ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

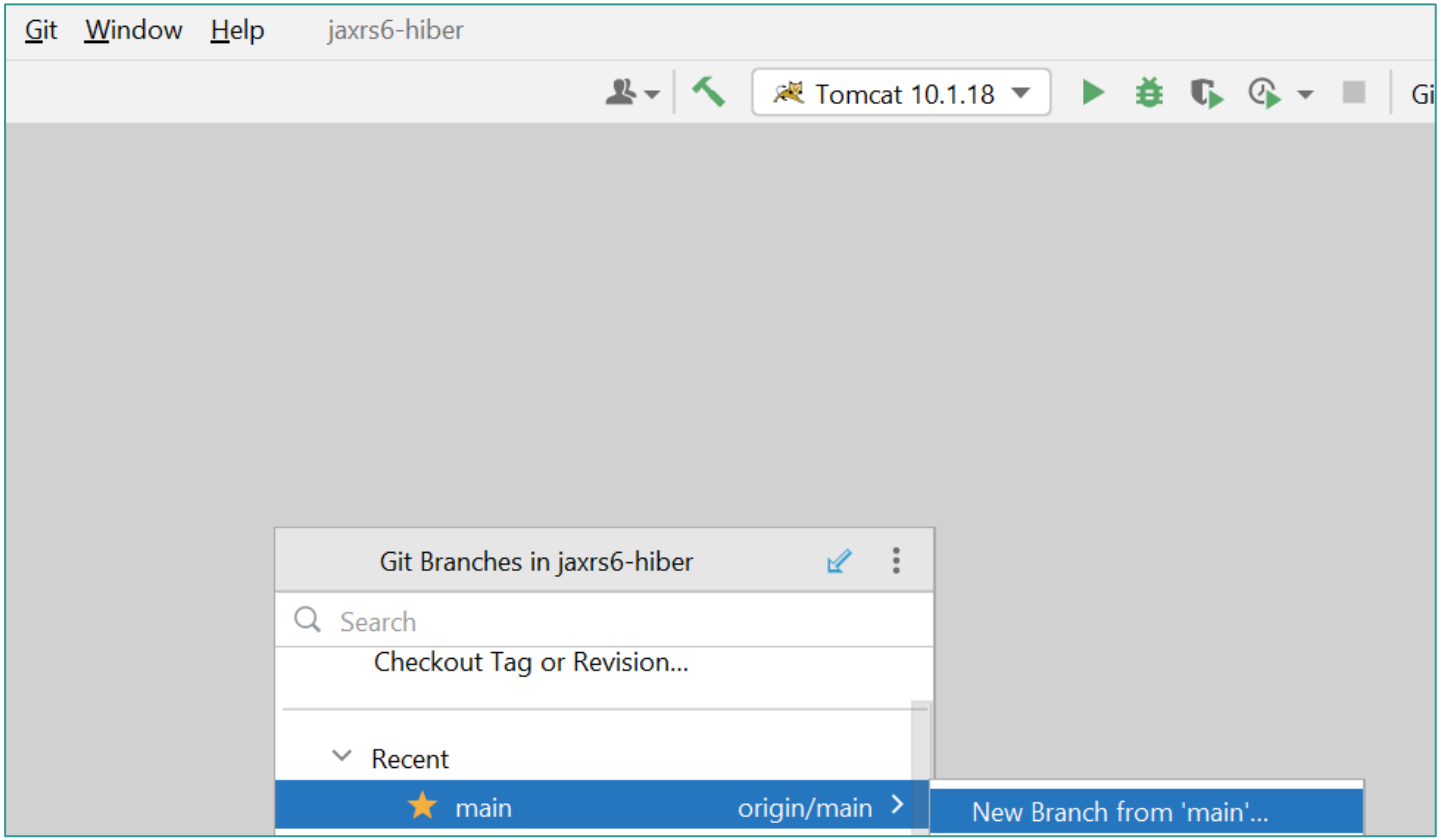
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Authentication με JSON Web Tokens Role-based Authorization

Αθ. Ανδρούτσος

Νέο branch auth



- Δημιουργούμε ένα νέο branch ξεκινώντας από το main. Το όνομα του νέου branch θα είναι auth και θα υλοποιήσουμε το authentication / authorization

Authentication

- Authentication σημαίνει να αποδείξει κανείς ότι είναι αυτός που ισχυρίζεται
- Το basic Authentication γίνεται τυπικά με username / password σε μία login page
- Το Authorization έχει να κάνει με τα δικαιώματα που έχει ένας logged-in user (view, edit, delete κλπ.)m

