Securing a Web App with Passwordless Web Authentication

Minimize and eliminate passwords!

What to expect from this workshop

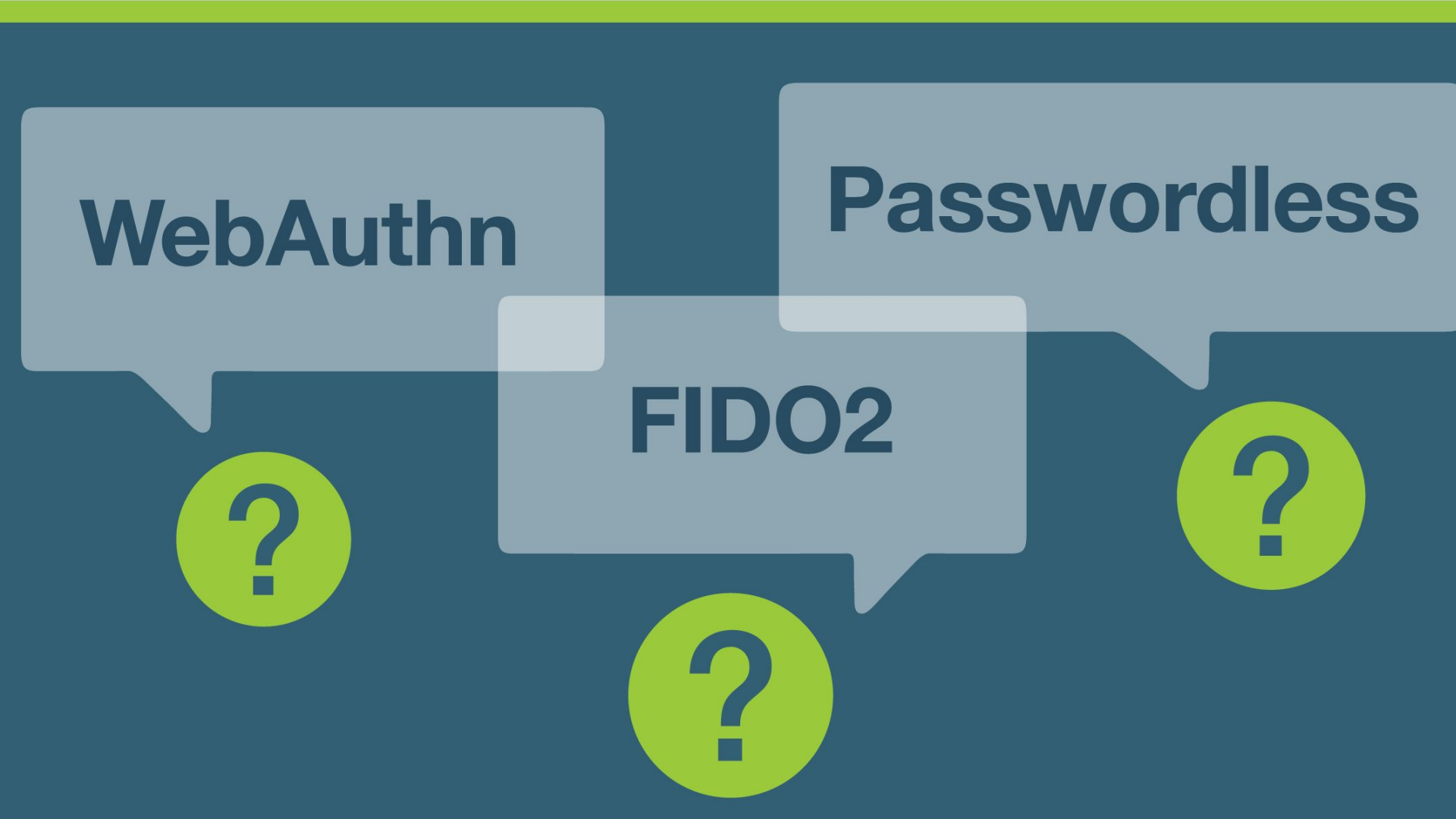
Learn how to implement passwordless authentication for a stand alone web app using:

- **Starter Spring Boot web app with traditional username/password**
- **WebAuthn**
  - Backend: Yubico WebAuthn Server Libraries
  - Frontend: JavaScript and W3C WebAuthn API
- **Client to Authenticator Protocol Version 2.0 Compatible Browser**
  - Resident Credentials enable passwordless authentication
- **FIDO2 Security Key**

You need

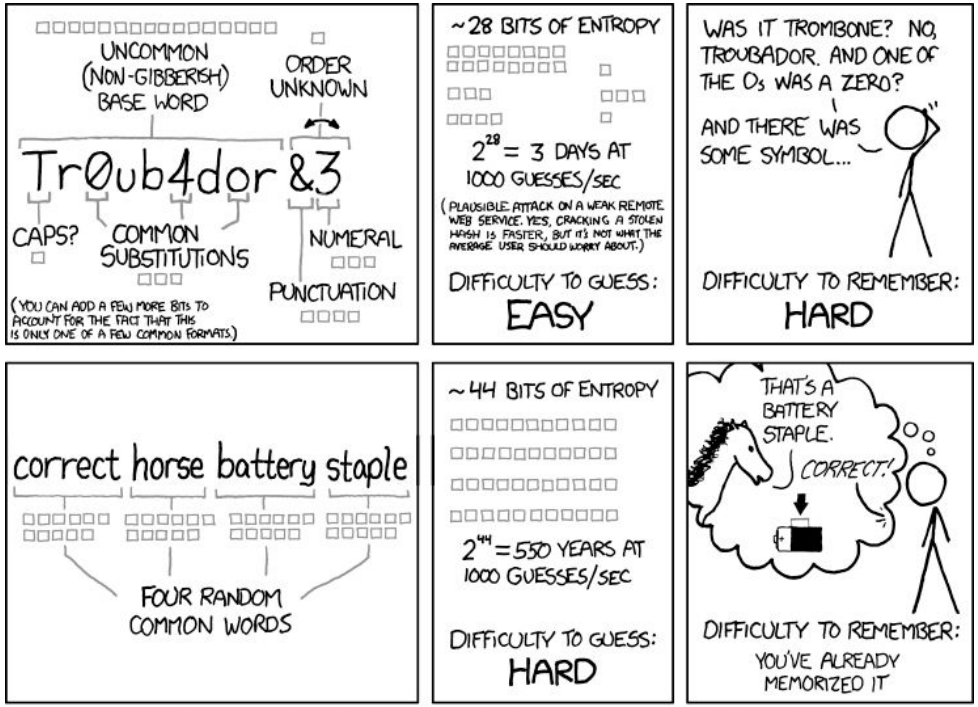
Some knowledge of:

- **Java**
- **Spring Framework**
- **JavaScript**
- **WebAuthn API**
  - Browser with resident credential capability
- **Security Key**
  - Download YubiKey Manager to reset FIDO credentials as needed
- **Optional**
  - Docker
  - Azure subscription for cloud native development


WebAuthn
FIDO2
Passwordless

Passwords

- Hard to remember
- Easy to crack
- Easy to phish
- Many strong passwords take lot's of effort!



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Source: https://xkcd.com/936/

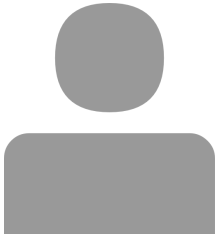
Authentication Factors

Something you **know**

- Password
- PIN

Something you **have**

- Smart card
- OTP dongle
- Mechanical key
- YubiKey



Something you **are**

- Fingerprint
- Face
- Voice
- Iris

Authentication Factors

Pros:

- Can’t be pickpocketed
- Can’t break
- Easy to replace

Cons:

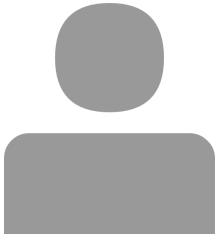
- Easy to steal remotely
- Hard to remember
- Theft is hard to detect

Pros:

- Can’t be stolen remotely
- Theft is easy to detect

Cons:

- Can be pickpocketed
- Can be forgotten, lost or destroyed



Pros:

- Natural to use
- Difficult to lose

Cons:

- Difficult to replace
- May change over time
- Environmental dependencies

Authentication Factors

Pros:

- Can’t be pickpocketed
- Can’t break
- Easy to replace

Cons:

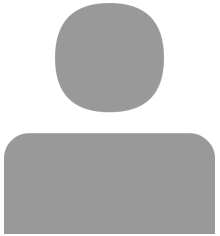
- Easy to steal remotely
- Hard to remember
- Theft is hard to detect

Pros:

- Can’t be stolen remotely
- Theft is easy to detect

Cons:

- Can be pickpocketed
- Can be forgotten, lost or destroyed



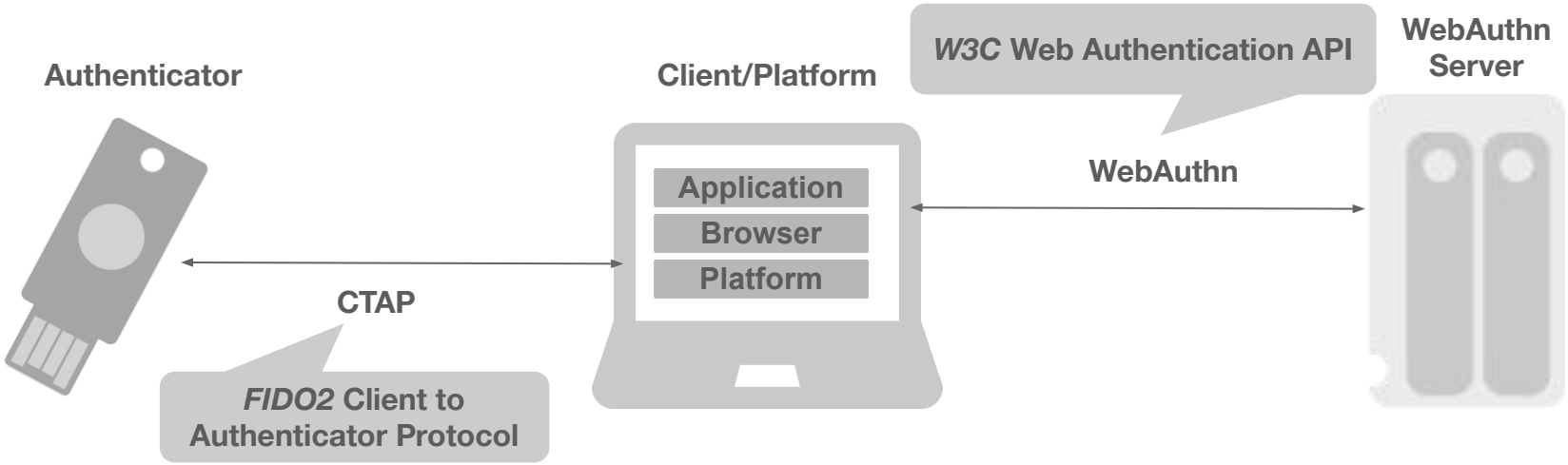
Pros:

- Natural to use
- Difficult to lose

Cons:

- Difficult to replace
- May change over time
- Environmental dependencies

What is FIDO2 / WebAuthn?



