



Week 3: Add Enterprise Qualities

Unit 5: Authorization and Authentication

Authorization and Authentication Overview

Intro: Security and SAP Cloud Application Programming Model

- Authentication, identification, and authorization
- Integration in SAP Cloud Application Programming Model

Part 1: Add security to existing project

- Add annotations

Part 2: Custom code

- Add custom code

Part 3: SAP Cloud Platform

- Configuration in the cockpit



**Authorization &
Trust Management**



Intro: Security and SAP Cloud Application Programming Model

Business services in the public cloud need special protection

Authentication vs authorization

- Authentication and identification:
 - "Who am I?"
 - Integration in standard technologies and centralized services
- Authorizations:
 - "What am I allowed to do?"
 - Ensure compliance and auditability

Support by SAP Cloud Application Programming Model

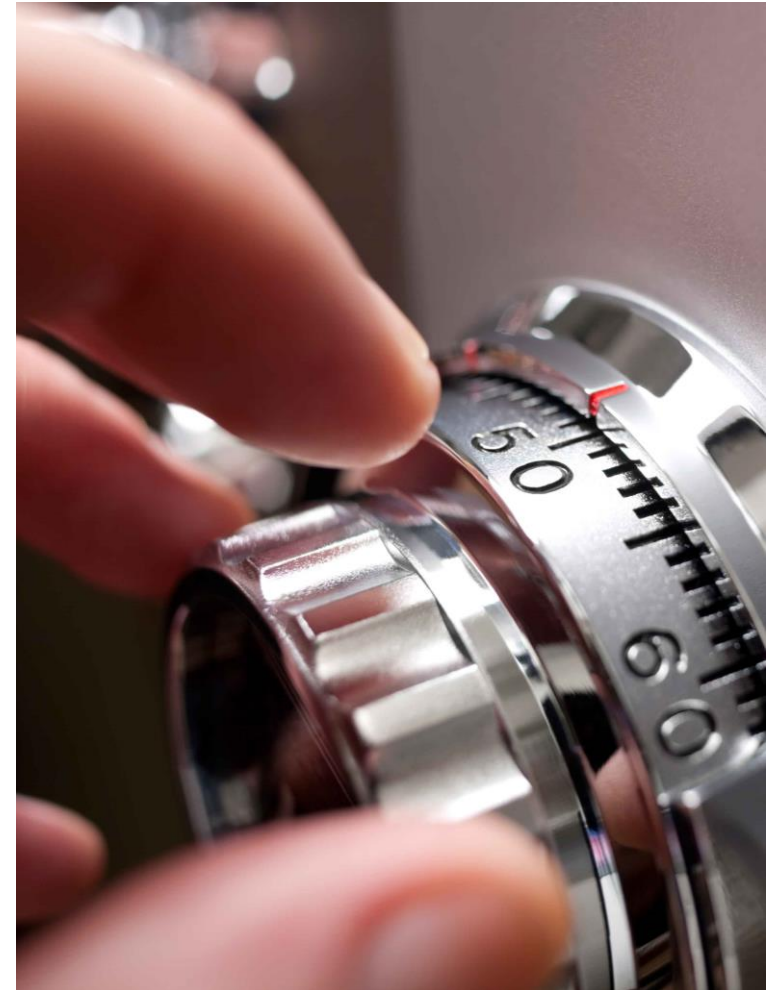
- Provides annotations to easily define the security settings in the CDS model
- Provides standard roles
- Provides APIs to enable authorization checks also in custom code
- Generates the authorization information for the `xs-security.json`



Intro: Security model

Basic ideas and principles

- Close integration between data/service model and authorization model
- Separation of concerns is possible
- Usage of the service-based modeling approach also for modeling the authorizations
(use data-centric authorizations only where it makes sense)



Intro: Annotations in CDS model

@restrict

- allows fine-grained control through an array of privileges given as grant statements in the form {grant:<operation>, to:<roles>, ...}.
- entity Orders @restrict: [{
grant: 'READ', to: 'authenticated-user',
where: 'createdBy = \$user'