Principal

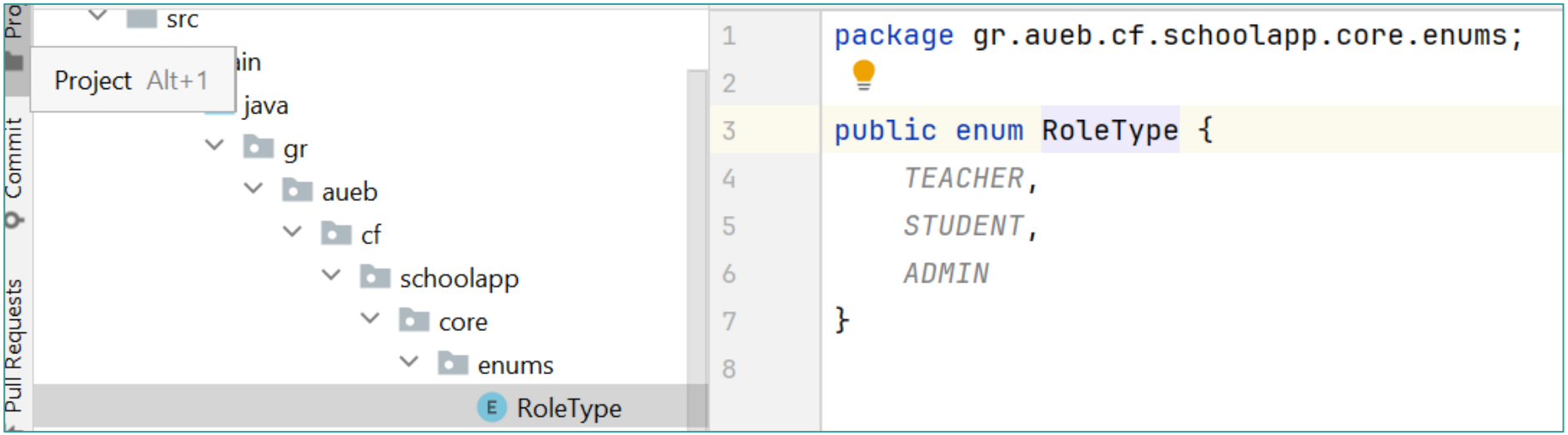
- Κεντρική δομή στα συστήματα authentication / authorization είναι ο principal που είναι ο logged-in user με τα στοιχεία που τον χαρακτηρίζουν
- Τα βασικό στοιχείο που χαρακτηρίζει έναν principal είναι το username ή γενικά κάποιο μοναδικό χαρακτηριστικό (id, vat, αν δεν υπάρχει username)
- Στην Jakarta ο **Principal είναι interface** που περιέχει το μοναδικό χαρακτηριστικό ενός χρήστη
- Θα πρέπει ο δικός μας User να κάνει implements τον Principal της Jakarta ώστε συνδέσουμε την πραγματική εφαρμογή με το προγραμματιστικό / συστημικό μέρος της Jakarta

User implements Principal


```java
package gr.aueb.cf.schoolapp.model;

import ...

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Builder
@Table(name = "users")
public class User extends AbstractEntity
        implements IdentifiableEntity, Principal {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;
    private String password;

    @Column(name = "role")
    @Enumerated(EnumType.STRING)
    private RoleType roleType;

    @Override
    public String getName() { return username; }
}
```

- Το Entity User υλοποιεί το interface Principal και επίσης το Identifiable Entity
- Η getName() που είναι μέθοδος του Principal έχει υλοποιηθεί ώστε να επιστρέφει το username
- Κρατάμε επίσης και τον ρόλο

Ρόλος



- Το Enum είναι όπως μία κλάση, αλλά περιέχει σταθερές. Ο ρόλος είναι ένα enum στο package core.enums
- Τα Enum είναι σταθερές που αντιστοιχούν σε ordinals, ξεκινώντας το 0. Συνεπώς, το TEACHER αντιστοιχεί στο 0, το STUDENT στο 1, το ADMIN στο 2

User


```
27      @Column(name = "role")
28      @Enumerated(EnumType.STRING)
29  a   private RoleType roleType;
30
```

- Αν δεν ορίσουμε στο Entity το enum να αποθηκεύεται ως String τότε αποθηκεύεται το ordinal
- Ωστόσο στον User ορίζουμε το enum value να αποθηκεύεται ως String
- Οπότε κατά το persist / update το TEACHER, STUDENT, ADMIN μετατρέπεται με την name() –που είναι σαν την toString- σε string "TEACHER", "STUDENT", "ADMIN"

User DAO


```
1   package gr.aueb.cf.schoolapp.dao;
2
3   import ...
6
7   public interface IUserDAO extends IGenericDAO<User> {
8       Optional<User> getByUsername(String username);
9       boolean isUserValid(String username, String password);
10      boolean isEmailExists(String username);
11  }
```

User DAO Impl


```java
package gr.aueb.cf.schoolapp.dao;

import ...

@ApplicationScoped
public class UserDAOImpl extends AbstractDAO<User> implements IUserDAO {

    @Override
    public Optional<User> getByUsername(String username) {
        String sql = "SELECT u FROM User u WHERE u.username = :username";

        try {
            User user = getEntityManager() EntityManager
                    .createQuery(sql, User.class) TypedQuery<User>
                    .setParameter("username", username)
                    .getSingleResult();
            return Optional.of(user);
        } catch (NoResultException e) {
            return Optional.empty();
        }
    }
```

- Η getByUsername() επιστρέφει ένα Optional<User> με βάση ένα username
- Αν υπάρχει το username επιστρέφεται ο User με Optional.of
- Αν δεν υπάρχει τότε η getSingleResult δίνει exception και επιστρέφουμε empty Optional

isUserValid


```
28      @Override
29      public boolean isUserValid(String username, String password) {
30          String sql = "SELECT u FROM User u WHERE u.username = :username";
31
32          try {
33              User user = getEntityManager() EntityManager
34                      .createQuery(sql, User.class) TypedQuery<User>
35                      .setParameter("username", username)
36                      .getSingleResult();
37              return SecUtil.checkPassword(password, user.getPassword());
38          } catch (NoResultException e) {
39              return false;
40          }
41      }
```