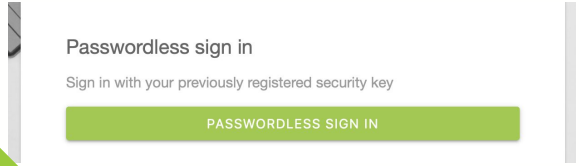
Open standards utilizing public-key cryptography with phishing protections to enable strong second-factor, first-factor, multi-factor authentication

Security Keys as Root of Trust

Anchoring FIDO2 / WebAuthn credentials in a root of trust is the cornerstone for building a secure identity model

- **A hardware-backed root of trust strengthens the account lifecycle**
  - Authentication, Step-Up Authentication, Account Recovery, Bootstrapping New Devices
- **An external authenticator, as the root of trust, is the anchor that creates a chain of trust with the internal authenticator**
  - Recording the authenticator used to register other authenticators creates a chain of trust that can be audited at a later date

Passwordless Migration Strategy

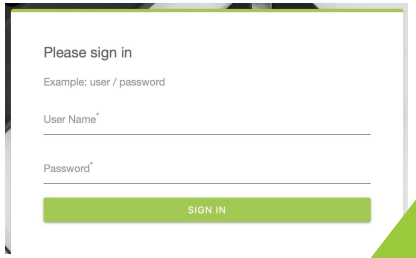


4. Eliminate Passwords

3. Transition users to passwordless deployment

2. Minimize use of passwords in user flows

1. Deploy the WebAuthn / FIDO2 credential management system across the account lifecycle



yubico

Workshop Modules

This workshop is split into multiple modules. Each module builds upon the previous module as you expand the application. You must complete each module before proceeding to the next.

1. Getting Started Instructions
2. Implement a Credential Repository
3. Implement WebAuthn Registration REST Endpoints
4. Implement WebAuthn Authentication REST Endpoints
5. Clean Up Instructions

Module 1 Getting Started Walkthrough

Development Environment

CTAP2 compatible platform / browser

- MacOS Safari Technology Preview version 71+
- Windows 10 version 1809+ with Edge

Security key is recommended. Platform authenticators can as well (Windows Hello, etc…)

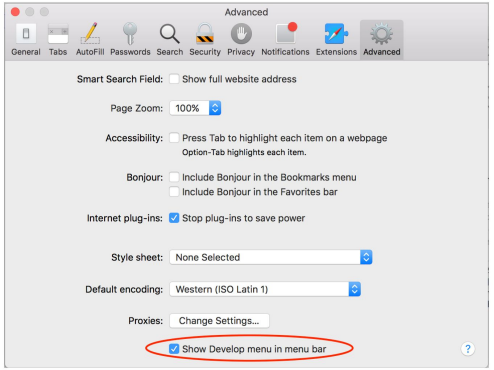
Local development

- Git
- JDK 1.8+
- Maven 3.2+
- Your preferred text editor or IDE
- [Optional] Docker

Cloud native development

- Azure Cloud Shell instructions are included

macOS Safari TP Tips



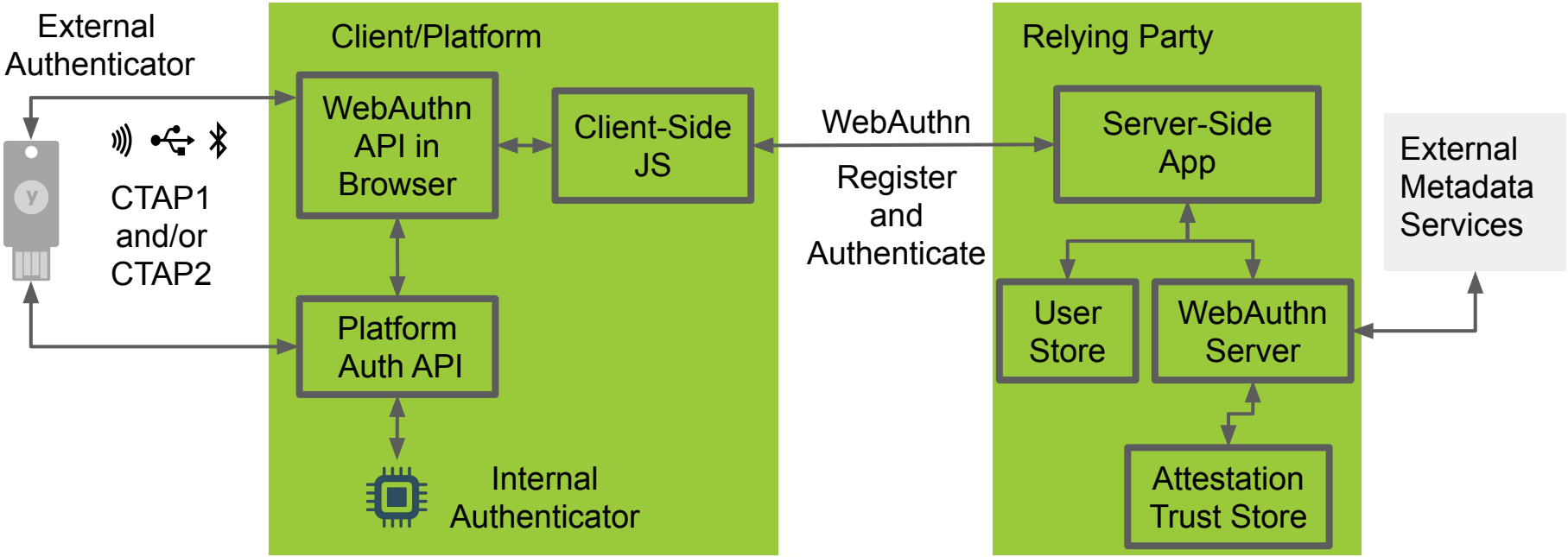
- **Enable the Develop Menu**
  - Choose Safari > Preferences, and click Advanced.
  - At the bottom of the pane, select the “Show Develop menu in menu bar” checkbox.
- **Enable Web Authentication Experimental Feature**
  - Choose Safari > Develop > Experimental Features
  - Verify “Web Authentication” is checked
- **Private Window**
  - Running a web app on https://localhost:8443 may require using new private window
- **No Security Key PIN Support**
  - Security keys with a PIN set may not work with Safari yet.
  - You can reset a security key back to factory default settings with the YubiKey Manager. Warning: A reset will remove all FIDO credentials.