@requires

- allows specifying one or more user roles (as a single string or an array of strings) that must be assigned to the current user (combined with OR)
- is just a convenience shortcut of @restrict
- service CustomerService @(requires:'authenticated-user')

Predefined roles

- authenticated-user, identified-user, system-user, any

Intro: Example with annotations on entity and service level

@requires on service level



service.cds

```
service CustomerService @(requires:'authenticated-user')  
  
service OwnerService @(requires:'owner_role')
```

@restrict on entity level



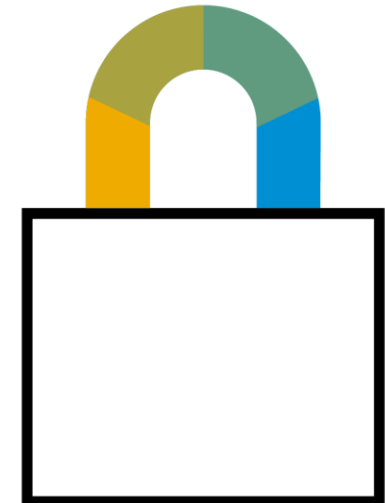
service.cds

```
entity Orders @restrict: [{  
  grant: 'READ', to: 'authenticated-user',  
    where: 'createdBy = $user'  
}]
```

Part 1: Add security to existing project

Goal:

- Use existing “bookshop” project and add security
 - Define `CustomerService` → can be invoked by all users who are authenticated
 - Define `OwnerService` → can only be invoked by users with role “owner_role”
- Privileges:
 - User with role “owner_role” can do everything
 - Authenticated user has the following permissions:
 - Can view all books and authors, but not create
 - Can create orders, but view only those orders which were created by her/him



Part 1: Add security to existing project

Implementation:

- No code required
- Add annotations to declare which roles are required



service.cds

```
service OwnerService @(requires: 'owner_role') {
    entity Books as projection on my.Books;
    . . .
}

service CustomerService @(requires: 'authenticated-user') {
    . . .
    entity Orders as projection on my.Orders;
}

annotate CustomerService.Orders with
    @restrict: [
        { grant: 'READ', to: 'authenticated-user', where: 'createdBy = $user' },
        { grant: 'UPDATE', to: 'authenticated-user', where: 'createdBy = $user' }
    ] ;
```


Part 1: Add security to existing project

Configuration for local execution:

- `package.json`
 - add and install dependency
`"passport": "*"`
- `.cdsrc.json`
 - configure mock users



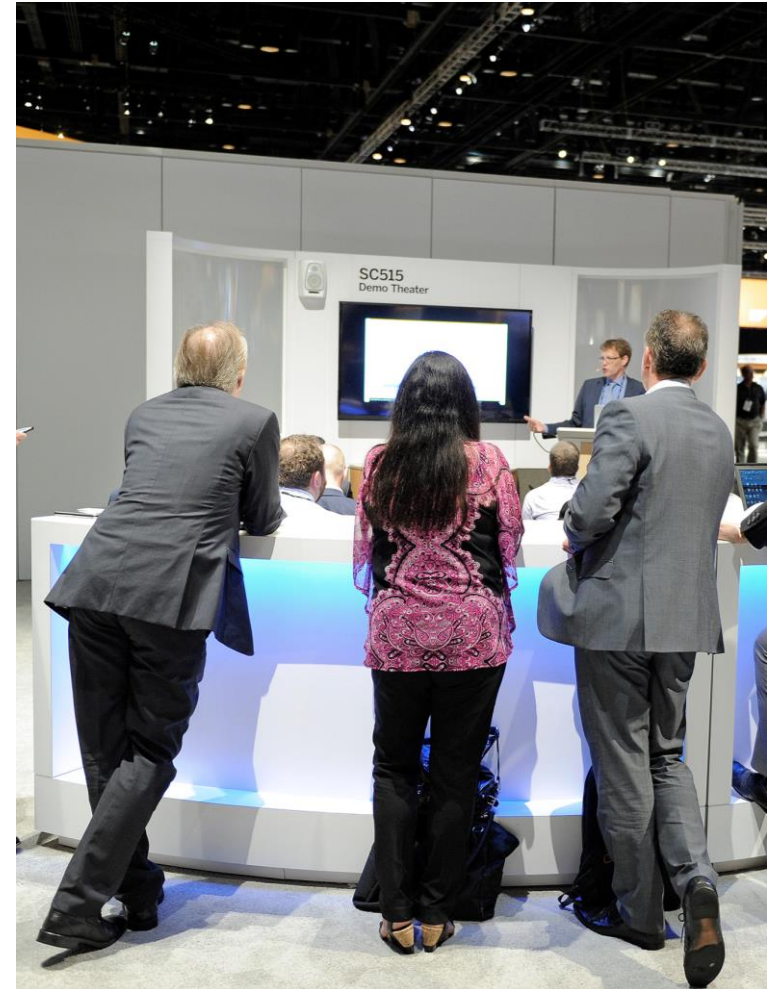
`.cdsrc.json`

```
"cds": {
  "auth": {
    "passport": {
      "strategy": "mock",
      "users": {
        "ottoowner": {
          "ID": "ottoowner@mail.com",
          "password": "123",
          "roles": [
            "owner_role"
          ]
        },
        "uteuser": {
          "ID": "uteuser@mail.com",
          "password": "123"
        }
      }
    }
  }
}
```

