C.A.R.E

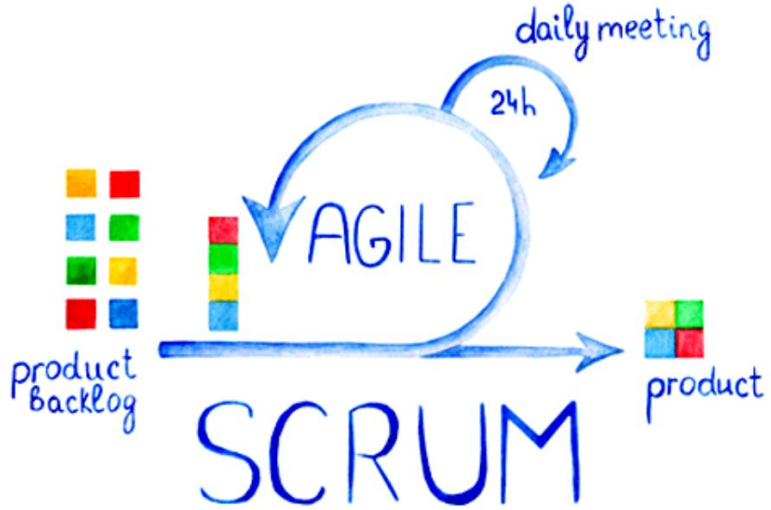
Centro **A**ccoglienza **R**egionale **E**matica

TEAM

Magliacane Hermann	**Team Leader**
Ranauro Giuliano	**Developer**
Ricciuto Luigi	**Tester & Developer**
Crovella Alessio	**Tester & Developer**
Polvere Donato	**Tester & Technical Writer**
Signoriello Sara	**Tester (usability & UX Tester)**
~~**Capriglione Vincenzo**~~	~~**Technical Writer**~~
~~**Quatraro Raffaele**~~	~~**Developer**~~

C.A.R.E

Team working



github.com/AccaEmme/CARE/commit/55997ee9cee4efc92a8e15941397e7789c4b3416#commitcomme

```
200 +     if(serial == "")        { alert("serial null");     k=1; }
201 +     if(note == "")   { alert("note null");       k=1; }
```

AccaEmme yesterday Owner

@Omni-star le note sono obbligatorie?

Omni-star 12 hours ago Author Collaborator

no, modificherò in seguito

Reply...

Problem Statement

CTT (Centro Trasfusionale Territoriale)

- Sacche di sangue locali CRUD
- Utenti locali CRUD
- Operazioni Offline
- Aggiunta e utilizzo di sacche
- Avviso sacche di sangue in scadenza

CCS (Centro Controllo e Smistamento)

- Aggiunta e rimozione di CTT
- Raccolta e gestione dati dei CTT
- Controllo prima e dopo la condivisione

I NOSTRI OBIETTIVI

Open Software

Il sistema C.A.R.E può essere utilizzato da qualsiasi ospedale senza costi perché open source

Facile Da Usare

Il sistema è facile da usare per chiunque nel reparto medico

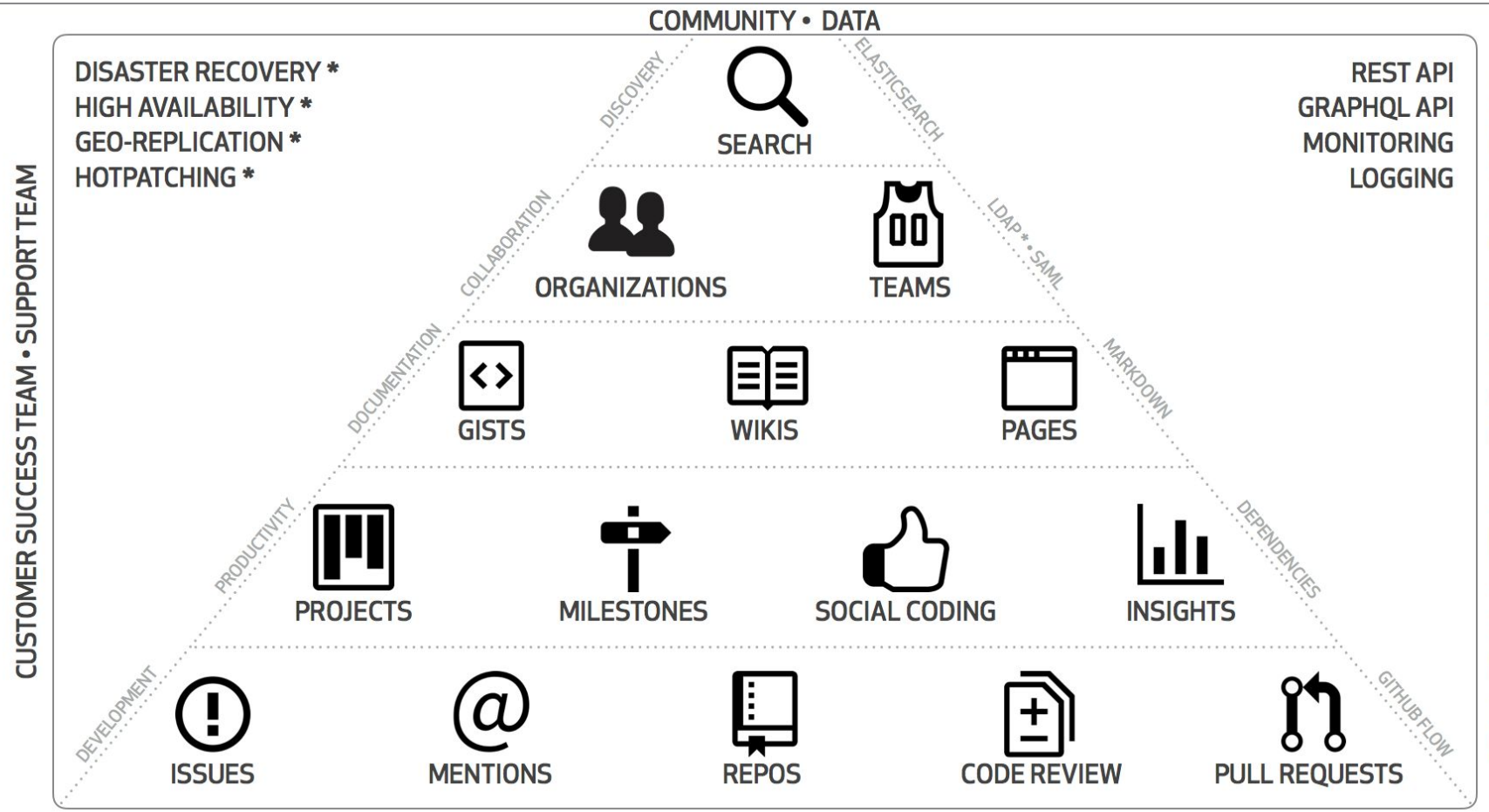
Niente Sangue Sprecato

Grazie alla comunicazione tra ospedali, le sacche non verranno più buttate

Work Offline

Anche se il CCS non risponde, il sistema funziona indipendentemente da ciò

Source Code Management



GitHub

- Source - https://github.com/AccaEmme/CARE
- Wiki - https://github.com/AccaEmme/CARE/wiki
- Discussions - https://github.com/AccaEmme/CARE/discussions
- Insights - https://github.com/AccaEmme/CARE/pulse
- Bug Tracking - https://github.com/AccaEmme/CARE/issues

