

System design

System introduction

The system provides the hospital with a system that allows different actors in a hospital to view and manage their electronic health records. It ensures that medical records stored in this system are not being viewed or changed by unauthorized people using a role-based control model which gives each type of user different permissions and access rights. The system supports AAA features (Authentication, Authorisation, Accountability).

System Requirements

- Functional requirements:
 - The system shall allow the user to login first time with the temporary password created by the user with administrator role
 - The system shall require the user to change their password after first login
 - The system shall allow to access data after successful validation
 - The system shall support 4 categories of users having different permissions
 - The system shall allow user with administrator permissions to create, delete and update users
 - The system shall allow users with administrator role to assign users with the permissions based on role
 - The system shall allow users with staff and regulator role to see the list of all registered patients
 - The system shall allow users with staff role to access and modify (read, write, update, delete) patients personal data and medical records
 - The system shall allow users with a regulator role to access and only read patients personal data and medical records.
 - The system shall allow users with the patient role to access and read personal data and medical records related to their user.
 - The system shall persist events happened in the system into a log on file system
 - The system shall verify the identity of the server and authenticity of it's messages
- Non-functional requirements:
 - The communication between the client and the server needs to be encrypted
 - The log on file system must be protected from deletion by the server application itself (must be append-only)
 - Graphical User Interface should be intuitive to navigate and easy to use

System components and functionality

Authentication Service - Registration and Login

The service supports the creation of new users by the Administrator. Administrator users are a special type of staff in the hospital that manages accounts for the medical staff, regulators and patients. When a patient is in hospital, their account gets created for them and they will get their credentials on a paper document (due to accessibility reasons for older people). When the patient safely gets home, they can log in with their credentials.

After the first successful login, the user is prompted to change their password in order to prevent any attacks that may have stemmed from, for example, leaving the paper document in a visible place.

The server proves it's identity to a client by using a public-key cryptosystem and establishes a secure symmetric connection between the client and server. All traffic between the server and client will be encrypted.

Password requirements:

- Minimum 8 characters long
- At least one Lower-Case Letter
- At least one Upper-Case Letter
- At least one digit
- At least one special character of !@#\$%&*()'"+,-./:;<=>?[]^_`{|}

A secure password policy is needed in order to extend the time required to break user's password using brute-force attack to a point where it's no longer feasible for the attacker to use brute-force attack. The time required to break a password grows exponentially as the complexity of the password increases. For example, according to [this source](#), breaking password "scctest" takes about 200 ms for an average computer. If we apply our password policy in a minimal way breaking this single "Scctest3#" password would take at least 3 weeks - and that's just a minimal requirement. Our intention was to make brute-force attacks less feasible for the attackers while still making it possible for users to remember their passwords in case they don't use a password manager. Because the last thing we want the user to do is to write down their password on a sticky note on their screen.

User login

Once a user (a regulator, staff employee or a patient) is given the credentials created by an administrator, they can login via the client application. For security reasons in cases their credentials are written on paper, after they log in for the first time, they are prompted to change their password. The password needs to conform to the rules of the system.

Login screen



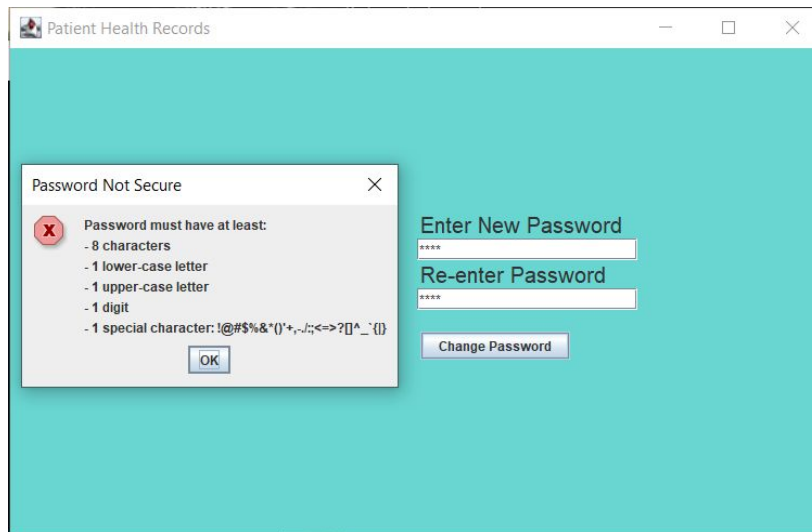
The screenshot shows a web application window titled "Patient Health Records". The background is a solid teal color. On the left side, the text "Patients Health Record" is displayed in a large, bold, black font. On the right side, there is a login form with two input fields: "Email" and "Password". The "Email" field contains the text "patient@email.com". The "Password" field contains a series of asterisks. Below the "Password" field is a "Login" button.