Getting Started

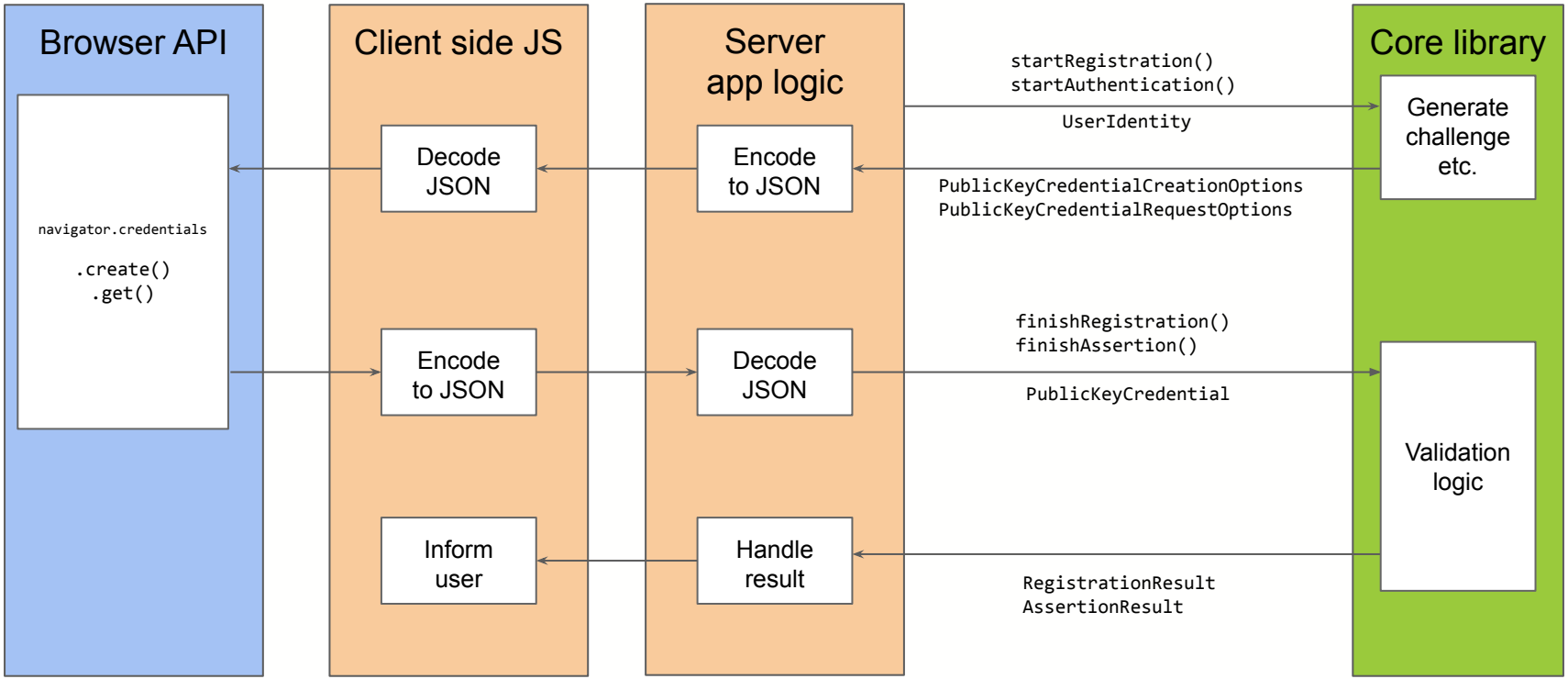
Start by cloning the git repository

https://github.com/YubicoLabs/java-webauthn-passwordless-workshop

WebAuthn Application Architecture



WebAuthn Demo Server Data Flow

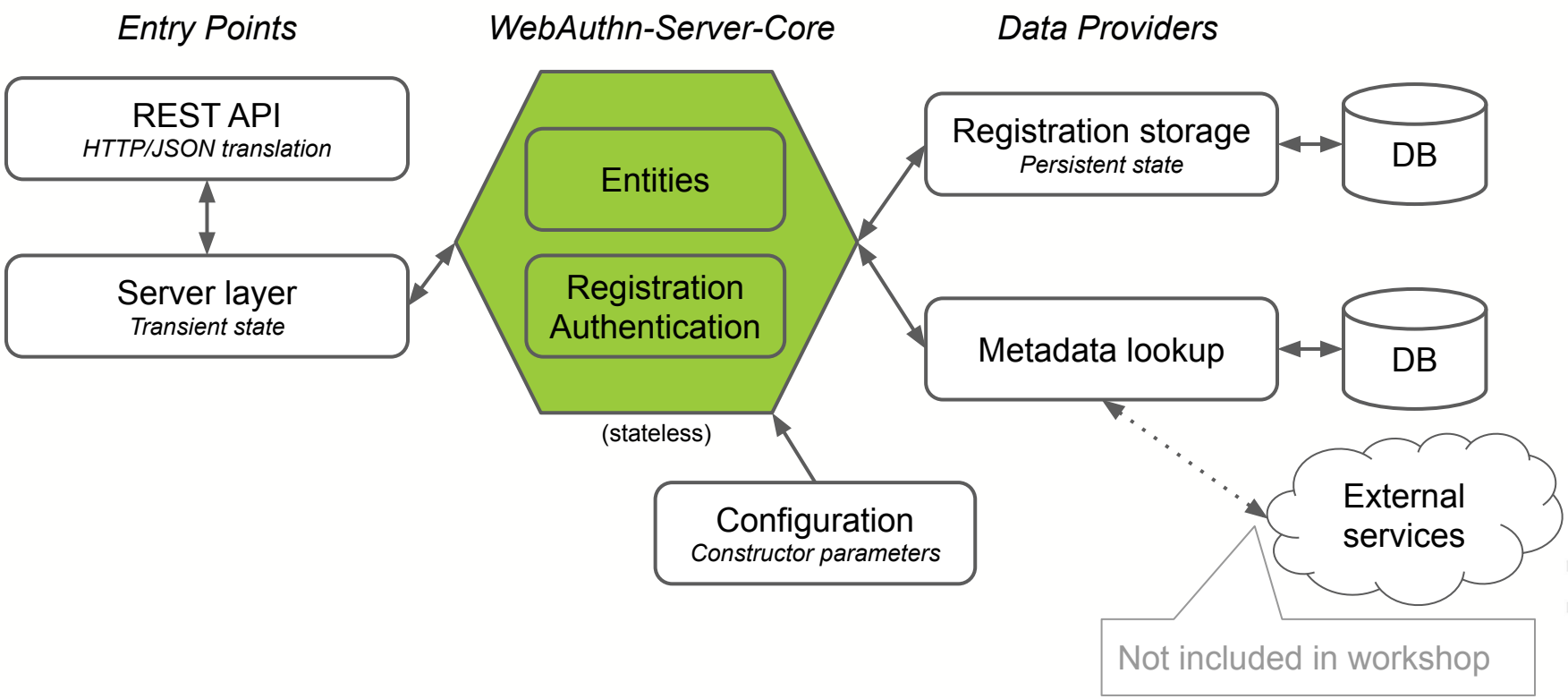


Yubico/java-webauthn-server

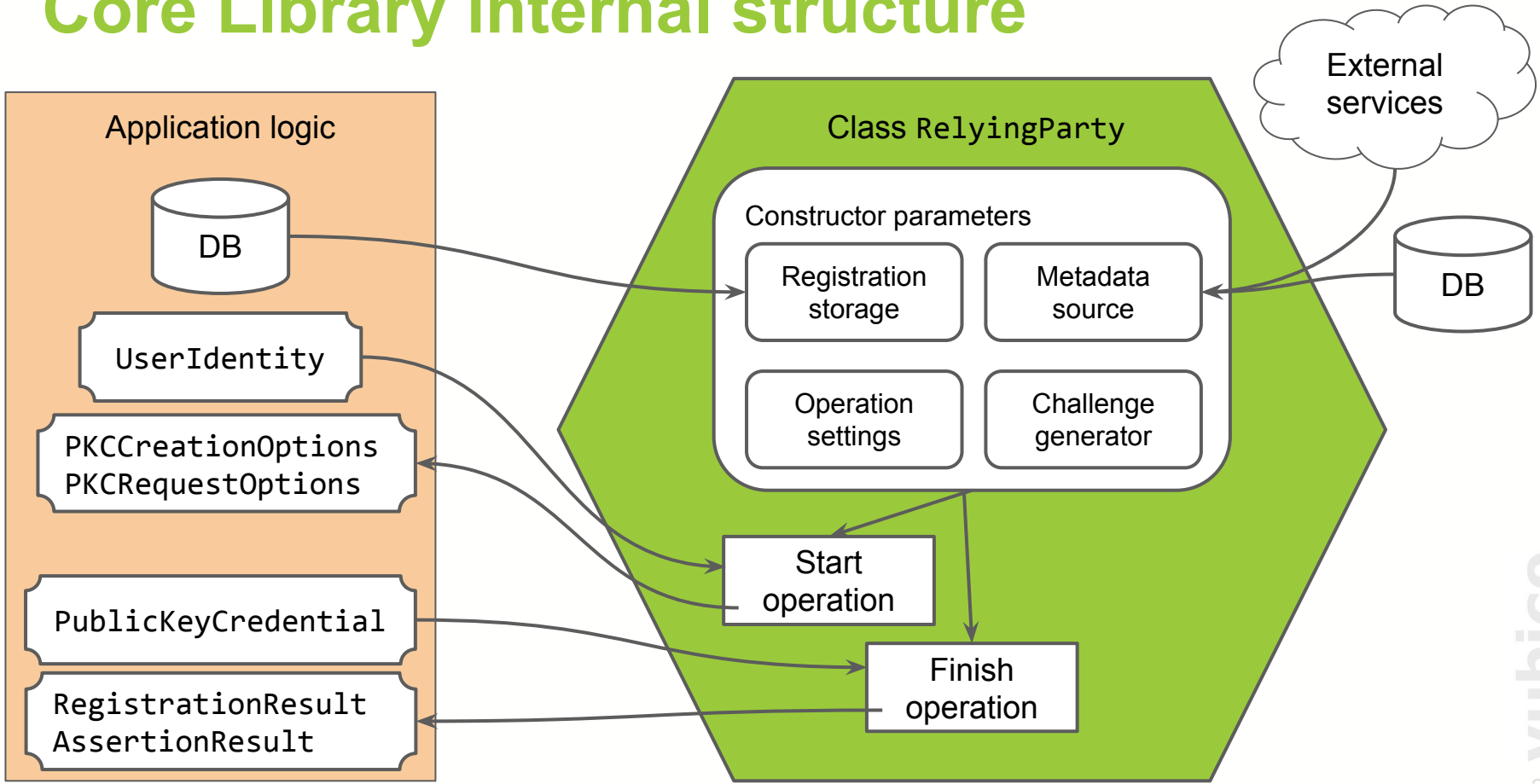
webauthn-server-core/

- **Entity Data Model**
  - Assertion Request
  - Assertion Result
  - Attestation Object
  - Authenticator Response
  - Public Key Credential
  - Public Key Credential Creation and Request Options
  - Registration Result
  - User Identity
  - Attestation
- **Data Providers**
  - Credential Repository
  - Metadata Service
- **Methods**
  - Registration
  - Authentication
  - Authenticated Actions

WebAuthn Demo Server Architecture



Core Library internal structure



What you need to provide

- **Storage for requests (temporary)**
  - Completely external to the library
  - Library simply returns request objects
  - finish* methods expect them to be passed back in
- **Storage for credentials (persistent)**
  - Library requires an adapter object (`CredentialRepository`)
  - Library only looks credentials up
  - You need to save new registrations to the DB
- **(Optional) Additional authenticator metadata sources**
  - In the future, the library will include a FIDO Metadata Service connector
  - Optional module with Yubico device metadata as static files

Module 2 Credential Repository Walkthrough

Module 2 Overview

1. **Add WebAuthn Server libraries to project**
2. **Implement a Credential Repository**
   2.1. Copy data entities, credential repository, and service layer from webauthn server demo into the project
3. **Implement a Model-View-Controller to manage credential registrations**
   3.1. Update the service layer to expose the registrations data model
   3.2. Create a controller for the account page
   3.3. Update the account page UI to add a table of registrations

2_Credential_Repository/TLDR.md

```
cd java-webauthn-passwordless-workshop/2_Credential_Repository/complete
```

```
mvn clean package spring-boot:run
```

or

```
docker build -t example/demo:module2 .
docker run -p 8443:8443 example/demo:module2
```

https://localhost:8443

Sign In: user / password

Dependency Configuration

Update pom.xml

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
<dependency>
    <groupId>com.yubico</groupId>
    <artifactId>webauthn-server-core</artifactId>
    <!--Check for the latest version at Maven Central-->
    <version>1.2.0</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>com.yubico</groupId>
    <artifactId>webauthn-server-attestation</artifactId>
    <!--Check for the latest version at Maven Central-->
    <version>1.2.0</version>
    <scope>compile</scope>
</dependency>
```

Implement the Credential Repository Interface

CredentialRepository.java