C.A.R.E

Overview Of The System

1. Languages used
2. Architecture
3. Resources
4. Data security
5. Pattern

Languages Used

Java

with IDEs: Eclipse, Visual Studio, IntelliJ

(Spring Framework)

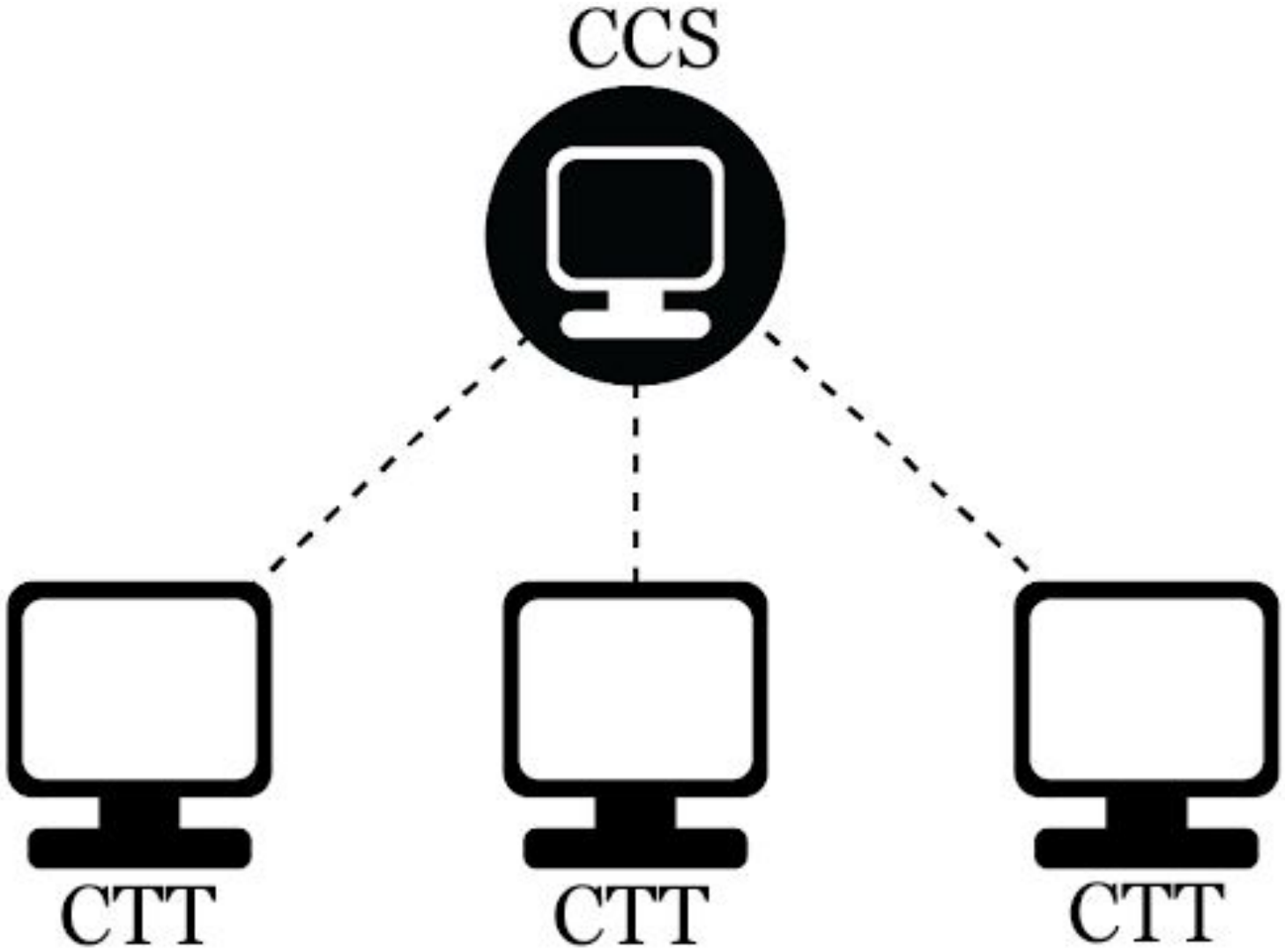
HTML, CSS, Vanilla Javascript

PHP

jQuery, RegExp

C.A.R.E

Architecture: client-server



Constraints & Definitions

- The bloodbag serial is an univoque protocol syntactically validated via RegExp (i.g. IT-NA206000-Apos-20210416-0001)
- The serial can't be set by users, set only by the system.
- Only administrator can create users
- "1900-01-01"(timestamp -2208996000) as null date
- Password complexity RegExp: "(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[^A-Za-z0-9])(?=.{8,})";

  Regex explaination of password secure requirements:

  * Password must contain at least one digit [0-9].
  * Password must contain at least one lowercase Latin character [a-z].
  * Password must contain at least one uppercase Latin character [A-Z].
  * Password must contain at least one special character like ! @ # & ( ).
  * Password must contain a length of at least 8 characters and a maximum of 20 characters.

```
^                       # start of line
    (?=.*[0-9])         # positive lookahead, digit [0-9]
    (?=.*[a-z])         # positive lookahead, one lowercase character [a-z]
    (?=.*[A-Z])         # positive lookahead, one uppercase character [A-Z]
    (?=.*[!@#&()–[{}]:;',?/*~$^+=<>]) # positive lookahead, one of the special character in this [..]
    .                   # matches anything
    {8,20}              # length at least 8 characters and maximum of 20 characters
$                       # end of line
```

- ...

Best Practices & Design Pattern

Some "Best Practice" used:

- **Indentation**: tabbed (not spaced) indentation to make code human readable easy
- **var names**: variables with significative name
- **Documentation**: every filesource is well commented, without useless info. In Java we provede javadoc too.
- **Code optimization**: reused properly functions(in javascript) and implemented simple and short classes in Java (no "God Classes" has been made").
- **Date timestamp** let's use timestamp instead of strings or object.
- **Enumerator**

- **SALT**
- **login anti-bruteforce** if attempts more than 3, user is blocked and should be managed by admin.

