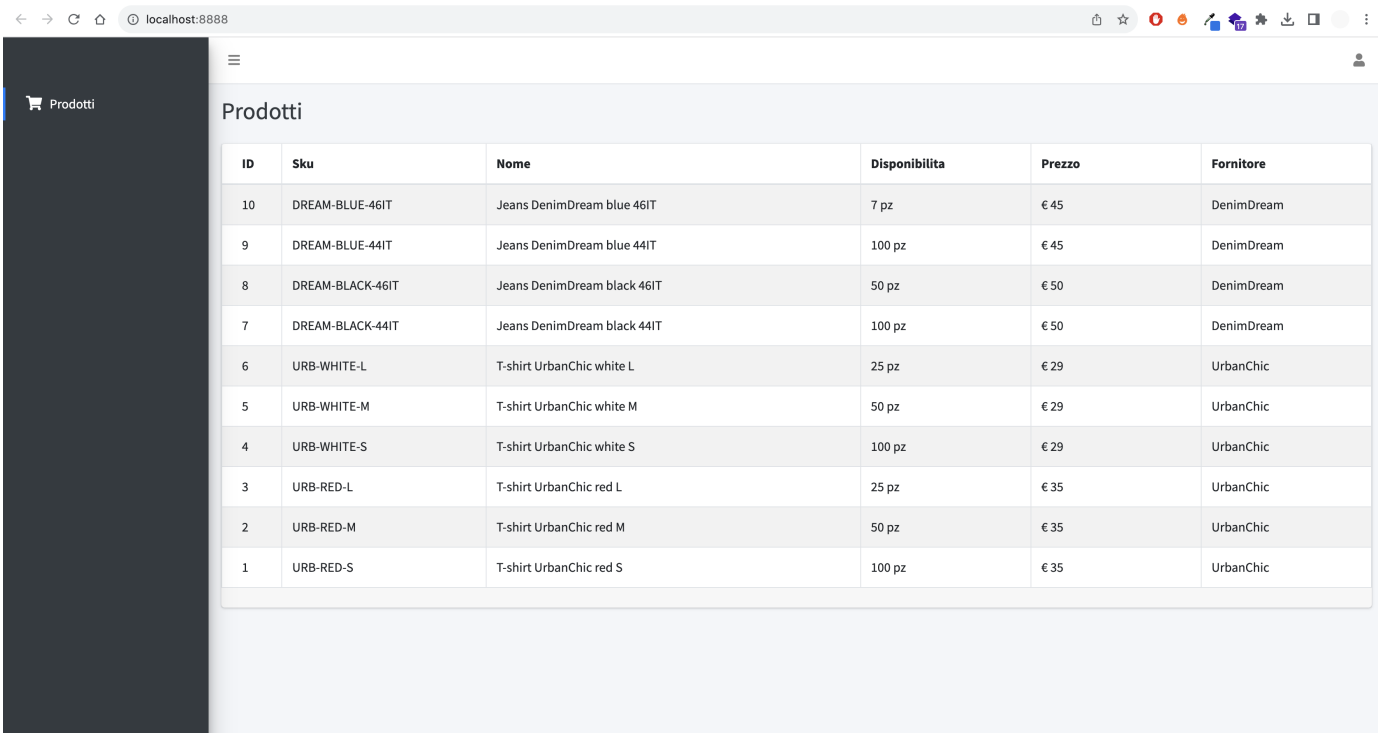
Presentazione

Introduzione

- **wls-productsales-app** è un esempio di ERP di wholesaling di base che consente agli utenti di autenticarsi con un server OAuth personalizzato e gestire diverse funzioni in base a diversi privilegi di accesso (chiamati scopes).
- **wls-auth-client** Un'applicazione PHP per l'autenticazione OAuth, che restituisce codici di autorizzazione e nomi utente per gli utenti autenticati a un endpoint modificabile
- **wls-backend-server** Server di autenticazione OAuth e backend del sistema di vendita all'ingrosso in TypeScript

Home

La home di **wls-productsales-app** non richiede autenticazione e permette a chi si collega di visualizzare i prodotti in vendita



Comunizazione con wls-backend-server

Di seguito è presente un esempio di comunicazione base tra wls-productsales-app e wls-backend-server. In questo caso viene mostrato l'ottenimento dei prodotti da parte del frontend al backend:

- ```
  public function getProdotti() {
  ```
  ```

```
    try {
        $result = $this->server->getProdotti();
        return $result["data"];
    } catch(Exception $e) {
        $this->failure( message: $e->getCode() . " " . $e->getMessage());
    }
}
```

```
public function getProdotti() {

    $url = $this->host . "wholesales/prodotti";

    try {
        $result = $this->get($url);
        return $result;

    } catch(Exception $e) {
        throw $e;
    }
}
```

```
private function get($url, $token = "", $data = array()) {

    $header = array(
        "ClientId: " . CLIENT_ID,
        "ClientSecret: " . CLIENT_SECRET,
        "Authorization: Bearer " . $token
    );

