
Team 27 Website Documentation

Leo McArdle, Chris Rios, Daniel Min

Apr 20, 2019

Contents:

1	Requirements	1
1.1	Background	1
1.2	Gathering Requirements	1
1.3	Personas	1
1.4	Essential Features - Must Haves	2
1.5	Possible Features in the Future - Could Haves	2
1.6	Use Cases	2
2	Research	5
2.1	openEHR	5
2.2	Existing solutions	5
2.3	Technologies	6
2.4	Summary of Final Decision	7
2.5	References	7
3	HCI - Human Computer Interaction	9
3.1	Design Principles	9
3.2	Initial Sketches	9
3.3	Personas Created	10
3.4	Interactive Wireframe	12
3.5	References	14
4	Design	15
4.1	System Architecture	15
4.2	Page Flow Diagram	17
4.3	Design Patterns	17
4.4	CDR Query Library	18
4.5	Electron App	19
4.6	Implementation of Key Functionalities in Deliverable Version	20
5	Testing	21
5.1	Testing Strategy	21
5.2	Unit and Integration Testing	21
5.3	Automated Testing	22
5.4	User Acceptance Testing	23
5.5	References	23

6	Evaluation	25
6.1	Summary of Achievements	26
6.2	Critical Evaluation of Project	27
6.3	Future Work	28
7	Management	29
7.1	Legal Issues	29
7.2	User Manual	29
7.3	Deployment Manual	32
7.4	Gantt Chart	33
8	Glossary	35
8.1	openEHR	35
8.2	EHR	35
8.3	CDR	35
8.4	AQL	35
8.5	Archetype	35
8.6	Templates	35
8.7	CORS	36
8.8	Federate	36
8.9	Zoom	36
9	Abstract	37
10	Key Features	39
11	Demonstration Video	41
12	Team Members	43

1.1 Background

openEHR - pronounced 'open air' - is an open specification for storing patient data used by clinicians and developers working in clinical contexts. However clinical data are often messy and mixed up, and each clinical application had its own models, so moving data was very hard. Everyone was building their models over and over again which kept causing problems. A new standard api to send models to but abstract data away from the database layer was needed. Furthermore, this meant that only one CDR could be queried at once, which slowed down the work of the developers in the field.

1.2 Gathering Requirements

We were able to gather the requirements from the users through direct semi-structured interviews with both our users and our client by using Zoom.

1.3 Personas

The system will be used by developers who are working within the clinical field. We therefore created two personas - an experienced developer who has been working in the clinical field for a long time, and a developer who has recently transferred to the clinical field. They can be found in more detail in HCI.

1.4 Essential Features - Must Haves

Requirements	Type
Register CDR environment variables*	Functional
Support different CDR authentication requirements*	Functional
Execute AQL on multiple CDRs and display the result	Functional
Federate the result of AQL statement from multiple CDRs	Functional
Check consistency of result of AQL statement from multiple CDRs	Functional
Commit composition to multiple CDRs	Functional
Easy-to-use GUI for new users	Non-Functional
System must run smoothly without any noticeable lag	Non-Functional

- Register CDR environment variables
 - Possibly import postman environment files
- Support different CDR authentication requirements
 - Ethersis requires session tokens
 - Marand ThinkEHR supports Basic authentication

1.5 Possible Features in the Future - Could Haves

Requirements	Type
Upload a template to multiple CDRs	Functional
List templates from multiple CDRs	Functional
Visualise templates and archetypes - show data constraints	Functional
Generate AQL from interacting with template visualisation	Functional
Authentication to support multiple users	Non-Functional
Generation of AQL from interaction with templates	Functional

1.6 Use Cases

Use Case	Logging In (UC1)
Description	User enters login details and clicks login button
Primary Actor	User
Secondary Actor	System
Pre-condition	None
Main flow	<ol style="list-style-type: none">1. User opens the app2. User enters his username3. Clicks button to go to main page and display CDRs
Post-condition	None
Alternative Flow	None

Use Case	Add CDR (UC2)
Description	User adds a new CDR from addition page
Primary Actor	User
Secondary Actor	System
Pre-condition	User has logged on
Main flow	<ol style="list-style-type: none"> 1. User enters add CDR page 2. User enters CDR details 3. Clicks button to add the CDR 4. CDR saved to the config file and a confirmation window pops up 5. Change is reflected in the main page
Post-condition	None
Alternative Flow	None

Use Case	Remove CDR (UC3)
Description	User removes a CDR from list by clicking the bin button
Primary Actor	User
Secondary Actor	System
Pre-condition	User has logged on and one or more CDRs exist in the list
Main flow	<ol style="list-style-type: none"> 1. User clicks on bin button next to CDR in list 2. The CDR is removed from the configuration file 3. Change is reflected in the main page
Post-condition	None
Alternative Flow	None

Use Case	Query AQL (UC4)
Description	User inputs an AQL query and clicks on the query button
Primary Actor	User
Secondary Actor	System
Pre-condition	User has selected CDRs they wish to query from the list
Main flow	<ol style="list-style-type: none"> 1. User inputs AQL query into text box 2. User clicks on the query button 3. System processes and interprets the query 4. Query result from the selected CDR(s) is returned as a JSON tree
Post-condition	None
Alternative Flow	Error: 400 Bad Request (i.e. Incorrect AQL query) (UC4.1)
Description	AQL query was incorrect and system has failed to process the query
Primary Actor	User
Secondary Actor	System
Pre-condition	User has selected CDRs they wish the query from the list
Main flow	<ol style="list-style-type: none"> 1. User inputs AQL query into text box 2. User clicks on the query button 3. System processes and interprets the query 4. System is unable to interpret the query as it is incorrect 5. Error: 400 Bad Request is shown in the results box

Use Case	Create JSON Table (UC5)
Description	User creates a table of results from the given JSON tree
Primary Actor	User
Secondary Actor	System
Pre-condition	User and system has successfully completed a query
Main flow	<ol style="list-style-type: none"> 1. User clicks on Create Table from JSON button 2. System creates a table from the JSON tree 3. The tree is displayed on a pop-up window
Post-condition	None
Alternative Flow	None