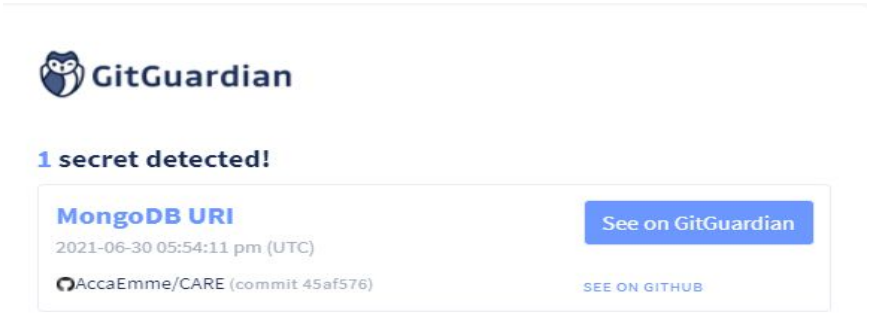
Some Pattern used:

- **MVC**: clear and easy file organization in Model-View-Control approach.
- **DAO PATTERN**: we defined Bean through "Data Access Object pattern" that represents the entity as in RDBMS. It will be used from "JPA Entity Manager"
- **Repository Pattern** provides an abstraction of data, so that your application can work with a simple abstraction that has an interface approximating that of a collection.
- **Iterator Pattern** In most of list we returned an iterator.
- **Composite Pattern** in report class we manage "report" for users and for bloodbag as composition of the same element
- **Decorator Pattern** CORS filter
- **Singleton Pattern** in SecurityContent we need to be ensure that we create only one instance of this class.
- **Builder Pattern**: our Password class that manages an object "Password". To avoid the overload of constructor User.

C.A.R.E

Data Security: Pattern & Best Practices

- Interfaces and patterns
- SALT
- Password cifrata nel database
- Password Complexity pattern constraint
- GitGuardian check uploaded code
- JWT token per i metodi esposti
- protocollo HTTPS per cifrare la comunicazione
- Attempts per evitare bruteforce
- Different DB users



id_user	active_user	creation_date	email	last_access	login_attempts	password	temppass	user_role	userr
1	0	1625071544336	NULL	-2208988800000	0	$2a$10$MsUNMafydo7uZ3nRzsPJjOzFjxkEhZQs...		ROLE_ADMINISTRATOR	giulian
2	0	1625073465660	NULL	-2208988800000	0	$2a$10$8aFKj1HEog2BNov.PN2mHu.N.bmPs5h...		ROLE_ADMINISTRATOR	giulian

Password.java | Constants.java

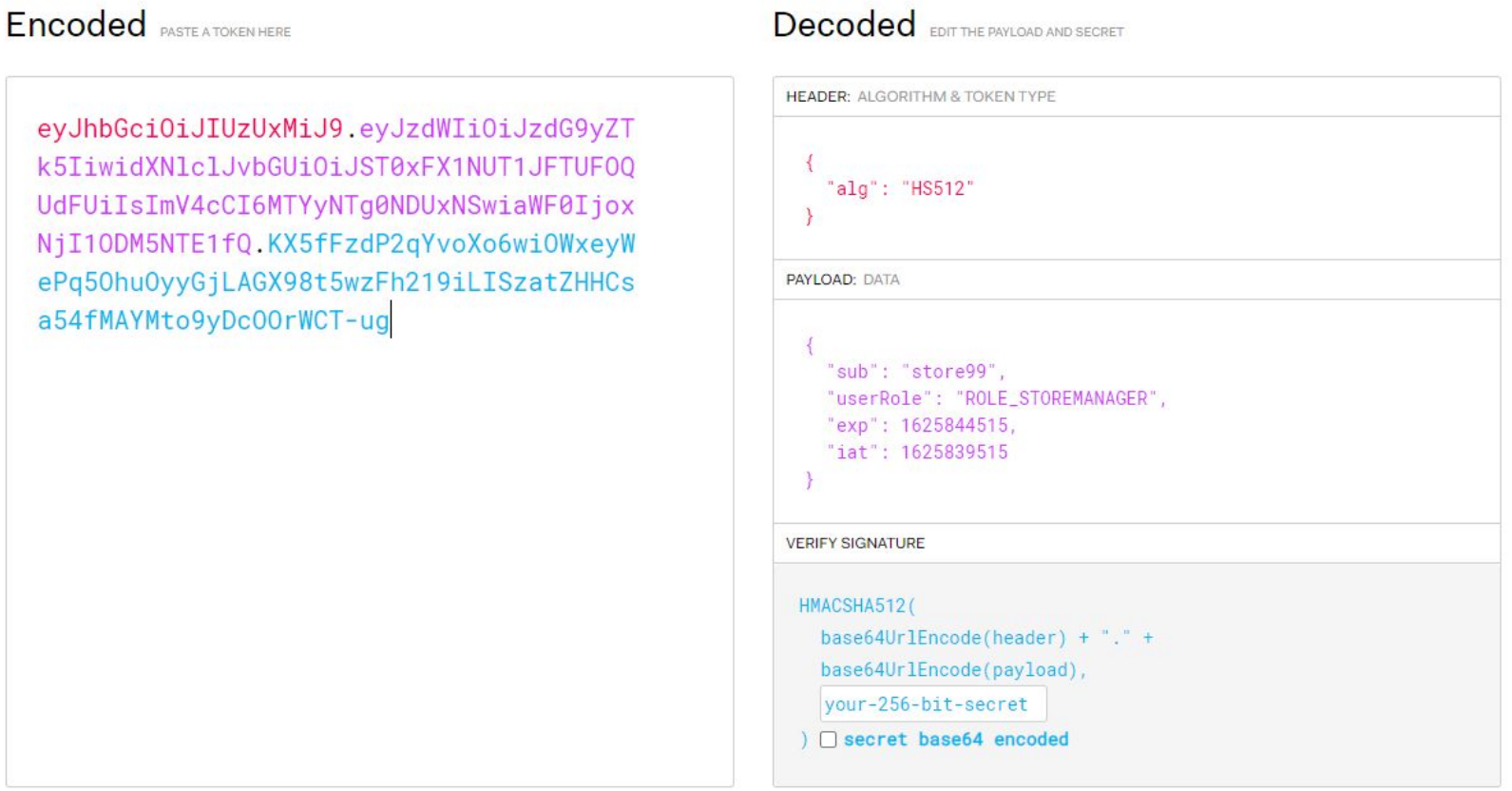
```
34
35      // ############################################## User Strings ######
36      public static final String PASSWORD_SALT = "CanforaMarkUs30L";
37      // public static final String USER_DEFAULT_TEMP_PASS = "CARE:changemenow";
38      public static final int USER_TEMPPASS_LENGTH = 10;
```

Password.java | Constants.java

```
87      public static String getBCrypt(String input) {
88
89          PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
90          input += Constants.PASSWORD_SALT;
91          return passwordEncoder.encode(input);
```