Demo 1: Add security to existing project

Demo:

- Add configuration – no code
- Test locally with mock users



Authorization and Authentication

Part 2: Writing custom code

API for accessing info of JWT token

- “Enforcement API”
- Used in handler implementation
- Manual check of required role
- Get info about the logged-in user

Examples

- Checking role:
- Checking user attribute:

 **service.js**

```
srv.before (['READ', 'CREATE'], 'Orders', req =>  
    req.user.is('admin') || req.reject(403)
```

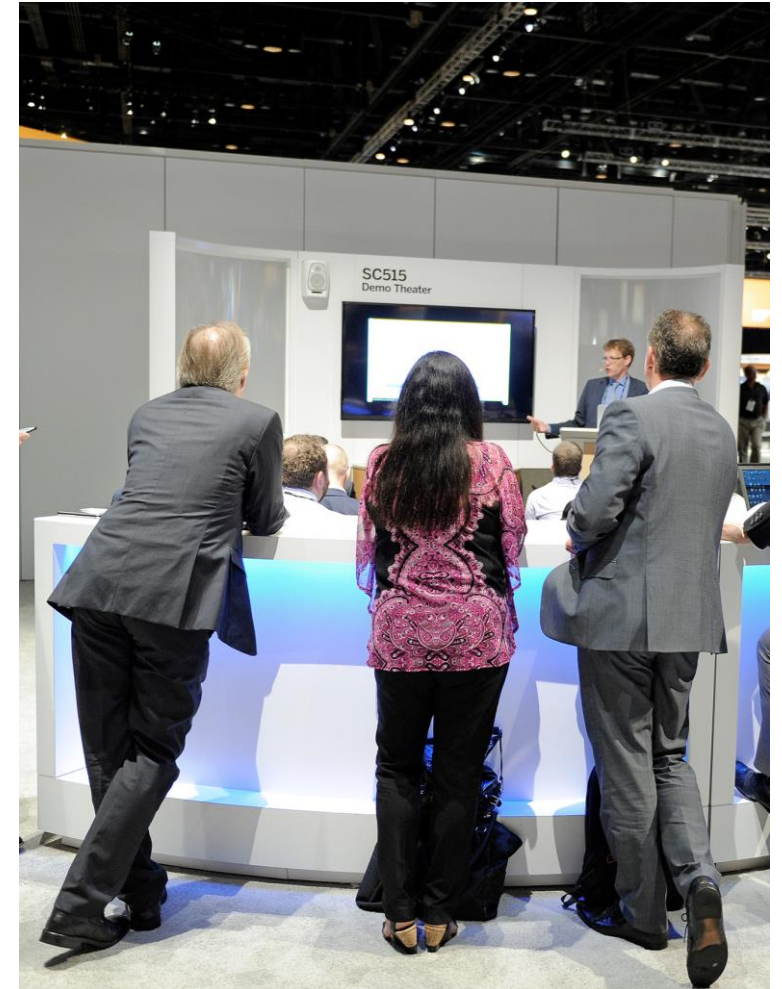
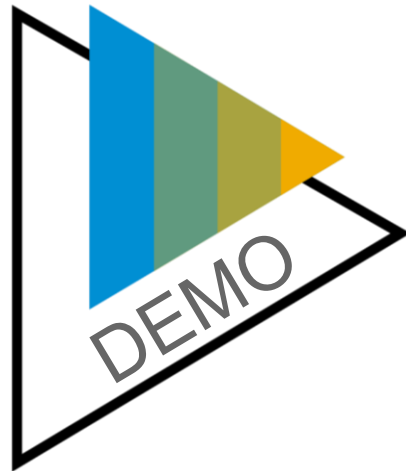
 **service.js**

```
srv.before ('*', 'Approval', req =>  
    Req.user.approverLevel > 1 || req.reject(403)
```