```
public interface CredentialRepository {
   Set<PublicKeyCredentialDescriptor> getCredentialIdsForUsername(String username);

   Optional<ByteArray> getUserHandleForUsername(String username);

   Optional<String> getUsernameForUserHandle(ByteArray userHandle);

   // Look up the public key and stored signature count for the given credential registered to the given
user.
   Optional<RegisteredCredential> lookup(ByteArray credentialId, ByteArray userHandle);

   Set<RegisteredCredential> lookupAll(ByteArray credentialId);
}
```

Implement the Registration Storage Interface

RegistrationStorage.java

```
public interface RegistrationStorage extends CredentialRepository {

   boolean addRegistrationByUsername(String username, CredentialRegistration reg);

   Collection<CredentialRegistration> getRegistrationsByUsername(String username);
   Optional<CredentialRegistration> getRegistrationByUsernameAndCredentialId(String username, ByteArray
userHandle);
   Collection<CredentialRegistration> getRegistrationsByUserHandle(ByteArray userHandle);

   boolean removeRegistrationByUsername(String username, CredentialRegistration credentialRegistration);
   boolean removeAllRegistrations(String username);

   void updateSignatureCount(AssertionResult result);
}
```

Copy Demo Resources

Copy webauthn demo server resources instead of implementing our credential repository from scratch

- GetLibs.sh copies the Java WebAuthn Server demo WebAuthnServer class, its Config file, associated Data Entities, and In-Memory Credential Repository Implementation to our project

Update Service Layer

Make WebAuthnServer class visible as a service via Spring

```
@Service
public class WebAuthnServer {
```

Get the registrations data model

```
public Collection<CredentialRegistration> getRegistrationsByUsername(String username)
{
    return this.userStorage.getRegistrationsByUsername(username);
}
```

Create an Account Controller

AccountController.java

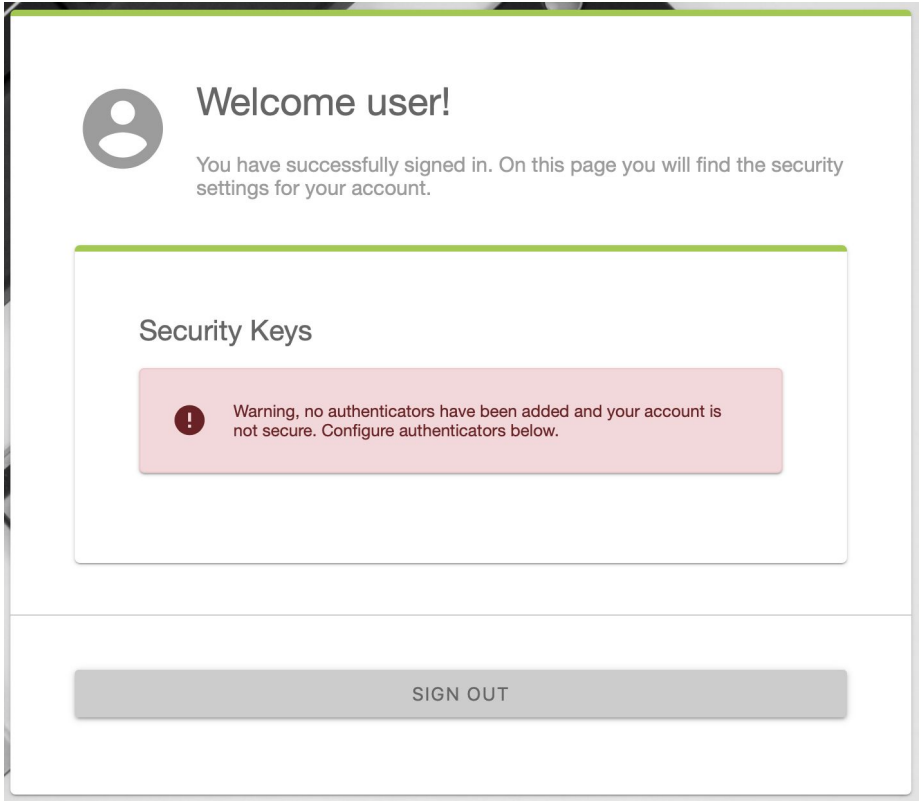
```
@Controller
public class AccountController {

    @Autowired
    private WebAuthnServer webAuthnServer;

    @GetMapping("/account")
    public String registerAll(Principal principal, Model model) {
        model.addAttribute("registrations",
webAuthnServer.getRegistrationsByUsername(principal.getName()));
        return "account";
    }
  }
```

Add Registrations Table UI to Accounts

account.html

```
<div class="card card--internal">
   <h2 class="section-header">Security Keys</h2>
   <table class="table" id="keys" th:classappend="${registrations.empty}? 'hide'">
       <thead><tr>
               <th> Nickname </th>
               <th> Registration Time </th>
           </tr></thead>
       <tbody id="keys">
           <tr th:each="registration : ${registrations}" >
               <td><span th:text="${registration.credentialNickname.get()}"> NickName </span></td>
               <td><span th:text="${registration.registrationTime}"> Registered </span></td>
           </tr>
       </tbody>
   </table>
</div>
```

List Registrations



Module 3 Registration Walkthrough

Module 3 Overview

1. **Update Service Layer**
   1.1. Remove the dependency on AuthenticatedActions
      1.1.1. Modify startRegistration() method to allow registration of multiple credentials
   1.2. Configure JSON Rendering
2. **Expose Registration REST Endpoints**
   2.1. Create a WebAuthn REST Controller
   2.2. Add start and finish registration endpoints
3. **Update UI to Enable Registration**
   3.1. Add JavaScript methods to call registration REST endpoints
   3.2. Add UI components to allow user to register a security key

3_Registration/TLDR.md

```
cd java-webauthn-passwordless-workshop/2_Registration/complete
```

```
mvn clean package spring-boot:run
```

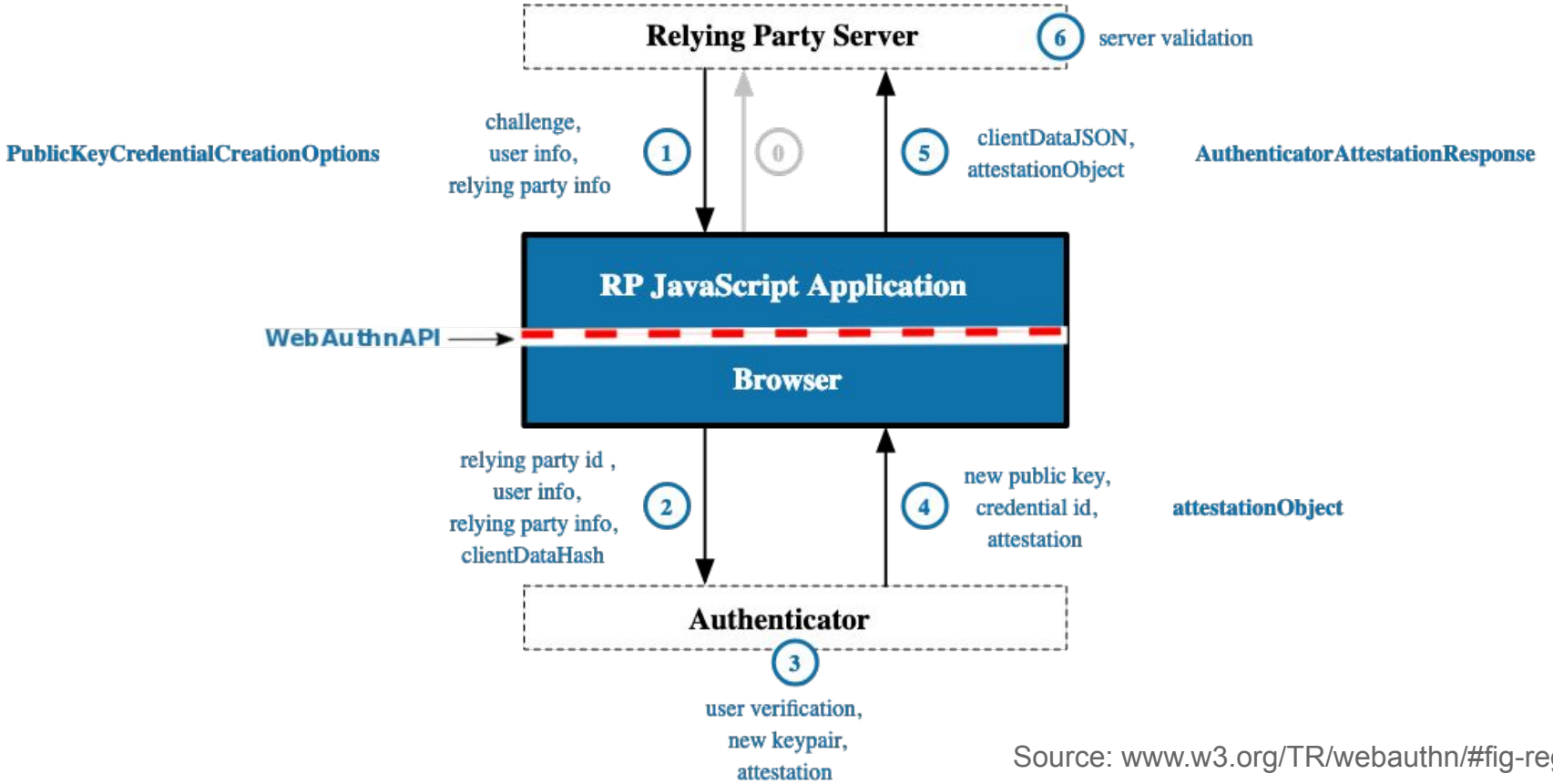
or

```
docker build -t example/demo:module3 .
docker run -p 8443:8443 example/demo:module3
```

https://localhost:8443

Sign In: user / password, **Register:** Register security key

Registration Flow



Source: www.w3.org/TR/webauthn/#fig-registration

Public Key Credential Creation Options