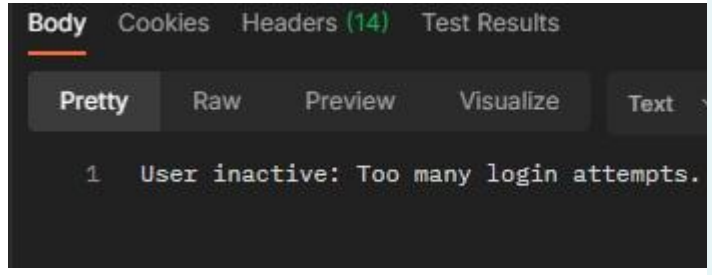
Data Security: Pattern & Best Practices

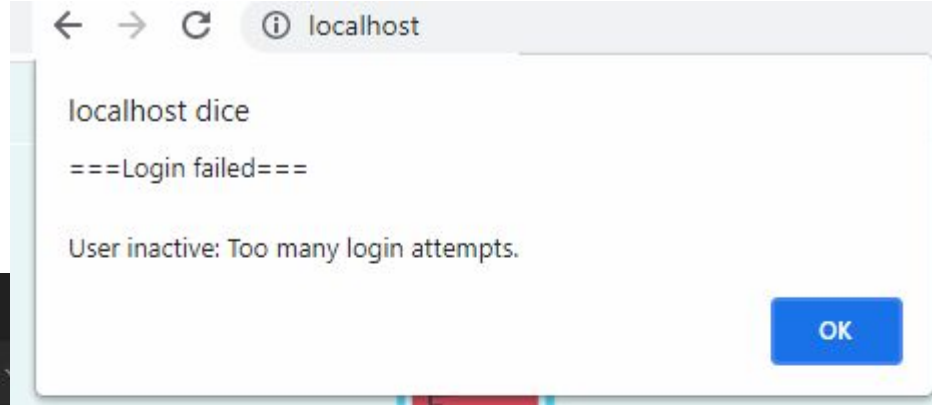
- Interfaces and patterns
- SALT
- Password cifrata nel database
- Password Complexity pattern constraint
- GitGuardian check uploaded code
- JWT token per i metodi esposti
- protocollo HTTPS per cifrare la comunicazione
- Attempts per evitare bruteforce
- Different DB users