Demo 2: Custom code

Demo:

- Goal:
Before order creation, the code in the custom handler should do a specific check that cannot be done by the framework, for example, whether the user is allowed to create this specific order.



In SAP Cloud Platform, XSUAA is used for

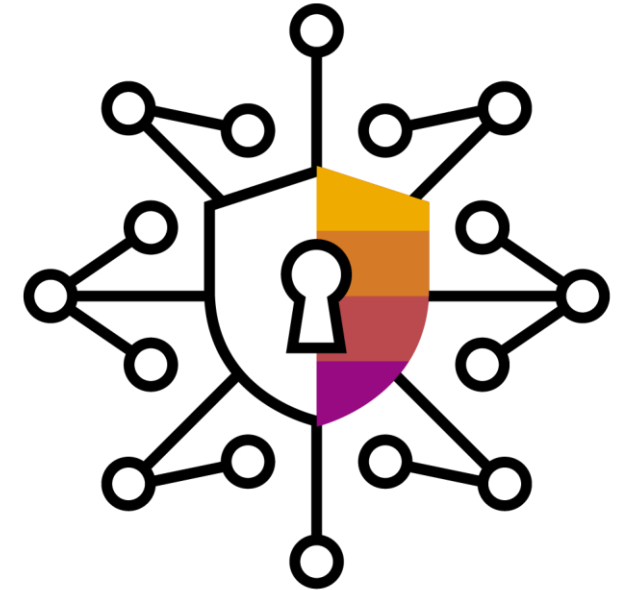
- OAuth2 security flow
- managing scopes/roles,
- validating users against identity provider, etc.

SAP Cloud Application Programming Model supports:

- generation of security artifacts out of CDS model

```
cds compile srv/ --to xsuaa > xs-security.json
```

- Node.js runtime uses `@sap/xssec` to communicate with XSUAA
- Runtime parses JWT token and provides “Enforcement API”



Implementation

- No code changes

Configuration

- `package.json`:
install required dependencies security configuration:
`xsuaa`
- `manifest.yml`:
add reference to the instance of *xsuaa* service
- `mta.yaml` (for bigger projects , same syntax))



package.json

```
"dependencies": {  
  "passport": "latest",  
  "@sap/xssec": "^2.2.3",  
  "@sap/audit-logging": "^3.0.2",  
  ...  
  "cds": {  
    "requires": {  
      "uaa": {  
        "kind": "xsuaa"  
      }  
    }  
  }  
}
```



manifest.yml

```
- name: myAppName  
  ...  
  services:  
    - xsuaaBookshop
```