Change password prompt



The screenshot shows the same web application window titled "Patient Health Records". The background is a solid teal color. On the left side, the text "Patients Health Record" is displayed in a large, bold, black font. On the right side, there is a form for changing the password with two input fields: "Enter New Password" and "Re-enter Password". Below these fields is a "Change Password" button.

If the new password does not adhere to the secure password requirements an error message pops up with the rules. This ensures any password used to access the system is harder to guess and safer from fraudulent activities.



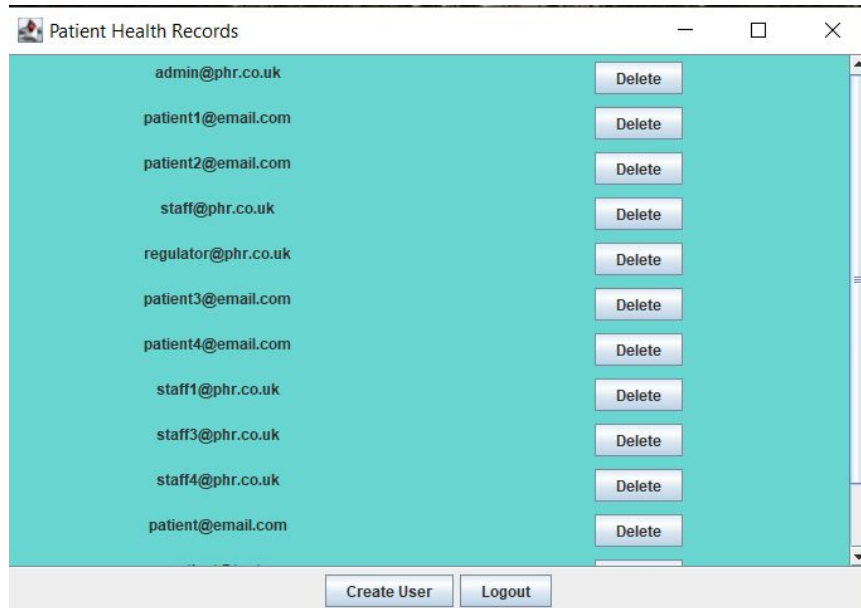
Once the user successfully logs in, they are redirected to their screen based on role - in this case a patient can see their health record. This means that each user can only access pages and data they are authorised for.



Account creation

An administrator is a special personele in the hospital which manages accounts. It doesn't see patient records, but is just responsible for processing requests when a new patient comes to the hospital and a new account and corresponding health record for this account will be created.

User management

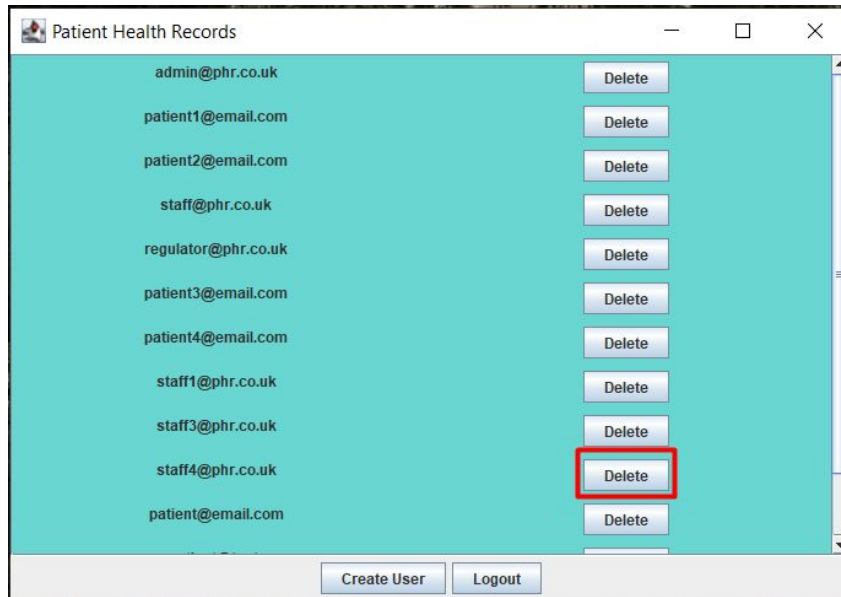


Creating new user (with one-time password)



Account deletion

The administrator can also revoke accounts (delete them) which will immediately stop any actions that staff/regulator or another admin account may take. For regular patients, that means their account will be deleted as well as their medical record. This is especially useful when an employee stops working for the hospital or in case that some account may have been compromised.



Database and password persistence

For storing the users' information there is a table called Users in SQLite database.

SQL Schema:

```
(id INTEGER PRIMARY KEY AUTOINCREMENT,  
email TEXT UNIQUE NOT NULL,  
password TEXT NOT NULL,  
role TEXT NOT NULL,  
should_update_password TEXT NOT NULL,  
salt TEXT NOT NULL)
```

Example of one entry the Users table might have:

id	email	password	role	should_update_password	salt
1	admin@phr.co.uk	3D8228DC11F5765A2...	admin	false	MvyShENkL1m17ldfDXpUUA==

Id and email

All users are assigned with unique id and email. Id's are automatically assigned once admin creates the user.