    $result = $this->http_request->get($url, $header, $data);

    if(isset($result["error"]))
        throw new Exception($result["error"], $result["httpCode"]);
    else
        return $result;
}
```

Per ogni tipo di endpoint (oauth, utenti, wholesaling), il flusso di esecuzione nel server di backend è strutturato nelle seguenti sezioni:

1. routes
   Definizione degli endpoint (get/post)

```
router.get( path: "/", auth.jwtAccessTokenAuthMiddleware, controller.listVendite);
router.get( path: "/testKey", auth.simpleAuthMiddleware, controller.testKey);
router.post( path: "/vendite/new/", auth.jwtAccessTokenAuthMiddleware, controller.newVendita);
router.get( path: "/vendite/:idVendita", auth.jwtAccessTokenAuthMiddleware, controller.getVendita);
router.get( path: "/vendite/venditore/:idVenditore", auth.jwtAccessTokenAuthMiddleware, controller.getVenditeVenditore);
router.get( path: "/vendite/cliente/:idCliente", auth.jwtAccessTokenAuthMiddleware, controller.getVenditeCliente);
router.get( path: "/vendite/prodotto/:idProdotto", auth.jwtAccessTokenAuthMiddleware, controller.getVenditeProdotto);
router.post( path: "/vendite/edit/", auth.jwtAccessTokenAuthMiddleware, controller.editVendita);
router.post( path: "/vendite/delete/", auth.jwtAccessTokenAuthMiddleware, controller.deleteVendita);

router.get( path: "/prodotti", auth.simpleAuthMiddleware, controller.listProdotti);
router.post( path: "/prodotti/new/", auth.jwtAccessTokenAuthMiddleware, controller.newProdotto);
router.get( path: "/prodotti/:idProdotto", auth.jwtAccessTokenAuthMiddleware, controller.getProdotto);
router.post( path: "/prodotti/edit/", auth.jwtAccessTokenAuthMiddleware, controller.editProdotto);
router.post( path: "/prodotti/delete/", auth.jwtAccessTokenAuthMiddleware, controller.deleteProdotto);

module.exports = router;
```

2. middleware di autenticazione
   In questa schermata è presente il middleware di autenticazione semplice che controlla le chiavi CLIENTID e CLIENTSECRET

```
export async function simpleAuthMiddleware(req: Request, res: Response, next: any) {
  const clientId = req.header( name: "clientId");
  const clientSecret = req.header( name: "clientSecret");

  const query = SqlString.format(
      sql: `SELECT id, nome, data_registrazione, stato FROM client WHERE clientId = ? AND clientSecret = ?`,
      args: [clientId, clientSecret],
  );

  db.db.query(query, function (err: unknown, results: any) {
    if (err) {
      res.status( code: 500).json( body: { error: "Auth: Internal server error" });
      return;
    } else if (results.length === 0) {
      res.status( code: 404).json( body: { error: "Auth: Client not found" });
      return;
    } else if (!results[0].stato) {
      res.status( code: 401).json( body: { error: "Auth: Unauthorized" });
      return;
    } else {
      next();
    }
  });
}
```

3. controller
   Gestisce le richieste HTTP e le operazioni necessarie per soddisfare una richiesta tra cui verifiche di sicurezza, e poi delega l'effettiva estrazione/modifica dei dati al provider. Infine, il controller restituisce una risposta HTTP al client.

```
export async function listProdotti(req: Request, res: Response, next: NextFunction) {
    try {
        const result = await provider.listProdotti();
        res.json(result);
    } catch (err) {
        next(err);
    }
}
```

4. provider
   Svolge un ruolo intermedio tra il controller e il datamodel. Se necessario segue controlli basati su token (verifica dei permessi) per garantire che l'utente abbia i diritti necessari per effettuare la modifica. In base ai permessi dell'utente, il provider può chiamare funzioni specifiche del datamodel per eseguire l'effettiva modifica dei dati.

```
export async function listProdotti() {
    return datamodel.getProdotti();
}
```

5. datamodel
   E' la parte più bassa dell'architettura ed è responsabile dell'accesso diretto al database o ai dati sottostanti e si preoccupa principalmente delle operazioni CRUD (Create, Read, Update, Delete) sul database.

```
export function getProdotti() {
    // tslint:disable-next-line:typedef
    return new Promise<any[]>( executor: function (resolve, reject) {
        const query = SqlString.format(
            sql: `SELECT * FROM prodotti ORDER BY id DESC`,
            args: [],
        );


        db.db.query(query, function (err: Error | null, results: any) {
            if (err) {
                return reject(new InternalServerError(err.message));
            }

            resolve(results);
        });
    });
}
```

Autenticazione con Oauth

L'autenticazione è obbligatoria per poter contattare gli endpoint che sono protetti da middleware che richiedono un'autenticazione più forte rispetto alla semplice autenticazione con CLIENTID e CLIENTSECRET.

In particolare, se l'utente vuole loggarsi, **wls-productsales-app** effettua un redirect a **wls-auth-client**