```
2   var publicKey = {
3     "rp": {
4       "name": "Yubico Web Authentication demo",
5       "id": "demo.yubico.com"
6     },
7     "user": {
8       "name": "a.user",
9       "displayName": "a.user",
10      "id": "weuI8XvDhZLEnAOeWaCMAAC67f..."
11    },
12    "challenge": "e6yvPpdAxgMqHp7fpvj30...",
13    "pubKeyCredParams": [
14      {
15        "alg": -7,
16        "type": "public-key"
17      }
18    ],
19    "excludeCredentials": [
20      {
21        "type": "public-key",
22        "id": "XeytXy7e_tiCjSgvTgjaoz..."
23      }
24    ],
25    "authenticatorSelection": {
26      "requireResidentKey": false,
27      "userVerification": "preferred"
28    },
29    "attestation": "direct"
30  }
```

rp: relying party data. **name** is required. If **id** is left out then origins effective domain is used

user: identity data. **name, displayName** (user friendly), and **id** (userHandle) are required

challenge: contains challenge for generating the newly created credential's attestationObject

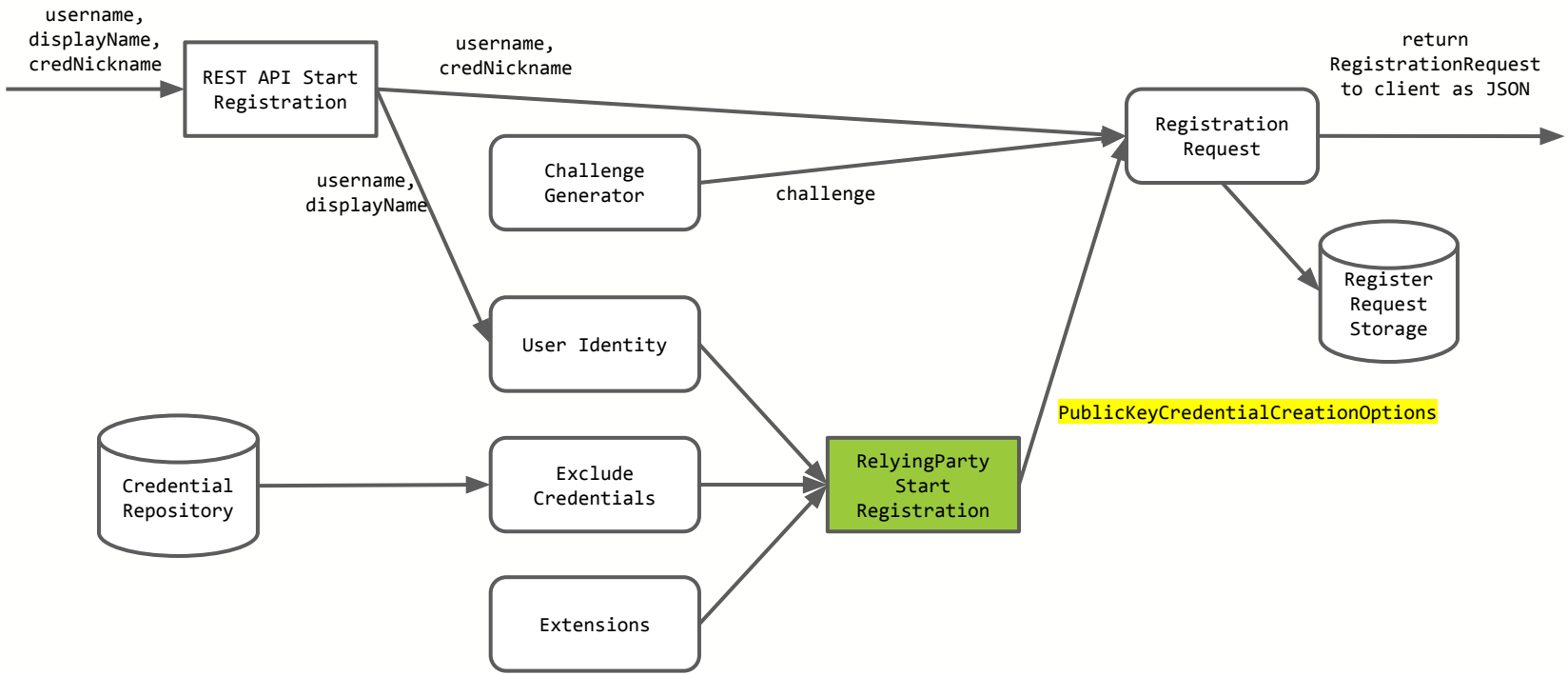
pubKeyCredParams: desired properties of credential to be created. **type:** only one type: "public-key". **alg:** crypto signature algorithm preference

excludeCredentials: limits creation of multiple creds for same account on a single authenticator. Credential descriptor includes cred **type** and cred **id**

authenticatorSelection: specify authenticator requirements. When the **requireResidentKey** is true the authenticator must create a client side resident private key. **userVerification** can be "preferred", "required", or "discouraged"

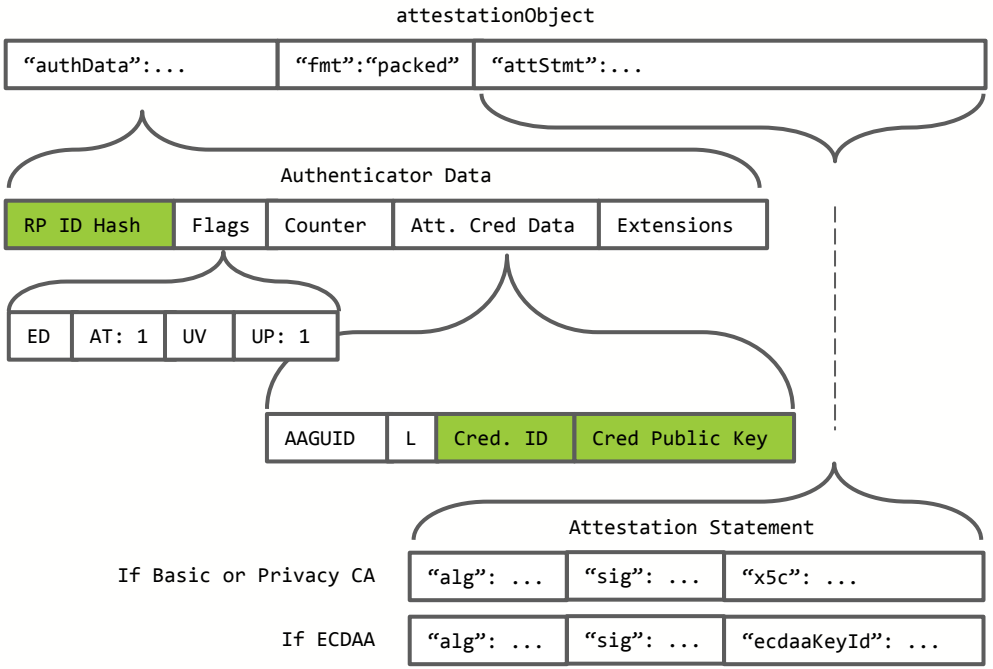
attestation: attestation conveyance preference. Default is "none". "direct" indicates the rp wants to receive the attestation statement. "indirect" indicates prefers an attestation statement

Start Registration Diagram



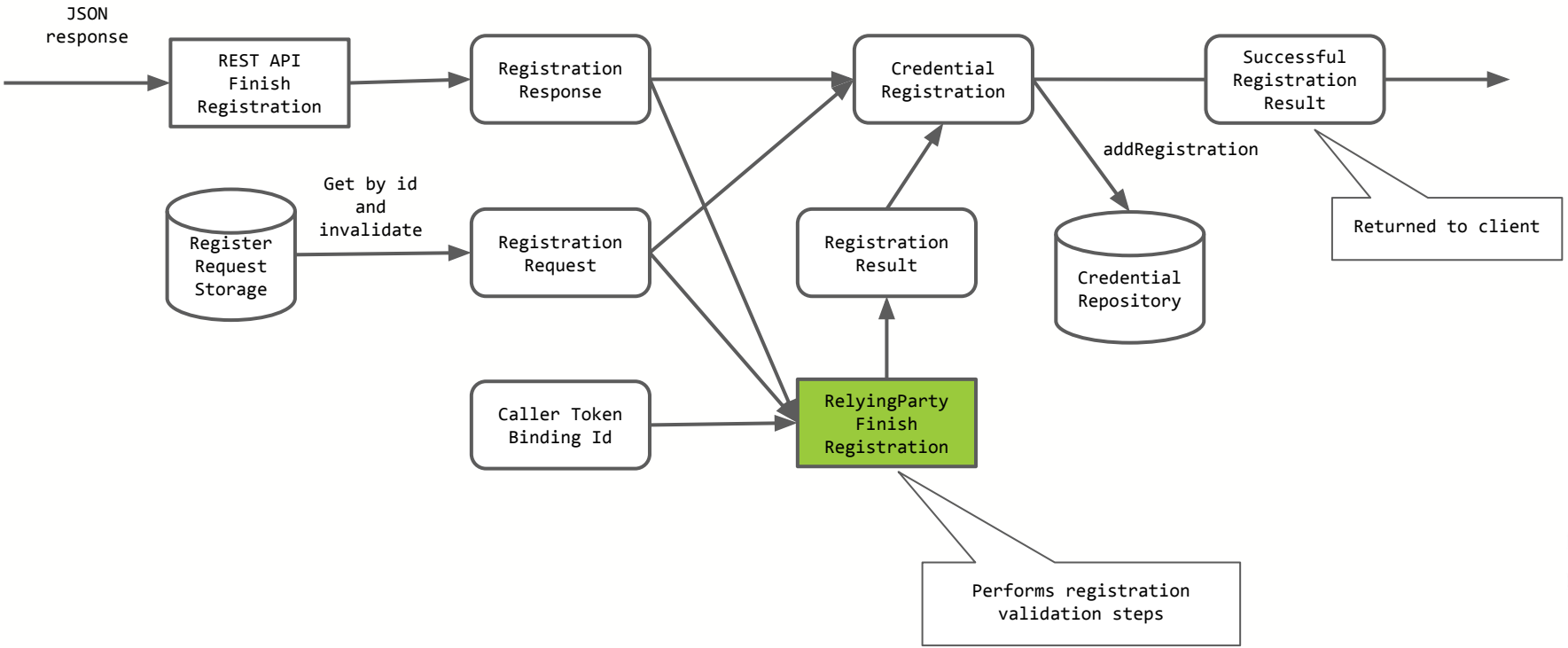
Credential Create Response