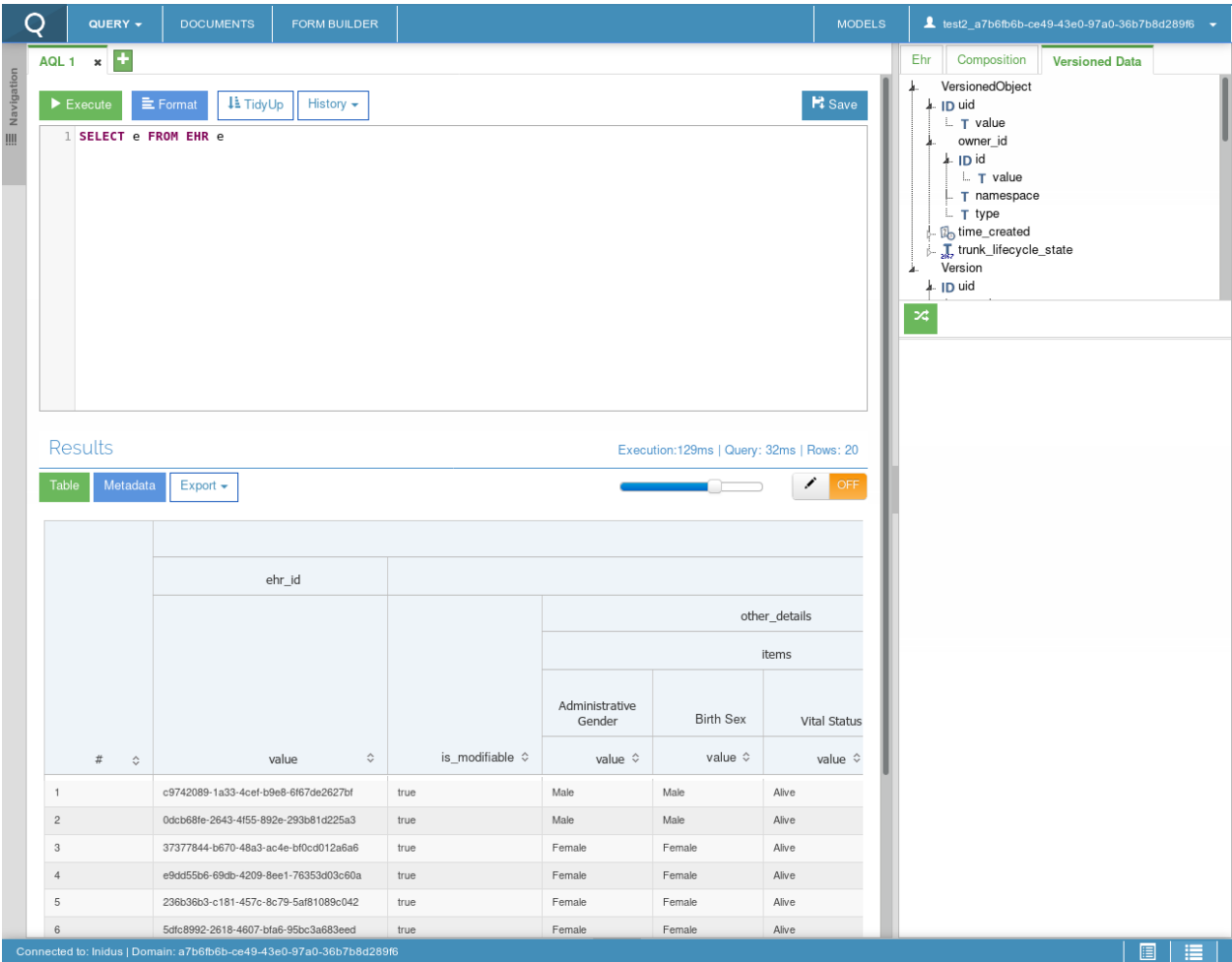
2.1 openEHR

openEHR is a technology for e-health, consisting of open specifications, clinical models and software that can be used to create standards, and build information and interoperability solutions for healthcare.[1]

Much of our research centred around understanding the openEHR ecosystem. Our client was an invaluable resource in this, taking the time to explain its concepts and elements to us. We also did some research of our own, from information available online[1][2].

2.2 Existing solutions

Our client gave us access to a similar proprietary tool to the one we'd be building, Think!EHR Explorer.



We tested the tool to get a feel what what potential users would be used to. Naturally, our aim was to go beyond what this tool did, and make a tool capable of querying multiple CDRs concurrently.

2.3 Technologies

2.3.1 Languages

We settled upon JavaScript since all members of our team had used it in the past or were familiar with it, and our client had recommended doing it this way.

We chose against making a browser-based webapp because CORS (Cross-Origin Resource Sharing) could have posed a problem with some CDRs not being configured correctly to be queried from a browser. So instead we decided to use Electron[3] to create a desktop application where this would not be a problem. Creating a desktop application also means that we can create executable files which can be easily distributed to users.

We have also decided to incorporate elements of modern JavaScript such as ES6 modules, Promises and the Fetch API, to build our project in a forward-thinking way and produce a future-proof solution. While some of these features aren't implemented in Node.js today, it is very likely that they will be in the very near future[4]. However for now, we can convert code written using them into JavaScript which current versions of Node.js can run using tools like Babel[5].

2.3.2 Test Frameworks

While researching on modern, up-to-standard and effective ways of JavaScript unit testing, we found an article on the Internet by Welldone Software[6] that had clearly laid out various information about testing on JavaScript. We therefore took the advices found on the article and looked into using Jest for the openEHR Explorer.

Its documentation showed it to be a powerful testing framework, and when using it we found it very pleasant and straightforward.

For mocking HTTP requests and responses in our tests we found Nock[7]. Nock is particularly powerful because it allows one to run a real HTTP request in a test, save the response, and then use those saved values in tests going forward. This is especially useful given our need to test querying against a complex external API, where manually running the queries and copying the result over would be too tedious and error prone.

2.3.3 Documentation Generators

JSDoc[8] is the standard for annotating JavaScript code with documentation, and we decided on using documentation.js[9] to turn that into readable documentation given its ease of use with minimal configuration.

2.4 Summary of Final Decision

Electron will allow us to create a multi-platform application and allow us to focus on the development of the system by removing the chance of CORS causing a problem. A desktop application means that the system can also be distributed easily with executable files to the users.

We will be using a multitude of modern tools such as Nock, Jest, etc. to effectively test our code as much as possible.

JSDoc will provide clear annotations and documentation for any future programmers who wish to contribute to in the development of the openEHR Explorer.

All of the above come together to reach our goal - creating a smooth, forward-minded system that is able to match the needs of the users.

2.5 References

- [1] https://www.openehr.org/about/what_is_openehr
- [2] <https://specifications.openehr.org/>
 - <https://specifications.openehr.org/releases/QUERY/latest/AQL.html>
 - <https://openehr.github.io/specifications-ITS-REST/>
 - <https://www.ehrscape.com/api-explorer.html>
- [3] <https://electronjs.org/>
- [4] <https://medium.com/@giltayar/native-es-modules-in-nodejs-status-and-future-directions-part-i-ee5ea3001f71>
- [5] <https://hackernoon.com/7-different-ways-to-use-es-modules-today-fc552254ebf4>
- [6] Vitali Zaidman, “An Overview of JavaScript Testing in 2018”, 9th February 2018 [Online] <https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2018-f68950900bc3>
- [7] <https://github.com/nock/nock#readme>
- [8] <http://usejsdoc.org/>