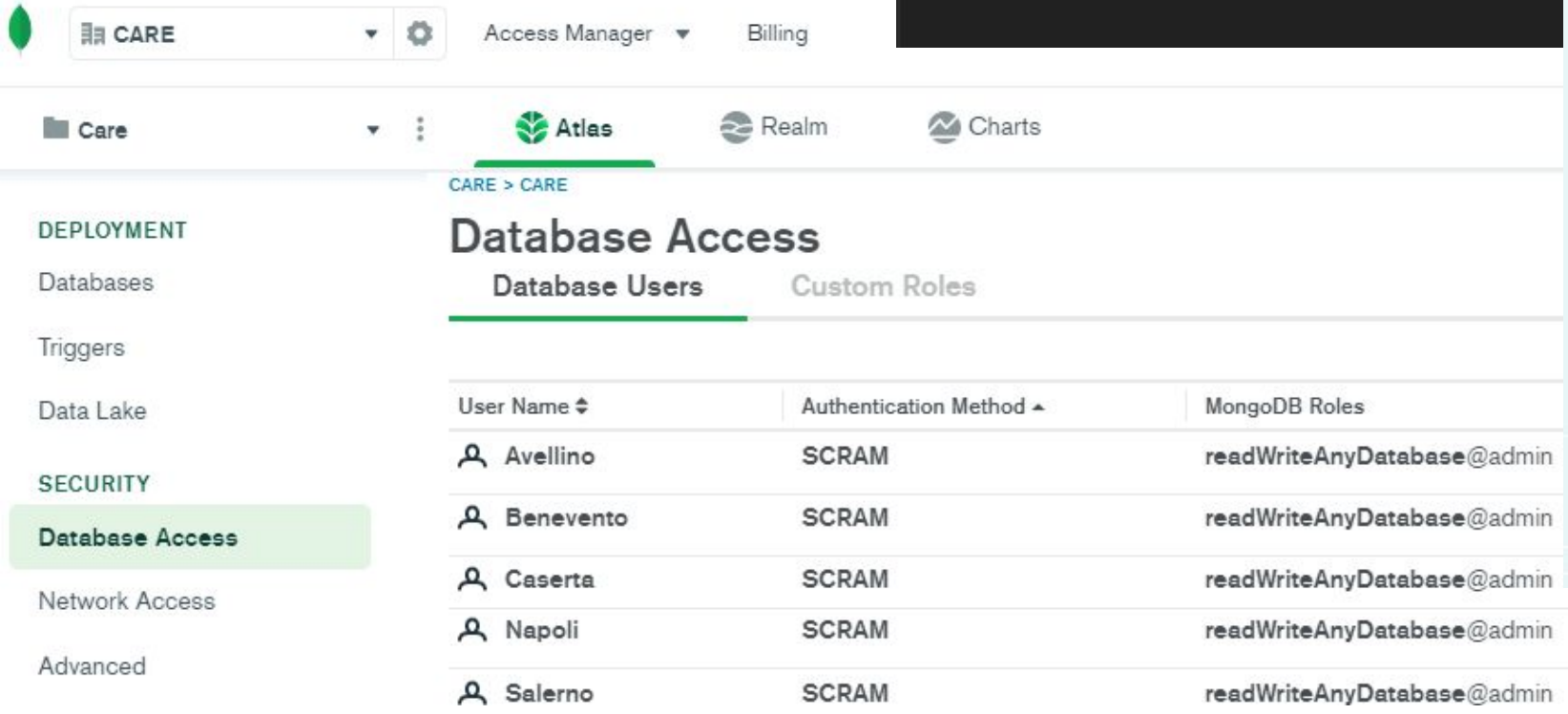


Data Security: Pattern & Best Practices

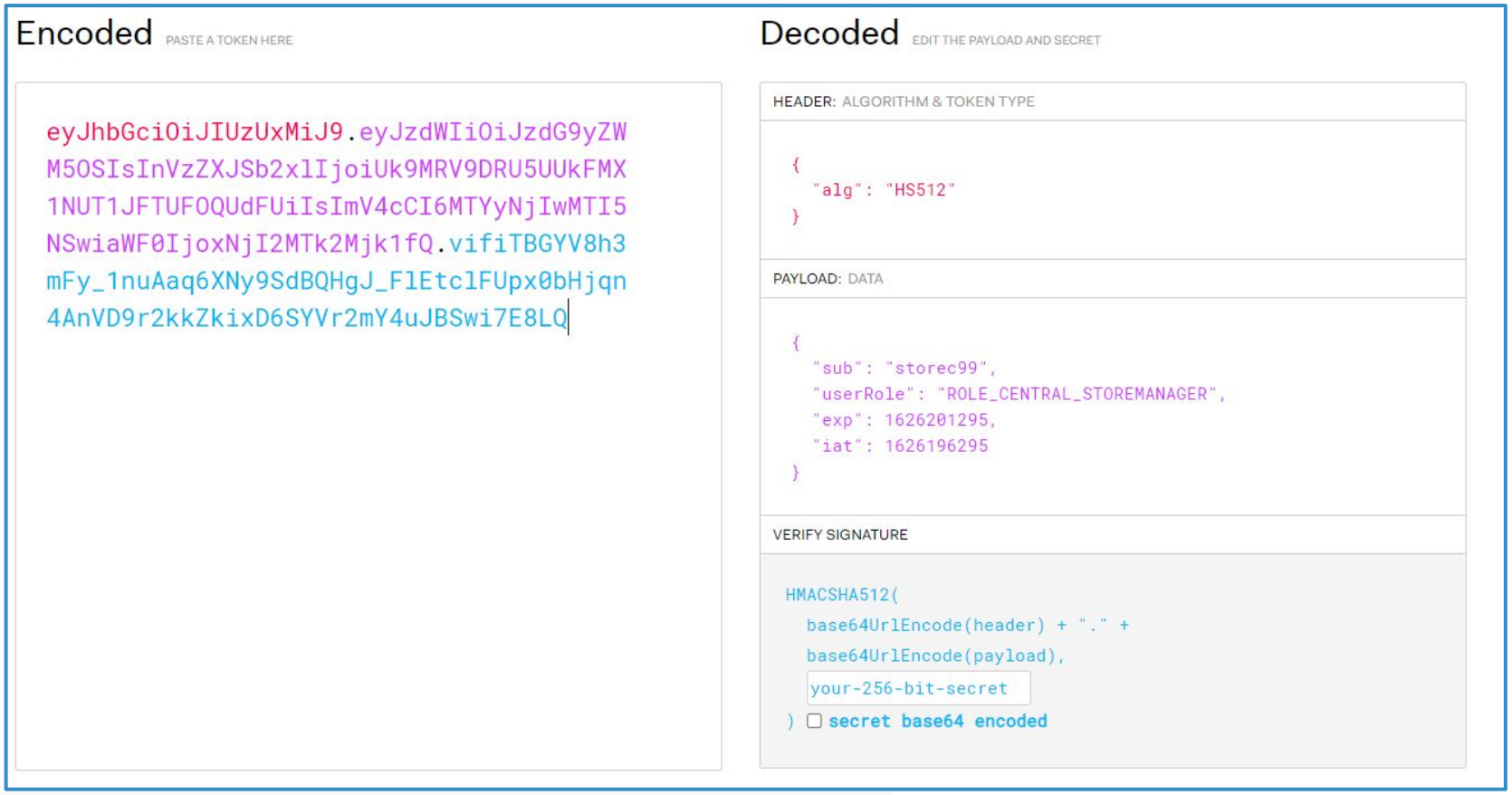
- Interfaces and patterns
- SALT
- Password cifrata nel database
- Password Complexity pattern constraint
- GitGuardian check uploaded code
- JWT token per i metodi esposti
- protocollo HTTPS per cifrare la comunicazione
- Attempts per evitare bruteforce
- Different DB users







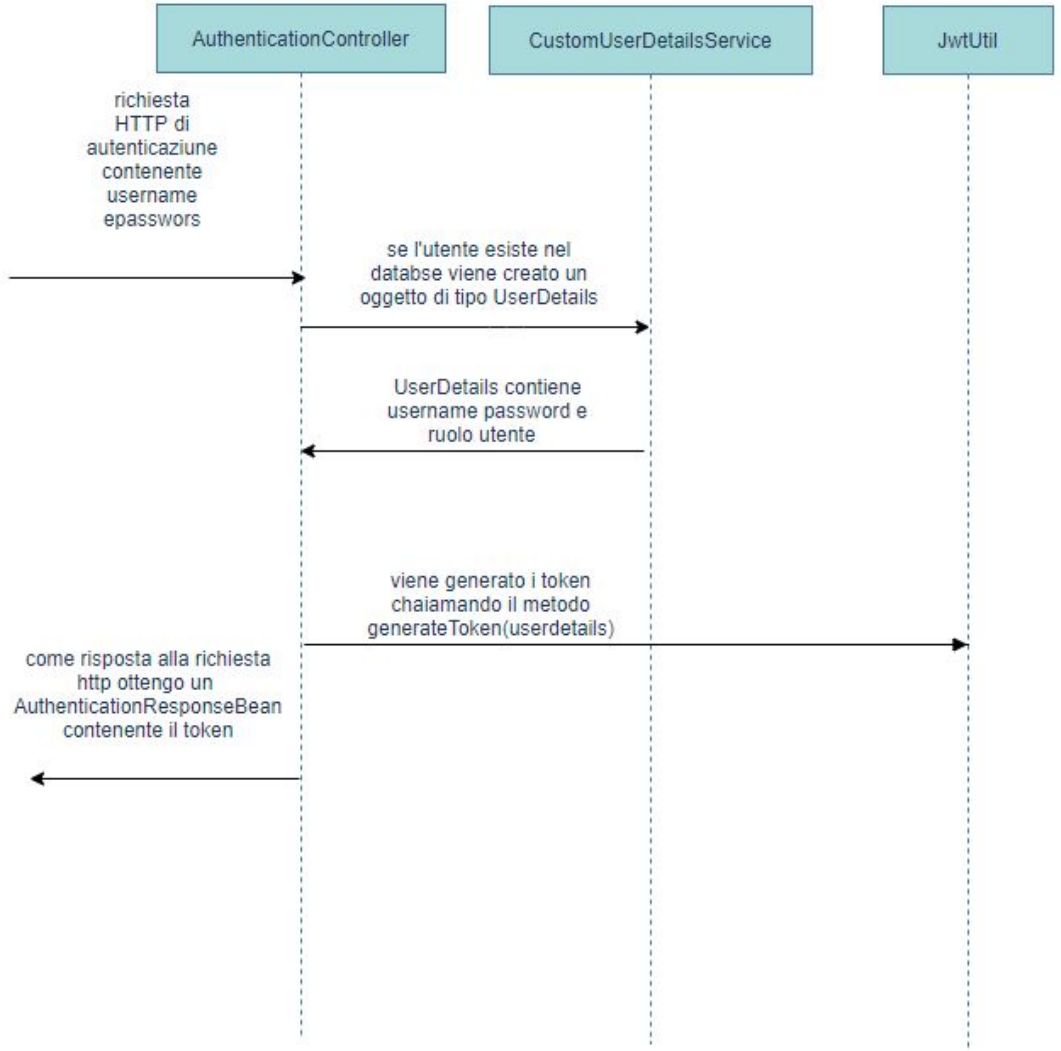
JSON-WEB-TOKEN



- Il JWT è uno standard utilizzato per l'autenticazione delle richieste http.
- Ha 3 campi Header,Payload,Signature
- Il token può essere crittografato e firmato utilizzando una chiave disponibile solo per il sistema che lo ha generato.
- Nel payload troviamo la durata del token e altre informazioni come il ruolo
- Il contenuto del payload non è crittografato, è importante che contenga dati sensibili.

