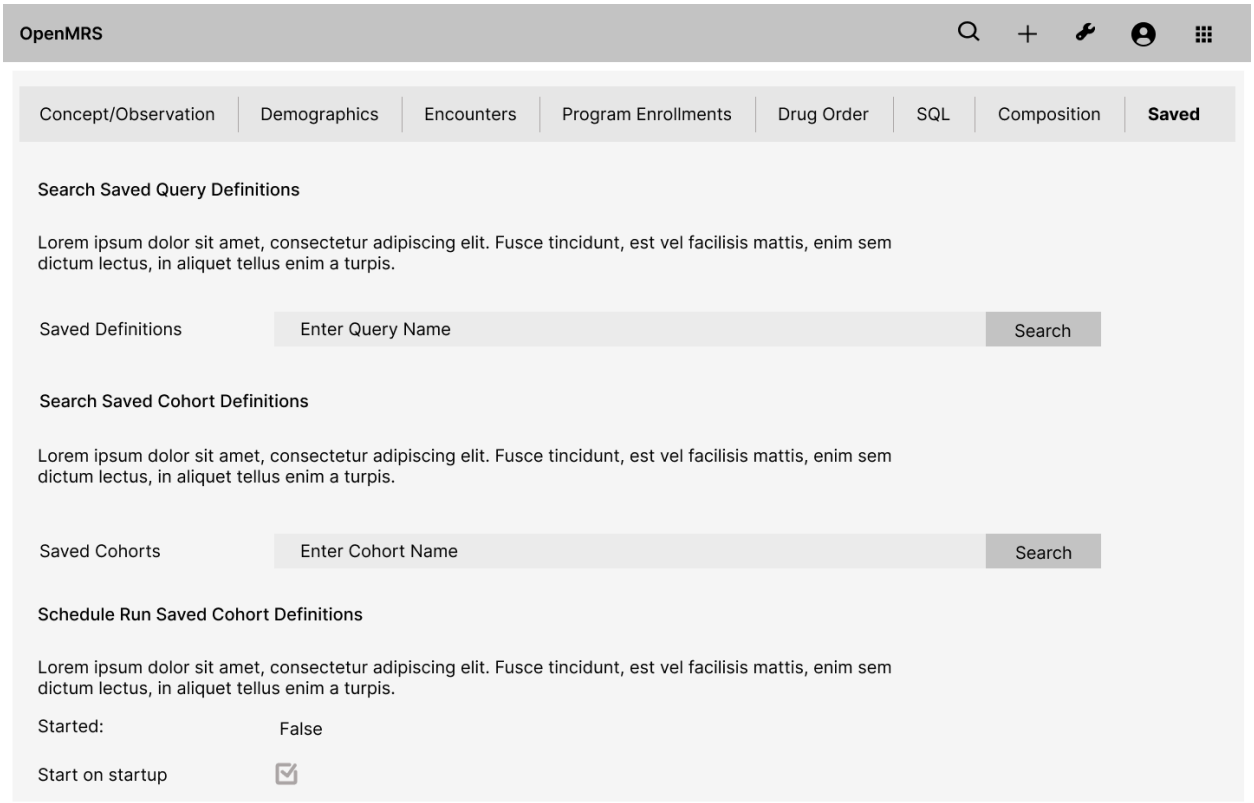


Figure 2.2.8.1. Wireframe of the default view

2.2.10. Cohort Run Scheduler

With the cohort run scheduler the user can configure the cohort runner which will run the saved cohorts at the scheduled time. The user can schedule, reschedule and stop running the saved cohort definitions.

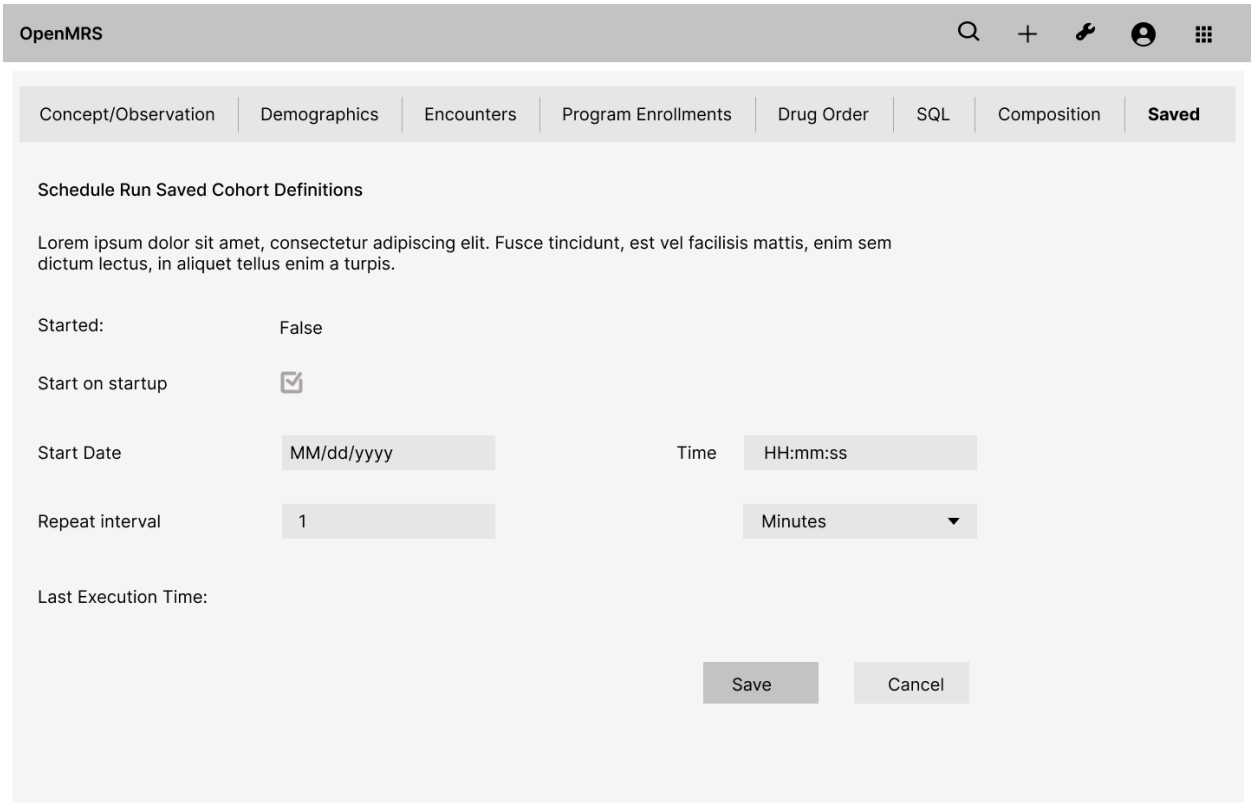


Figure 2.2.10.1. Wireframe of the default view

2.2.11. Cohort Results

This is where the results of the cohort will be shown. This list downs the patient on a table to give more readability and show more details about the patient. This component includes pagination to handle long lists of patients in order to improve the user experience.