- [9] <https://documentation.js.org/>

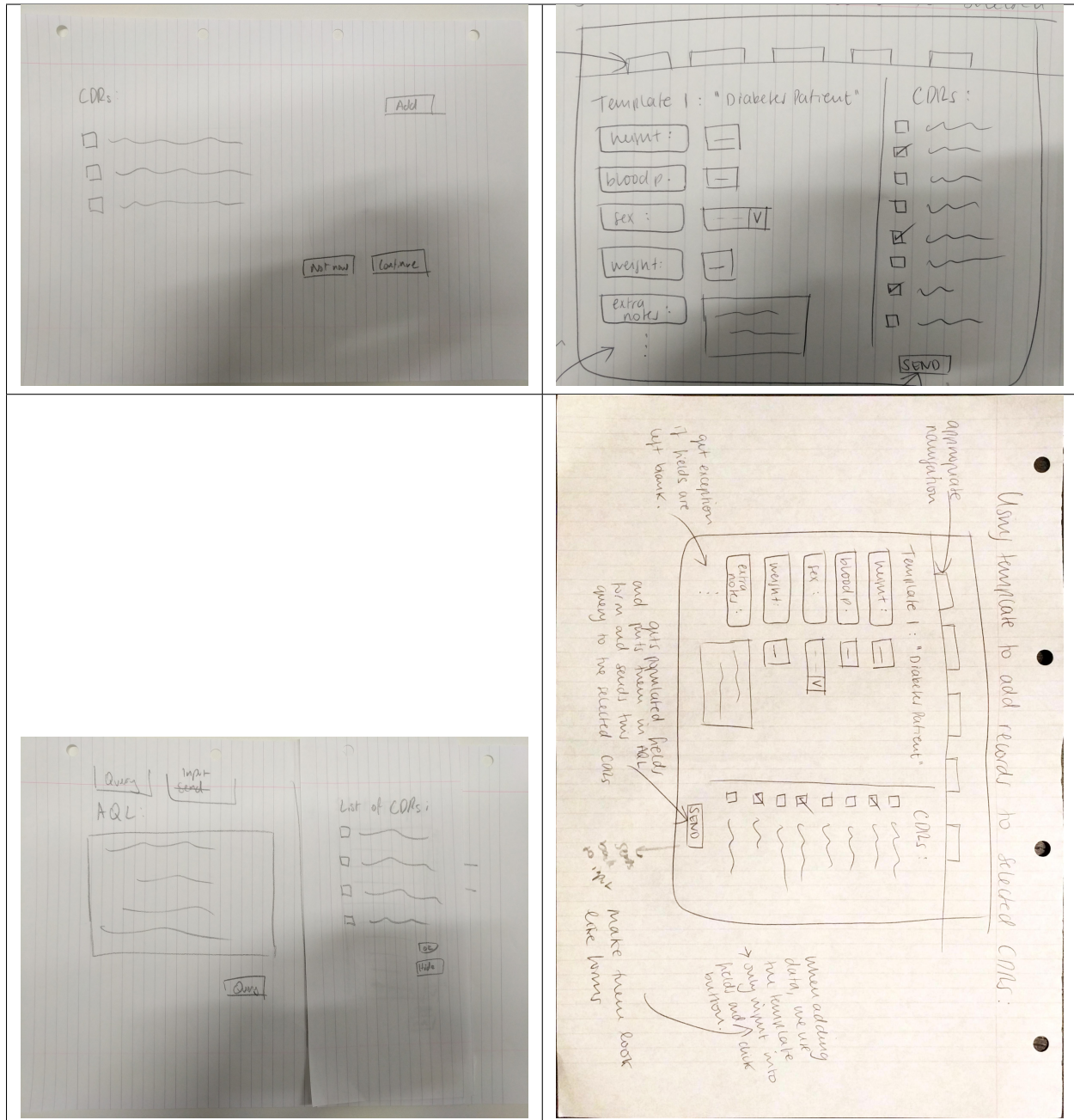
3.1 Design Principles

Ben Shneiderman outlines that there are eight design principles of HCI[1]. The team has decided to adopt some of the principles of Shneiderman that we believed to be particularly important. This section will explore how the principles affected the team's implementation of the HCI:

- Consistency
- Our system has a universal theme/layout which prevents from causing confusion for a first time user.
- Shortcuts for frequent users
- While a simple and consistent design provides advantages for new users, frequent users may find this to be repetitive in certain parts of the system. We have therefore minimised screen movement in our system to allow the user to carry out similar tasks in quick succession.
- Easy reversal of actions
- It is always possible that even an experienced user will carry out an incorrect action, and being able to easily revert any changes in the system will provide relief for the user - for example in openEHR Explorer, a wrong CDR might be added; we have therefore allowed the user to easily remove any CDRs that is unnecessary/incorrect.
- Reduce short-term memory load
- 'limitation of human information processing in short-term memory requires that displays be kept simple' - the team has followed Shneiderman's statement and has reduced multiple page displays as much as possible, condensing the crucial functions of the system to be contained within a single page.

3.2 Initial Sketches

After we had gathered the requirements from the client and the users, we started sketching possible solutions prior to the creation of the first iteration of a prototype individually to increase the chances of each member devising a unique solution. Once the sketches were completed by all three members, we compared the results to come up with a version that we used to create the first iteration of the prototype.

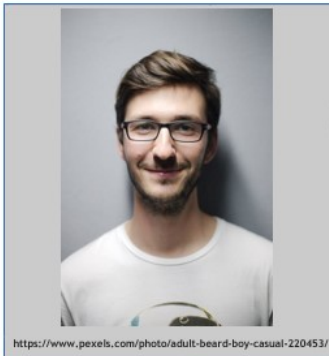


3.3 Personas Created

The team created two personas. The personas helped the team to come up with a HCI that would be enjoyed by a range of users.

- The first persona is of an experienced developer

Leonard Smith



“It would be much easier if templates could be uploaded to different CDRs at the same time”

Leonard is an experienced developer who works for NHS Digital. He finds it tedious to have to send queries to different Clinical Data Repositories as he has to use various platforms to do so, which makes the job hard. He enjoys querying in AQL and acts as a consultant for the rest of the team.

Name Leonard Smith
Type 30 year old
Role Developer

Needs

- Switching between different target CDRs.
- Lists of available openEHR artefacts.
- Displaying of results in different formats.

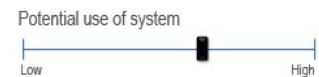
Would like to see

- Uploading templates to different CDRs at the same time.
- Manipulation of CDR made easier, including running AQL statements on them and getting results.
- Generation of GraphQL interfaces.
- Friendly User Interface to improve the flow of work and ease of use and learning.

Frustrations

- Inability to browse openEHR artefacts held in design-time repositories.
- Some CDRs vendors do not provide additional tooling, this is a setback to developers when it comes to building queries and browsing data repositories.

Behaviours



ThoughtWorks®

- The second persona is of a developer who is not yet familiar with the medical industry.

John Acton-Wright



“It can be hard trying to work with the current solution as I’m still getting used to unfamiliar terms. I hope a new solution can fix this.”

John is a young developer who has recently started working for NHS Digital. While he is comfortable with general aspects of development, he has yet to familiarise himself with openEHR(Electronic Health Record), CDR(Clinical Data Repositories) and AQL. A system that even new users can use without difficulty would be greatly appreciated by John.

Image reference: <https://hackemoon.com/11-things-developers-love-hearing-from-non-developer-co-workers-ea94805cf05d>

Name John Acton-Wright
Type Developer
Role Potential User

Needs

- A concise interface where AQL would be input.
- Clarity of the purpose of different parts of the system – i.e. between selecting a CDR and selecting a template.
- Displaying AQL results in different formats.

Would like to see

- Usability for those unfamiliar with medical terms.
- Generation of GraphQL results.
- Possibly a documentation of the proposed solution that could help future developers who are not experienced in the given industry.

Frustrations

- Lack of explanation of medical terms in existing solution.
- Difficulty of navigation of different parts of the existing solution.