- Είναι βασική μέθοδος γιατί ελέγχει αν υπάρχει ο User με βάση το username και το password. Πρώτα ελέγχουμε το username και μετά αν υπάρχει το username, ελέγχουμε το password με **SecUtil.checkPassword** την οποία θα δούμε στην επόμενη διαφάνεια

SecUtil (1)


```java
package gr.aueb.cf.schoolapp.security;

import org.mindrot.jbcrypt.BCrypt;

public class SecUtil {

    private SecUtil() {

    }

    public static String hashPassword(String inputPasswd) {
        int workload = 12;
        String salt = BCrypt.gensalt(workload);
        return BCrypt.hashpw(inputPasswd, salt);
    }

    public static boolean checkPassword(String inputPasswd, String storedHashedPasswd) {
        return BCrypt.checkpw(inputPasswd, storedHashedPasswd);
    }
}
```

- Πρόκειται για δική μας Utility Κλάση που υλοποιεί την κρυπτογράφηση με τον αλγόριθμο Blowfish

SecUtil (2)

- Ο Blowfish είναι συμμετρικός (το ίδιο secret για κρυπτογράφηση και αποκρυπτογράφηση) αλγόριθμος κρυπτογράφησης που χρησιμοποιείται σε διάφορες εφαρμογές, όπως
  - Open SSL
  - Password Hashing
  - File Encryption
  - Network Security Protocols (VPN)

jBCrypt (1)

- Ο αλγόριθμος bcrypt προέρχεται από τον Blowfish αλλά έχει βελτιστοποιηθεί για **password hashing**
- Έχει εισάγει ένα work factor που εισάγει latency ώστε να είναι δύσκολο να σπάσει ο αλγόριθμος σε όρους χρόνου ιδιαίτερα με σύγχρονο hardware όπως GPUs
- Η Bcrypt αυτόματα εισάγει salt σε κάθε password πριν εξάγει το hashed password. Το salt είναι random data ώστε δύο ίδια password να έχουν τελικά διαφορετικό hash

jBCrypt (2)


```
<dependency>
    <groupId>org.mindrot</groupId>
    <artifactId>jbcrypt</artifactId>
    <version>0.4</version>
</dependency>
```

- Το dependency για τον jBCrypt που έχουμε εισάγει στο POM.xml είναι το παραπάνω
- Ο bcrypt προσφέρει one-way hashing, δεν είναι reversable

User DAO Impl


```
43      @Override
44      public boolean isEmailExists(String username) {
45          String sql = "SELECT COUNT(u) FROM User u WHERE u.username = :username";
46
47          try {
48              Long count = getEntityManager() EntityManager
49                      .createQuery(sql, Long.class) TypedQuery<Long>
50                      .setParameter("username", username)
51                      .getSingleResult();
52              return count > 0;
53          } catch (NoResultException e) {
54              return false;
55          }
56      }
57  }
```

UserInsertDTO


```java
package gr.aueb.cf.schoolapp.dto;

import ...

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class UserInsertDTO {

    @Email(message = "Invalid username")
    private String username;

    @Pattern(regexp = "^(?=.*?[a-z])(?=.*?[A-Z])(?=.*?\\d)(?=.*?[@#$!%&*]).{8,}$",
            message = "Invalid Password")
    private String password;

    @Pattern(regexp = "^(?=.*?[a-z])(?=.*?[A-Z])(?=.*?\\d)(?=.*?[@#$!%&*]).{8,}$",
            message = "Invalid Password")
    private String confirmPassword;

    @NotEmpty(message = "Role can not be empty")
    private String role;
}
```

UserLoginDTO


```java
package gr.aueb.cf.schoolapp.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class UserLoginDTO {
    private String username;
    private String password;
}
```

UserReadOnlyDTO


```java
package gr.aueb.cf.schoolapp.dto;

import ...

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class UserReadOnlyDTO {
    private Long id;
    private String username;
    private String password;
    private String role;
}
```

Mapper


```
51 @    public static User mapToUser(UserInsertDTO dto) {
52          return new User(null, dto.getUsername(), SecUtil.hashPassword(dto.getPassword()),
53                  RoleType.valueOf(dto.getRole()));
54      }
55
56 @    public static UserReadOnlyDTO mapToUserReadOnlyDTO(User user) {
57          return new UserReadOnlyDTO(user.getId(), user.getUsername(), user.getPassword(),
58                  user.getRoleType().name());
59      }
60  }
```

- Προσθέτουμε στον Mapper το mapping για τον User

User service interface (API)


```java
package gr.aueb.cf.schoolapp.service;

import ...

public interface IUserService {

    UserReadOnlyDTO insertUser(UserInsertDTO dto) throws AppServerException;
    UserReadOnlyDTO getUserByUsername(String username) throws EntityNotFoundException;
    boolean isUserValid(String username, String password);
    boolean isEmailExists(String username);
}
```

User Service Impl


```
1   package gr.aueb.cf.schoolapp.service;
2
3   import ...
20
21  @ApplicationScoped
22  @RequiredArgsConstructor(onConstructor = @__(@Inject))
23  public class UserServiceImpl implements IUserService {
24      private static final Logger LOGGER = LoggerFactory.getLogger(UserServiceImpl.class);
25      private final IUserDAO userDAO;
26
```

- Εισάγουμε Logger ενώ επίσης κάνουμε inject

Lombok @Slf4j


```
24    @Slf4j
25    public class UserServiceImpl implements IUserService {
26        //private static final Logger LOGGER = LoggerFactory.getLogger(UserServiceImpl.class);
27        private final IUserDAO userDAO;
```

```
private static final Logger log = LoggerFactory.getLogger(YourClassName.class);
```