```
public function startOauthAuthentication() {
    header(
        header: "Location: ". OAUTH_FRONTEND .
        "?callback=" . base64_encode( string: base_url() . "/oauth_callback.php") .
        "&clientId=" . CLIENT_ID .
        "&clientSecret=" . CLIENT_SECRET
    );
    exit();
}
```

wls-auth-client verifica le chiavi CLIENTID e CLIENTSECRET

```
public function getClientInfo() {
    $url = $this->host . "oauth/clientInfo/";

    $data = array(
        "clientId" => $this->clientId,
        "clientSecret" => $this->clientSecret
    );

    try {
        $result = $this->post($url, token: "", $data);
        return $result;

    } catch(Exception $e) {
        throw $e;
    }
}
```

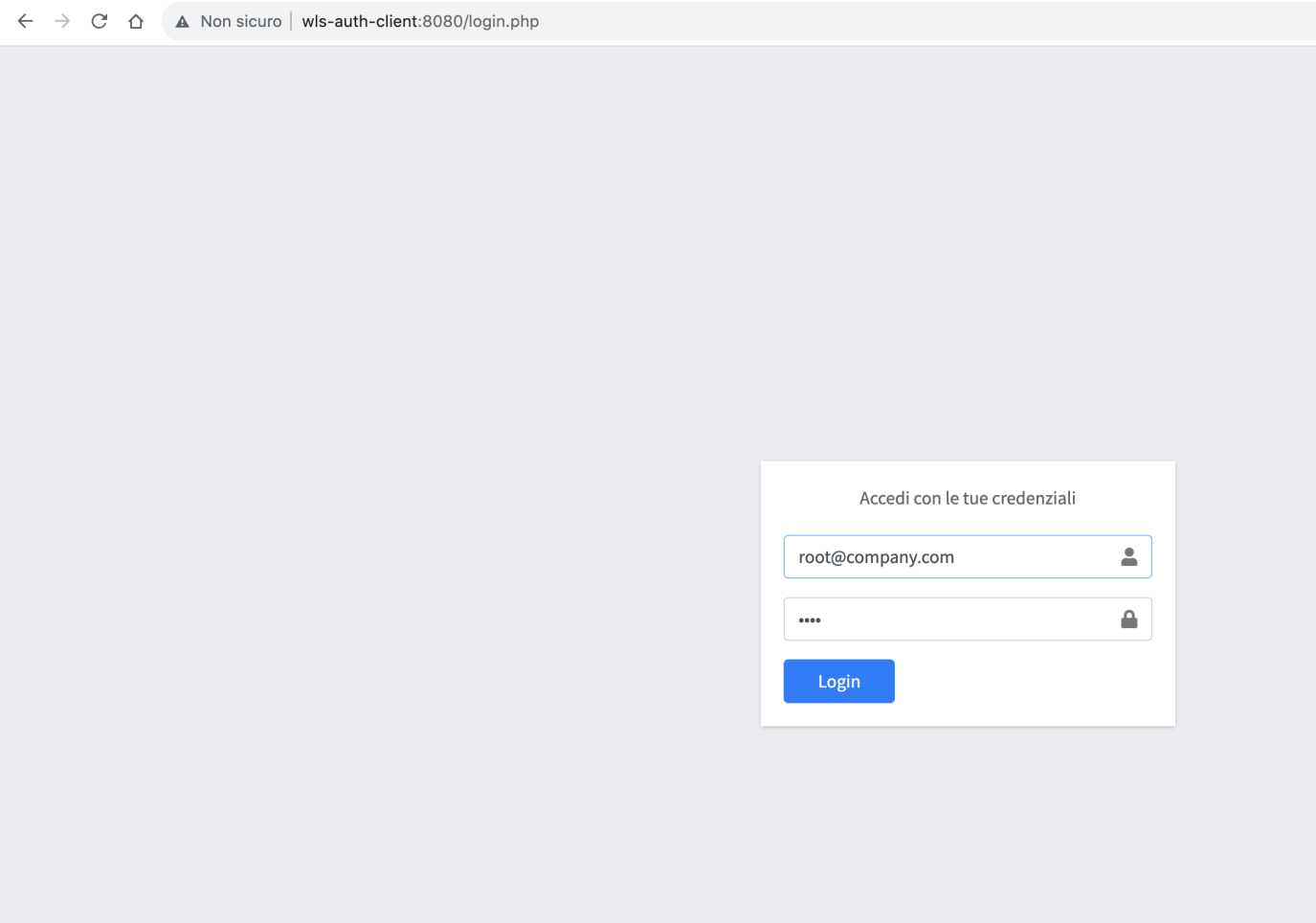
```
export function getClientInfo(clientId: string, clientSecret: string) {
  return new Promise( executor: function (resolve, reject) {
    const query = SqlString.format(
        sql: `SELECT id, nome, data_registrazione, stato FROM client WHERE clientId = ? AND clientSecret = ?`,
        args: [clientId, clientSecret],
    );

    db.db.query(query, function (err: Error | null, results: any) {
      if (err) {
        return reject(new InternalServerError(err.message));
      }

      if (results.length === 0) {
        reject(new NotFound( msg: "Client not found"));
        return;
      }

      resolve(results);
    });
  });
}
```

Se la verifica va a buon fine allora all'utente si apre la pagina dove è possibile inserire le credenziali



Vengono verificate le credenziali e viene creato un primo token JWT (userInfoToken) contenente i dati degli utenti e i propri scopes