Behaviours



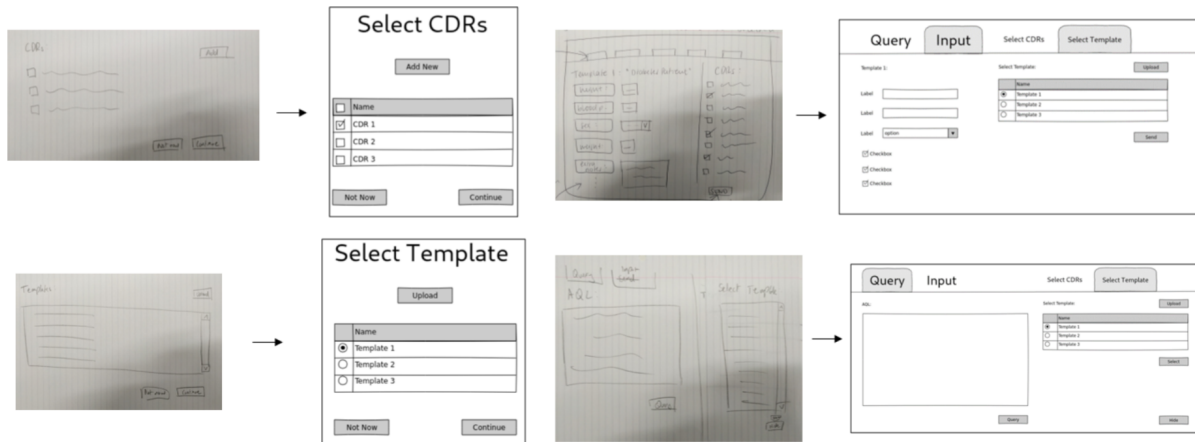
ThoughtWorks®

3.4 Interactive Wireframe

https://projects.invisionapp.com/share/V9OIKT03CQS#/screens/325133873_Select_Cdr

3.4.1 First Iteration

From the gathered requirements and the sketches produced, we were able to create our first iteration of a prototype:



Feedback

After we had completed the first prototype of the HCI, we allowed our potential users to experiment with it and give feedback on the first version.

COMMENT MODE

Select CDRs

Add New

<input type="checkbox"/>	Name
<input checked="" type="checkbox"/>	CDR 1
<input type="checkbox"/>	CDR 2
<input type="checkbox"/>	CDR 3

Not Now **Continue**

☐ Mark as resolved

IM **Ian McNicoll** 3 months ago

Suggest an import option to allow a postman environment file to be dropped in.

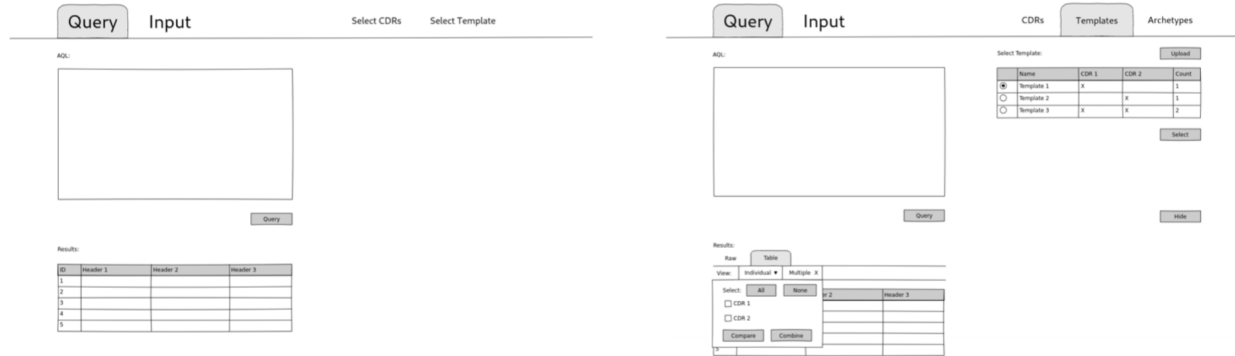
Add a new comment

Then, the group once again drew further sketches on a section of the solution that we had previously missed out - manipulation and the visualisation of the results data.

3.4.2 Second Iteration

Using the feedback and the new set of sketches, we were able to build upon the first prototype to expand its features.

First prototype: → Second prototype:

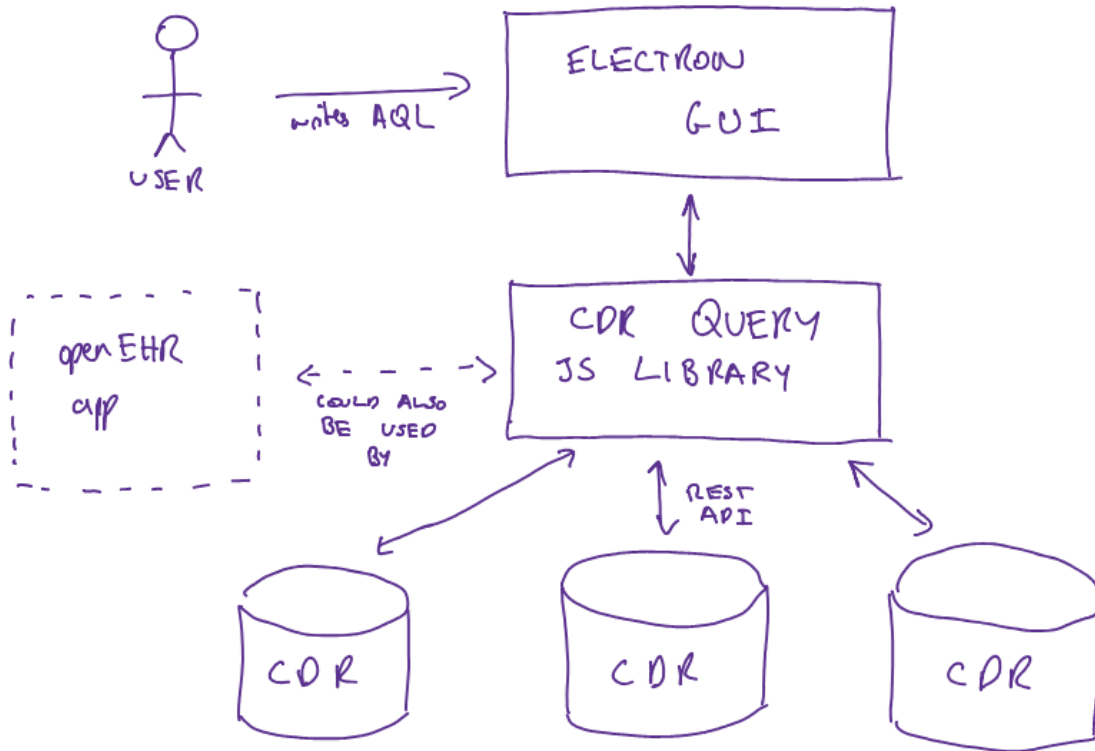


3.5 References

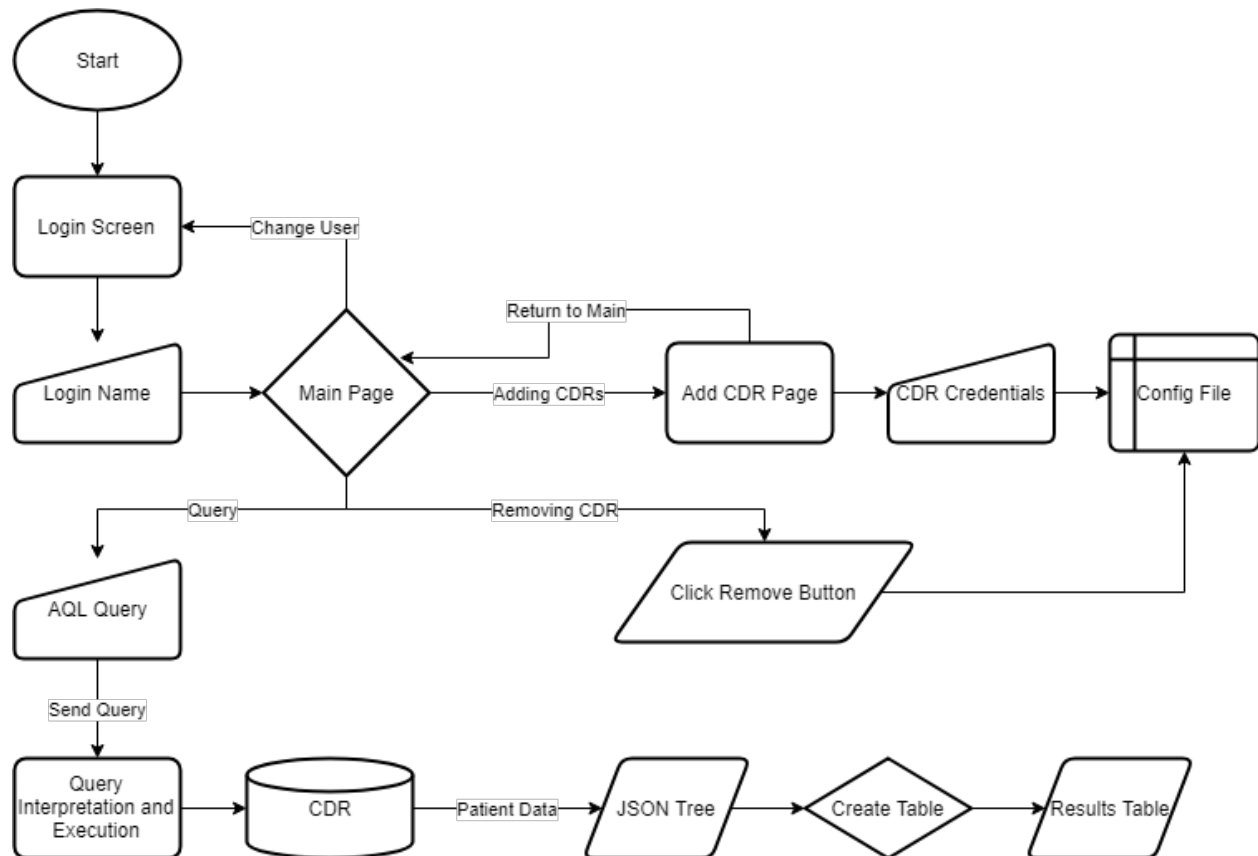
- [1] Ben Shneiderman, 'Shneiderman's "Eight Golden Rules of Interface Design"', 12th September 2013 [Online] <https://www.designprinciplesftw.com/collections/shneidermans-eight-golden-rules-of-interface-design>

4.1 System Architecture

We decided to separate out the project into two components to make developing them concurrently easier, to make testing easier in isolation, and finish the project with deliverables which are maximally useful to the openEHR ecosystem.



4.2 Page Flow Diagram



4.3 Design Patterns

4.3.1 Iterators

Iteration solves the problem of accessing and traversing objects without exposing the data structure and representation.

In openEHR Explorer, an iterator is used to traverse through the list of CDRs for actions such as displaying the list itself.

4.3.2 Asynchronous Methods

Asynchronous methods solve the problems that could be caused by potentially long-running programs by allowing the next method in the thread of execution to be invoked.

Promises allows for asynchronous programming to be employed in JavaScript and has therefore been extensively used for the back-end of openEHR Explorer.

4.3.3 Test-Driven Development

Test-Driven Development (also commonly referred to as TDD) is an increasingly important aspect of programming. TDD ensures that there is a very high coverage of testing of the code as each functionality has a corresponding test.

The team has followed the TDD methodology to ensure the back-end of the code has been thoroughly tested.

4.3.4 Factory

A factory is a that forms the basis of many software design patterns. A factory is an object that is used to create other objects - a factory method is a subroutine that returns an object of varying class.

Factories are mainly used in openEHR Explorer to create the CDR objects.

4.3.5 Balking

Balking refers to the execution of an action when an object is in a particular state, and the method will not return anything when the object is in an inappropriate state. For example, if an object needed to access a method in a file but it has been zipped, an exception would be thrown/the object would 'balk' at the request.

The team has taken a slightly different approach to balking - we were unhappy with the idea of 'no return' defined by balking, so instead we have decided to ensure that no balking would happen. For example, to avoid AQL queries from being executed on unwanted CDRs, openEHR Explorer will deselect all CDRs by default upon startup.

4.3.6 Proxy

A proxy is an object serving as an interface to another class. This provides a controlled access to a particular object, and could additionally provide functionalities when accessing the object.

In openEHR Explorer, all CDRs that have been saved by the users are stored on a configuration file - constantly reading this file could be very resource-intensive if the file increases in size significantly. Therefore, a proxy has been used to provide access to the stored values without having to load the configuration file over and over again.

4.4 CDR Query Library

[Initial prototype source code.](#)

To handle retrieving data from CDRs, federating it, and committing new data to CDRs. In this initial prototype all that the library is able to do is send an AQL query to one or multiple CDRs, and concatenate their results.

However it's been built with modern development practices, which makes extending this functionality incredibly simple:

- We followed the test-driven development methodology, so the code is thoroughly tested:

```

456 $ npm test
457
458 > openehr-cdr-query@0.0.3 test /home/travis/build/ucl-openehr-explorer/openehr-cdr-query
459 > node_modules/standard/bin/cmd.js && ./node_modules/jest/bin/jest.js
460
461 PASS tests/index.test.js
462   querying CDR
463     ✓ successfully (27ms)
464     ✓ returns no results (4ms)
465     ✓ with malformed AQL (11ms)
466     ✓ with invalid authentication (34ms)
467   querying multiple CDRs
468     ✓ successfully (9ms)
469
470 Test Suites: 1 passed, 1 total
471 Tests:      5 passed, 5 total
472 Snapshots:  0 total
473 Time:       1.61s
474 Ran all test suites.
475 The command "npm test" exited with 0.

```

- We used JSDoc and documentation.js to generate extensive API documentation for the code:

CDR
A connection to a single Clinical Data Repository

Parameters

- **config** **Object** CDR configuration
 - **config.url** **String** API URL (without a trailing /)
 - **config.authentication** **Object** Authentication configuration
 - **config.authentication.type** **"basic"** Type of authentication (currently only supports Basic)
 - **config.authentication.username** **String** Username for authentication
 - **config.authentication.password** **String** Password for authentication

query
Runs an AQL query against the CDR

Parameters

- **aql** **String** The AQL query to run

Returns **Promise<Object>** A promise resolving with the parsed JSON result of the query from the CDR, or rejecting with a **CDRError**

- We used the latest and greatest additions to JavaScript (ES6 modules, Promises, the Fetch API, among others), utilising Babel to make them backwards compatible with older JavaScript engines and versions of Node.js

By building this library in isolation from the Electron app, we give developers in the openEHR ecosystem the option of incorporating this library into their own projects.

While in its prototype phase it only runs in Node.js, with only a little work it could be made to work in the browser too.

