Tool Assessment

Application Integration

To send data over HTTP (i.e. communication between the client and the server), we can use either GraphQL or REST. Using either technologies for development ensures Separation of Concerns, where the client-side code will largely only concern itself with the View / UI. It does not need to concern itself with writing SQL statements or the logic for accessing the database. Furthermore, by having Separation of Concerns:

1. Development of user interface, the server and the data storage can be performed independently as they are loosely coupled to each other.
2. Debugging and spotting security flaws in our project becomes easier.

We have decided to use GraphQL over REST as REST faces the following issues:

1. Overfetching: Client downloads more information than is required.
2. Underfetching: A specific endpoint doesn’t provide enough required information. This means that the client has to make multiple calls to the server through various endpoints to retrieve the required information.
3. Slowing development: As we are not clear about the requirements / data to be displayed to the various users, we may need many iterations of user testing to verify that the data displayed to each user is as desired by the user. Because of the way REST works, both the UI & backend code needs to be updated to support these changes. This is not the case for GraphQL; only the UI needs to be updated.
4. Harder to pick up: Good API commented that "when time is the essence… when only one client that you control consumes your API, when you can’t afford to study REST or to learn HTTP in-depth or when you can’t hire someone with the expertise to help you, GraphQL might be the better way to go" simply because "True REST APIs are incredibly hard to pull off."
5. Our project requirement has the need of uploading and downloading files. GraphQL makes this much easier to program as REST is more suited for CRUD applications only.

Our project uses Spring Boot to create a web service using GraphQL because:

1. Creating a web service can be done quickly as many of the configurations are performed automatically by Spring Boot.
2. It's easy to connect and perform operations on a MySQL database.
3. Most of us are familiar with Java, and Spring works on Java platform.

Node.js is an alternative to Spring Boot. While Node.js has perks such as being easier to set up, we chose Spring Boot as:

1. Not many of us are familiar with JavaScript.
2. Both Java & Spring are more mature than JavaScript & Node.js. Therefore, there's many more high-quality third-party libraries available in Java, such as the k-anonymity library (see below).
3. Spring works better than Node.js on relational databases (MySQL is a relational database).
4. IDE support for Spring is better than that of Node.js. Since we aren't familiar with both Spring and Node.js, having stronger IDE support is definitely helpful in speeding up our development.

Front-end Technology

As discussed above, the client-side code will largely only concern itself with the View / UI. Since UI isn't an important aspect of this project, there's no need for us to use frameworks such as Bootstrap for front-end development. We will simply code in HTML, CSS and JavaScript.

Anonymising Data

Anonymising data makes it impossible to:

1. Single out an individual in the dataset.
2. Link records concerning the same individual.
3. Infer the value of one attribute based on other values.

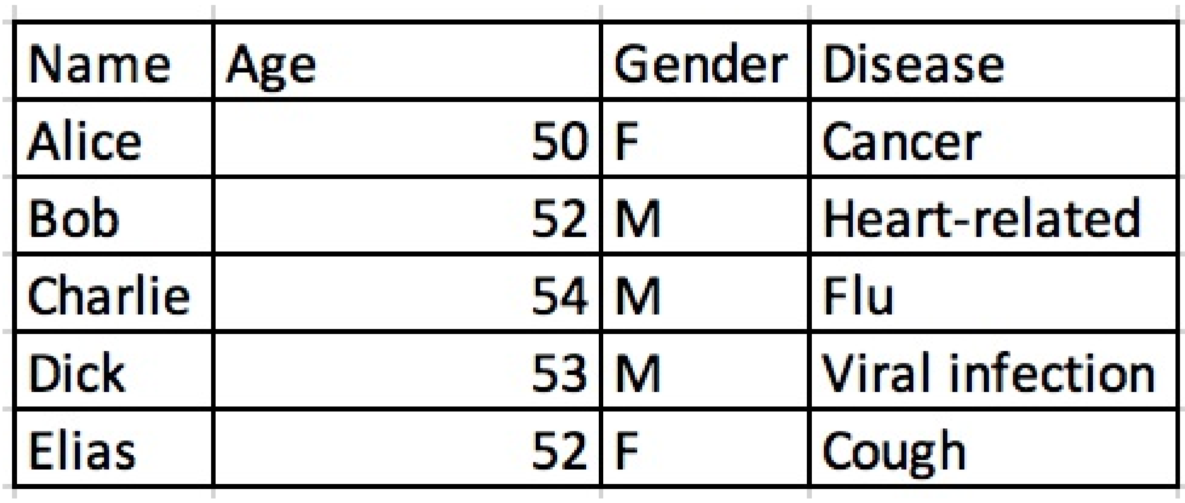
This allows for researchers to retrieve data for research purposes without violating privacy concerns.