```
export function getUserInfo(username: string, password: string): Promise<string> {
  return new Promise<string>( executor: (resolve, reject) => {

    const query = SqlString.format(
        sql: `SELECT id, nome, cognome, username, type FROM utenti WHERE username = ? AND password = ?`,
        args: [username, password],
    );

    db.db.query(query, function (err: Error | null, results: any) {
      if (err) {
        return reject(new InternalServerError(err.message));
      }

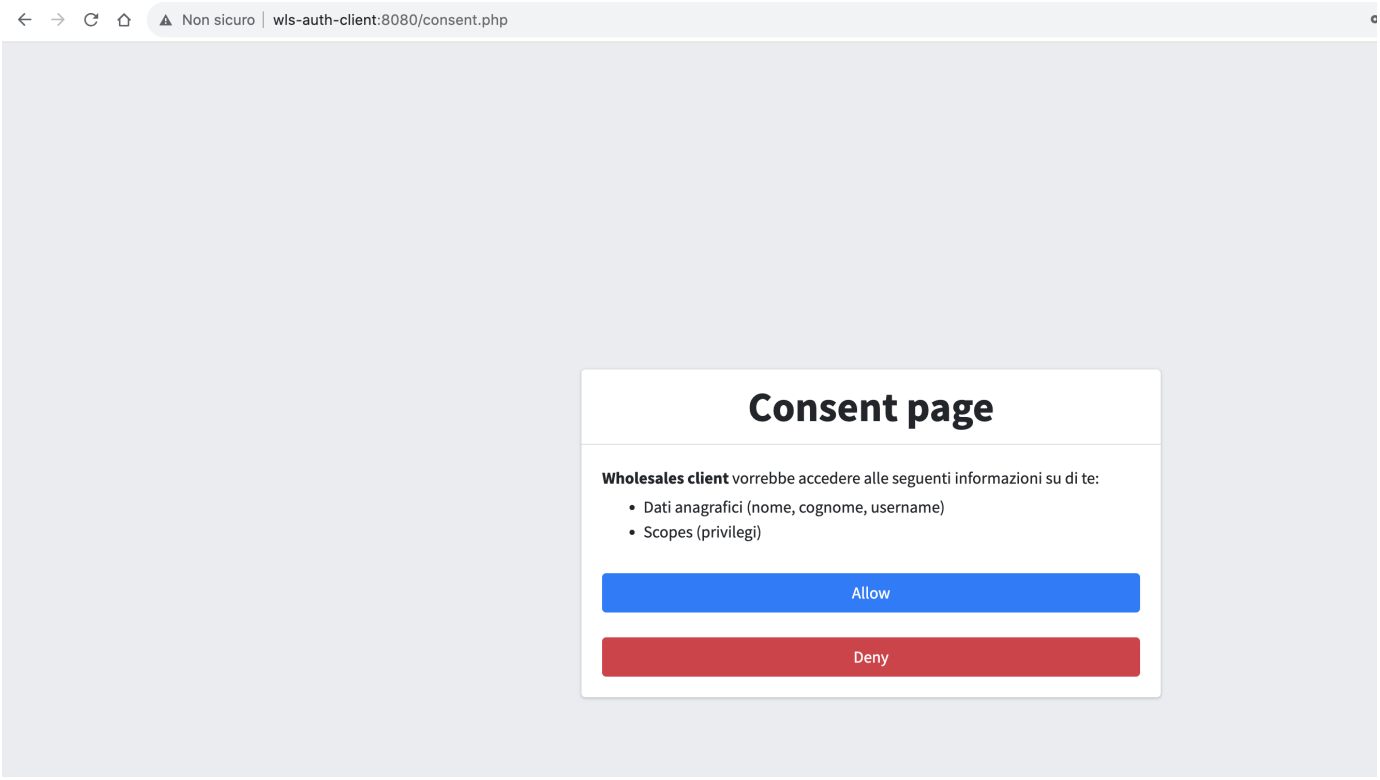
      if (results.length === 0) {
        reject(new NotFound( msg: "User not found"));
        return;
      } else {
        const user = results[0];
        const scopes = oauthScopes.getScopesForType(user.type);

        const payload = {
          tokenType: "userInfo",
          userInfo: user,
          scopes: scopes,
        };