4.5 Electron App

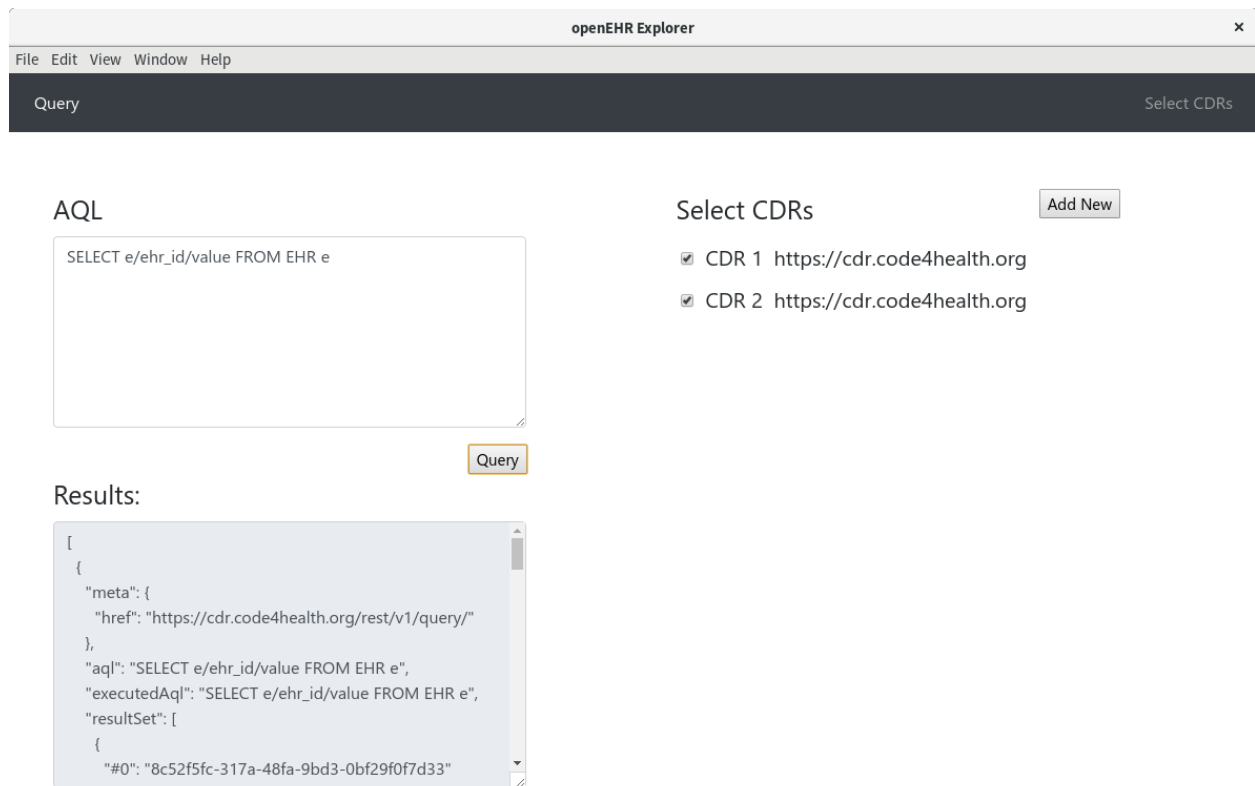
4.5.1 Data Storage

We have decided to use JSON to store the credentials of the CDRS as it is human readable and is widely used as a configuration file across the industry.

4.5.2 Initial Prototype

Initial prototype source code.

The Electron app utilises the CDR Query Library to provide a GUI which users can use to query CDRs.



In this initial prototype, users can add CDRs, send an AQL query to a subset of them, and get the raw data back.

4.6 Implementation of Key Functionalities in Deliverable Version

- **Login system for users with different lists of CDRs**
 - Different users will have their own lists of CDRs they wish to query - openEHR Explorer can accommodate this problem.
- **AQL querying to multiple CDRs**
 - The existing solution only allows the user to query a single CDR - openEHR Explorer allows querying of multiple CDRs at once.
- **Return results from multiple CDRs in one federated JSON tree**
 - The client and users have expressed that they would like to see all the results combined into one.
- **Creating a table from the JSON tree for greater readability**
 - While JSON trees are human-readable, a table can be used with ease by users with little previous experience.
- **Adding CDRs from multiple vendors with different authentication systems**
 - Further strengthens the ability to query multiple CDRs at once.
- **Removing saved CDRs from list**
 - User can easily remove any unused CDRs or CDRs with incorrect credentials from the list, preventing cluttering.

5.1 Testing Strategy

From the beginning of the development of the back-end, we have followed the TDD methodology to ensure high coverage of the code. This will also help us create unit and integration tests.

We have used Jest - a testing framework developed and maintained by Facebook[1]. It is open-source and has become increasingly popular over the last few years. Jest's primary advantage is that it works 'out of the box' without any configuration and it is very simple to use.

Nock has been used to simulate/mock HTTP requests and responses as mentioned in Research.

5.2 Unit and Integration Testing

The TDD methodology has meant that thorough unit and integrations have been written during development. The integration tests have been created as a composition of multiple unit tests that we seemed belonged together as a part of a group. For example, unit tests that are querying CDRs have been grouped and can be performed in a series - below shows a snippet of the test of openEHR Explorer:

```
describe('querying CDR', () => {
  test('successfully', async () => {
    awaitnockBack('query_success.json')
    const res = cdr.query('SELECT TOP 1 e/ehr_id/value AS id FROM EHR e')
    await expect(res).resolves.toEqual({
      'meta': {
        'href': 'https://cdr.code4health.org/rest/v1/query/'
      },
      'aql': 'SELECT TOP 1 e/ehr_id/value AS id FROM EHR e',
      'executedAql': 'SELECT TOP 1 e/ehr_id/value AS id FROM EHR e',
      'resultSet': [
        {
          'id': '8c52f5fc-317a-48fa-9bd3-0bf29f0f7d33'
        }
      ]
    })
  })
})

test('returns no results', async () => {
  awaitnockBack('query_none.json')
  const res = cdr.query('SELECT TOP 0 e FROM EHR e')
  await expect(res).resolves.toEqual({})
})
```

Below shows an example of the above tests being run in a sequence.

```
abcalx1@DESKTOP-CFM002H:/mnt/c/Users/Daniel Min/Documents/Projects/openehr-cdr-query$ npm test
> openehr-cdr-query@0.0.3 test /mnt/c/Users/Daniel Min/Documents/Projects/openehr-cdr-query
> node_modules/standard/bin/cmd.js && ./node_modules/jest/bin/jest.js

PASS tests/index.test.js
  querying CDR
    ✓ successfully (27ms)
    ✓ returns no results (3ms)
    ✓ with malformed AQL (4ms)
    ✓ with invalid authentication (3ms)
```

5.3 Automated Testing

Naturally, following the TDD methodology has given way to test automation. They can be run quickly through Jest and can be repeated many time without difficulty.

5.4 User Acceptance Testing

User acceptance testing is especially important as a system which the user and the client are not satisfied with will see very little use even if the system has thorough functionality testing. We have therefore given great attention to constantly receive feedback from our expected users and client.

Our client is distinct from other usual clients in that the client will also be a user. Therefore our client was able to provide views of openEHR Explorer both as a client and an user which is invaluable. We believe this has also significantly reduced the time taken of the user acceptance testing.

Each week we updated our Teaching Assistant in laboratory sessions and our client and users through a combination of slack. The continuous stream of feedback we were able to received contributed greatly in building our system to meet with their requirements and create a software that is up to expectations. Positive and negative feedback was given each time along with advices which the team took into consideration whenever an element was being edited.