There are 3 ways to ensure data anonymity:

1. k-anonymity, where at least k individuals in the dataset share the same set of attributes. This is implemented by:

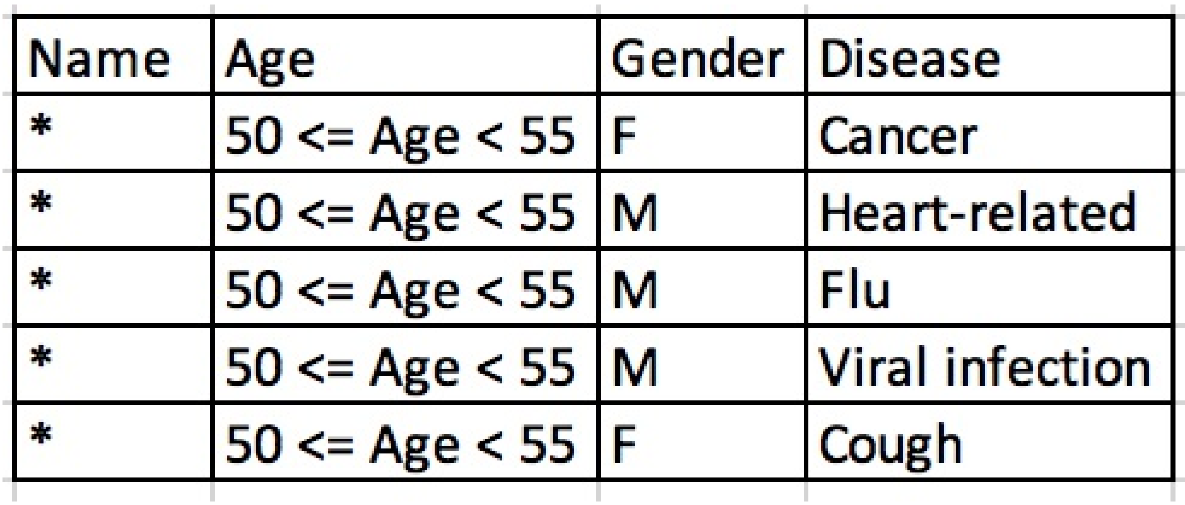
    i. Suppression. For instance, all or some values of a particular column may be omitted through replacing them with "*".
    ii. Generalisation. For instance, the medical condition can be replaced with a more general term (e.g., “Cardiovascular disease” replaced with “Heart-related”).

Example of pre-anonymised data:



Name	Age	Gender	Disease
Alice	50	F	Cancer
Bob	52	M	Heart-related
Charlie	54	M	Flu
Dick	53	M	Viral infection
Elias	52	F	Cough

After applying 2-anonymity to the data. Notice that this data has 2-anonymity with respect to the attributes "Age" and "Gender", but not for the attribute "Disease":



Name	Age	Gender	Disease
*	50 <= Age < 55	F	Cancer
*	50 <= Age < 55	M	Heart-related
*	50 <= Age < 55	M	Flu
*	50 <= Age < 55	M	Viral infection
*	50 <= Age < 55	F	Cough

2. Adding noise to the data. This can be implemented by swapping cells within columns.

3. Generating fake data, where the fake data must be a good representation of the actual data.

Since anonymising data is not the key focus of the project, we decided not to spend time to implement the algorithms if possible. Out of the 3 methods discussed above, only k-anonymity has an existing well-established library, complete with examples. Therefore, k-anonymity will be used to ensure data anonymity.

Secure Data Transfer

We intend to have a web application in which authorization is handled by Apache Shiro and authentication by JSON Web Token (JWT). This would ensure secure data transfer when a user logs in with his/her own credentials.