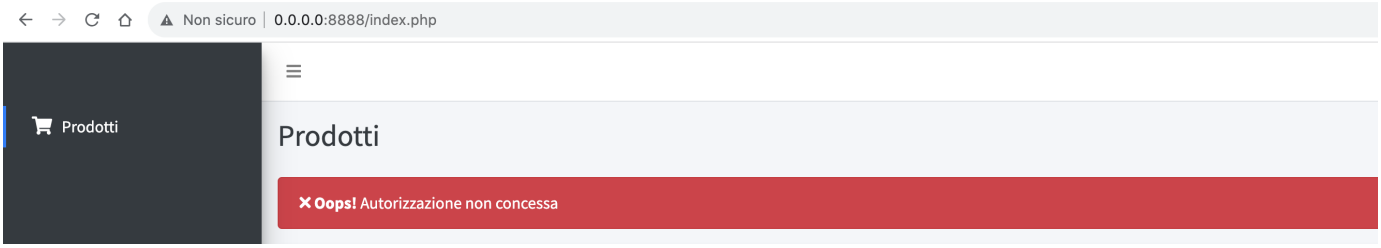
        const expiresIn = "1m";

        jwtUtils.getEncryptedToken(payload, expiresIn) Promise<string>
            .then((token: string) => {
              resolve(token);
            }) Promise<void>
            .catch((error: any) => {
              reject(error);
            });
      }
```

Se le credenziali sono valide, l'utente vede la pagina di consenso dove può scegliere se approvare o negare l'accesso ai propri dati



Es. Accesso negato



Es. Accesso approvato, l'userInfoToken viene scambiato per un authorizationCode, il quale viene restituito ad un endpoint di callback di **wls-productsales-app**. L'authorizationCode è un altro token JWT che contiene id, username dell'utente e gli scopes

```php
if(isset($_POST["allow"])) {
    $authorizationCode = null;
    try {
        $authorizationCode = $_SESSION["oauthServerConnector"]->getAuthorizationCode($_SESSION["userInfoToken"]);

        $url = base64_decode(
                $_SESSION["callback"]) .
                "?username=" . base64_encode($_SESSION["username"]) .
                "&authorizationCodeEncoded=" . base64_encode($authorizationCode["data"]
            );

        header( header: 'Location: ' . $url);
        exit;

    } catch(Exception $e) {
        $httpCode = $e->getCode();
        $error = $e->getMessage();

        $_SESSION['consent_failure'] = $httpCode . " " . $error;
        header( header: 'Location: consent.php');
        exit;
    }
}

if(isset($_POST["deny"])) {
    $url = base64_decode($_SESSION["callback"]) . "?deny=true";

    header( header: 'Location: ' . $url);
    exit;
}
```

```typescript
export function getAuthorizationCode(userInfoToken: string): Promise<string> {
  return new Promise<string>( executor: async (resolve, reject) => {

    try {
      const decryptedUserInfo = await jwtUtils.getDecryptedToken(userInfoToken);
      jwtUtils.checkTokenType(decryptedUserInfo,  type: "userInfo");

      const payload = {
        tokenType: "authorizationCode",
        id: decryptedUserInfo.userInfo.id,
        username: decryptedUserInfo.userInfo.username,
        scopes: decryptedUserInfo.scopes,
      };

      const expiresIn = "1m";

      jwtUtils.getEncryptedToken(payload, expiresIn) Promise<string>
          .then((token: string) => {
            resolve(token);
          }) Promise<void>
          .catch((error: any) => {
            reject(error);
          });
    } catch (error) {
      reject(error);
    }
  });
}
```

wls-productsales-app lo invierà a wls-backend-server per scambiarlo con un primo access token ed un refresh token.

wls-productsales-app userà gli access token come chiave API Bearer per tutte le chiamate agli endpoint di wls-backend-server che lo richiedono ed userà il refresh token (se ancora valido) per ottenere nuovi access token.

Con il primo access token, wls-productsales-app chiederà a wls-backend-server alcuni dati che servono per inizializzare l'applicativo di visualizzazione e gestione dei prodotti/vendite.

N.B. Il refresh token e l'expiration vengono salvati nei cookie del browser in modo da consentire un ripristino della sessione qualora il refresh token fosse ancora valido. In qualsiasi altro caso, all'utente viene richiesto di autenticarsi di nuovo.

```php
require_once BASE_PATH . '/lib/AuthManager/AuthManager.php';
session_start();


if($_SERVER["REQUEST_METHOD"] !== "GET")
    die("Method not allowed");

$_SESSION['auth_manager'] = new AuthManager();
if(isset($_GET["authorizationCodeEncoded"]) && isset($_GET["username"])) {
    $result = $_SESSION['auth_manager']->exchangeAuthorizationCodeAccessToken(base64_decode($_GET["authorizationCodeEncoded"]));
    $_SESSION['auth_manager']->setScopes($result["accessToken"], $result["refreshTokenExpiration"]);

    $access_token = $_SESSION['auth_manager']->getAccessToken($result["refreshToken"]);
    $_SESSION['auth_manager']->setUserData(base64_decode($_GET["username"]), $access_token, $result["refreshTokenExpiration"]);

    $_SESSION['user_logged_in'] = true;

    header( header: "Location: index.php");
} else if(isset($_GET["deny"])) {
    $_SESSION['auth_manager']->failure( message: "Autorizzazione non concessa",  location: "index.php");
}
```