- Μπορούμε με Lombok να πάρουμε τον Logger με την διαφορά ότι το όνομα του Logger δεν είναι customizable
- Τυπικά το @Slf4j δίνει μία δήλωση του Logger όπως η παραπάνω
- Δίνει ως όνομα το **log**, ενώ εμείς έχουμε κάνει customize το όνομα ως LOGGER

Insert User (Register)


```java
@Override
public UserReadOnlyDTO insertUser(UserInsertDTO dto)
        throws AppServerException {
    try {
        JPAHelper.beginTransaction();
        User user = Mapper.mapToUser(dto);
//          if (userDAO.getByUsername(dto.getUsername()).isPresent()) {
//              throw new EntityAlreadyExistsException("User", "User with username: " + dto.getUsername()
//              + " already exists");
//          }
        UserReadOnlyDTO readOnlyDTO = userDAO.insert(user) Optional<User>
                .map(Mapper::mapToUserReadOnlyDTO) Optional<UserReadOnlyDTO>
                .orElseThrow(() -> new AppServerException("User", "User with vat: " + dto.getUsername() +
                        " not inserted"));
        JPAHelper.commitTransaction();
        LOGGER.info("User with username: {} inserted", dto.getUsername());
        return readOnlyDTO;
    } catch (AppServerException e) {
        JPAHelper.rollbackTransaction();
        LOGGER.error("Error. User with username: {} not inserted", dto.getUsername());
        throw e;
    } finally {
        JPAHelper.closeEntityManager();
    }
}
```

- Με σχόλια είναι η λογική του Idempotency την οποία όπως θα υλοποιήσουμε στον Custom Validator

getByUsername


```
55      @Override
56      public UserReadOnlyDTO getUserByUsername(String username) throws EntityNotFoundException {
57          try {
58              JPAHelper.beginTransaction();
59
60              UserReadOnlyDTO userReadOnlyDTO = userDAO.getByUsername(username) Optional<User>
61                      .map(Mapper::mapToUserReadOnlyDTO) Optional<UserReadOnlyDTO>
62                      .orElseThrow(() -> new EntityNotFoundException("User", "User with username: " +
63                              username + " not found"));
64              JPAHelper.commitTransaction();
65              return userReadOnlyDTO;
66          } catch (EntityNotFoundException e) {
67              LOGGER.warn("Warning. User with username {} not found", username);
68              throw e;
69          } finally {
70              JPAHelper.closeEntityManager();
71          }
72      }
```

Other services


```
74      @Override
75      public boolean isUserValid(String username, String password) {
76          try {
77              JPAHelper.beginTransaction();
78              boolean isValid = userDAO.isUserValid(username, password);
79              JPAHelper.commitTransaction();
80              return isValid;
81          } finally {
82              JPAHelper.closeEntityManager();
83          }
84      }
85
86      @Override
87      public boolean isEmailExists(String username) {
88          try {
89              JPAHelper.beginTransaction();
90              boolean mailExists = userDAO.isEmailExists(username);
91              JPAHelper.commitTransaction();
92              return mailExists;
93          } finally {
94              JPAHelper.closeEntityManager();
95          }
96      }
```

Αρχιτεκτονική Auth

- Για να υλοποιήσουμε το Authentication θα χρειαστούμε
  - Ένα **authentication provider** που θα μας ενημερώνει αν ένα username υπάρχει στη ΒΔ μας
  - Ένα **JwtService** που θα μας παρέχει μεθόδους δημιουργίας JSON Web Token καθώς και validation του token που λαμβάνουμε καθώς και μεθόδους για extraction πληροφορίας από το token

Authentication Provider


```java
package gr.aueb.cf.schoolapp.authentication;

import gr.aueb.cf.schoolapp.dto.UserLoginDTO;
import gr.aueb.cf.schoolapp.service.IUserService;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import lombok.RequiredArgsConstructor;

@ApplicationScoped
@RequiredArgsConstructor(onConstructor = @__(@Inject))
public class AuthenticationProvider {

    private final IUserService userService;

    public boolean authenticate(UserLoginDTO userLoginDTO)  {
        return userService.isUserValid(userLoginDTO.getUsername(),
                userLoginDTO.getPassword());
    }
}
```

- Η authenticate επιστρέφει true/false καλώντας την userService.isUserValid()

JSON Web Token (1)

- Στα Web Services δεν υπάρχει η λογική του backend session object όπου κρατάγαμε πληροφορίες (state) για τον logged-in user όπως το username
- Στις εφαρμογές με Web Services υπάρχει μία λογική **stateless authentication**. Το backend ελέγχει τα login credentials του χρήστη και αν είναι έγκυρα, αποστέλλει πίσω ένα token (container) που περιέχει πληροφορίες για τον χρήστη, όπως username

JSON Web Token - Header

- Το JSON Web Token (JWT) είναι ένα string που περιέχει τρία μέρη (Header, Payload, Signature):
- **Header**: Περιέχει τον αλγόριθμο για hashing που συνήθως είναι ο HS256 (που χρησιμοποιεί τον SHA256 για τη δημιουργία ενός hash από τον header και το payload). Η μορφή του header είναι: **{"alg": "HS256", "typ": "JWT"}**. Το Header μετατρέπεται σε Base64URL μορφή. Αυτό σημαίνει ότι μπορεί εύκολα να αποκωδικοποιηθεί αλλά όχι να τροποποιηθεί χωρίς να ακυρωθεί η υπογραφή (το signature)