Password and salt

Password data stored in the database is hashed HEX String, which consists of the user's password plus random salt assigned to each user. Hash is a secure way of storing the password, because it is a one-way function and doesn't contain human readable text. Rather than storing the initial password user typed in, we store the hashed copy of it and when the user types in the password the program computes its hash one more time and compares with one stored in the database. Hashing algorithm used is SHA-256. Algorithms from the SHA-2 family are a lot more complicated than MD5 or SHA-0/1 and are still considered safe.

Because dictionary attacks are still possible we also generate and store salt. Salting makes passwords more complex and longer making attacks upon them far harder. Salt is a random generated data used as an additional input to function that hashes the password. It is stored in the database as a byte array converted to String and is generated by cryptographically strong RNG (random number generator) once the admin creates the user. With salt, attackers must compute hashes of all dictionary words once for each password entry.

Role

Each user is assigned with the role, which is used to give permissions specific to its role once user logging into the system. Available roles are: patient, regulator, staff and admin.

Should change password

This column has two possible values: "true" or "false". Once a user is created by the admin with a temporary password `should_change_password` is automatically assigned with the value "true", which means that user is required to change password. After first login, with credentials given to the new user system checks this value and if it is "true", it requires the user to update the password and sets the value to "false", meaning that the password was updated by the user.

Client-Server Session Negotiation (Login)

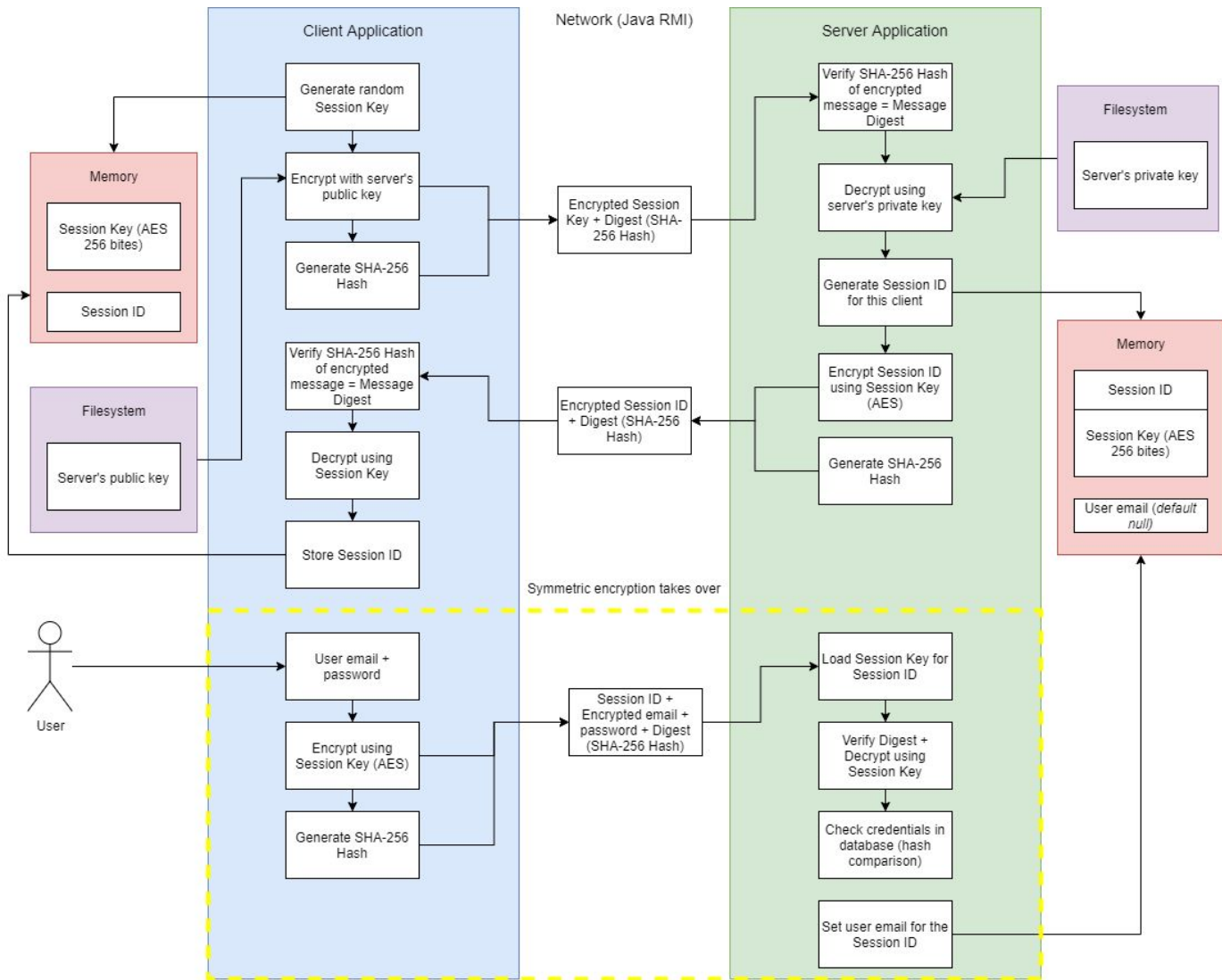
The client application connects to the server application via Java RMI via network (could be both running on localhost for testing purposes). Our application has a custom protocol between client and the server. At the start client-server negotiates a session using asymmetric cryptography and then switches to use symmetric encryption for every message object transmitted via Java RMI.