```
//navigator.credentials.create() response:
{
  "id": "rfzwEiU4eHowfAW54Z_Hkp5ULF...",
  "response": {
    "attestationObject": "o2NmbXRoZm...",
    "clientDataJSON": "eyJjaGFsbGVuZ..."
  },
  "clientExtensionResults": {}
}

//Parsed clientDataJSON response:
{
  "challenge": "p9erRYiAG98K8CU4...",
  "origin": "https://demo.yubico.com",
  "tokenBinding": {
    "status": "not-supported"
  },
  "type": "webauthn.create"
}
```



Refer to W3C Web Authentication API for more details on attestation statements

Finish Registration Diagram



Registration Recap

1. **Client calls startRegistration() endpoint**
   - With userName, displayName, and credentialNickname args
2. **Relying Party generates RegistrationRequest**
   - With userName, credentialNickname, and PublicKeyCredentialCreationOptions
3. **Client calls navigator.credentials.create()**
   - With data from the RegistrationRequest
4. **Client calls finishRegistration() endpoint**
   - With authenticatorAttestationResponse JSONObject
5. **Relying Party verifies attestation signature**
   - After validation, add user and associated credential to the credential repository

Note

- Unexpected behavior can occur after 20 credentials registered

Start Registration Method

WebAuthnServer.java

```
rp.startRegistration(
    StartRegistrationOptions.builder()
        .user(user)
        .authenticatorSelection(Optional.of(AuthenticatorSelectionCriteria.builder()
            .requireResidentKey(requireResidentKey)
            .authenticatorAttachment(AuthenticatorAttachment.CROSS_PLATFORM)    // Default
to roaming security keys (CROSS_PLATFORM). Comment out this line to enable either PLATFORM
or CROSS_PLATFORM authenticators
            .build()
        ))
        .build()
)
```

JSON Rendering

```java
@Bean
public ObjectMapper objectMapper() {
    ObjectMapper mapper = new ObjectMapper();
    mapper.registerModule(new Jdk8Module());
    mapper.setVisibility(PropertyAccessor.FIELD, Visibility.ANY);
    mapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, false);
    mapper.setSerializationInclusion(Include.NON_NULL);
    mapper.setSerializationInclusion(Include.NON_ABSENT);
    return mapper;
  }
```

WebAuthnServer.java


```
"publicKeyCredentialCreationOptions":{
"rp":{
 "id":"localhost"
},
"user":{
 "name":"user",
 "id":"sYr36b..."
},
"challenge":"BD0n...",
"pubKeyCredParams":[
 {"alg":-7,
  "Type":"public-key"}],
"excludeCredentials":[],
"authenticatorSelection":{
 "authenticatorAttachment":"cross-platform",
 "requireResidentKey":true,
 "userVerification":"preferred"
},
"attestation":"direct",
"extensions":{}
}
```

Registration REST Endpoints

WebAuthnController.java

```
class WebAuthnController {
    ...
    @PostMapping("/register")
    ResponseEntity<RegistrationRequest> startRegistration(...) {
                Either<String, RegistrationRequest> result = webAuthnServer.startRegistration(username,
displayName, credentialNickname, requireResidentKey);
                return ResponseEntity.status(HttpStatus.OK).body(result.right().get());
    }
    @PostMapping("/register/finish")
    ResponseEntity<WebAuthnServer.SuccessfulRegistrationResult> finishRegistration(...) {
        Either<List<String>, WebAuthnServer.SuccessfulRegistrationResult> result =
webAuthnServer.finishRegistration(responseJson);
        return ResponseEntity.status(HttpStatus.OK).body(result.right().get());
    }
  }
```

Enable registration on UI

account.html

```
function register() {
            return fetch('/register', {
                ...
                    username, displayName, credentialNickname, requireResidentKey,
                ...
            })
            .then(response => response.json())
            .then(function(request) {
                return webauthn.createCredential(request.publicKeyCredentialCreationOptions)
                .then(webauthn.responseToObject)
                .then(function (publicKeyCredential) {
                    return submitResponse('/register/finish', request.requestId, publicKeyCredential);
                ...
}
```

Enable Registration on UI

account.html

```
<h2 class="section-header">Register a Security Key</h2>
<label class="input-group">
    <input type="text" id="inputNickname">
    <span>Nickname</span>
</label>
<button onclick="register()">Register</button>
<p id="status"></p>
<div id="takeAction">
    <p>Please insert and take action on the security key.</p>
    <div class="loader-container" role="status">
        <svg class="loader" viewBox="22 22 44 44"><circle class="loader-circle" cx="44" cy="44" r="20.2"
fill="none" stroke-width="3.6"></circle></svg>
    </div>
</div>
```

Register a Security Key

Security Keys

Nickname	Registration Time
YubiKey!	2019-05-22T08:21:36.525Z

Register a Security Key

Nickname

REGISTER

Success!

Module 4 Authentication Walkthrough

Module 4 Overview

1. **Expose Authentication REST Endpoints**
   1.1. Add start and finish authentication endpoints
   1.2. Given successful WebAuthn authentication, manually authenticate user in Spring Security
2. **Update UI to Enable Passwordless Authentication**
   2.1. Add JavaScript methods to call authentication REST endpoints
   2.2. Add UI components to enable passwordless authentication

4_Authentication/TLDR.md

```
cd java-webauthn-passwordless-workshop/2_Registration/complete
```

```
mvn clean package spring-boot:run
```

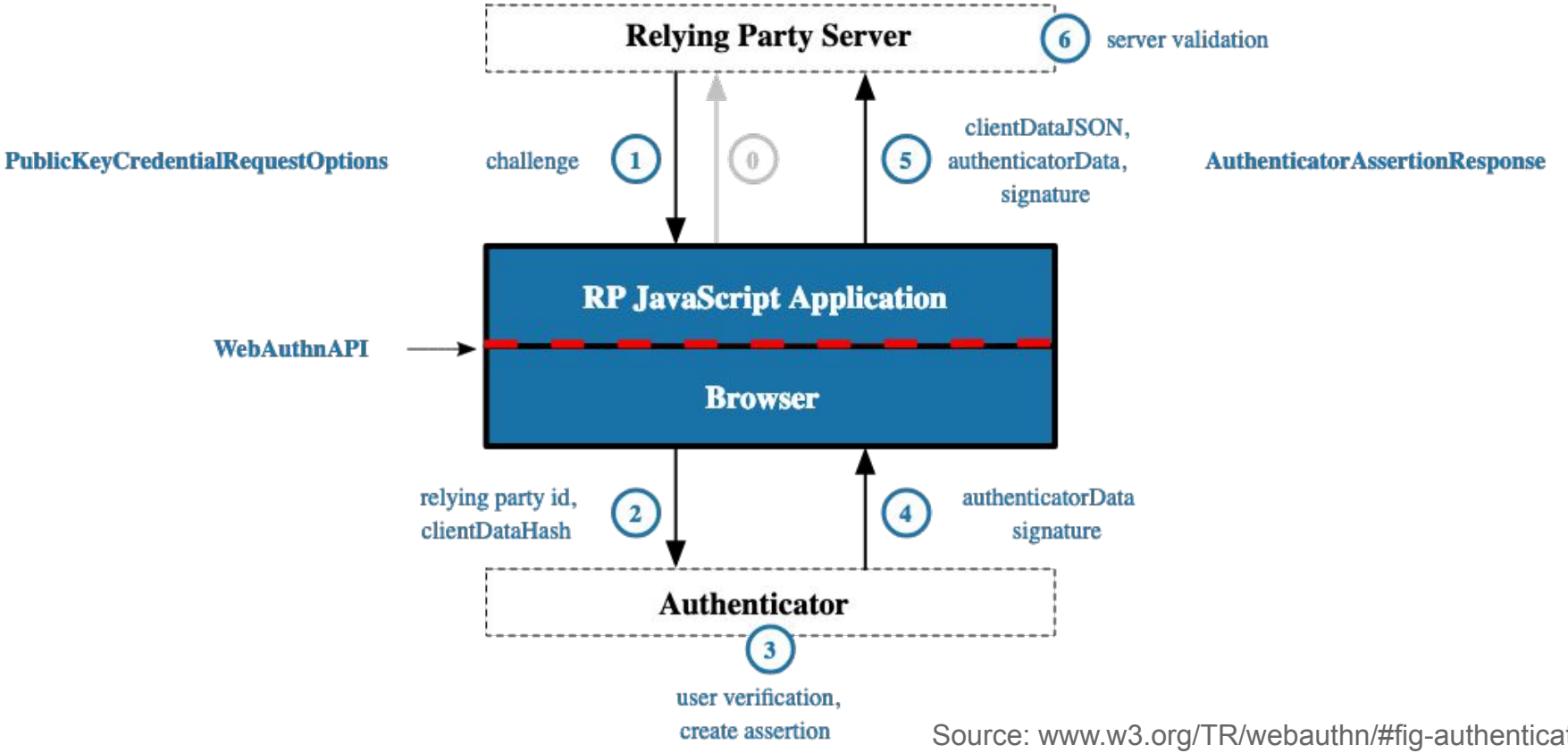
or

```
docker build -t example/demo:module4 .
docker run -p 8443:8443 example/demo:module4
```

https://localhost:8443

Sign In: user / password, **Register:** Register security key, **Sign out, Passwordless sign in**

Authentication Flow



Source: www.w3.org/TR/webauthn/#fig-authentication

Public Key Credential Request Options