JWT– Payload (1)

- **Payload**: Περιέχει τα δεδομένα που μεταφέρουμε. Το payload ενός JWT (JSON Web Token) περιέχει τα **claims**, τα οποία είναι πληροφορίες σχετικά με τον χρήστη καθώς και πρόσθετα μεταδεδομένα. Το payload έχει κωδικοποίηση Base64URL, που σημαίνει ότι μπορεί εύκολα να αποκωδικοποιηθεί αλλά όχι να τροποποιηθεί χωρίς να ακυρωθεί η υπογραφή (το signature). Γιαυτό δεν πρέπει να περιέχει ευαίσθητα προσωπικά δεδομένα, όπως password. Για παράδειγμα η μορφή του payload μπορεί να είναι: {"sub": "thanasis", "iat": 1516239022 } . Το iat είναι issued at και είναι registered claim

JWT– Payload (2)

- Το payload περιλαμβάνει claims. Υπάρχουν τρεις τύποι claims
  - Registered
  - Public
  - Private

Registered Claims

- Τα registered claims είναι predefined claims που προτείνεται από το JWT Specification (RFC 7519) να ορίζονται στο JWT

Claim	Description
`iss`	Issuer: Identifies who issued the token (e.g., your authentication server).
`sub`	Subject: Identifies the subject (user or entity) the token is referring to (e.g., user ID).
`aud`	Audience: Identifies the intended audience of the token (e.g., your API).
`exp`	Expiration Time: The timestamp (in seconds) after which the token is considered expired.
`nbf`	Not Before: The timestamp before which the token is not valid.
`iat`	Issued At: The timestamp when the token was issued.
`jti`	JWT ID: A unique identifier for the token, useful for one-time tokens or token revocation.

Παράδειγμα σε εφαρμογή

```
{
   "iss": "cf.aueb.gr",
   "sub": "thanasis",
   "iat": 1634024506,
   "exp": 1634110906
}
```

Public Claims

- Τα public claims είναι proprietary claims της εφαρμογής, όπου ορίζουμε μεταδεδομένα που πιθανώς χρειαζόμαστε να μεταφέρει το JWT, όπως name, role, email, vat, κ.α.
- Αυτά τα claims μπορεί να βρίσκουν εφαρμογή σε πολλά apps, μπορεί δηλαδή πολλές εφαρμογές που λαμβάνουν από εμάς το JWT να τα χρησιμοποιούν

```
{
  "sub": "thanasis",
  "name": "Αθανάσιος Ανδρούτσος",
  "role": "admin",
  "email": "a8anassis@gmail.com"
}
```

Private Claims

- Τα private claims είναι proprietary claims της εφαρμογής, όπου ορίζουμε μεταδεδομένα που πιθανώς χρειαζόμαστε να μεταφέρει το JWT, όπως walletMoney, κ.α.
- Η διαφορά με τα public claims, είναι ότι τα private claims εφαρμόζονται μόνο σε συγκεκριμένες εφαρμογές και όχι γενικά

Hash-based signatures

- Η υπογραφή (**signature**) ή τεχνικά **MAC (Message Authentication Code)** είναι μια κρυπτογραφική λειτουργία που δημιουργεί μια "υπογραφή“, χρησιμοποιώντας ένα HMAC αλγόριθμο (Hash-based Message Authentication Code, όπως HS256, HS384, HS512).
- Οι υπογραφές (MACs) δημιουργούνται με one-way hashing algorithms. Δεν υπάρχει δηλαδή decrypt, παρά μόνο encrypt με ένα secret (το κλειδί της κρυπτογράφησης που είναι μυστικό)
- Σκοπός των hash-based signatures είναι να εγγυηθούν το **integrity** και το **authenticity** ενός μηνύματος

Integrity / Authenticity

- Το γενικό σενάριο είναι το εξής
  - Ο χρήστης (client) κάνει login και αν είναι επιτυχές λαμβάνει ένα MAC (signature) σε μορφή Base64URL (βλ. επόμενη διαφάνεια)
  - Σε κάθε επικοινωνία με τον Server, ο client στέλνει τα data και το signature.
  - Ο Server από τα data που λαμβάνει, υπολογίζει το signature χρησιμοποιώντας το secret και συγκρίνει με το signature που λαμβάνει. Αν δεν διαφέρουν τα data δεν έχουν γίνει tamper (αλλοιωθεί). Άρα εγγυάται το **integrity** (ακεραιότητα) των data. Επίσης, εφόσον τα signatures ταιριάζουν, υπάρχει εγγύηση ότι τα data είχαν σταλεί από κάποιον που γνώριζε το secret key, άρα εγγυάται το **authenticity**

Base64 vs Base64URL

- Το Base64 mapping scheme χρησιμοποιείται για την μετατροπή binary data (εικόνες, βίντεο, αρχεία pdf, κλπ.) σε text ώστε να μεταφερθούν με text-based πρωτόκολλα όπως το HTTP
- Χρησιμοποιεί 64 text χαρακτήρες ([a-z] [A-Z] [0-9] + / =) ενώ κωδικοποιεί 3 bytes σε 4 χαρακτήρες (6-bit encoding). Αν δεν είναι τα input data πολλαπλάσιο του 3-bytes χρησιμοποιείται ένας padding χαρακτήρας, το =
- Το Base64URL έχει την ίδια λογική αλλά χρησιμοποιεί χαρακτήρες που είναι URL-Safe. Αντί για + / = χρησιμοποιεί - _ και no padding (το = δεν χρησιμοποιείται στο Base64URL)