1. The client application contains the server's trusted public key stored on filesystem and uses it to encrypt messages sent to the server. It generates a random *SessionKey* (AES 256-bit key) and sends it encrypted along with a SHA-256 Hash to the server.
2. The server decrypts the message using its private key and verifies the message's integrity by re-calculating SHA-256 Hash and comparing it to the one received.
3. The server generates random string *SessionID* for this client and stores the decrypted *SessionKey* in in-memory dictionary where *SessionID* is the key and `{SessionKey & userEmail = Null }` is the value of that item (to enable efficient lookup for multiple clients).
4. The server encrypts the *SessionID* it generated with *SessionKey* using AES, generates SHA-256 Hash and sends to the client

5. The client decrypts the message using *SessionKey* and AES algorithm, checks the integrity and stores the *SessionID* which will be sent with every request to the server. It needs it to distinguish which symmetric key to use for the decryption (in case of multiple clients).
6. The client enters email + password and encrypts it using *SessionKey* and AES algorithm, generates SHA-256 Hash and sends to the server along with plaintext *SessionID*
7. The server looks up *SessionKey* based on *SessionID* of the request, tries to decrypt it using AES and that *SessionKey* and checks the integrity of the message.
(The same applies for all client-server and server-client communication from this point on.)
8. The server uses encrypted credentials, checks whether they are valid and if so, sets *UserEmail* property value in the Session dictionary meaning this session represents the user which has logged in.

The advantage of using this mixed approach is that we mitigated most of the disadvantages of asymmetric cryptography while keeping its benefits. For example, RSA encryption is slow and plaintext payload is limited and correlates to the key size which means it's impractical for rapid communication. In contrast, symmetric encryption is fast and lightweight, but has the problem of key exchange. By combining those two cryptosystems together we used asymmetric encryption to verify the server's identity and exchange keys which will be used throughout the session for symmetric encryption between the client and the server.

RSA public-private cryptosystem provides authorization of the server, RSA encryption and AES encryption provides confidentiality and SHA-256 ensures integrity of the exchanged data on an unsecured network.



Full resolution for this diagram is available at [this link](#).

Authorisation Service - Role-based Access Model

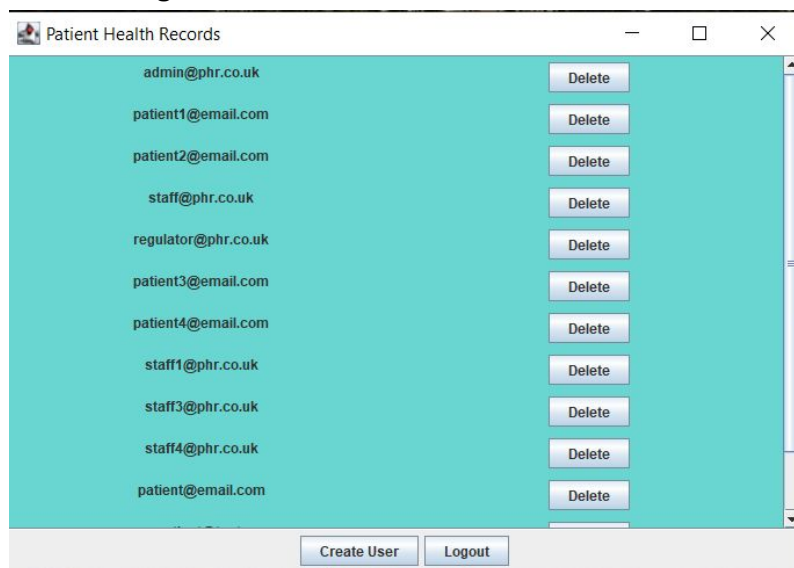
The system works under a role based access control model which should only allow users to access and see data that is within their authorised scope. Following the principle of least privilege, users have just the permissions they need:

- **Administrator** users are only able to manage (create/delete) user accounts, they cannot see medical records.
- **Patients** are only able to see health information about themselves and they cannot edit any of the data.
- The hospital **staff** is able to see information about all the patients and can create, delete, edit and read their records.
- The **regulators** are only able to read all records but are unable to change or edit them in any way.