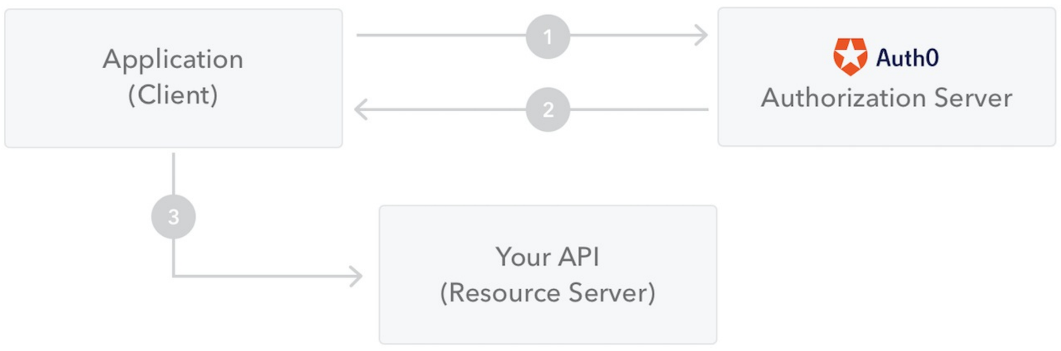
Apache Shiro is a powerful and easy to use Java security framework that offers developers an intuitive yet comprehensive solution to:

1. Authentication
2. Authorization

Although Apache Shiro already has an authorization functionality built into its library, we decide to employ the use of JWT for this purpose instead as JWT is much easier to implement. JWT removes the need for cookies and session which is required in Apache Shiro's framework. Since JWT only provides authorization, we would still require Apache Shiro for authentication. A justification for the use of JWT is because it comes with an extensive library for Java programming which we are employing in our backend server code.

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. A JWT is actually a string consisting of three parts: Header, Payload, and Signature. The JWT can encrypt the secret using the Hash-based Message Authentication Code (HMAC) algorithm or sign it using the Rivest–Shamir–Adleman (RSA) public and private key. The purpose of encryption is to ensure data transfer traffic is not susceptible to potential interceptors. This JWT is then presented every time the subject requests protected resources.

The following diagram shows how a JWT is obtained and used to access APIs or resources:



When we use Apache Shiro to implement the login system, a JWT information is returned to the front-end, and it stores the token information in the request header when it interacts with the back-end server. We will then configure a custom interceptor to intercept all URL requests, retrieve the token information in the request header information, and verify the token information. If the token information generated during the login is correct, the user is logged in. Otherwise, we will reject the request and return a 401 error.

An alternative to Apache Shiro is Spring Security. Although both frameworks function equally well, Apache Shiro requires lesser dependencies than Spring Security, making it lightweight in contrast to the heavyweight latter. Hence, we are choosing to go with Apache Shiro.

Source Code Control & Issue Management

Our project uses GitHub for source code control and issue management because most of the team are familiar and comfortable with using GitHub. This is not the case for the alternatives.

There are other alternatives such as:

1. Bitbucket. It allows free hosting of private repositories. However, the free version only allows for up to 5 users per repository. As our team has 6 members, we are not able to use the free version. Furthermore, there is no requirement to host our code on private repositories.

   Therefore, there's no compelling reason for us to use Bitbucket over GitHub.

2. GitLab. GitLab has inbuilt CI/CD, which means that we do not have to manually integrate external CI/CD tools to our repositories. This is a plus point, though, not a very big one since it's easy to manually integrate external CI/CD tools.

   However, GitLab is lacking some features compared to GitHub. For example, an issue can only be assigned to a single developer. There may be occasions when we need to assign multiple developers to resolve a certain issue, perhaps because that issue is hard.

   Therefore, there's no compelling reason for us to use GitLab over GitHub.

Reports will be written in .md (instead of .doc) to allow for version control of reports as well.

DevOps Tools

Configuration Management Tool

Our project will use Ansible as our configuration management tool to ensure that our remote servers are set up properly and simply. This doc will help us with setting up.

A configuration management tool is great for our project as we are managing 3 remote servers with different functionalities, most likely serving their roles as a web server, an application server and a database server.

We considered an alternative which was Octopus Deploy. However, due to our limitation of using Linux it is not viable to use this. The cost for this tool was too high as well.