JWT Token


```
HMACSHA256(
    base64UrlEncode(header) + "." + base64UrlEncode(payload),
    secretKey
)
```

- Το Signature / MAC δημιουργείται όπως παραπάνω. Είναι μία one-way διαδικασία κρυπτογράφησης με το secret key των δύο Base64URL (Header και Payload)
- Αφού δημιουργηθεί το MAC και μετατραπεί και αυτό σε Base64URL προστίθεται στο τέλος του String
- Το τελικό string, που ονομάζεται JSON Web Token, έχει τρία μέρη που διαχωρίζονται με τελεία

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Secret Key Length

- Το secret μπορεί να έχει οποιοδήποτε μήκος αλλά σχετικά μεγάλο και τυχαίο ώστε να μην μπορεί εύκολα να βρεθεί με αλλεπάλληλες δοκιμές (brute-force)
- Για τον HS256 (HMAC-SHA256) προτείνεται ως ελάχιστο μήκος 256-bit κλειδί (32-bytes)
- Τα 32-bytes αντιστοιχούν σε 43 χαρακτήρες Base64URL

Python Script

main.py | Share | Run | Output

```python
import secrets
import base64

# Generate 32 random bytes (256 bits)
key = secrets.token_bytes(32)

# Base64URL encode it for easy use (URL safe, no padding)
base64_key = base64.urlsafe_b64encode(key).rstrip(b'=').decode
    ('utf-8')

print(f"Generated 32-byte key (Base64URL, 43 chars):
    {base64_key}")
```

```
Generated 32-byte key (Base64URL, 43 chars):
    FvArDZiJ1hvR9k3Ks1J6s8FqbmL6rRnlmTL5J3jNiT8

=== Code Execution Successful ===
```

- Με Python και τις βιβλιοθήκες secrets και base64 δημιουργούμε ένα random 32 bytes sequence από bytes (integers από 0 - 255) και στη συνέχεια δημιουργούμε το αντίστοιχο Base64URL. Με rstrip(b'=') διαγράφουμε τυχόν padding (με b'=' παίρνουμε το = σε μορφή byte). Με decode('utf-8') παίρνουμε το utf-8 string
- Σε ένα online python editor εκτελούμε και παίρνουμε ένα 32-byte key που είναι ένα string με 43-χαρακτήρες

JwtService (1)


```java
package gr.aueb.cf.schoolapp.security;

import ...

@ApplicationScoped
public class JwtService {

//    private String secretKey = System.getenv("SECRET_KEY");
//    private String secretKey = "FvArDZiJ1hvR9k3Ks1J6s8FqbmL6rRnlmTL5J3jNiT8";

//    Strong security 384-bits = 48 bytes = 64 Base64URL characters
    private String secretKey = "5ce98d378ec88ea09ba8bcd511ef23645f04cc8e70b9134b98723a53c275bbc5";
    private long jwtExpiration = 10800000;  // 3 hours in milliseconds

//    if use refresh expiration token
//    private long refreshExpiration = 604800000;
```

- Το JwtService επιστρέφει ένα JWT token, το επαληθεύει όταν το ξαναλάβει και κάνει extract τα περιεχόμενα του payload (βλ. επόμενη διαφάνεια)

JwtService (2)


```
34      public String generateToken(String username, String role) {
35          var claims = new HashMap<String, Object>();
36          claims.put("role", role);
37          return Jwts
38                  .builder()
39                  .setIssuer("self") // todo
40                  .setClaims(claims)
41                  .setSubject(username)
42                  .setIssuedAt(new Date(System.currentTimeMillis()))
43                  .setExpiration(new Date(System.currentTimeMillis() + jwtExpiration))
44                  .signWith(getSignInKey(), SignatureAlgorithm.HS256)
45                  .compact();
46      }
47
48  @   public boolean isTokenValid(String token, User user) {
49          final String subject = extractSubject(token);
50          return (subject.equals(user.getName())) && !isTokenExpired(token);
51      }
```

- Στο JWT έχουμε εισάγει το subject (username), το iat (currentTimeMillis), το exp (currentTimeMillis + exp), τον Issuer (θα είναι το deployment URL, π.χ. "https://api.aueb.gr")
- Η isTokenValid, ελέγχει αν το JWT είναι valid

JwtService (3)


```
53      public String extractSubject(String token) {
54          return extractClaim(token, Claims::getSubject);
55      }
56
57  @   public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
58          final Claims claims = extractAllClaims(token);
59          return claimsResolver.apply(claims);
60      }
61
62      private boolean isTokenExpired(String token) {
63          return extractExpiration(token).before(new Date());
64      }
65
66      private Date extractExpiration(String token) {
67          return extractClaim(token, Claims::getExpiration);
68      }
69
70      private Claims extractAllClaims(String token) {
71          return Jwts
72                  .parserBuilder()
73                  .setSigningKey(getSignInKey())
74                  .build()
75                  .parseClaimsJws(token)
76                  .getBody();
77      }
```