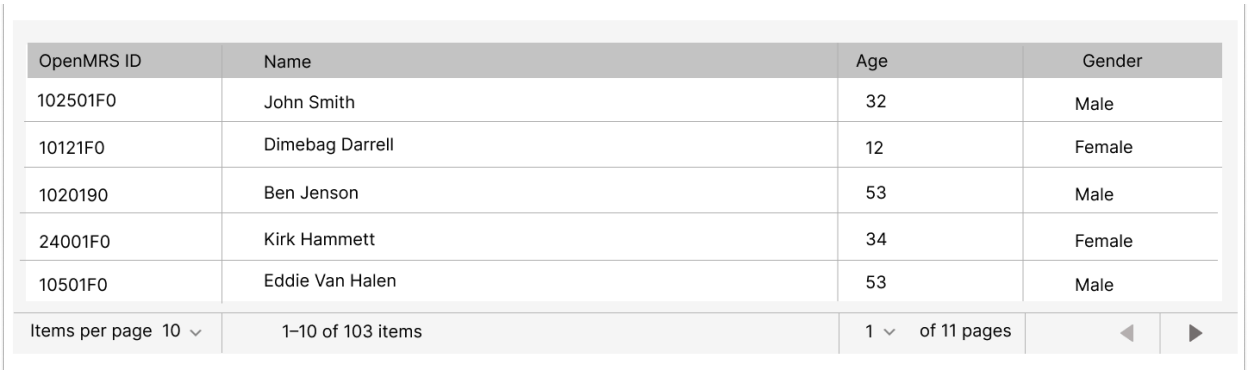


Figure 2.2.11.1. Wireframe of the results table

Clicking on a patient name will redirect to the patient summary view where the patient data is shown.

2.2.12. Chapter Summary

In this chapter the proposed design was described. Each wireframe clarified consistent ways for displaying particular types of information on the user interface.

2.3. Implementation

In this chapter implementation will be presented based on the above-discussed design. This includes the selection of the tools, and implementation of the views and the chapter will be ended with the testing section.

2.3.1. Selection of tools and technologies

OpenMRS Frontend 3.0. uses a microfrontends-based architecture. Micro frontends are in-browser javascript modules (ESMs) that provide application UI. For the Cohort Builder ESM module following technologies will be used.

Languages and libraries
TypeScript provides highly productive development tools for JavaScript IDEs and practices, like static checking. TypeScript simplifies JavaScript code, making it easier to read and debug.

OpenMRS is pretty much established with React and all the micro frontend modules are built with React.

UI Components
Carbon Design System is a free and open-source design system and library created by IBM. Following a design system helps to maintain consistency throughout the system. OpenMRS style Guide will be used for the customised components.

Testing
Jest is a JavaScript testing framework designed to ensure the correctness of any JavaScript codebase. Testing enables you to see what the software does and how well it does it so that the organisation can measure the quality of the software before it goes live.

Bundling
Webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser. Webpack is one of the most popular build tools out there.

OpenMRS ESM Template App
This repository provides a starting point for creating a new OpenMRS Microfrontend. ESM template comes with everything configured with the above technologies.

The following modules give the required APIs,

1. OpenMRS Web services Rest
2. OpenMRS Reporting Module Rest

For the creation of new REST APIs reporting compatible module will be referred to.

2.3.2. Implementation of views

In this section the implementation will be discussed based on each view.

2.3.2.1. Navigation Tabs

Navigation Tabs contain the links required to switch between the different cohort building methods. namely,

1. Concepts/Observations
2. Demographics
3. Encounters
4. Program enrollments
5. Drug Order
6. SQL
7. Composition
8. Saved
9. Search History
10. Cohort Results

With the navigation tabs user should be able to navigate through the different views easily. The following mockups show the final look of the navigation tabs.

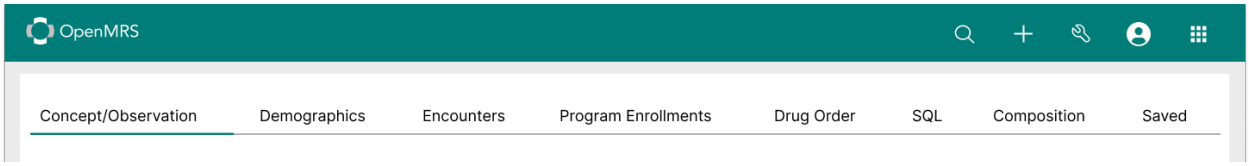


Figure 2.3.2.1. Proposed UI of the Navigation Tab

For the navigation tabs, Tabs carbon component will be used and the line variant matches the OpenMRS design.

2.3.2.2. Concepts/Observations

Concepts/Observations has two search types that take different inputs from the user. Based on that there are two main subcomponents in this view.

1. Concept Component
2. Observation Component

Concept Component

This includes the search input to get the concepts. When the user types concepts it will return the concepts that match the user input. For the search component, OMRS Search bar will be used because it helps to keep the whole platform consistent.

The following APIs will be used to implement the search by concepts function.

```
Concept {
      uuid :     String,
      display:   String,
      name:      {},
      datatype:  {},
      conceptClass:    {},
      descriptions:    [],
      answers:   [],
}
```

Get all the concepts

GET {baseUrl}/concept?v=full
Request Body: N/A
Response:
On Success: 200 OK List<Concept>
On Failure: 401 Unauthorized (When the user is not logged in)

Get a specific concept

GET {baseUrl}/concept?v=full&q={conceptName}
Request Body: N/A
Response:
On Success: 200 OK Concept
On Failure: 401 Unauthorized (When the user is not logged in)

Observation Component

This component includes the following,
Dropdown to select whether the cohort is for patients who have these observations or not
Date range selector to get the obsDateTime
Two inputs to get the last months or dates

"*What values*" dropdown contains the following values,

Less Than <
Less Equal <=
Equal =
Greater Than >
Greater Equal >=

Enter value input will be used to get the concept value in relevant units (ex: kg, cm).

The following components will be used for this view:
OMRS Primary Button
Input
Date Picker
Dropdown

The following APIs will be used to implement the search by concepts function.
Hl7 abbreviations are required to run the search.

```
hl7AbbrevTypes {
      CWE: 'codedObsSearchAdvanced',
      NM:  'numericObsSearchAdvanced',
      DT:  'dateObsSearchAdvanced',
      ST:  'dateObsSearchAdvanced',
      TS:  'textObsSearchAdvanced',
      ZZ:  'codedObsSearchAdvanced',
      BIT: 'codedObsSearchAdvanced'
    }
```

```
parameters {
      hl7AbbrevType:  [{
            name: String,
            value: String
   }]
]
```

Run the concept query

POST {baseUrl}/reportingrest/adhocquery?v=full

Request Body: parameters

Response:

On Success: 201 Created List<Patient>

On Failure: 401 Unauthorized (When the user is not logged in)