Comparison with other tools

The most popular configuration management tools out there are Ansible, Salt, Puppet, and Chef.

Puppet and Chef are pull-based configuration management tools so that means that they pull all the configurations from a central server, which will not serve our needs. They also constrict users to using Ruby. Ansible and Salt, on the other hand, are push-based configuration tools which means users can directly push their configurations onto their node machines. We chose Ansible over Salt as Ansible has excellent security using SSH. It also has a less steep learning curve than Salt, which is important given that we do not have experience with these tools.

Security Tools

To scan our web application for vulnerabilities like XSS, we will use OWASP ZAP. This article by Upguard makes a comparison between two of the most popular open source web application penetration testing tools - Arachni and OWASP ZAP and we have decided on ZAP as it has more extensive community resources.

To find and fix known vulnerabilities in open-source dependencies, we would also use Synk,over SourceClear, which is not free, and Synk can be integrated easily with our Github repository.

As we would need to uphold our security claims to our users, we will also consider using BDD-Security to launch automated scans with these specific security claims. There are other automated security testing frameworks like GauntIt, but BDD-Security has more example tests that we can employ, and it is much easier to setup with fewer prerequisites needed on the system.

Penetration Testing Tools

For our penetration testing exercise, we will use the following tools for specific purposes:

Web front end

1. Tamper Data and Paros Proxy for testing the application on web based attacks (e.g. SQL injection, XSS etc). This will also be used to find logical errors in their web application.
2. Wireshark to observe if any sensitive data being transmitted unsafely.
3. Netwox to attempt man in the middle attacks.

Back end

1. Nessus for a general scan of the servers to check for any vulnerabilities. Nessus is one of the best scanners in the industry.
2. Nexpose is another scanner we will use to scan for vulnerabilities. It has the added bonus of being created by the company who created Metasploit.
3. Metasploit in order to exploit any vulnerabilities in an attempt to install a payload (Malware etc).
4. John the ripper for any password cracking purposes.
5. NMAP for identification and general reconnaissance.

Most of these tools can be found installed on Kali Linux.

Technology of the Tag

We will be communicating with the RFduino tag over Bluetooth Low Energy (BLE).

Due to it being architecturally less complex for us to implement a web application instead of a native Android/iOS app that communicates with the tag and also due to the lack of other alternative Bluetooth tools for web applications, we have decided to make use of the Web Bluetooth API which is written in JavaScript. As Web Bluetooth is still a work in progress right now, only certain platforms have Web Bluetooth fully implemented & supported in the browsers. The full status of its implementation can be found here. Chrome OS, Mac and Android have it fully supported and implemented. Windows and Linux have it partially implemented and a flag must be enabled.

Due to the I/O capabilities of the tag (no input and output capabilities), it is only possible to use the JustWorks method of BLE which offers no way of verifying the devices taking part in the connection so a MITM attack cannot be prevented. We would look at having some output capabilities for the tag.

The tag shall have 2 different functions (for 2-Factor Authentication and for supporting the validation of the ownership of health data).
The first function of the tag is to participate in a 2-Factor Authentication system for patients.

The steps of the 2-Factor Authentication for patients are as follows:

1. Patient logs into his account on the web app with his IC and a password.
2. Web app verifies the patient's login credentials.
3. If the verification is successful, patient brings his tag close to the device running the web app.
4. Patient's tag sends a pair of values (a randomly generated number and an encrypted value of the randomly generated number using the patient's private key) to the web app.
5. Web app verifies the patient's tag using the patient's public key.
6. If the verification is successful, patient logs in successfully.
7. Otherwise, if any of the verifications fail, patient will not be logged in.

The second function of the tag is to support the validation of the ownership of health data. In the scenario where a therapist photographs a wound on an unconscious patient, we would need to ensure that the photo belongs to that patient.

The steps of that scenario are listed below:

1. Therapist logs into his account on the web app and types in the patient ID.
2. Therapist then takes a photo and uploads onto the web app.
3. Web app pairs with the tag nearby on the patient.
4. Web app sends a hash of the image to the tag.
5. The tag signs it with its private key and sends it back to the web app.
6. Web app decrypts with the tag's public key. If the decrypted content is the same as the original hash sent, the picture is now ensured to belong to the patient because only the patient has the tag.