JWT Validation

- Το JWT που στέλνουμε στον Client αποθηκεύεται στο Local Storage του Browser (όχι αυτόματα, ο JS client το αποθηκεύει) ή στο Cookie αν χρησιμοποιούμε cookies με HTTP-only για security (να μην μπορεί να γίνει access από JS) και στη συνέχεια σε κάθε request στέλνεται και το JSON Web Token
- Στην πλευρά του Server μπορεί να έχουμε ένα **filter που να ελέγχει κάθε request αν έχει Authentication header που ξεκινάει με Bearer** και ακολουθεί το JSON Token. Άρα το JWT ξεκινάει από τη θέση 7 και μετά του συνολικού String, π.χ.

```
Bearer 3167b13d1f9fe8205fda45d31639d21aa497191d3cfc33fd25e0a4053cbbe11b
```

JWT Authentication Filter

- Αν το Authentication header που βρίσκεται μέσα σε ένα request ελεγχθεί από τον Server και έχει valid JWT, τότε θα πρέπει να δημιουργηθεί για αυτό το request ένα Security Context
- Θα δημιουργήσουμε εμείς ένα Custom Security Context ώστε η υλοποίηση του Security Context να έχει τον δικό μας Principal δηλ. τον δικό μας User

Security Context


```java
package gr.aueb.cf.schoolapp.security;

import ...

@AllArgsConstructor
public class CustomSecurityContext implements SecurityContext {

    private User user;

    @Override
    public Principal getUserPrincipal() { return user; }

    @Override
    public boolean isUserInRole(String role) {
        return user.getRoleType().name().equals(role);
    }

    @Override
    public boolean isSecure() { return false; }

    @Override
    public String getAuthenticationScheme() { return "Bearer"; }

    public String getUserRole() { return user.getRoleType().name(); }
}
```

- Το SecurityContext είναι ένα interface
- Είναι βασικά μια δομή στην οποία κρατάμε πληροφορίες για τον user του κάθε request
- Είναι αντίστοιχο του session object, όμως το SecurityContext είναι short-lived και παρέχει contextual info για τον Principal του κάθε request

JWT Authentication Filter (1)


```java
package gr.aueb.cf.schoolapp.authentication;

import ...

@Provider
@Priority(Priorities.AUTHENTICATION)
@RequiredArgsConstructor(onConstructor = @__(@Inject))
public class JwtAuthenticationFilter implements ContainerRequestFilter {

    private final IUserDAO userDAO;
    private final JwtService jwtService;

    @Context
    SecurityContext securityContext;

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
```

- Επειδή το filter είναι μέρος του JAX-RS Runtime, μεταξύ του Web Server και πρέπει να χαρακτηριστούν ως @Provider ώστε το JAX-RS runtime να δημιουργήσει ένα instance του φίλτρου, διαφορετικά δεν θα δημιουργηθεί

JWT Authentication Filter (2)


```
35      @Override
36  @   public void filter(ContainerRequestContext requestContext) throws IOException {
37
38          UriInfo uriInfo = requestContext.getUriInfo();
39
40          // getPath() returns the Controller/method path,
41          // e.g. "auth/register", not "api/auth/register"
42          String path = uriInfo.getPath();
43
44          if (isPublicPath(path)) {
45              return;
46          }
47
48          String authorizationHeader = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION);
49
50          if (authorizationHeader == null || !authorizationHeader.startsWith("Bearer ")) {
51              throw new NotAuthorizedException("Authorization header must be provided");
52          }
53          String token = authorizationHeader.substring("Bearer ".length()).trim();
54
```

- Το substring("Bearer".length()) επιστρέφει 7. Αφού το string ξεκινάει από το 0, η θέση 7 είναι η θέση που ξεκινάει το JWT

JWT Authentication Filter (3)


```
55          try {
56              String username = jwtService.extractSubject(token);
57
58              if (username != null && (securityContext == null
59                      || securityContext.getUserPrincipal() == null)) {
60                  User user = userDAO.getByUsername(username).orElse(null);
61                  if (user != null && jwtService.isTokenValid(token, user)) {
62                      requestContext.setSecurityContext(new CustomSecurityContext(user));
63                  } else {
64                      System.out.println("Token is NOT valid: "+ requestContext.getUriInfo());
65                  }
66              }
67          } catch (Exception e) {
68              throw new NotAuthorizedException("Invalid token");
69          }
70      }
71
72  @   private boolean isPublicPath(String path) {
73          // Define paths that should bypass the authorization filter
74          return path.equals("auth/register") || path.equals("auth/login");
75      }
76  }
```

Auth Rest Controller - register


```
1   package gr.aueb.cf.schoolapp.rest;
2
3   import ...
28
29  @ApplicationScoped
30  @RequiredArgsConstructor(onConstructor = @__(@Inject))
31  @Path("/auth")
32  public class AuthRestController {
33
34      private final IUserService userService;
35      private final AuthenticationProvider authProvider;
36      private final JwtService jwtService;
37
38      @POST
39      @Path("/register")
40      @Consumes(MediaType.APPLICATION_JSON)
41      @Produces(MediaType.APPLICATION_JSON)
42      public Response registerUser(UserInsertDTO userInsertDTO, @Context UriInfo uriInfo) {
43          UserReadOnlyDTO userReadOnlyDTO;
44
```