Feedbacks have been mostly positive - the GUI and the querying has both been approved by the users. However, there are parts in openEHR Explorer that are less developed and has been mention and talked with the client. These will be mentioned in Evaluation.

5.5 References

1. <https://jestjs.io/>

6.1 Summary of Achievements

6.1.1 Achievements Table

Table

ID	Requirements	Priority	State	Contributors
1	Register CDR environment variables	Must	✓	Christian
2	Support basic authentication on CDRs	Must	✓	Leo
3	Support different CDR authentication requirements	Must	✓	Leo
4	Execute AQL on multiple CDRs	Must	✓	Leo, Christian
5	Display results	Must	✓	Christian
6	Federate Results from multiple CDRs	Must	✓	Leo
7	Check consistency of result of AQL statement from multiple CDRs	Must	✓	Leo, Christian
8	Easy-to-use GUI	Must	✓	Christian, Daniel
9	System runs smoothly without lags	Must	✓	Christian, Daniel
10	Upload template to CDRs	Should	✖	Leo, Christian, Daniel
11	List templates from CDRs	Could	✖	Leo, Christian, Daniel
12	Visualise templates and archetypes	Could	✖	Christian, Daniel
13	Authentication to support multiple users	Could	✓	Daniel
14	Generation of AQL queries from template	Could	✖	Leo, Daniel

6.1.2 Contribution Table

Work Packages	Leo	Christian	Daniel
Client Liaison	40%	40%	20%
Requirements Analysis	34%	33%	33%
Research	33%	34%	33%
UI Design	30%	35%	35%
Programming	40%	40%	20%
Testing	100%	0%	0%
Bi-weekly Reports	33%	33%	34%
Project Website	20%	5%	75%
Poster Design	0%	50%	50%
Video Editing	0%	50%	50%
Overall Contribution	20%	40%	40%
Main Roles	Back-end, Tester	Front-end, Research	Editor, Front-end

6.2 Critical Evaluation of Project

6.2.1 User Experience

Our foremost focus was user experience. openEHR Explorer was created with inexperienced developers in mind who are likely to not yet be comfortable in the clinical field. We strongly believe that the GUI is very user-friendly and users of any skill level will not find openEHR Explorer difficult to fully use.

6.2.2 Functionality

While we have achieved all of the essential features regarding the management and the querying of CDRs in general, we could not complete the more complicated tasks specifically related to templates and archetypes.

6.2.3 Stability

The application itself functions without problems even if one or more CDRs in the list is unaccessible or down. The UI is robust and is highly unlikely that a user will be able to directly cause problems that will affect the stability of the system.

We have used Electron to develop the application - this ensures that it can be used cross-platform. An executable file can be created to easily distribute the software whenever necessary.

6.2.4 Efficiency

The simple-yet-straightforward UI means that the user will have a clear idea of what needs to be done and how to carry out the actions that he desires to do. In extreme situations, a developer could change the application's file system to be executed asynchronously if the need rises for reasons such as an exceptionally long configuration file of CDRs.

6.2.5 Maintainability

TDD ensures that as much of the testable code is covered by unit tests, reducing the chance of errors and exceptions rising in the future when the codebase is likely to have expanded.

Comments have been written within the code in parts where we believe it could cause confusion for any future developers who are looking at the code.

Version control has been used since the beginning of the project and versions can be tracked back. Branches were used in situations where more than one person contributed to the code to minimise conflicts when developing concurrently.

The back-end was developed independently from the front-end so that future developers would be able to incorporate their own libraries or user interface from or into the current openEHR Explorer ecosystem.

6.2.6 Project Management

We had clearly communicated with each other from the very beginning to know each member's strengths and weaknesses. Tasks were allocated so that all the members did not feel they were unjustly overpowered.

Over time, we had formed strong friendships with each other and the team was able to bond exceptionally well. When Leo McArdle had to leave due to personal health considerations, the remaining members - albeit devastated - did their utmost best to fill the gap and carry out any tasks yet undone without lowering the standard.

6.3 Future Work

The team recognises that openEHR Explorer has yet to reach its full potential and that there are features that would be greatly valued but are yet to be implemented into openEHR Explorer.

The missing aspect of openEHR Explorer with the greatest significance is the lack of features regarding templates - from uploading templates into CDRs to the listing templates from multiple CDRs. The client had mentioned briefly that the ultimate goal eventually for openEHR Explorer is to be able to create AQL queries by interacting with the templates belonging to CDRs as AQL is not a language that is widely used and is quite specialised - new developers in the clinical field often struggle. This means that the system will need to have features regarding templates in order to reach its future goal. However, the client has also noted that visualisation of templates is a very difficult task to manage and that it is not absolutely necessary as users will eventually become comfortable with AQL queries.

Nevertheless we were not able to implement any features regarding templates in the given timeframe, but we believe that if further time was given, it would be possible to integrate template functionalities into openEHR Explorer and bring the system closer to its ultimate goal of being able to build AQL queries by templates which would greatly lower the difficulty of querying CDRs.

Other possible future implementations include: - Finer authentication system - More robust user identification system - Friendlier method of sorting and selecting CDRs to query - Deleting multiple CDRs at once - Allowing user to alter the layout of openEHR Explorer - More extensive use of the top navigation bar - Export the generated table to a saveable file

7.1 Legal Issues

Download the Legal Issues Document [here](#).

7.2 User Manual

7.2.1 Logging in

When the application is opened, you will be greeted with the login screen:

Log In

Username

Continue

Enter any name you wish (Case sensitive, Whitespace sensitive) and click the 'Continue' button.

You will then be transferred to the main page:

[Query](#) [Change User](#) [Select CDRs](#)

AQL

Write your AQL code here


Query


Results:

Create Table

Select CDRs

Add New

 ☐ c4h_nutshell <https://cdr.code4health.org>

 ☐ ichirf <https://cdr.ichirf.inidus.cloud>

7.2.2 Manage CDRs

After logging in, you can add a new CDR to the list clicking the 'Add New' button, which will transfer you to the addition page:

Query Main

Add CDR

Name
URL
Username
Password

Continue

Fill in the details and press 'Continue'. This will add the CDR to the list and automatically save the changes to the configuration file, saving the list for use at another time. The text boxes will be emptied so allow you to add additional CDRs.

To return to the main page, click on the 'Main' button on the right of the top navigation bar.

CDRs can be removed from the list by clicking on the bin image next to the CDRs.

7.2.3 Querying

In the main page, you can query one or more CDRs from the list by clicking and selecting the corresponding boxes.