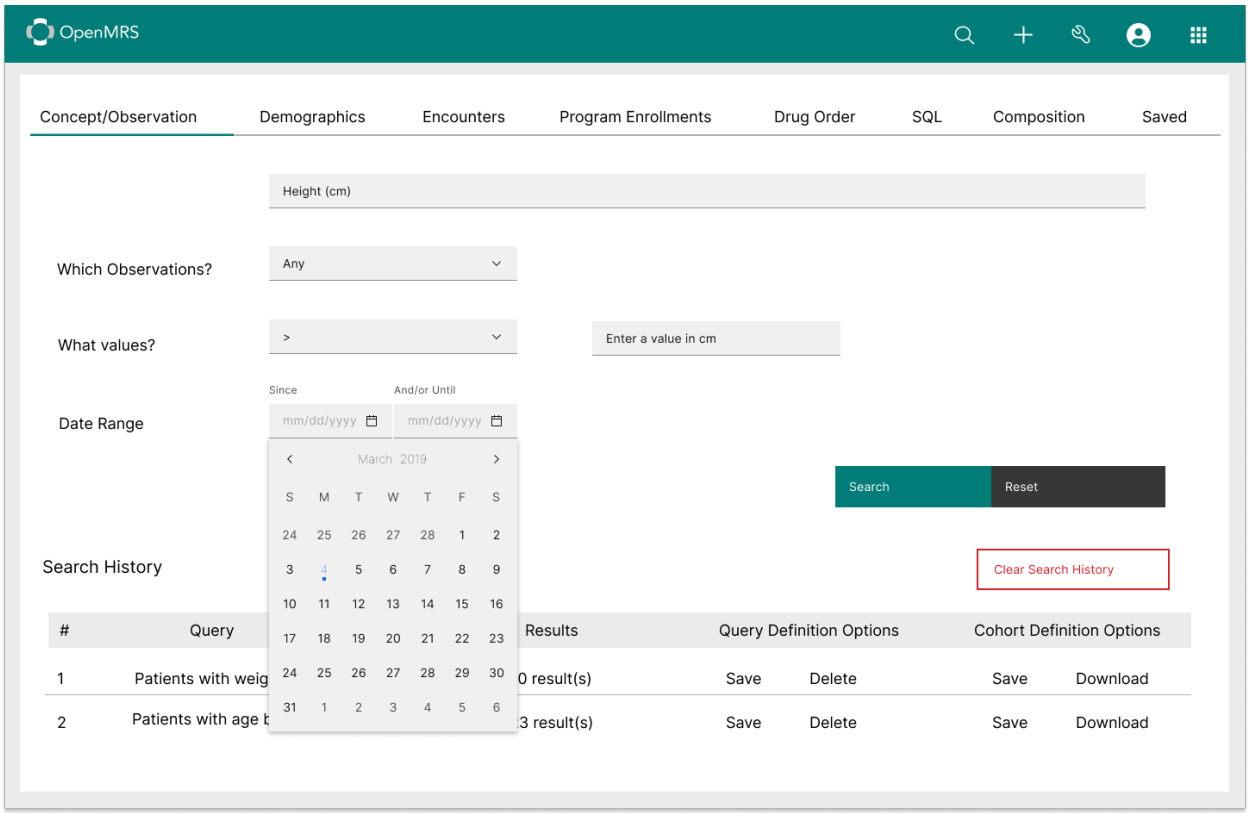


Figure 2.3.2.3.1. Proposed UI of Concept/Observation component

2.3.2.3. Demographics

Demographics component contains two search methods. For these two methods, the following components will be created.

Search by Demographics

This component contains the following components,

- A dropdown to get the gender with (Male and Female values)
- Two numerical inputs for getting the age range
- Birthdate range date picker
- Date of death range date picker
- A radio button to select alive or not alive

Search by Person Attributes
To get the person attributes a dropdown will be used and an input to get the value for the relevant attribute.

```
PatientAttributeTypes  {
            uuid: String,
            display: String,
            links: [
                {
                    rel: String,
                    uri:  String
                }
            ]
        }
```

Get the patient attributes

GET {baseUrl}/personattributetype
Request Body: N/A
Response:
On Success: 200 OK List<PatientAttributeTypes>
On Failure: 401 Unauthorized (When the user is not logged in)

```
searchParameters {
      gender: String,
      ageRangeOnDate: [
        { name: 'minAge', value: Number },
        { name: 'maxAge', value: Number}
      ],
      atLeastAgeOnDate: [
          { name: 'minAge', value: Number }
      ],
      upToAgeOnDate: [
          { name: 'maxAge', value: Number }
      ].
      bornDuringPeriod: [
        { name: 'startDate', dataType: 'date', value: Date },
        { name: 'endDate', dataType: 'date', value: Date }
      ]
}
```

Run the demographics query

POST {baseUrl}/reportingrest/adhocquery?v=full

Request Body: searchParams

Response:

On Success: 201 Created List<Patient>

On Failure: 401 Unauthorized (When the user is not logged in)

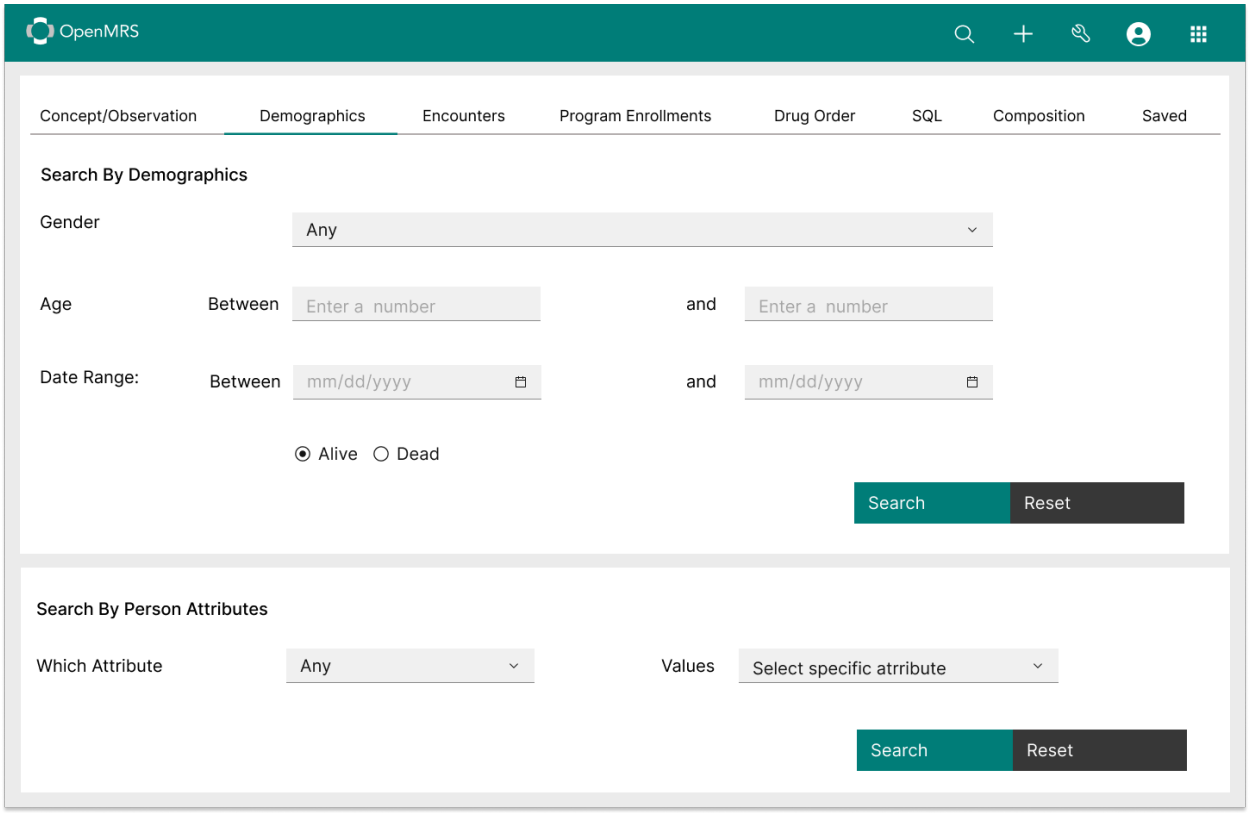


Figure 2.3.2.4.1. Proposed UI of Search by Demographics view

2.3.2.4. Encounters

Since there are two ways to search in the encounter view there will be two subcomponents as well.

1. Search by encounter component
2. Search by location component

Search by encounter
There are three dropdowns for selecting the encounter type, forms and locations. The carbon Dropdown component will be used for this and the following APIs will be used to get the data.