```
navigator.credentials.get({
  publicKey: {
    "challenge": "Z37e0ba2cSru6mu43R
    "rpId": "demo.yubico.com",
    "allowCredentials": [
      {
        "type": "public-key",
        "id": "RvPR8sycnQMH32jNKtxA_
      }
    ],
    "userVerification": "preferred"
  }
});
```

challenge: contains challenge that the authenticator signs as part of the authentication assertion

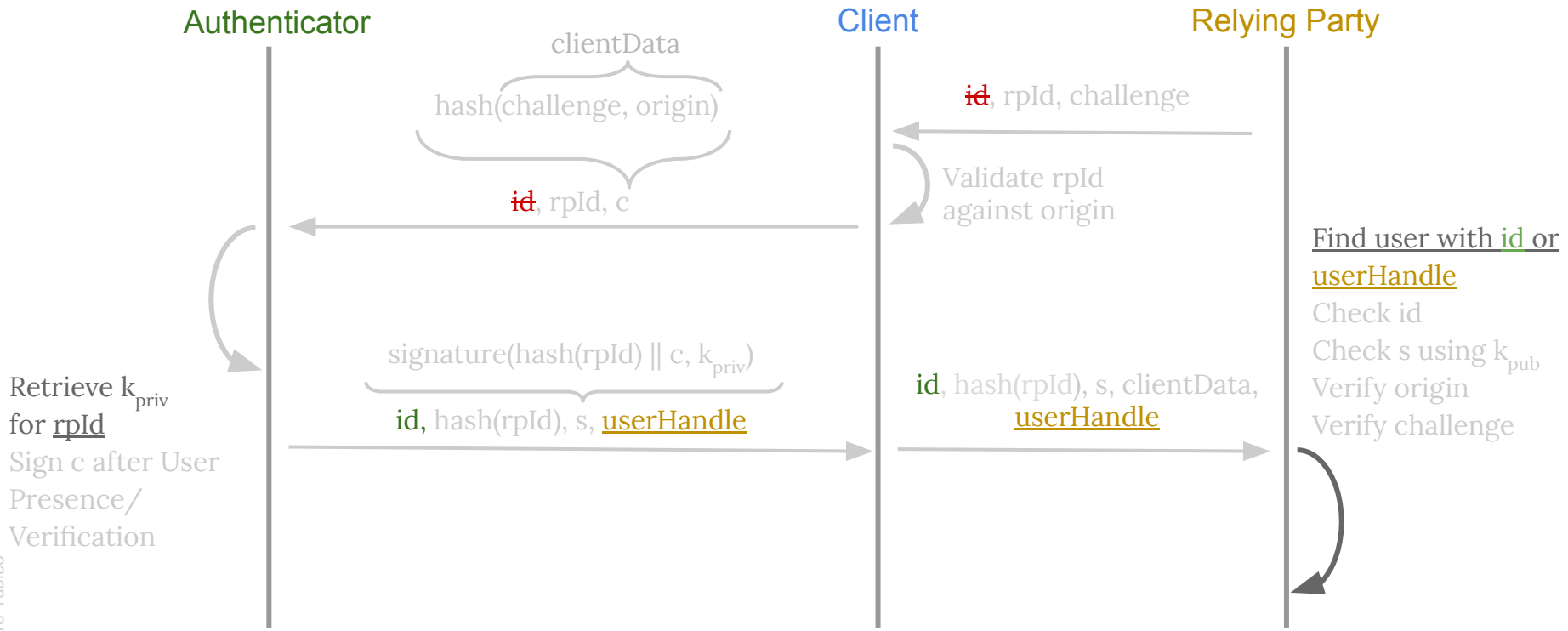
rpId: relying party identifier claimed by the caller

allowCredentials: list of public key credentials acceptable to the caller, can be omitted for username-less authentication. **type:** only one type: “public-key”. **id:** credential Id of the public key credentials

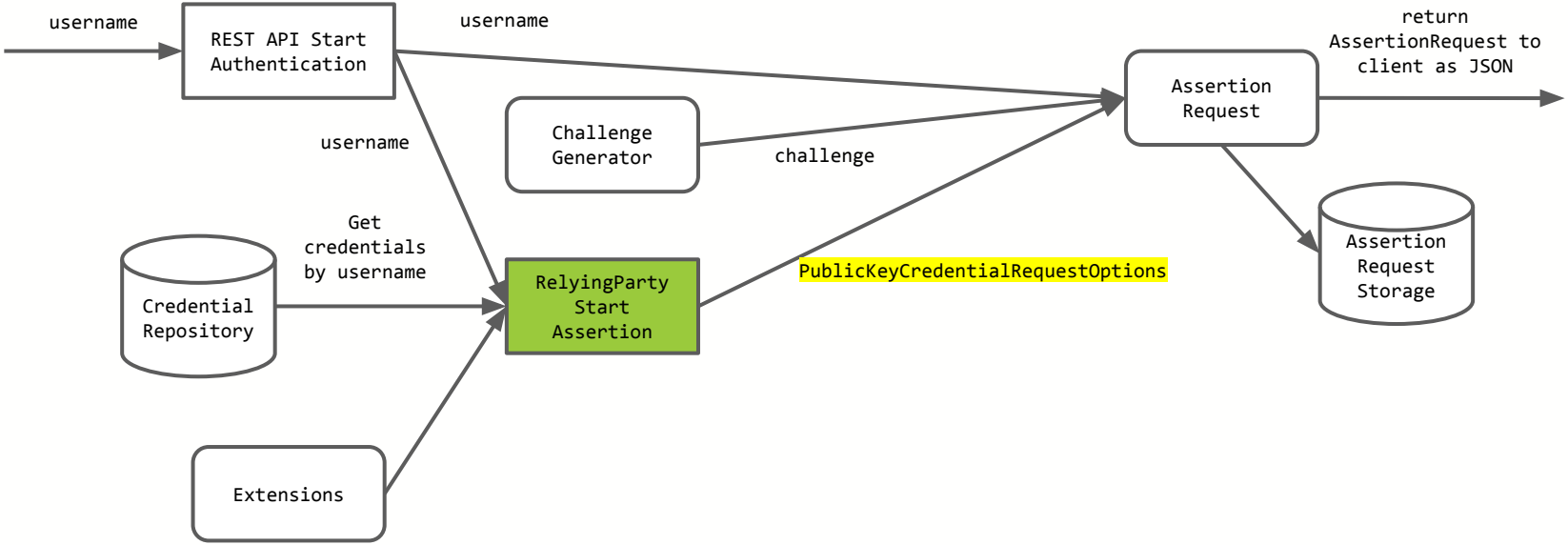
userVerification: the default is “preferred”. Can also be set to “required” or “discouraged”

Authentication Sequence

First factor mode

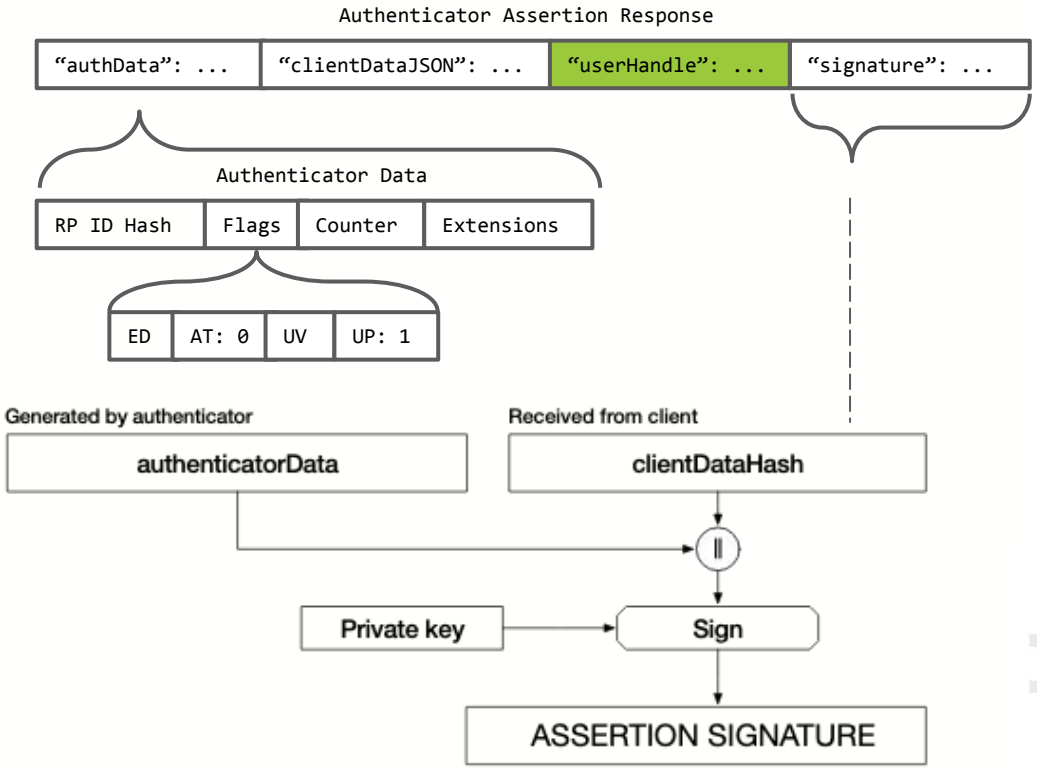


Start Authentication Diagram

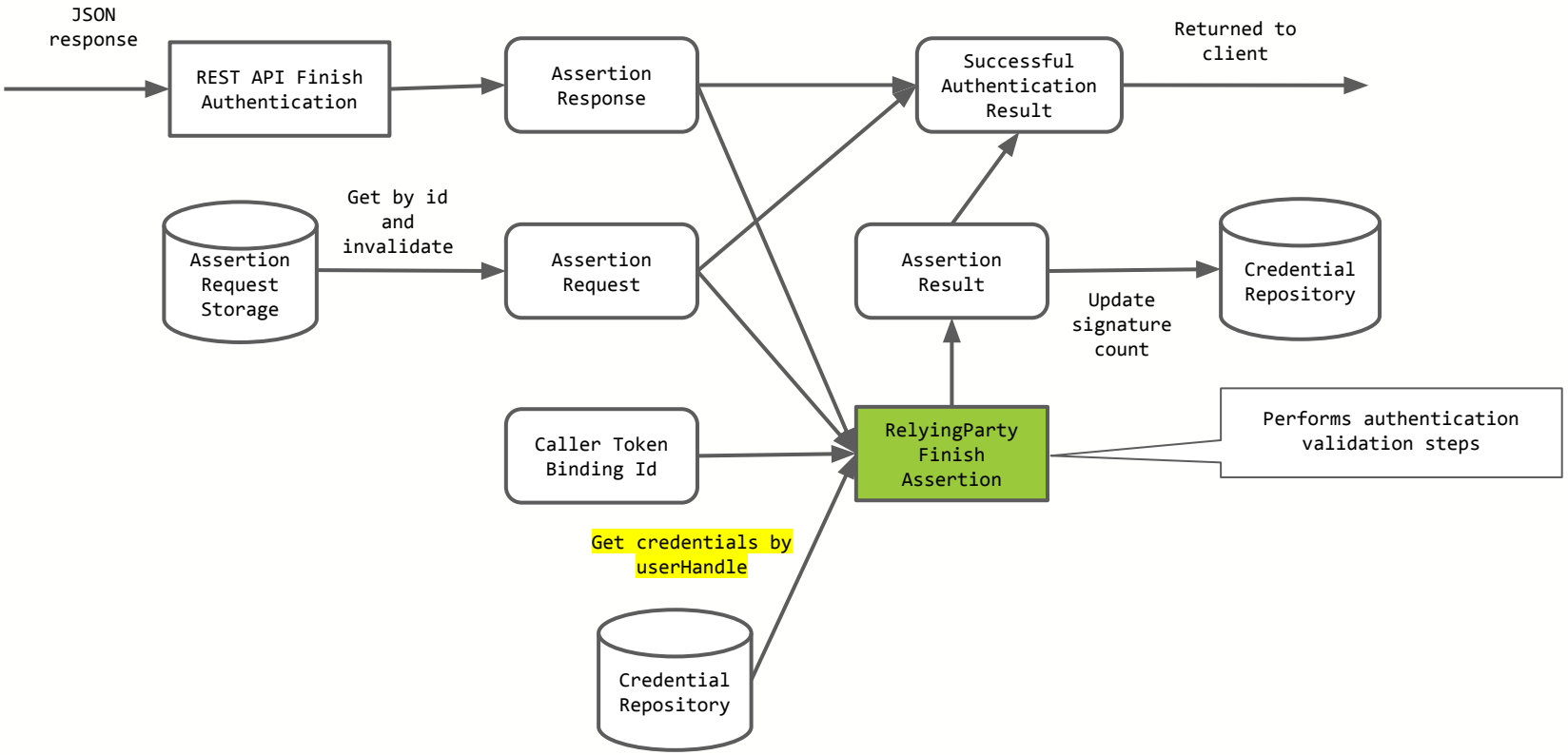


Client - Get Credential Response