After the user logs in, the client app asks the server for the role of the logged in user and shows only actions which the current user can take. This obviously would not be a security barrier in itself, so in addition to that whenever the server receives a request from the client it always verifies whether the user which is logged has sufficient role to perform this operation. Even if somebody observed the traffic in, for example Wireshark, and made different requests without the UI, the server would reject it.

Administrator UI

User management

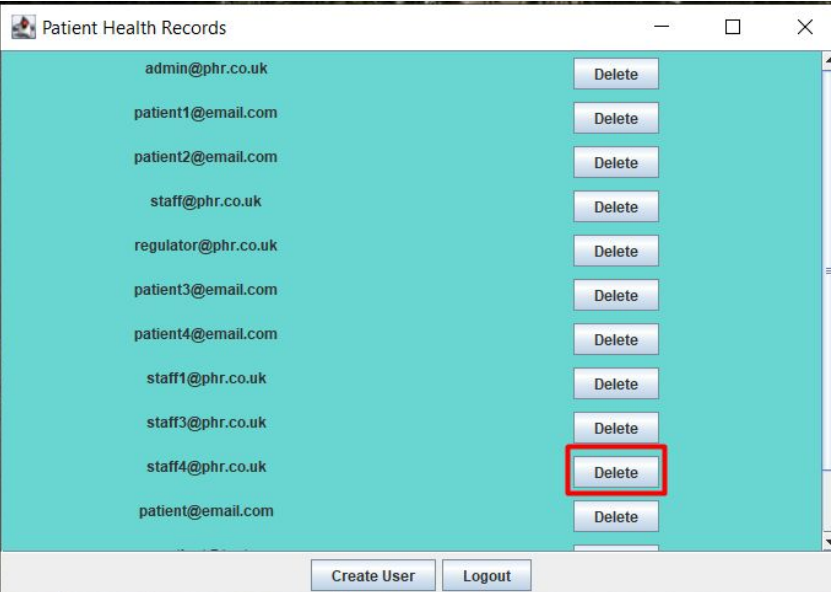


Creating new user (with one-time password)



The screenshot shows a window titled "Patient Health Records" with a teal background. On the left, there are two input fields: "Email" with the value "newuser@email.com" and "Password" with the value "OneTimePass123#". To the right, under the heading "Role:", there are four radio button options: "Patient" (selected), "Staff", "Regulator", and "Admin". At the bottom center, there are two buttons: "Create User" and "Back".

Deleting an account



The screenshot shows the same "Patient Health Records" window, but now displaying a list of users. The list has two columns: email addresses and "Delete" buttons. The email addresses are: admin@phr.co.uk, patient1@email.com, patient2@email.com, staff@phr.co.uk, regulator@phr.co.uk, patient3@email.com, patient4@email.com, staff1@phr.co.uk, staff3@phr.co.uk, staff4@phr.co.uk, and patient@email.com. The "Delete" button for "staff4@phr.co.uk" is highlighted with a red rectangle. At the bottom, there are two buttons: "Create User" and "Logout".

Email	Action
admin@phr.co.uk	Delete
patient1@email.com	Delete
patient2@email.com	Delete
staff@phr.co.uk	Delete
regulator@phr.co.uk	Delete
patient3@email.com	Delete
patient4@email.com	Delete
staff1@phr.co.uk	Delete
staff3@phr.co.uk	Delete
staff4@phr.co.uk	Delete
patient@email.com	Delete

Patient UI

Health record detail (their own)

Patient Health Records

ID: 28

Surname: Jordan

Blood Group: A+

Disabilities: None

Name: Michael

DOB: 17/02/1963

Allergies: None

GP: Dr Williams Parker

Logout

Staff UI

Patient records (view + search)

Patients Health Record

ID	Name	Surname	Date Of Birth	Blood Group	Allergies	Disabilities	GP
2	Joshua	Wood	01-01-1989	A+	fish	null	Dr. Clara Smith
3	Mark	Robbenson	12-12-1967	A+	cats	null	Dr. Mila Parker
7	Clara	Brown	05-06-1975	B-	dairy products	null	Dr. Mila Parker
8							
23	TEST	test	05-06-1975	B-	dairy products	null	Dr. Mila Parker
27							
28	Michael	Jordan	17/02/1963	A+	None	None	Dr Williams Parker


Patients Health Record

Search User

Clear Search

Log Out

Patient record detail (view + edit)