The encounter methods dropdown will have the following values,

Any Encounter
Most Recent Encounter
Earliest Encounter

```
EncounterType  {
            uuid: String,
            display: String,
            links: [
```

```
            {
                rel: String,
                uri:  String
            }
        ]
    }
```

Get the encounters

GET {baseUrl}/encountertype
Request Body: N/A
Response:
On Success: 200 OK List<EncounterType>
On Failure: 401 Unauthorized (When the user is not logged in)

```
Form  {
        uuid: String,
        display: String,
        links: [
            {
                rel: String,
                uri:  String
            }
        ]
    }
```

Get the forms

GET {baseUrl}/form
Request Body: N/A
Response:
On Success: 200 OK List<Form>
On Failure: 401 Unauthorized (When the user is not logged in)

```
Locations  {
        uuid: String,
        display: String,
        links: [
            {
                rel: String,
                uri:  String
            }
        ]
```

```
        }
```

Get the locations

GET {baseUrl}/locations
Request Body: N/A
Response:
On Success: 200 OK List<Locations>
On Failure: 401 Unauthorized (When the user is not logged in)

```
searchParams = { encounterSearchAdvanced: [
 { name: onOrAfter, value: Date },
 { name: onOrBefore, value: Date },
 { name: atLeastCount, value: Number },
 { name: atMostCount, value: Number },
 { name: formList, value: String[] },
 { name: locationList, value: String[] },
 { name: encounterTypeList, value: String[] },
] }
```

Run the encounter query

POST {baseUrl}/reportingrest/adhocquery?v=full
Request Body: searchParams
Response:
On Success: 201 Created List<Patient>
On Failure: 401 Unauthorized (When the user is not logged in)

Search by location

This has the locations dropdown and the encounter methods dropdown.

Get the locations

GET {baseUrl}/locations
Request Body: N/A
Response:
On Success: 200 OK List<Locations>
On Failure: 401 Unauthorized (When the user is not logged in)

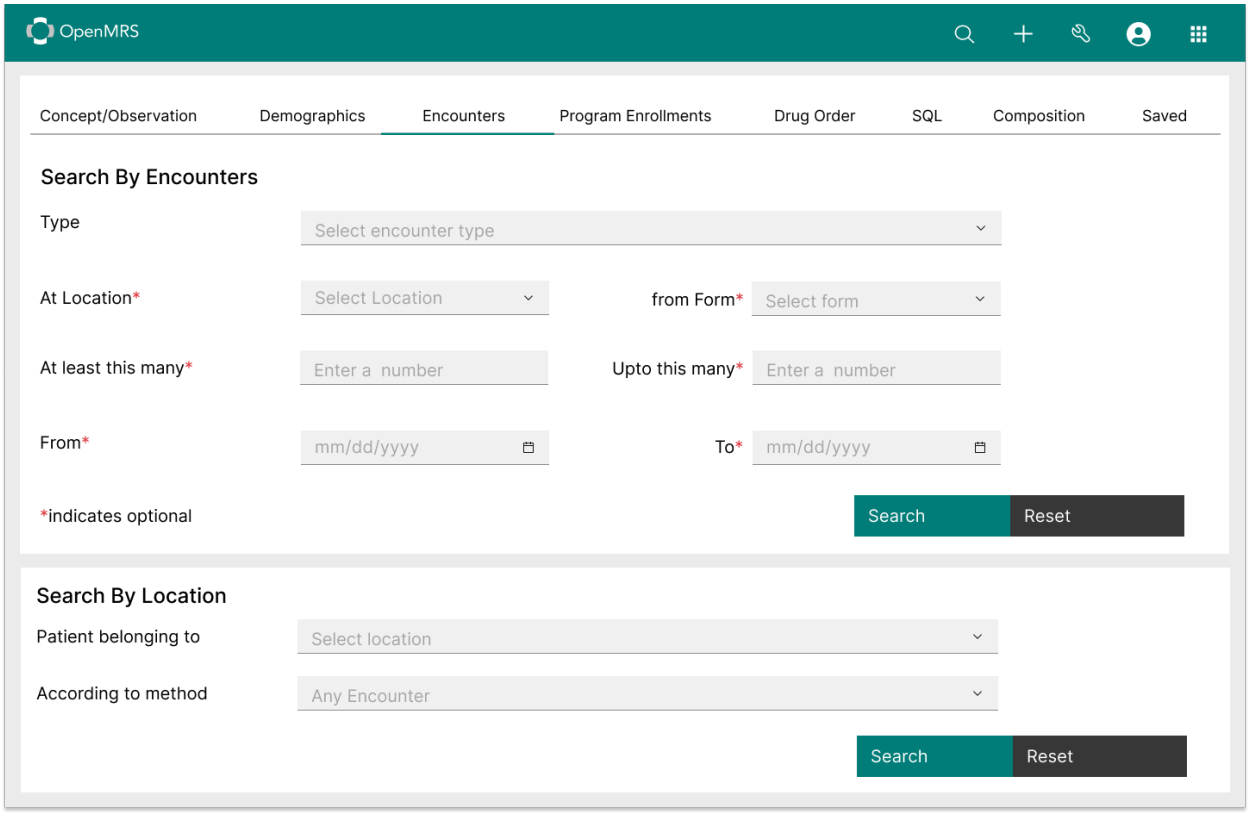


Figure 2.3.2.4.1. Proposed UI of Search by Encounters view

2.3.2.5. Program enrollments

Programs dropdown contains all the available programs that come from the following API. For selecting the date range there will be a [carbon datepicker](#).

```
Program  {
            uuid: String,
            name: String,
            allWorkflows: [],
            links: [
                {
                    rel: String,
                    uri:  String
                }
            ]
        }
```

Get the programs

GET {baseUrl}/program
Request Body: N/A
Response:
On Success: 200 OK List<Program>
On Failure: 401 Unauthorized (When the user is not logged in)