Enter AQL in the top left textbox and click the 'Query' button to execute the AQL. The results will be returned as a JSON tree, shown below:

Query Change User Select CDRs

AQL

COMPOSITION.problem_list.v1
contains (
EVALUATION.b_a[openEHR-EHR-
EVALUATION.problem_diagnosis.v1] or
CLUSTER.b_b[openEHR-EHR-
CLUSTER.problem_status.v0])
where a/name/value='Problem list'

Query

Select CDRs

Add New

☒ c4h_nutshell https://cdr.code4health.org
☐ ichirf https://cdr.ichirf.inidus.cloud

Results:

```
[
  {
    "meta": {
      "href": "https://cdr.code4health.org/rest/v1/query/"
    },
    "aql": "select\n  e/ehr_id/value as ehrId,\n  e/ehr_status/subject/external_ref/id/value as subjectId,\n  e/ehr_status/subject/external_ref/namespace as\n  subjectNamespace,\n  a/composer/name as
```

Create Table

Clicking on the 'Create Table' button will generate a table from the JSON tree and display it on a new resizable window:

The screenshot shows a web application interface with a dark header bar containing 'Query' and 'Change User' on the left, and 'Select CDRs' on the right. The main area is divided into two panels. The left panel, titled 'AQL', contains a query editor with the following text:

```
COMPOSITION.problem_list.v1]
contains (
  EVALUATION_b_a[openEHR-EHR-
  EVALUATION.problem_diagnosis.v1] or
  CLUSTER_b_b[openEHR-EHR-
  CLUSTER.problem_status.v0])
where a/name/value='Problem list'
```

Below the editor is a 'Query' button. The right panel, titled 'Query Results', displays a table with the following columns: `composerId`, `Problem_Diagnosis_value`, `Body_site_code`, `compositionId`, `Problem_Diagnosis_code`, `subjectId`, and `entryId`. The table contains four rows of data:

0	composerId	Problem_Diagnosis_value	Body_site_code	compositionId	Problem_Diagnosis_code	subjectId	entryId
1	"G33567"	"Hypertension"	null	"f1e1fb56-4a38-4fb4-b8b4-7796fa78b20f:98aa716e-8bc6-40f6-a5a3-84518c4c60ef:1"	"38341003"	"99999999068"	"09334b-9c87-4e870b-e44d65c"
2	"G33567"	"Recurrent urinary tract infection"	null	"113a0793-b465-49f2-99d2-381e4803ecfc:98aa716e-8bc6-40f6-a5a3-84518c4c60ef:1"	"197927001"	"99999999068"	"3575b9-f09f-4ecaa49-b4bd9d"
3	"G33567"	"Dupuytren's contracture"	"85151006"	"28a1869b-4fe9-477c-8739-c97002c2e504:98aa716e-8bc6-40f6-a5a3-84518c4c60ef:1"	"274142002"	"99999999068"	"b05e02-5064-43a50e-8aac103"
4	"G33567"	"Migraine"	null	"b98d6e0d-4088-4960-b744-0a401c8372dc:98aa716e-8bc6-40f6-a5a3-84518c4c60ef:1"	"37796009"	"99999999068"	"dd595e-cbca-4b99b1-6e7a56c"

At the bottom of the 'Query Results' panel is a 'Create Table' button.

7.3 Deployment Manual

1. Clone repository from GitHub - [found here](#).
2. Go to the repository in command line (e.g. Windows - cmd, Unix - Bash, etc.)
3. Ensure npm is installed. To check, enter

```
npm --version
```

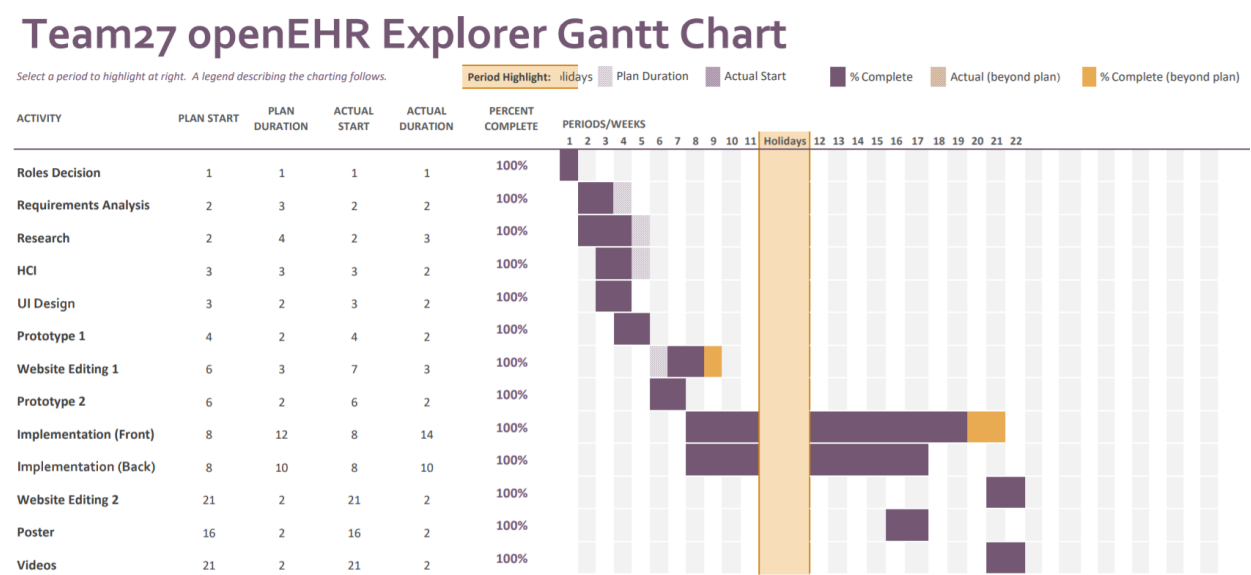
If not installed, download [NodeJS](#) here.

4. Run the following commands:

```
#install dependencies
npm install
#starts the app
npm start
```

For instructions in greater detail, please refer to the README.md in the GitHub repository.

7.4 Gantt Chart



Download: [Excel](#), [PDF](#).

8.1 openEHR

Open source standard for storing and handling EHRs. [Official Website](#), [What is openEHR?](#)

8.2 EHR

Electronic Health Record: A collection of digitalised information about a patient(s).

8.3 CDR

Clinical Data Repository: A place where patients' EHRs are stored.

8.4 AQL

Archetype Query Language: The query language used on CDRs.

8.5 Archetype

Little chunks of medical record. e.g. Medication orders, blood tests, etc.

8.6 Templates

Templates are a defined data set created by multiple archetypes as components. They are built independantly of CDRs.

8.7 CORS

Cross-Platform Resource Sharing: a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.

8.8 Federate

Combine data from multiple sources, in this context multiple CDRs. This is useful for the following requests(situations):

- “I don’t care what CDR this data lives in, I just want a list of blood pressures for this patient regardless of whether they were taken at the GP or hospital.”
- “Give me every patient with hypothyroidism across 12 London hospitals.”

8.9 Zoom

An online communications and conferencing software that uses cloud computing. Provides services such as video conferencing, online chats, mobile contributions, etc.

CHAPTER 9

Abstract

openEHR Explorer is an open-source application to query openEHR CDRs, targeted at developers working in a clinical context. Previously, developers did not have a standardised method of querying, and openEHR Explorer was created to solve the problem. openEHR Explorer is able to query multiple CDRs concurrently with a single AQL query and federate their results into a table.

CHAPTER 10

Key Features

- Login system to accomodate multiple users with different lists of CDRs.
- Supports authentication of CDRs from multiple vendors.
- One AQL to query one or more CDRs at once and federate results into a table.

CHAPTER 11

Demonstration Video

The video can be found at: <https://youtu.be/jmtJvnSaQUg>

CHAPTER 12

Team Members

- Leo McArdle - leo.mcardle.17@ucl.ac.uk

Team Leader, Back-end Developer, Client Liason.



- Christian Martin Rios - chris.rios.17@ucl.ac.uk

Front-end Developer, Deputy Leader, Research.



- Daniel Kyung-Hwan Min - daniel.min.17@ucl.ac.uk
Secondary Front-end Developer, Editor, UI Design.