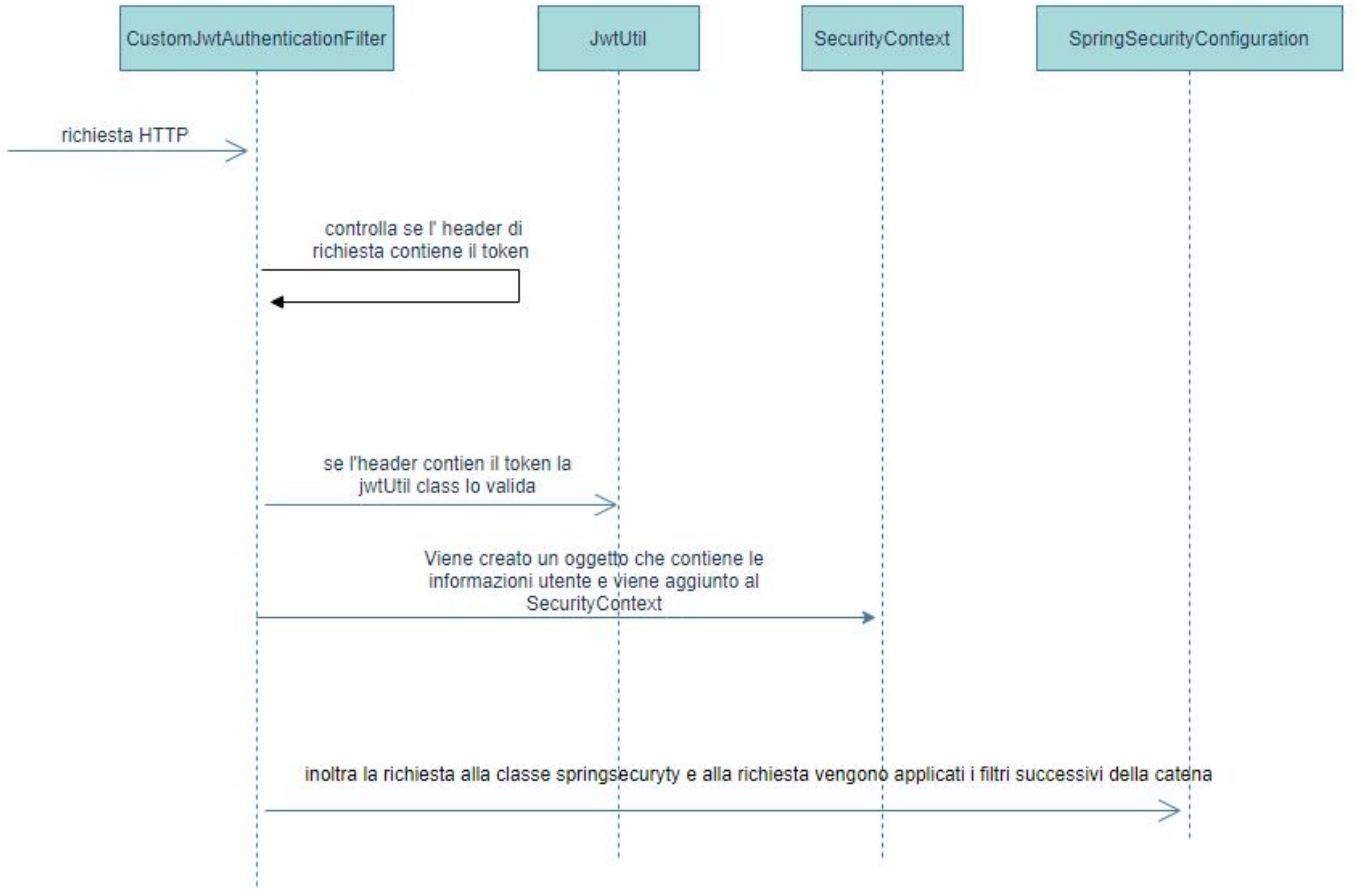
```
# jwt Authentication
jwt.secret = CARE@Unisannio
jwt.expirationDateInMs=5000000
```

Autenticazione



- l'endpoint di autenticazione è accessibile a tutti l'header di richiesta HTTP non ha bisogno del JWT.

Autenticazione basata su ruolo



TOKEN SCADUTO

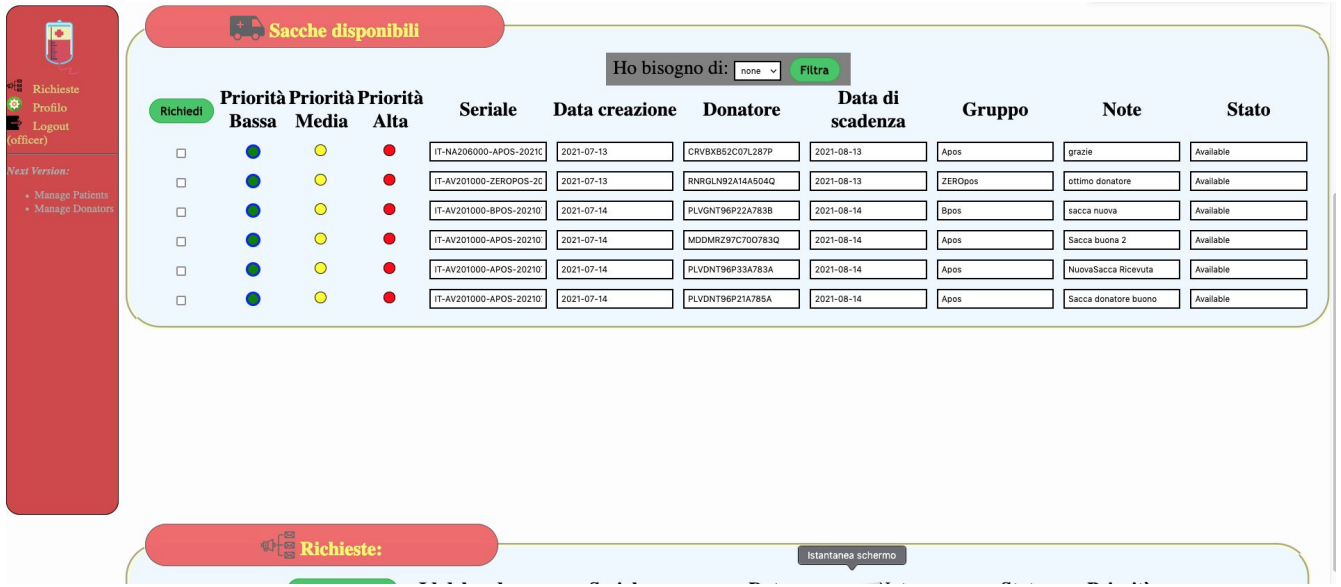
```
{
    error: "JWT expired at 2021-07-14T15:31:28Z. Current time: 2021-07-14T22:39:35Z, a difference of 25687271 milliseconds.  Allowed clock skew: 0 milliseconds."
}
```