```
export function exchangeAuthorizationCodeAccessToken(clientId: string, authorizationCode: string): Promise<{ accessToken. string, re
  return new Promise<{ accessToken: string, refreshToken: string, refreshTokenExpiration: string }>( executor: async (resolve, reject)

    const decryptedAuthorizationCode = await jwtUtils.getDecryptedToken(authorizationCode);
    jwtUtils.checkTokenType(decryptedAuthorizationCode, type: "authorizationCode");

    const accessTokenPayload = {
      tokenType: "accessToken",
      id: decryptedAuthorizationCode.id,
      username: decryptedAuthorizationCode.username,
      scopes: decryptedAuthorizationCode.scopes,
    };

    const refreshTokenPayload = {
      tokenType: "refreshToken",
      id: decryptedAuthorizationCode.id,
      username: decryptedAuthorizationCode.username,
      scopes: decryptedAuthorizationCode.scopes,
    };

    const accessTokenExpiresIn = "1m";
    const refreshTokenExpiresIn = "1h";

    const createAccessTokenPromise = jwtUtils.getEncryptedToken(accessTokenPayload, accessTokenExpiresIn);
    const createRefreshTokenPromise = jwtUtils.getEncryptedToken(refreshTokenPayload, refreshTokenExpiresIn);

    const refreshTokenExpiration = format(addHours(new Date(), amount: 1), format: "yyyy-MM-dd HH:mm:ss");

    Promise.all( values: [createAccessTokenPromise, createRefreshTokenPromise]) Promise<(Awaited<string>)[]>
        .then(([accessToken : string , refreshToken : string ]) => {
          jwtUtils.addUniqueAccessToken(clientId, accessToken);
          resolve( value: {accessToken, refreshToken, refreshTokenExpiration});
        }) Promise<void>
        .catch((error: any) => {
          reject(error);
        });
```

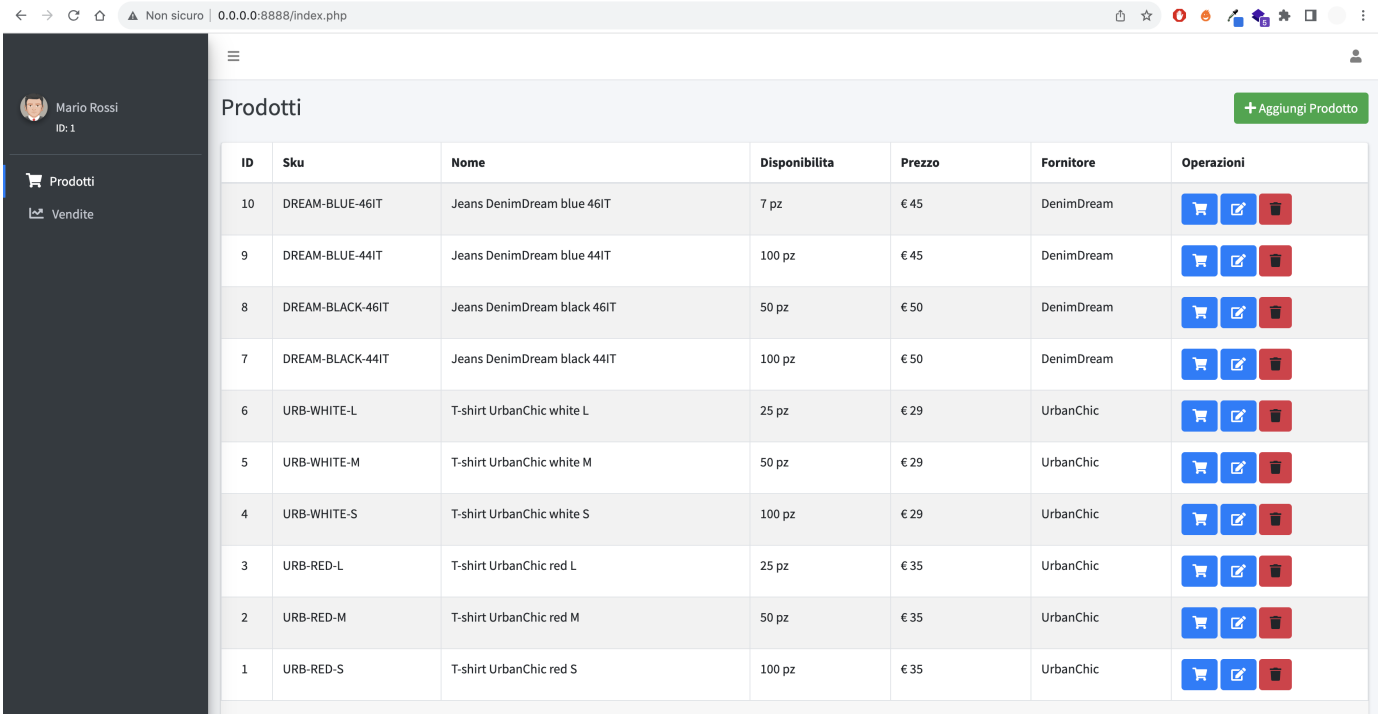
```
export function getUserScopes(accessToken: string): Promise<string[]> {
  return new Promise<string[]>( executor: async (resolve, reject) => {
    try {
      const decryptedAccessToken = await jwtUtils.getDecryptedToken(accessToken);

      const userScopes: string[] = decryptedAccessToken.scopes;
      resolve(userScopes);
    } catch (error) {
      reject(error);
    }
  });
}
```