Patient Health Records

ID: 2

Name: Joshua

Surname: Wood

DOB: 01-01-1989

Blood Group: A+

Allergies: fish

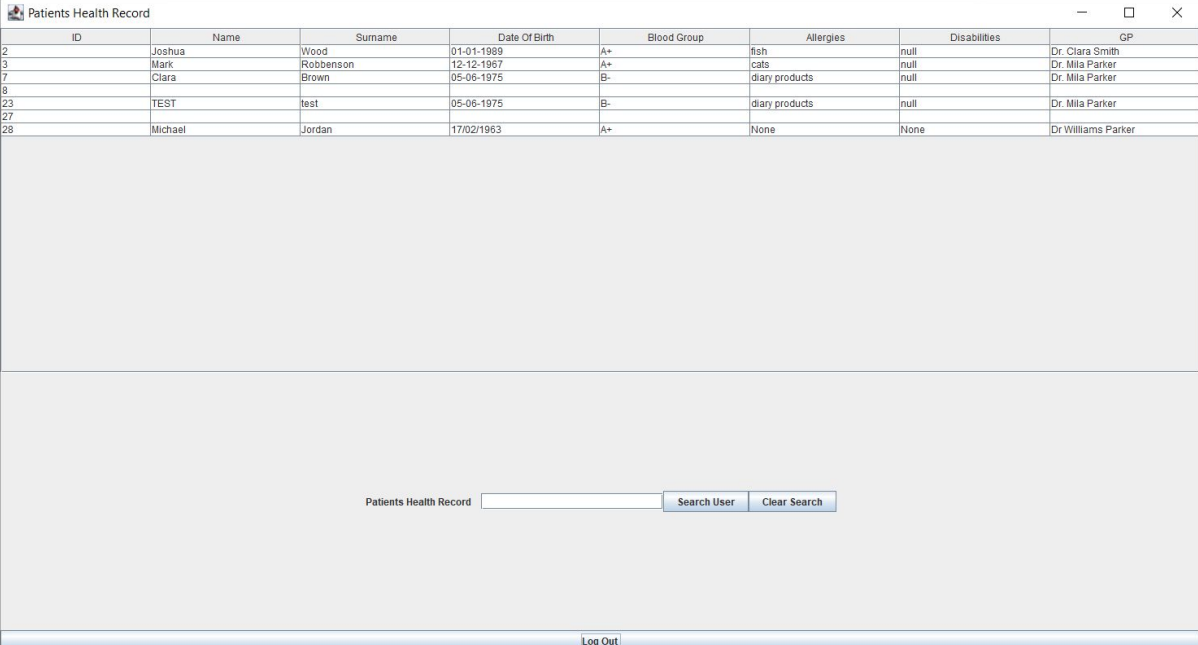
Disabilities: null

GP: Dr. Clara Smith

Save Back

Regulator UI

Patient records (just view + search)



ID	Name	Surname	Date Of Birth	Blood Group	Allergies	Disabilities	GP
2	Joshua	Wood	01-01-1989	A+	fish	null	Dr. Clara Smith
3	Mark	Robbenson	12-12-1967	A+	cats	null	Dr. Mila Parker
7	Clara	Brown	05-06-1975	B-	diary products	null	Dr. Mila Parker
8							
23	TEST	test	05-06-1975	B-	diary products	null	Dr. Mila Parker
27							
28	Michael	Jordan	17/02/1963	A+	None	None	Dr Williams Parker

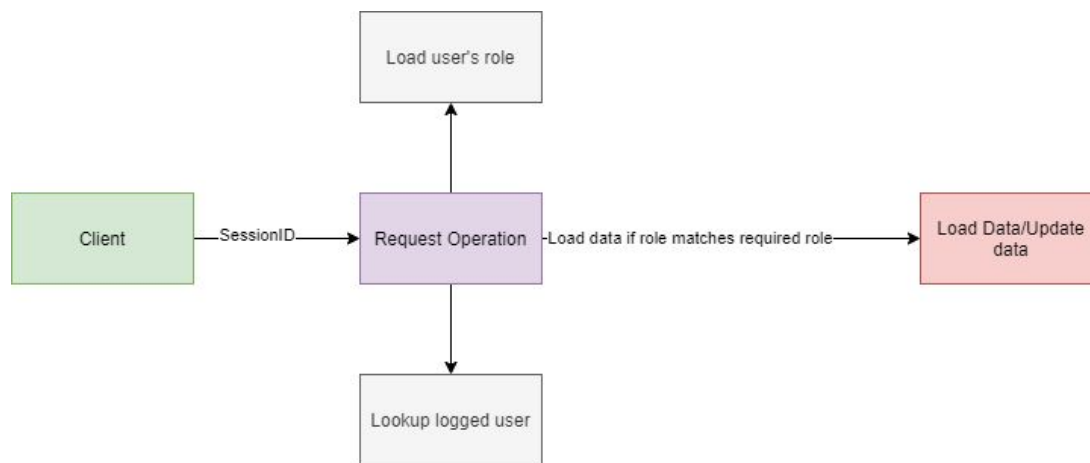
Patients Health Record

Search User Clear Search

Log Out

Role-based access control in the Server

Since the server holds the confidential data, it needs to be responsible for enforcing the authorization based on the user's role. Each operation the server provides (e.g. to list all patient records or to create a new user) has a specification in code which says which role can perform it. When the server processes a request, the client also sends *SessionID* and the server looks up the email of the logged user for that session in in-memory dictionary.