/register


```
45          List<String> beanErrors = ValidatorUtil.validateDTO(userInsertDTO);
46          if (!beanErrors.isEmpty()) return Response
47                  .status(Response.Status.BAD_REQUEST).entity(beanErrors).build();
48
49          Map<String, String > otherErrors = UserValidator.validate(userInsertDTO);
50          if (!otherErrors.isEmpty()) return Response
51                  .status(Response.Status.BAD_REQUEST).entity(otherErrors).build();
52
53          try {
54              userReadOnlyDTO = userService.insertUser(userInsertDTO);
55          } catch (Exception e) {
56              return Response
57                      .status(Response.Status.BAD_REQUEST)
58                      .entity("User Error in insert")
59                      .build();
60          }
61
62          return Response.created(uriInfo.getAbsolutePathBuilder()
63                          .path(userReadOnlyDTO.getId().toString())
64                          .build())
65                  .entity(userInsertDTO).build();
66      }
```

/login


```
68      @POST
69      @Path("/login")
70      @Consumes(MediaType.APPLICATION_JSON)
71      @Produces(MediaType.APPLICATION_JSON)
72      public Response login(UserLoginDTO loginDTO, @Context Principal principal) {
73          boolean isUserValid =  authProvider.authenticate(loginDTO);
74          if (!isUserValid) return Response
75                  .status(Response.Status.UNAUTHORIZED)
76                  .build();
77          try {
78              String username = principal.getName();
79              if (loginDTO.getUsername().equals(username)) {
80                  return Response.status(Response.Status.OK).entity("Already Authorized").build();
81              }
82              UserReadOnlyDTO userReadOnlyDTO = userService.getUserByUsername(loginDTO.getUsername());
83              String role = userReadOnlyDTO.getRole();
84              String token = jwtService.generateToken(loginDTO.getUsername(), role);
85              AuthenticationResponseDTO responseDTO = new AuthenticationResponseDTO(token);
86              return Response.status(Response.Status.OK).entity(responseDTO).build();
87          } catch (EntityNotFoundException e) {
88              return Response
89                      .status(Response.Status.UNAUTHORIZED)
90                      .build();
91          }
92      }
```

Authentication Response DTO


```
1   package gr.aueb.cf.schoolapp.authentication.dto;
2
3   import ...
7
8   @NoArgsConstructor
9   @AllArgsConstructor
10  @Getter
11  @Setter
12  public class AuthenticationResponseDTO {
13      private String token;
14  }
```

- Αυτό που επιστρέφεται από το /login είναι ένα AuthenticationResponseDTO

Register a Student

POST {{base_url}} /api/auth/register Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1  {
2      "username": "niki@aueb.gr",
3      "password": "Ath@n@s1os",
4      "confirmPassword": "Ath@n@s1os",
5      "role": "Student"
6  }
```

Body Cookies Headers (6) Test Results 201 Created 10.65 s 324 B

Pretty Raw Preview Visualize JSON

```
1  {
2      "username": "niki@aueb.gr",
3      "password": "Ath@n@s1os",
4      "confirmPassword": "Ath@n@s1os",
5      "role": "Student"
6  }
```

/login as a Student

POST {{base_url}} /api/auth/login Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "username": "niki@aueb.gr",
    "password": "Ath@n@s1os"
}
```

Body Cookies Headers (5) Test Results 200 OK 16.59 s 332 B Save Response

Pretty Raw Preview Visualize

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIjoiU3R1ZGVudCIsInN1YiI6Im5pa2lAYXVlYi5nciIsImlhdCI6MTcyODIyNzYyMCwiZXhwIjoxNzI4MjM4NDIwfQ.
6P5SkSKuTHoYIMuMNRQWneL4qIyzBxuICqJQ__Fkyrw"}

unauthorized

GET {{base_url}} /api/teachers/13 Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Cache-Control		no-cache	
Postman-Token		<calculated when request is sent>	
Host		<calculated when request is sent>	
User-Agent		PostmanRuntime/7.42.0	
Accept		*/*	
Accept-Encoding		gzip, deflate, br	
Connection		keep-alive	
Authorization		Bearer eyJhbGciOiJIUzI1NiJ9.eyJyb2xlIj...	
Key		Value	Description

Body Cookies Headers (7) Test Results 401 Unauthorized 2.18 s 950 B

Add a teacher

POST {{base_url}} /api/auth/register Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "username": "nikoleta@aueb.gr",
    "password": "Ath@n@s1os",
    "confirmPassword": "Ath@n@s1os",
    "role": "Teacher"
}
```

Body Cookies Headers (6) Test Results 201 Created 17.51 s 329 B

Pretty Raw Preview Visualize JSON

```
{
    "username": "nikoleta@aueb.gr",
    "password": "Ath@n@s1os",
    "confirmPassword": "Ath@n@s1os",
    "role": "Teacher"
}
```

/login as a Teacher

POST {{base_url}} /api/auth/login Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1  {
2      "username": "nikoleta@aueb.gr",
3      "password": "Ath@n@s1os"
4  }
```

Body Cookies Headers (5) Test Results 200 OK 1701 ms 337 B

Pretty Raw Preview Visualize


```
{"token":"eyJhbGciOiJIUzI1NiJ9.
eyJyb2xlIjoiVGVhY2hlciIsInN1YiI6Im5pa29sZXRhQGF1ZWIuZ3IiLCJpYXQiOjE3MjgyMjkwOTAsImV4cCI6MTcyODIzOTg5MH0.
Mg6y16nvinxigVAdua5rQuubttFfXFZ7aRJ2Krj9Aes"}
```

Εργασία

- Στην εφαρμογή που αναπτύξατε προσθέστε Authentication
- Τεστάρετε όλα τα use cases