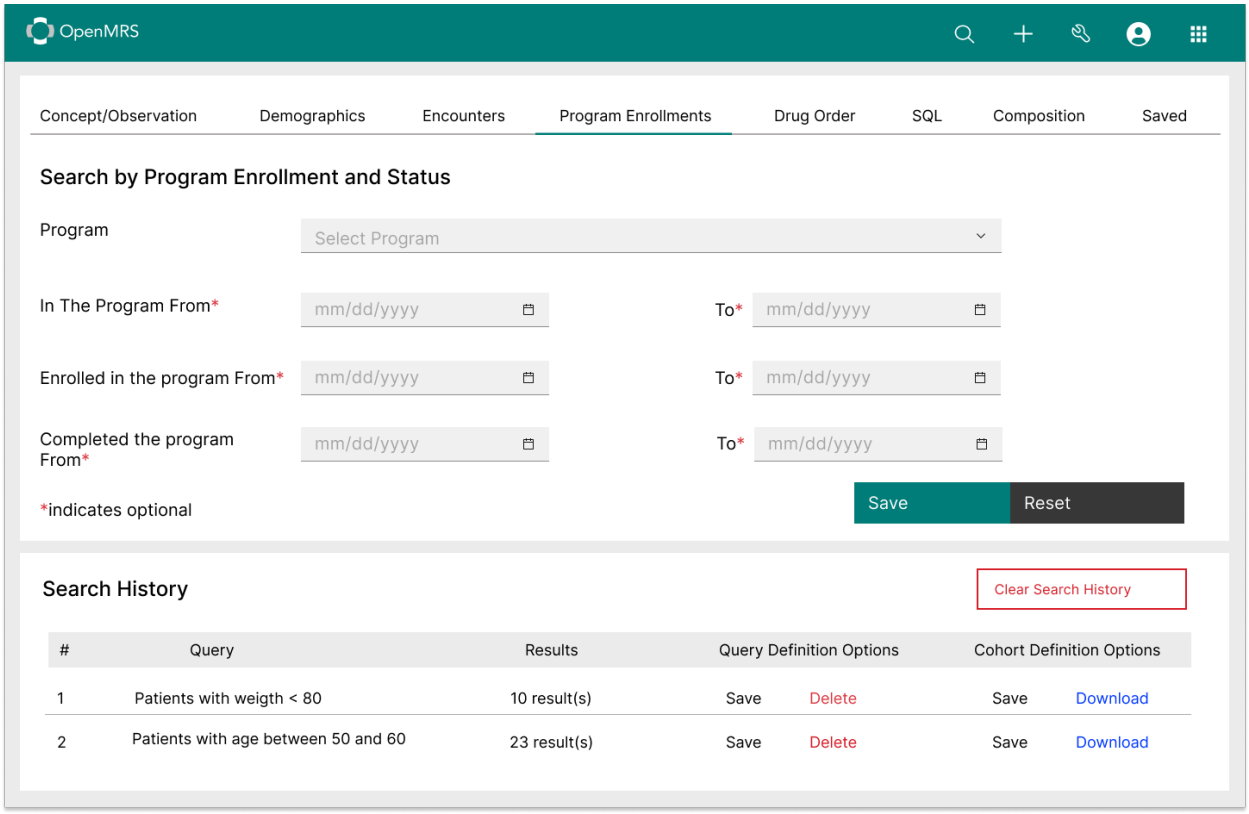


Figure 2.3.2.5.1. Proposed UI of Search by Program Enrollments

2.3.2.6. Drug Order

Drug Order components have two main subcomponents that serve the two search types.

1. Drug Order
2. Drug Usage

Drug Order
Drug Order component has a date range selector to get the observation date with time and two inputs to get the last months or dates values.

```
Drug  {
            uuid: String,
            display: String,
            links: [
                {
```

```
            rel: String,
            uri:  String
        }
    ]
}
```

Get the drugs

GET {baseUrl}/drugs
Request Body: N/A
Response:
On Success: 200 OK List<Drug>
On Failure: 401 Unauthorized (When the user is not logged in)

Drug Usage
Drug Usage component has the same component as the DrugUsage component but has two other dropdowns with the drug generics and drug order stop reasons.

This requires two new REST APIs to get the reasons for stopping the drugs and to get drug generics.

Get the drugs

GET {baseUrl}/drugs/generics
Request Body: N/A
Response:
On Success: 200 OK List<Drug>
On Failure: 401 Unauthorized (When the user is not logged in)

```
Reason  {
    conceptId: String,
    name: String,
}
```

Get the drugs

GET {baseUrl}/drugs/order-stop-reasons
Request Body: N/A
Response:
On Success: 200 OK List<Reason>
On Failure: 401 Unauthorized (When the user is not logged in)

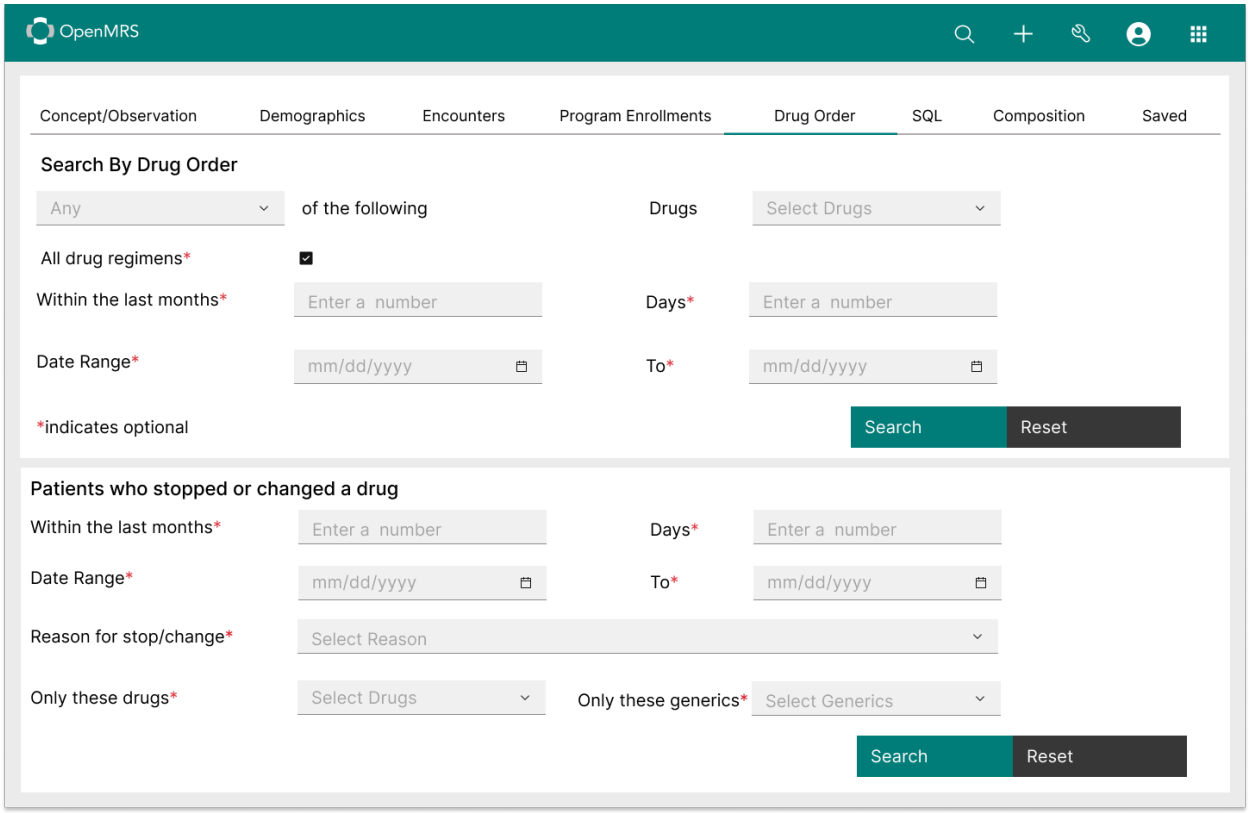


Figure 2.3.2.8.1. Proposed UI of Search by Drug Order

2.3.2.7. SQL

The user needs to input the SQL query into the text box. In this implementation, there will be a new REST API for running the query. Which takes the query in the body and returns the filtered patients.

```
Query  {
     query: String
}
```

Run the SQL query

POST {baseUrl}/sql-patient-filter

Request Body: Query

Response:

On Success: 201 Created List<Patient>

On Failure: 401 Unauthorized (When the user is not logged in)