Then the server makes a database call to retrieve the user role from the database for that email. It compares it to the required role for the given operation and if it matches, returns the result (e.g. list of all patient records).

Database

As it was stated in the above section users' roles are stored in the database table called "Users". There are 4 possible roles that might be assigned to the user (only one per user): patient, regulator, staff and admin.

Example table with users of all possible roles:

id	email	password	role	should_shange_password	salt
1	admin@phr.co.uk	3D8228DC11F572E...	admin	false	MvyShENkL1m17ldfDXpUUA==
2	staff@phr.co.uk	11A9F295EBF16E4...	staff	false	SjMSPWUT6q76u7IVpsCLuw==
6	regulator@phr.co.uk	D9E14A17225BAA...	regulator	false	KqmyZgtX+z5xELZBn3m62g==
11	patient1@email.com	15BE6D8924B5FEE...	patient	true	v+a62oVWzDGqQU+VrOaaRw==

Roles are assigned to the user during creation. They are keys in making decisions to the system what access control to give to a specific user. System checks the user's role by making a call with user email as input and gets a role as output.

If the user role is "patient" it is automatically added to the "Patients" table with only filled value "user_id" as foreign key references id from "Users" table and others are NULL.

Example entry in Patient table of just added user with id:11 (see table above) and role Patient:

user_id	firstname	surname	dob	blood_group	allergies	disabilities	gp
11	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Example entry of same user with added/modified record by staff member:

user_id	firstname	surname	dob	blood_group	allergies	disabilities	gp
11	Peter	Parker	01-01-2020	A+	fish	NULL	Dr. Janna Wood

Accountability - Logging service

The system persists append-only logs to the filesystem of the server. Every log stores the hash value of the previous log entry as well as its log value. The server application code itself can only add new logs, so no one no matter their permission can delete any of the logs, provided the file system is protected by the operation system of the server. Also, we assume that the safest place for the logs to be in is the filesystem as the logs are not being transferred over the network. Usually, if a company updates its OS often or uses well-established cloud providers for managing their VM's, having logs stored there is one of the most secure places of the system.

The server application logs various events that happen, which may be important when trying to determine the root cause or scope of an attack. All logs contain metadata which should help to identify an attacker or compromised accounts. The system logs have different TAGs:

- ERROR - Occurs when an exception is thrown in the application (could be attack or application bug).
- EVENT - Occurs when significant system action takes places (e.g. patient record was updated by staff or patient logged in)
- WARNING - Occurs when the system is experiencing potentially malicious actions from some actors

All the logs contain important metadata about the events in the system. For example, if a patient record was updated in the system then it contains both the id of that record and also the email address of the staff account which performed the update.