```php
public function getAccessToken($refreshTokenInput = null) {
    $refreshToken = "";
    if($refreshTokenInput == null && !$this->checkRefreshToken())
        $this->startUserAuthentication();
    else if($refreshTokenInput != null)
        $refreshToken = $refreshTokenInput;
    else
        $refreshToken = $this->getRefreshToken();

    try {
        if(!$refreshToken)
            throw new Exception( message: "Invalid refresh token",  code: 500);

        $result = $this->oauthServerConnector->exchangeRefreshTokenAccessToken($refreshToken);

        $this->setRefreshToken($result["data"]["refreshToken"], $result["data"]["refreshTokenExpiration"]);

        return $result["data"]["accessToken"];
    } catch(Exception $e) {
        $this->failure( message: $e->getCode() . " " . $e->getMessage(),  location: "index.php");
    }
}
```

```typescript
export function exchangeRefreshTokenAccessToken(clientId: string, refreshToken: string): Promise<{ accessToken: string,  4  1  4
  return new Promise<{ accessToken: string, refreshToken: string, refreshTokenExpiration: string }>( executor: async (resolve, reject)

    const decryptedRefreshToken = await jwtUtils.getDecryptedToken(refreshToken);
    jwtUtils.checkTokenType(decryptedRefreshToken,  type: "refreshToken");

    const accessTokenPayload = {
      tokenType: "accessToken",
      id: decryptedRefreshToken.id,
      username: decryptedRefreshToken.username,
      scopes: decryptedRefreshToken.scopes,
    };

    const refreshTokenPayload = {
      tokenType: "refreshToken",
      id: decryptedRefreshToken.id,
      username: decryptedRefreshToken.username,
      scopes: decryptedRefreshToken.scopes,
    };

    const accessTokenExpiresIn = "1m";
    const refreshTokenExpiresIn = "1h";

    const createAccessTokenPromise = jwtUtils.getEncryptedToken(accessTokenPayload, accessTokenExpiresIn);
    const createRefreshTokenPromise = jwtUtils.getEncryptedToken(refreshTokenPayload, refreshTokenExpiresIn);

    const refreshTokenExpiration = format(addHours(new Date(),  amount: 1),  format: "yyyy-MM-dd HH:mm:ss");

    Promise.all( values: [createAccessTokenPromise, createRefreshTokenPromise]) Promise<(Awaited<string>)[]>
        // tslint:disable-next-line:no-shadowed-variable
        .then(([accessToken : string , refreshToken : string ]) => {
          jwtUtils.addUniqueAccessToken(clientId, accessToken);
          resolve( value: {accessToken, refreshToken, refreshTokenExpiration});
        }) Promise<void>
        .catch((error: any) => {
          reject(error);
```

```php
public function setUserData($username, $accessToken, $expiration) {
    try {
        $result = $this->oauthServerConnector->getUtente($username, $accessToken);
        $userData = serialize(array_values($result['data']));
        $expires = strtotime($expiration);

        setcookie('userData', $userData, $expires, '/');
    } catch(Exception $e) {
        $this->failure( message: $e->getCode() . " " . $e->getMessage(), location: "index.php");
    }
}
```

In questo caso l'utente root (privilegio di livello 0) ha accesso a tutte le funzionalità (CRUD su prodotti, CRUD su vendite)



Crittografia dei token

Tutti i token JWT sono firmati con la chiave privata e hanno il payload crittografato con la chiave pubblica al fine di ottienere **autenticità, integrità e riservatezza dei dati**:

- **Autenticità**: Utilizzando la chiave privata per firmare il token JWT, si fornisce una prova che il token è stato emesso da una fonte attendibile. Poiché chiunque riceva il token può verificare che sia autentico.
- **Integrità**: La firma digitale creata con la chiave privata garantisce che il contenuto del token non sia stato modificato durante la sua trasmissione e quando il destinatario verifica la firma del token con la chiave pubblica corrispondente, può essere sicuro che il token non sia stato alterato da terze parti durante il trasporto.
- **Riservatezza**: La crittografia con la chiave pubblica assicura che solo chi possiede la chiave privata corrispondente sarà in grado di decifrare il contenuto del payload. Pertanto, i dati nel payload rimarranno confidenziali per chiunque non abbia accesso alla chiave privata (ce l'ha solo il server).