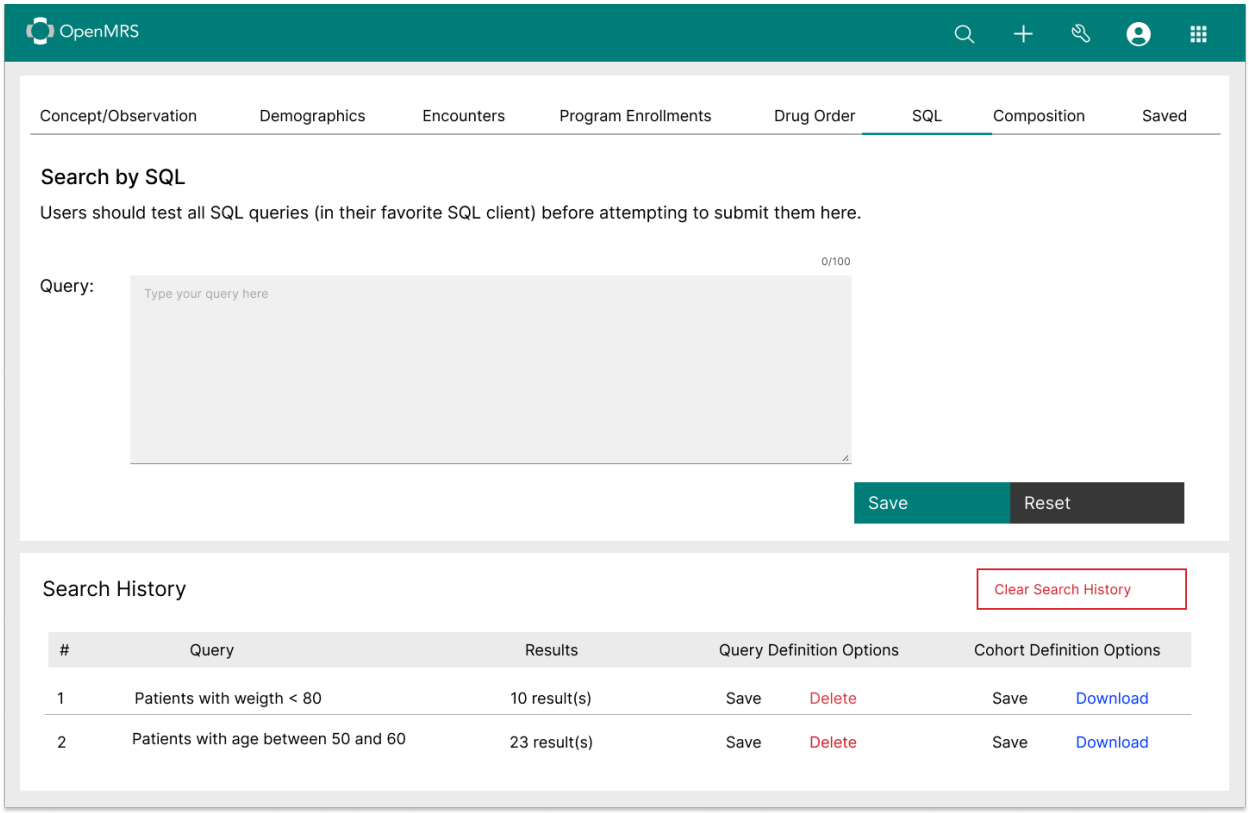


Figure 2.3.2.7.1. Proposed UI of Search by SQL view

2.3.2.8. Composition

There's an input for entering the operations.
Using the description input user enters a name or a description for the search composition that is being created

For the input component carbon Input component will be used.
Regex will be used to validate user input and to avoid errors.

```
Columns: [
    {
        "name": "firstname",
        "key":
"reporting.library.patientDataDefinition.builtIn.preferredName.givenName",
        "type":
"org.openmrs.module.reporting.data.patient.definition.PatientDataDefinition
"
    },
    {
        "name": "lastname",
```

```
        "key":
"reporting.library.patientDataDefinition.builtIn.preferredName.familyName",
        "type":
"org.openmrs.module.reporting.data.patient.definition.PatientDataDefinition
"
    },
    {
        "name": "gender",
        "key": "reporting.library.patientDataDefinition.builtIn.gender",
        "type":
"org.openmrs.module.reporting.data.patient.definition.PatientDataDefinition
"
    },
    {
        "name": "age",
        "key":
"reporting.library.patientDataDefinition.builtIn.ageOnDate.fullYears",
        "type":
"org.openmrs.module.reporting.data.patient.definition.PatientDataDefinition
"
    },
    {
        "name": "patientId",
        "key": "reporting.library.patientDataDefinition.builtIn.patientId",
        "type":
"org.openmrs.module.reporting.data.patient.definition.PatientDataDefinition
"
    }
]


AdhocDataset {
      type:
"org.openmrs.module.reporting.dataset.definition.PatientDataSetDefinition",
      customRowFilterCombination: String,
      rowFilters: [{type: String, key: String}],
      columns: Columns
}
```

Run the composition query

POST {baseUrl}/reportingrest/adhocquery?v=full

Request Body: AdhocDataset

Response:

On Success: 201 Created List<Patient>

On Failure: 401 Unauthorized (When the user is not logged in)

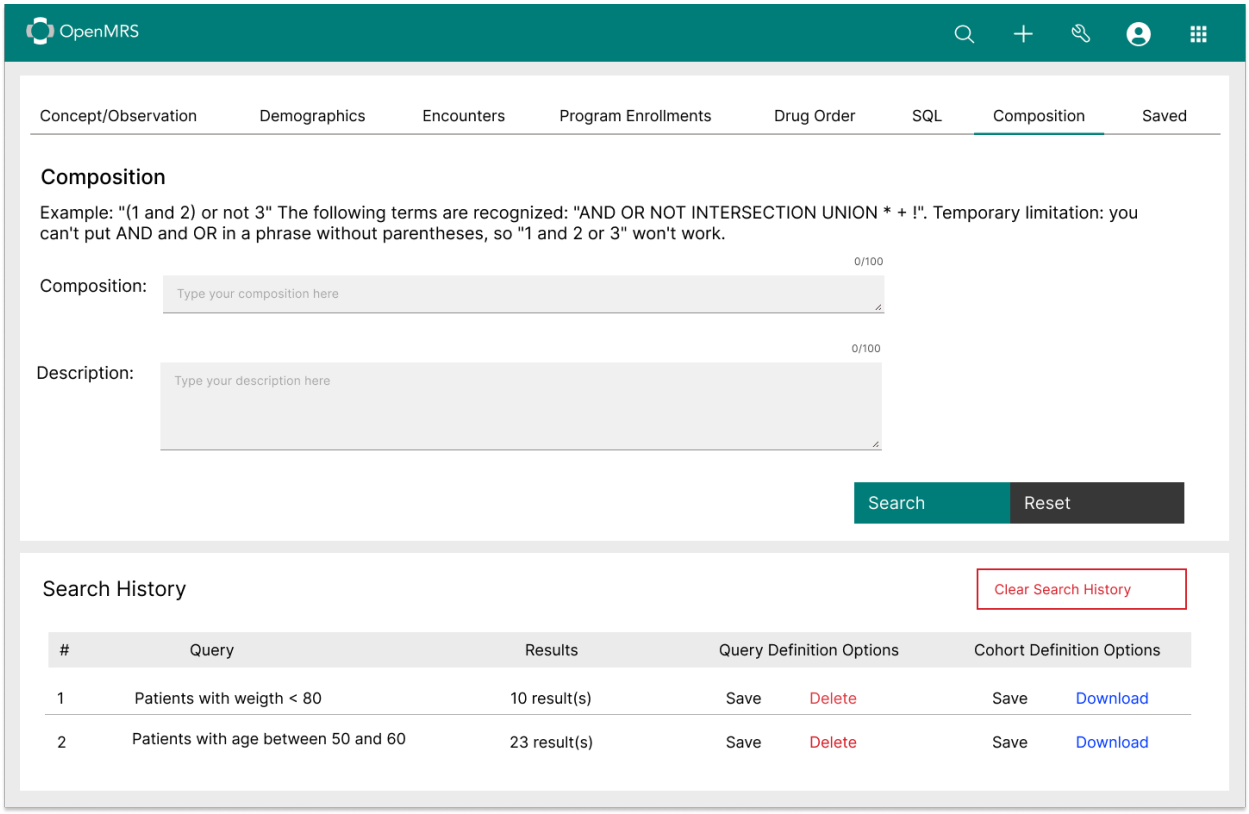


Figure 2.3.2.9.1. Proposed UI of Composition component

2.3.2.9. Saved Definitions

Saved Definitions component includes two search inputs for search saved definitions and saved cohorts. For the search inputs, the carbon Input component will be used. The following APIs will be used to implement this component.