```
// authenticatorAssertionResponse
{
  "id": "RvPR8sycnQMH32jNKtxA_wKuqMAvJILXFE
  "response": {
    "authenticatorData": "xGzvgq0bVGR3WR0Ai
    "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJa
    "signature": "MEUCIQDjRKF0AiI7sLKM0Q0nM
  },
  "clientExtensionResults": {
    "appid": false
  }
}

//clientDataJSON
{
  "challenge": "Z37e0ba2cSru6mu43RDe78d
  "new_keys_may_be_added_here": "do not
  "origin": "https://demo.yubico.com",
  "tokenBinding": {
    "status": "not-supported"
  },
  "type": "webauthn.get"
}
```



Finish Authentication Diagram



Authentication Recap

1. **Client calls startAuthentication() endpoint**
   - With (or without) userName
2. **Relying Party generates AssertionRequest**
   - With userName, and PublicKeyCredentialRequestOptions
3. **Client calls navigator.credentials.get()**
   - With data from the AssertionRequest
4. **Client calls finishAuthentication() endpoint**
   - With authenticatorAssertionResponse JSONObject
5. **Relying Party verifies signature**
   - After validation, authentication is successful

Authentication REST Endpoints

WebAuthnController.java

```
@PostMapping("/authenticate")
public ResponseEntity<AssertionRequestWrapper> startAuthentication(@RequestParam("username")
Optional<String> username) {
    Either<List<String>, AssertionRequestWrapper> result = webAuthnServer.startAuthentication(username);
    return ResponseEntity.status(HttpStatus.OK).body(result.right().get());
}
@PostMapping("/authenticate/finish")
public ResponseEntity<WebAuthnServer.SuccessfulAuthenticationResult> finishAuthentication(
        @RequestBody String responseJson) {
    Either<List<String>, WebAuthnServer.SuccessfulAuthenticationResult> result = webAuthnServer
            .finishAuthentication(responseJson);

    if (result.isRight()) {
        // Manually authenticate user
        ...
```

Manually Authenticate

WebAuthnController.java

```
    if (result.isRight()) {
        // Manually authenticate user
        String username =
result.right().get().getRegistrations().iterator().next().getUserIdentity().getName();
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        UserDetails u = userDetailsService.loadUserByUsername(username);
        Authentication newAuth = new UsernamePasswordAuthenticationToken(u, auth.getCredentials(),
                u.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication(newAuth);
        return ResponseEntity.status(HttpStatus.OK).body(result.right().get());
    } else {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, result.left().get().toString());
    }
```

Call Authenticate Endpoints

login.html

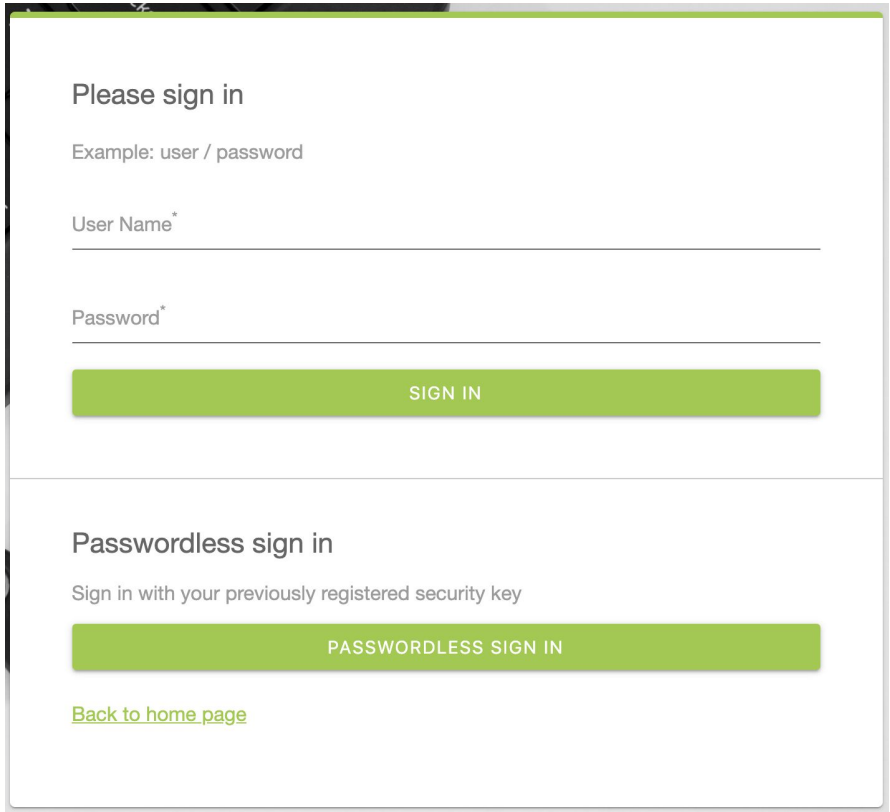
```
function authenticate() {
            return fetch('/authenticate', {
                ...
                })
                .then(response => response.json())
                .then(function (request) {
                    return webauthn.getAssertion(request.publicKeyCredentialRequestOptions)
                        .then(webauthn.responseToObject)
                        .then(function (publicKeyCredential) {
                            return submitResponse('/authenticate/finish', request.requestId,
publicKeyCredential);
                        })
                        ...
          }
  }
```

Passwordless Sign In UI

login.html

```
<h2 class="form-signin-heading">Passwordless sign in</h2>
<p>Sign in with your previously registered security key</p>
<p id="status"></p>
<p><button onclick="authenticate()">Passwordless Sign in</button><br />
```

Passwordless Sign In!



Usernameless Passwordless Sign In!

Security Keys

Nickname	Registration Time
YubiKey!	2019-05-22T08:21:36.525Z

Register a Security Key

Nickname

REGISTER

Success!

Best Practices

Best Practices

- **Store the verbatim attestation object**
  - Enables future re-evaluation of trust
- **Allow registering more than one credential per account**
  - Consider allowing credential nicknames
  - Unexpected behavior may occur when greater than 20 credentials registered
- **Weigh pros vs cons of *requiring* attestation**
  - Pros:
    - Higher assurance
  - Cons:
    - Maintenance for attestation trust store
    - Compatibility issues for unknown/new authenticators (not in attestation trust store)
- **Security and Privacy Considerations**
  - W3C WebAuthn spec https://www.w3.org/TR/webauthn/#security-considerations

Recap

- Plan your passwordless migration strategy across the account lifecycle
- Anchor resident credentials on security keys to enable roaming passwordless authentication scenarios
- Record attestation and build a chain of trust
- Allow users to register multiple credentials
- WebAuthn libraries can jumpstart your journey to passwordless (eg. Yubico Java Webauthn Server libraries)

Resources

- **Workshop**
  - https://github.com/YubicoLabs/java-webauthn-passwordless-workshop
- **FIDO2/WebAuthn Developer Guide**
  - https://developers.yubico.com/FIDO2/FIDO2_WebAuthn_Developer_Guide
- **Java WebAuthn Server**
  - https://github.com/Yubico/java-webauthn-server
- **Yubico Developer Videos**
  - https://www.yubico.com/why-yubico/for-developers/developer-videos/
- **W3C Web Authentication API**
  - https://www.w3.org/TR/webauthn
- **FIDO Client to Authenticator Protocol V 2.0**
  - https://fidoalliance.org/specifications/download/

Yubico Developer Program

yubi.co/devs

Workshops, Webinars, Documentation, Implementation Guides, Reference Code, APIs, SDKs

yubico

Trust the Net