```
1 export function getEncryptedToken(
2   payloadInput: any,
```

```
  expiresIn: string,
  options?: any): Promise<string> {
    return new Promise<string>((resolve, reject) => {

        const fs = require("fs");

        // tslint:disable-next-line:no-shadowed-variable
        const crypto = require("crypto");
        const encryptionOptions = {
            key: fs.readFileSync(
              filepath.join(__dirname, "key", "public-key.pem"), "utf8"),
            format: "pem",
            padding: crypto.constants.RSA_PKCS1_OAEP_PADDING,
        };

        const encryptedPayload = crypto.publicEncrypt(
          encryptionOptions, Buffer.from(JSON.stringify(payloadInput)));
        const base64EncryptedPayload = encryptedPayload.toString("base64");

        const payload = {
            data: base64EncryptedPayload,
        };

        const Jwtoptions: jwt.SignOptions = {
            algorithm: "RS256",
            expiresIn,
        };

        if (options) {
            Object.assign(Jwtoptions, options);
        }

        const privateKeyPEM = fs.readFileSync(
          filepath.join(__dirname, "key", "private-key.pem"), "utf8");
        const privateKey = crypto.createPrivateKey({
            key: privateKeyPEM,
            format: "pem",
            passphrase: process.env.OPENSSL_PASSPHRASE,
        });

        jwt.sign(payload, privateKey, Jwtoptions, (err, token) => {
            if (err) {
                reject(err);
            } else {
                resolve(token || "");
            }
        });
    });
}
```

```
export async function getDecryptedToken(inputToken: string):
Promise<any> {
    return new Promise((resolve, reject) => {
        // tslint:disable-next-line:no-shadowed-variable
        const crypto = require("crypto");
        const privateKeyPath = filepath.join(__dirname, "key", "private-key.per
        const publicKeyPath = filepath.join(__dirname, "key", "public-key.pem"
        const fs = require("fs");
        const privateKeyPEM = fs.readFileSync(privateKeyPath, "utf8");
        try {
            const publicKey = fs.readFileSync(publicKeyPath, "utf8");
            const verifiedToken: any = jwt.verify(
              inputToken, publicKey, { algorithms: ["RS256"] });
            const base64EncryptedPayload = verifiedToken.data;
            const encryptionOptions = {
                key: privateKeyPEM,
                format: "pem",
                passphrase: process.env.OPENSSL_PASSPHRASE,
                padding: crypto.constants.RSA_PKCS1_OAEP_PADDING,
            };
            const decryptedPayloadBuffer = crypto.privateDecrypt(
              encryptionOptions, Buffer.from(base64EncryptedPayload, "base64"))
            const decryptedPayload = JSON.parse(
              decryptedPayloadBuffer.toString("utf8"));
            resolve(decryptedPayload);
        } catch (err) {
            reject(err);
        }
    });
}
```

Access token monouso

Ogni access token generato ha una durata fissata di 1 minuto, tempo sufficiente per la generazione e la richiesta ad un endpoint che richiede tale token per l'autenticazione (potrebbe essere anche un tempo molto più basso).

Inoltre è stata implementata una protezione in più per rendere gli access token monouso al fine di evitare i replay attack.

In particolare all'avvio del software viene inizializzata una mappa che conterrà gli access token generati e che una volta ricevuti dal server stesso saranno da essa rimossi.

```
const uniqueAccessTokenMap: Record<string, string[]> = {};

export function addUniqueAccessToken(clientId: string, token: string) {
    if (!uniqueAccessTokenMap[clientId]) {
        uniqueAccessTokenMap[clientId] = [];
    }
    uniqueAccessTokenMap[clientId].push(token);
}

export function removeTokenFromUniqueTokenMap(clientId: string, token: string) {
    const tokens = uniqueAccessTokenMap[clientId];

    if (tokens) {
        const tokenIndex = tokens.indexOf(token);
        if (tokenIndex !== -1) {
            tokens.splice(tokenIndex, deleteCount: 1);
        } else {
            throw new InternalServerError( msg: "jwt access token already used");
        }
    } else {
        throw new NotFound( msg: "clientId not found");
    }
}

export function checkTokenExpiration(token: any) {
    const currentTime = Math.floor( x: Date.now() / 1000);
    if (token.exp < currentTime) {
        throw new InternalServerError( msg: "Token has expired");
    }
}
```

```
Promise.all( values: [createAccessTokenPromise, createRefreshTokenPromise]) Promise<(Awaited<string>)[]>
    .then(([accessToken : string , refreshToken : string ]) => {
      jwtUtils.addUniqueAccessToken(clientId, accessToken);
      resolve( value: {accessToken, refreshToken, refreshTokenExpiration});
    }) Promise<void>
    .catch((error: any) => {
      reject(error);
    });
```

Per evitare che un client generi troppi token e che rimangano inutilizzati, la mappa viene automaticamente svuotati dopo un determinato periodo di tempo

```
export function clearUniqueAccessTokens() {
    // tslint:disable-next-line:forin
    for (const clientId in uniqueAccessTokenMap) {
        delete uniqueAccessTokenMap[clientId];
    }
}

/*
 Clears the map of used jwts every hour
 */
export function startTokenCleanupInterval() {
    setInterval(clearUniqueAccessTokens,  ms: 3600000);
}
```

Basato su Wiki.js