ACCESSO VIETATO

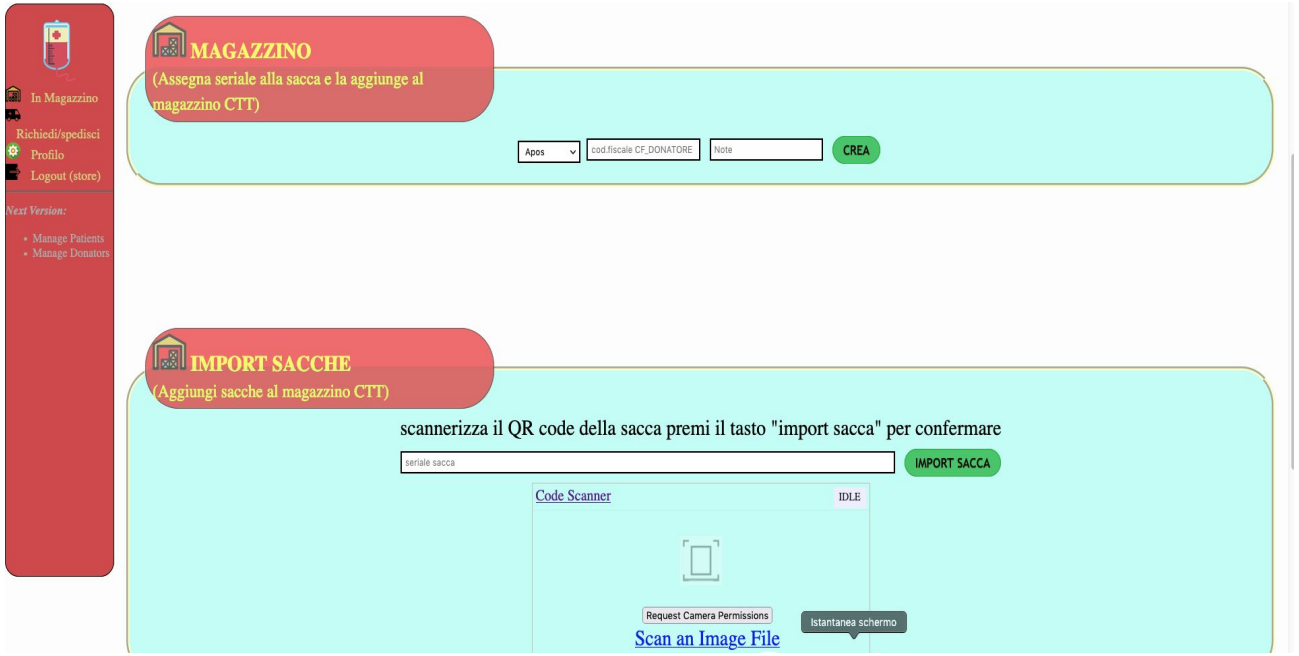
```
{
    timestamp: "2021-07-15T12:36:18.909+00:00",
    status: 403,
    error: "Forbidden",
    path: "/bloodbag/add"
}
```

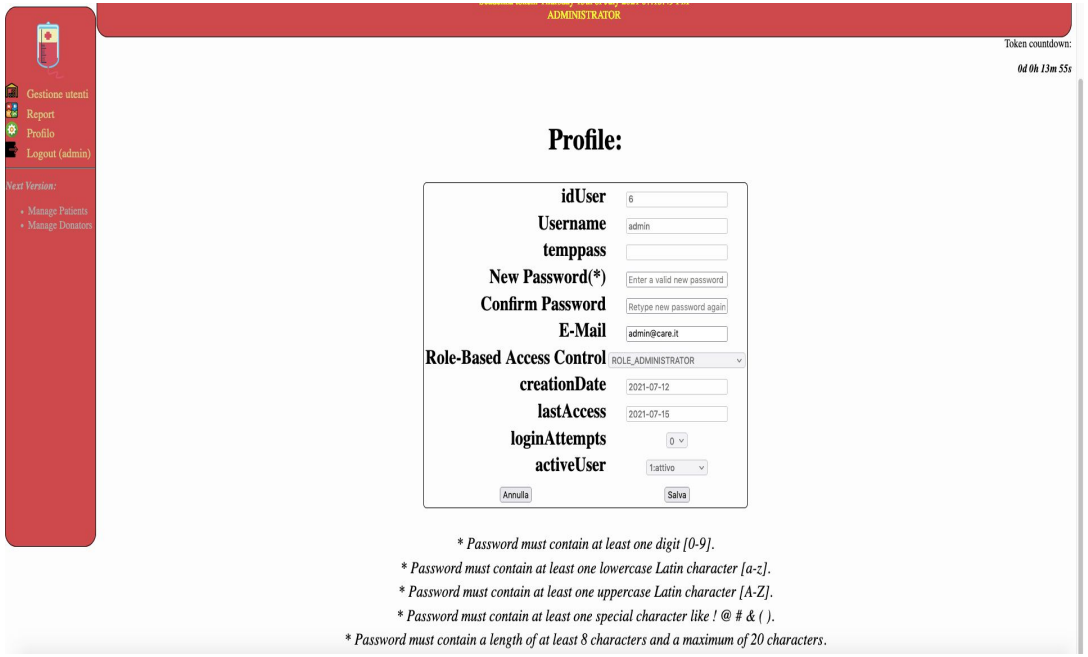
- Alcuni servizi per essere utilizzati hanno bisogno di una autenticazione jwt basata su ruolo.
- Ogni richiesta deve avere nell'header un token.
- SecurityContextHolder usa utilizza una variabile ThreadLocal per archiviare il securityContext, il che significa che il SecurityContext è sempre disponibile per i metodi nello stesso thread di esecuzione.