Get saved definitions

GET {baseUrl}/reportingrest/dataSetDefinition?v=full&q={definitionsQuery}

Request Body: N/A

Response:

On Success: 200 OK AdhocDataset

On Failure: 401 Unauthorized (When the user is not logged in)

Get saved cohort

GET {baseUrl}/cohort?v=full&q={cohortsQuery}

Request Body: N/A

Response:

On Success: 200 OK List<Cohort>
On Failure: 401 Unauthorized (When the user is not logged in)

Delete cohort

DELETE {baseUrl}/cohort/{cohortId}
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

Delete definition

DELETE {baseUrl}/reportingrest/adhocdataset/{uuid}?purge=true
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

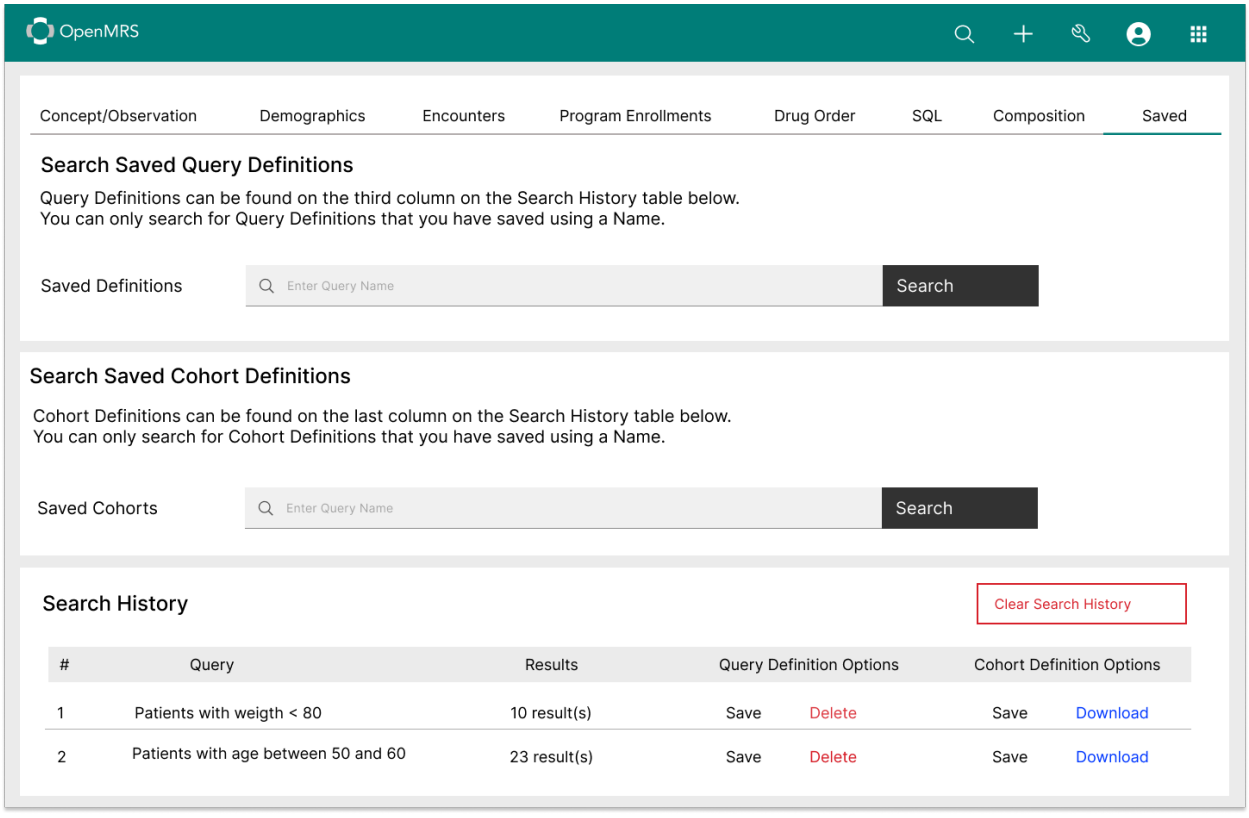


Figure 2.3.2.9.1. Proposed UI of Search Definitions view

2.3.2.10. Search History

To preserve the user's search history, Session Storage will be used because unlike local storage, session storage only keeps data for a particular session. Seach history will be an array of search history objects. Which includes the description, the patients and the parameters.

To show the search history Carbon Data Table will be used. The clear search history button will clear out the session storage and will be visible only if there's a history to show. OMRS Danger button will be used for the clear button. Before clearing out the history, a confirmation modal will be shown. OMRS Modal will be used for this.

The download link will download the data in CSV format.
The following APIs will be used to implement this component.

Save cohort

POST {baseUrl}/cohort
Request Body: Cohort
Response:
On Success: 201 Created Cohort
On Failure: 401 Unauthorized (When the user is not logged in)

Save search

POST {baseUrl}/reportingrest/adhocdataset
Request Body: Query
Response:
On Success: 201 Created
On Failure: 401 Unauthorized (When the user is not logged in)

Delete cohort

DELETE {baseUrl}/cohort/{cohortId}
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

Get dataset

GET {baseUrl}/reportingrest/dataSet/{uuid}
Request Body: N/A
Response:
On Success: 200 OK AdhocDataset
On Failure: 401 Unauthorized (When the user is not logged in)

Delete Adhoc dataset

DELETE {baseUrl}/reportingrest/adhocdataset/{uuid}?purge=true
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

The following figure shows the proposed UI for the search history component,

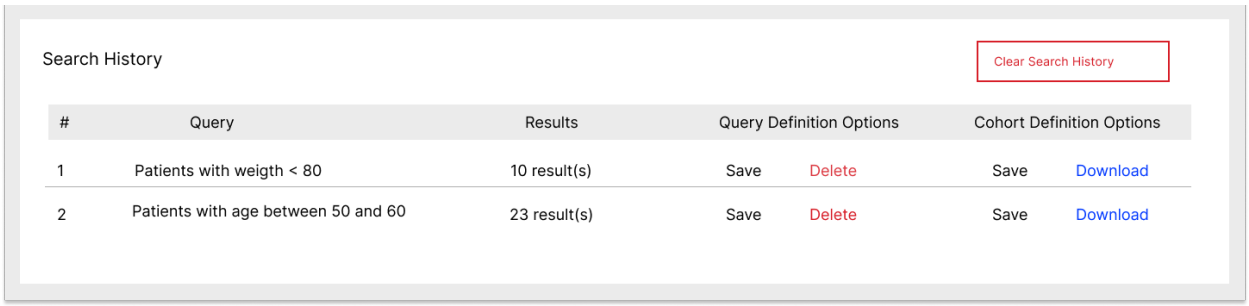


Figure 2.3.2.10.1. Proposed UI of Search History component

2.3.2.11. Cohort Results

Cohort results component list downs the cohort members in a carbon data table component with pagination. This receives the patients as a prop. When the user clicks on a patient name it redirects the user to the {baseURL}/patient/{uuid}/chart/Patient%20Summary which is the patient summary page that already exists.
The following figure shows the proposed UI of the search result component.