To ensure the integrity of the log entries, each entry contains SHA-256 Hash of the previous log entry which prevents it from tampering. Example log file looks like this (all logs from one day go into one file):

```
2021-02-08.txt - Notepad
File Edit Format View Help
2021-02-08 15:14:50|---|EVENT|SessionSuccessfullyNegotiated:SessionId: aZEdEP0JxjQdtw==
2021-02-08 15:15:13|P52TdPjpkGpSLvCCgZNBxZ0+00xR5t0WGP1u/3RZmB4=|EVENT|SuccessfulLogin:Email: admin@phr.co.uk SessionId aZEdEP0JxjQdtw==
2021-02-08 15:15:38|p/WGJzcDgF/5YJZRUTFE/DYKQIjyYZxwZipRnlggegk=|EVENT|UserCreated:CreatedEmail: test1@test.com CreatorEmail: admin@phr.co.uk
2021-02-08 15:15:59|bnqJTfv+oLtGSDanCQ2QuSrid53Kj55DEzb6VBWwbak=|EVENT|SessionSuccessfullyNegotiated:SessionId: rh84IZbl1VT0Qw==
2021-02-08 15:16:08|t/65S740moyPmfHd8e5qgC2j13hONHOISmRapGHbPds=|EVENT|SuccessfulLogin:Email: admin@phr.co.uk SessionId rh84IZbl1VT0Qw==
2021-02-08 15:17:26|uE47jTniWfth5M0fg0QpTq5jrdH+h1Fg/1jbf48vNM=|EVENT|SessionSuccessfullyNegotiated:SessionId: C5X2FjrmsB2TzQ==
2021-02-08 15:17:33|zjHATDwD9nYnaU0NZpazt0GB7wzYUGEqbdQ0PJcWLT=|EVENT|SuccessfulLogin:Email: patient1@email.com SessionId C5X2FjrmsB2TzQ==
2021-02-08 15:17:42|UXmiS8MB4KT1fieciS6ChleaBZVFzIhCwb0wx8ZTTXE=|EVENT|SessionSuccessfullyNegotiated:SessionId: FfpXk1F2ILfLHw==
2021-02-08 15:22:29|tAbk0skkbj602eguvDmgfH5I9BDR3xIb3AfwxVZ9H8=|EVENT|SuccessfulLogin:Email: staff@phr.co.uk SessionId FfpXk1F2ILfLHw==
2021-02-08 15:22:57|jab/W7BTiVa8upmRBPf+ycVwFK+LSD0ba0ZooWARD2I=|EVENT|SessionSuccessfullyNegotiated:SessionId: LRnkqYx7jhumxg==
2021-02-08 15:23:16|dW03S1r+6i0e6DPsFMEh9ZB6FsStsm2BIT0fgHBQa=|EVENT|SuccessfulLogin:Email: staff@phr.co.uk SessionId LRnkqYx7jhumxg==
2021-02-08 15:23:33|UvXL24VwNrIhRdQTBegLrtJWNYLWF9NQJtcMkxiduuc=|EVENT|PatientRecordUpdated:RecordId: 2 UpdatorEmail: staff@phr.co.uk
2021-02-08 15:23:34|szTH/LjxvLU8oe52ik1Ddy0PBI9NeKQNPi8t/KC0ao8=|EVENT|PatientRecordUpdated:RecordId: 2 UpdatorEmail: staff@phr.co.uk
2021-02-08 15:23:43|xIM7rj3xFK/pYf1Gq5/WzQZiG4evFjMAFFZlSE8VnBE=|EVENT|SessionSuccessfullyNegotiated:SessionId: cXxyY8MCDhHlw==
2021-02-08 15:23:51|0990AwzYHjdRwy9etinSvp36bg9Xk0T85/0NpEhLzj4=|EVENT|SuccessfulLogin:Email: staff@phr.co.uk SessionId cXxyY8MCDhHlw==
2021-02-08 18:37:45|SoCrZQb5NfgvC3ZLyqZoxjhKfDcocPpsZRPy7e3q98=|WARNING|Unsuccessful login attempt for email admin@phr.co.uk using SessionId: U74FiucbqoSdxQ==
2021-02-08 18:37:47|TCCwiq/QKKRhhEPZv8kAVmbF0jieIj23t8+ddjWCck=|WARNING|Unsuccessful login attempt for email admin@phr.co.uk using SessionId: U74FiucbqoSdxQ==
2021-02-08 18:37:47|zmeajG6ZyT45QL2oNFzEg7Pauue1VZQ9sw4LjnBZp10=|WARNING|Unsuccessful login attempt for email admin@phr.co.uk using SessionId: U74FiucbqoSdxQ==
```

Types of logs in the system

Log	Event Type	Metadata
Invalid message integrity	WARNING	
Session successfully negotiated	EVENT	SessionId
Successful login	EVENT	Email, SessionId
Unsuccessful login attempt	WARNING	Email, SessionId
Login attempt for non-existing session	WARNING	SessionId
User Created	EVENT	CreatedEmail, CreatorEmail
Patient Record updated	EVENT	RecordId, UpdatorEmail
UserDeleted	EVENT	UserId, DeletorEmail
Exception	ERROR	Stacktrace

Attack prevention and countermeasures

Brute-force + dictionary attacks:

Strong password policy forces users to use passwords which are harder to guess in case of dictionary attacks. Also, the more complex a password is the longer it takes to break using brute-force attack.

Rainbow Table Attack

Hashing passwords in combination with unique salt for each user means that pre-computed attacks which use Rainbow tables are not possible to perform. Rainbow table attacks usually rely on the system using just hashing algorithms thus making it easy to pre-compute hash values of any table of passwords.

Man-in-the-middle attack:

The client verifies the server's identity by validating cryptographic signatures by using a public-private key algorithm (RSA). Also, the communication between the server and client is encrypted using a symmetric AES encryption using shared key negotiated for the session and generated by client and transmitted in an encrypted form in which only the server can decrypt it.

Encryption ensures that "man-in-the-middle" cannot read any of the communication between client and the server. Integrity checks ensure that third parties cannot add extra information or manipulate the information while in transit on an unsecured network (requests which may have been tampered are rejected on the server).

Ex-employee misuse:

The system allows the admin user to delete old users from the database. Every succeeding login for that user will fail and all current sessions will be revoked (the user role won't be available anymore to make decisions) leading to effective revocation of permissions in case someone stops working at the hospital or his account is compromised.

Initial-password loss:

Since we use an administrator which creates accounts for other users, the users at hospital are given an initial password on a paper document. This document could be left in a public place and for that reason the system prompts the user to change their password after first successful login. After the user changes their password, leak of the initial password document does not present any risk.

Team member contributions

Michal Zatloukal (34851631) - Implemented client Java RMI backend + server logic, session negotiation & encryption, logging

Aleksandr Goidin (34833455) - SQLite DB design and implementing different calls by the server, password checking, hashing & salting, server database storage of records, UI mockup

Jonathan Lau () - Helped to build the front end client side, creation of table page, login and password creation check frontend.

Daniel Nestor (37527622) - Helped build the front end for the app and applied the methods to connect the front end to the back end using the server connector.