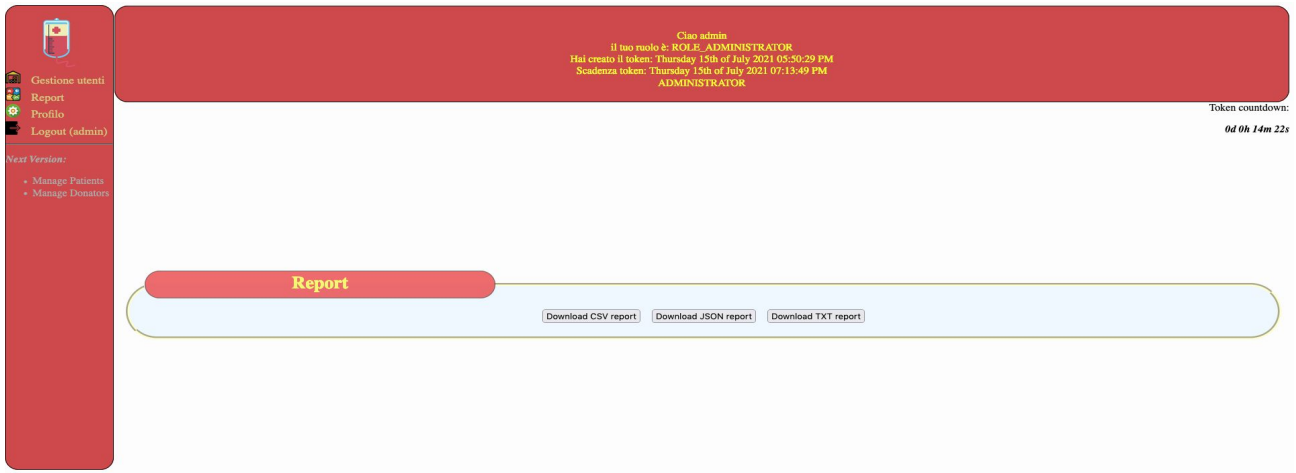
Graphic examples C.A.R.E



Graphic examples C.A.R.E







JUnit Test

```java
/**
    * Junit creation for verifying the insertion of an incorrect passwor
d combo through the static streams
    * @param password the password to be tested
    */
    @ParameterizedTest(name =
"#{index} - Run test with valid password complexity pattern = {0}")
    @MethodSource("validPasswordsProvider")
    public void test_password_regex_valid(String password) {
        Assert.assertTrue( Password.validatePlaintextPasswordPattern
(password) );
    }

    static Stream<String> validPasswordsProvider() {
        return Stream.of(
                "AAAbbbccc@123",
                "Hello world$123",
                "A!@#&()–a1",               // valid: punctuation part 1
                "A[{}]:;',?/*a1",           // valid: punctuation part 2
                "A~$^+=<>a1",               // valid: symbols
                "0123456789$abcdefgAB",     // valid: 20 chars
                "123Aa$Aa"                  // valid: 8 chars
        );
    }
```

Al fine di garantire maggiore qualità al committente, il sistema CARE è stato testato sia mediante l'approccio **glass-box** sia in **black-box**.

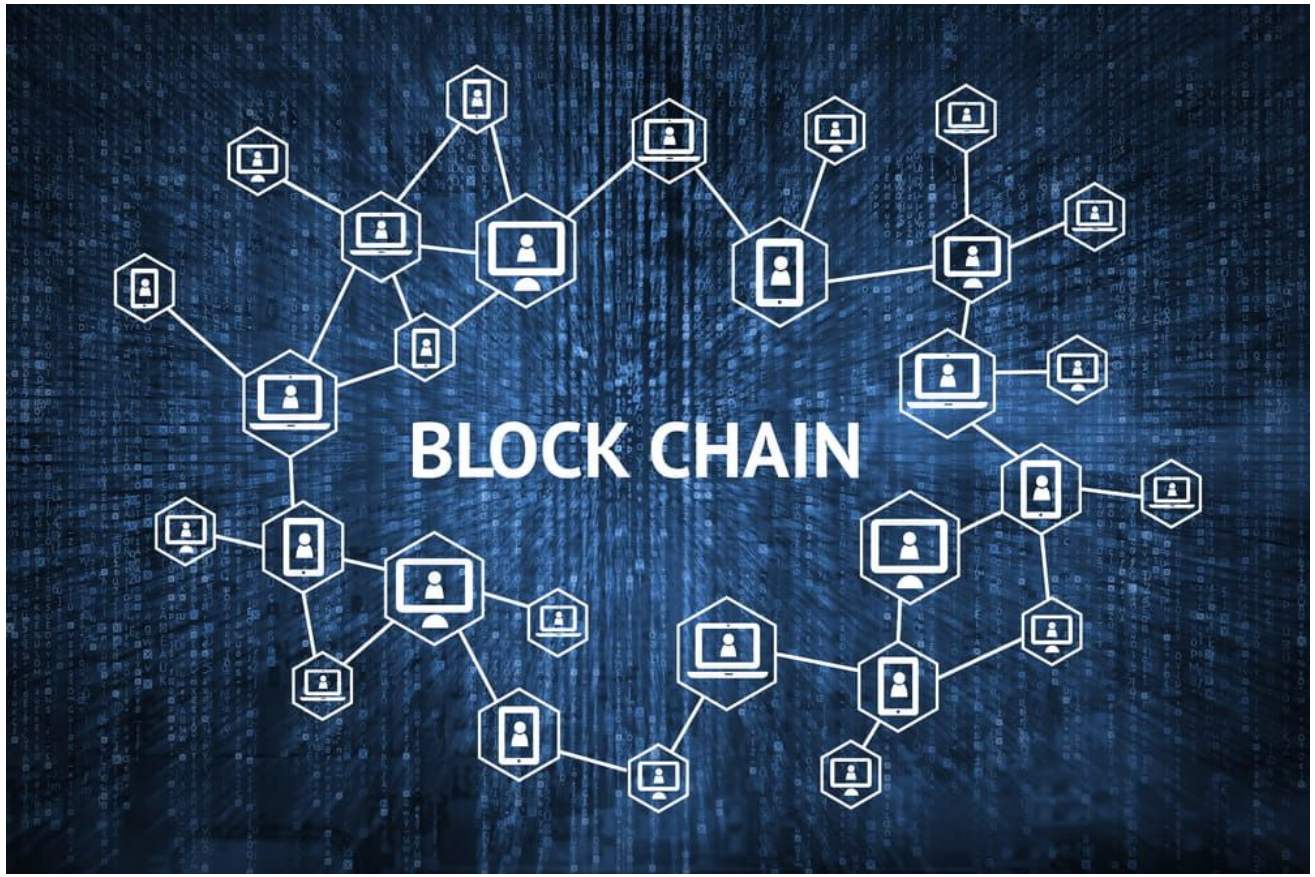
In **White-box** (o "glass-box") analizziamo la percentuale di **coverage** nonché i rami potenzialmente non coperti.

In **black-box**, nello specifico **JUnit 5,** si garantisce il testing dei casi d'uso non conoscendo le istruzioni che verranno eseguite, ma garantendo mediante una ipotesi che la premessa di utilizzo dei metodi ritorni quanto previsto.

Nell'esempio in questione i JUnit Test delle classi di modellazione base.
I test non garantiscono l'impeccabilità del codice, ma in modo ragionevole

RequestState.java	0,0 %	0	74	74
BloodBag.java	88,1 %	496	67	563
RequestPriority.java	0,0 %	0	34	34
Serial.java	93,1 %	231	17	248
BloodGroup.java	97,6 %	483	12	495
BloodBagState.java	100,0 %	64	0	64
it.unisannio.CARE.model.util.Logger	23,3 %	91	299	390
it.unisannio.CARE.model.util	81,6 %	1.053	237	1.290
XMLHelper.java	29,2 %	73	177	250
QRCode.java	88,0 %	300	41	341
Password.java	90,9 %	159	16	175
Constants.java	98,7 %	229	3	232
LabelGenerator.java	100,0 %	292	0	292

Future Implementations




thing...