Figure 2.3.2.11.1. Proposed UI of Search Results component

2.3.2.12. Cohort Scheduler

A new CohortScheduleTask will be created in the OMRS Task Scheduler and TaskResources in the webservices rest will be used to implement this.

```
Task {
```

```
        startTime: Date,
        repeatInterval: Number;
        startOnStartup: Boolean,
        taskClass: String,
        name: String,
        description: String
 }
```

Create the cohort schedule

POST {baseUrl}/taskdefinition
Request Body: Task
Response:
On Success: 201 Created
On Failure: 401 Unauthorized (When the user is not logged in)

Get the cohort schedule

PUT {baseUrl}/taskdefinition/{taskID}
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

Update the cohort schedule

PUT {baseUrl}/taskdefinition/{uuid}
Request Body: Task
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

Delete the cohort schedule

DELETE {baseUrl}/taskdefinition/{uuid}?purge=true
Request Body: N/A
Response:
On Success: 200 OK
On Failure: 401 Unauthorized (When the user is not logged in)

The following figure shows the Cohort run scheduler default view.

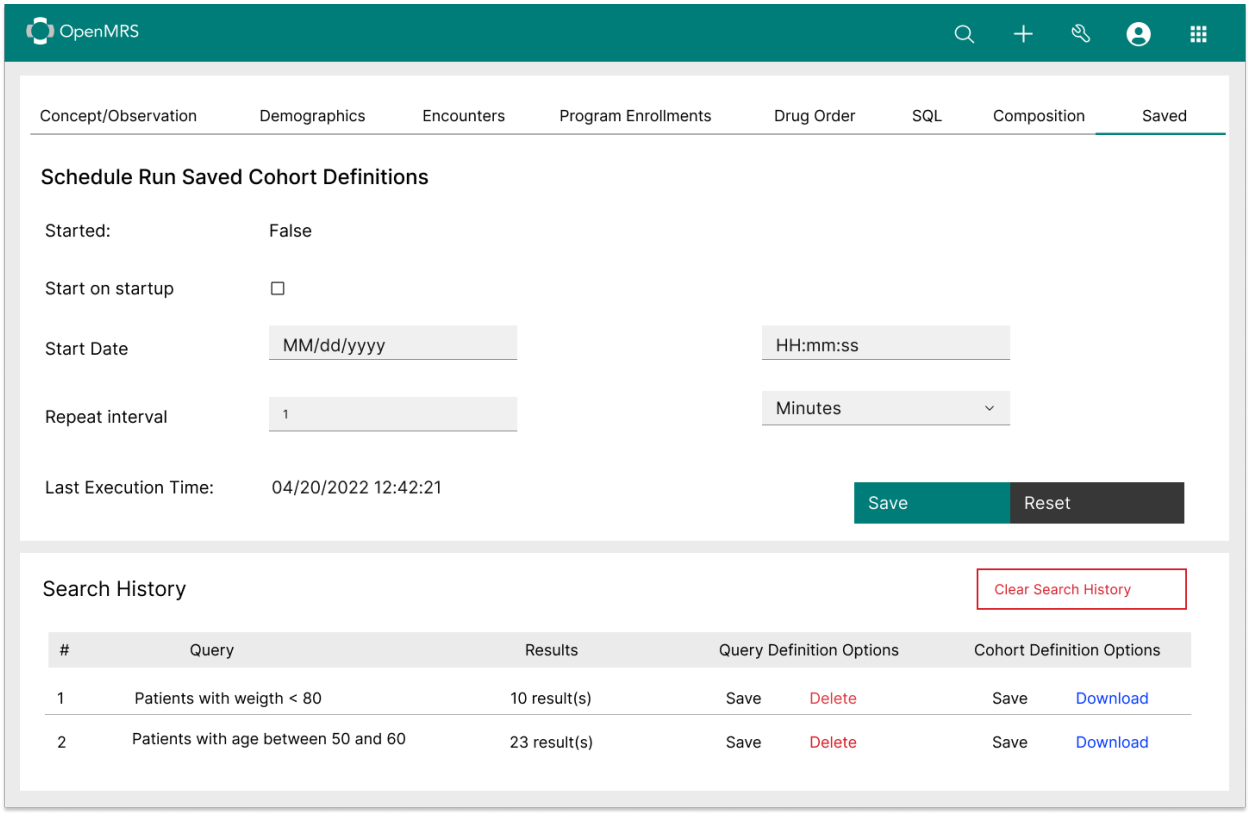


Figure 2.3.2.12.1. Proposed UI of Cohort Run Scheduler

2.3.3. Testing

Unit and integration testing will be done using Jest. E2E tests will be written in the [OpenMRS QA framework](#).

2.3.4. Chapter Summary

In this chapter the implementation of the solution was described using the different views alongside the inputs, data types and relevant APIs.

2.4. Risks associated with the project

The following table contains the identified risks, the likelihood and impact of each risk, and the mitigation plans.

Risk	Likelihood	Impact	Mitigation plan
Technical Difficulties	Low	Medium	- Planned to get the help from the community when the technical difficulties are met

Lack of time	Medium	High	- Planned small sprints to deliver a working product at the end of each sprint - Improvements will be considered towards the end of the project

2.5. Timeline

Period	Duration	Deliverable	Description
21th May - 28th May	1 week	Setting up the project	Finalize the GUI Mockups, APIs, Read documentation
29th May - 9th June	1.5 week	FR-01	Initial cohort builder UI with search by concept component
10th June - 19th June	1 week	FR-09	Creation of the Search History component
20th June - 29th June	1 week	FR-02	Implementation of the search by demographics component
30th June - 8th July	1 week	FR-03	Implementation of the search by encounters component
9th July - 17th July	1 week	FR-04	Implementation of the search by program enrollments component
18th July - 28th July	1.5 week	FR-05	Implementation of the search by drug order component

29th July - 6th August	1 week	FR-06	Create the Search by SQL UI and the REST API
7th August - 24th August	2 week	FR-08	Create the cohort run scheduler and create the required APIs
25rd August - 5th August	1.5 week	FR-10	Implementation of the saved component
6th September - 12th September	0.8 weeks	Testing the Implementation for the maintenance	Test the UI implementation to identify the bugs and fix them

Further Enhancements & Ideas

Create dedicated REST APIs for querying data that currently use the Ad Hoc because Adhoc API is hard to maintain and not developer-friendly.

2.6. Summary

This proposal provided the overview of the GSoC project Redo Legacy UI Cohort Builder. It began by describing the motivation behind selecting the project. Then the existing solutions were described followed by the problems of the existing solutions. Thereafter it described the functional and non-functional requirements of the project. After describing the design of the proposed solution implementation was presented. In the end, the timeline and the further enhancements were presented.

2.7. References

OpenMRS WIki.[2022] OWA User Guide [Online]. Available at: https://wiki.openmrs.org/display/docs/Cohort+Builder+Open+Web+App+User+Guide [Accessed 16 April 2022]

OpenMRS Guide.[2022] Cohort Builder [Online]. Available at: https://guide.openmrs.org/en/Using%20Data/cohort-builder.html [Accessed 16 April 2022]

OpenMRS WIki.[2022] Cohort Builder Features [Online]. Available at:
https://wiki.openmrs.org/display/projects/Expanded+Cohort+Features
[Accessed 16 April 2022]

OpenMRS Rest API.[2022] API Doc [Online]. Available at:
https://rest.openmrs.org/
[Accessed 16 April 2022]

OpenMRS Github [2022] Reporting Compatibility Module [Online]. Available at:
https://github.com/openmrs/openmrs-module-reportingcompatibility
[Accessed 17 April 2022]

OpenMRS Github [2022] Reporting Rest [Online]. Available at:
https://github.com/openmrs/openmrs-module-reportingrest
[Accessed 17 April 2022]

OpenMRS Github [2022] Webservices Rest [Online]. Available at:
https://github.com/openmrs/openmrs-module-webservices.rest
[Accessed 18 April 2022]