SAP Cloud Platform: security configuration

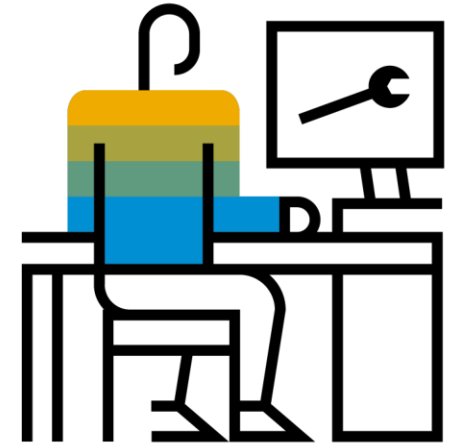
- Create XSUAA service instance with MTA
 - Add path to generated `xs-security.json` in `mta.yaml`:

```
- name: capire-bookshop-uaa
  type: org.cloudfoundry.managed-service
  parameters:
    path: ./xs-security.json
```

- Create role and role collection
 - Role is available according to CDS
 - Create role collection with name of choice
- Adapt trust configuration
 - Assign role collection to demo user
 - Add a second user without role assignment

Test

- Use REST client
 - to fetch token, then call service

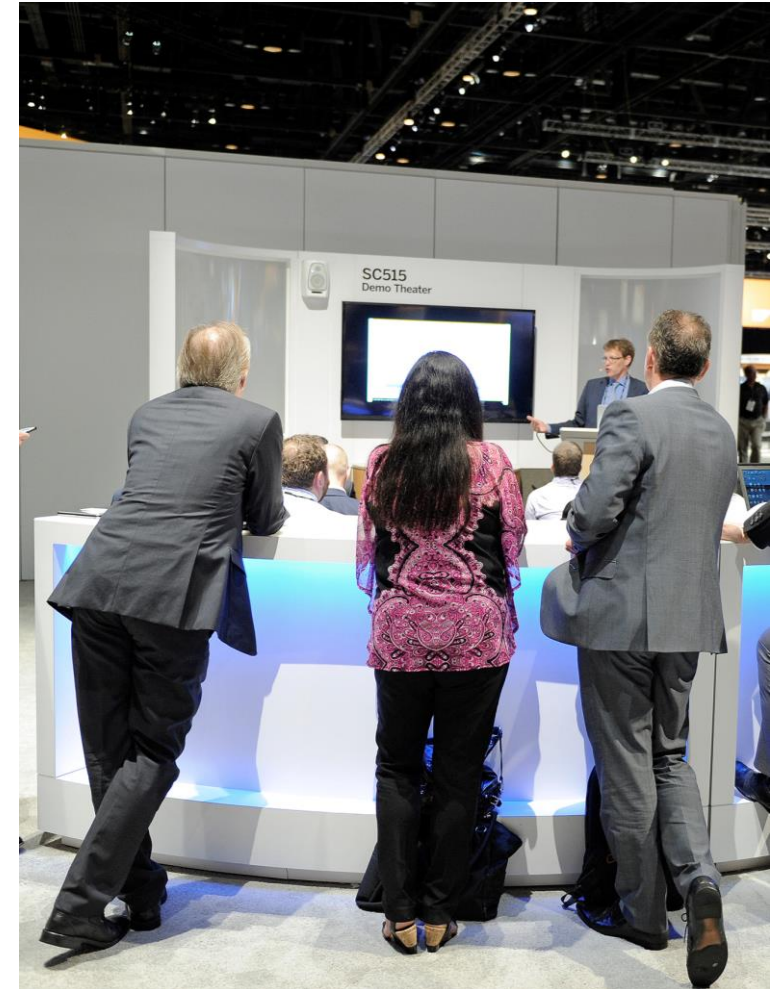
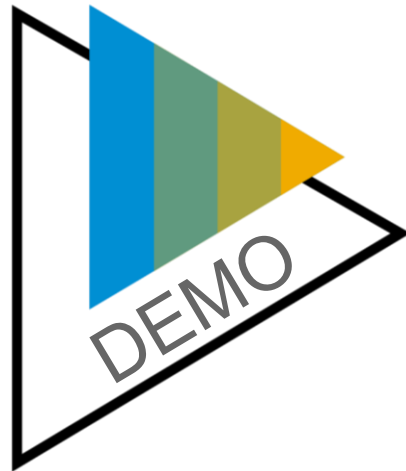


Authorization and Authentication

Demo 3: SAP Cloud Platform

Demo:

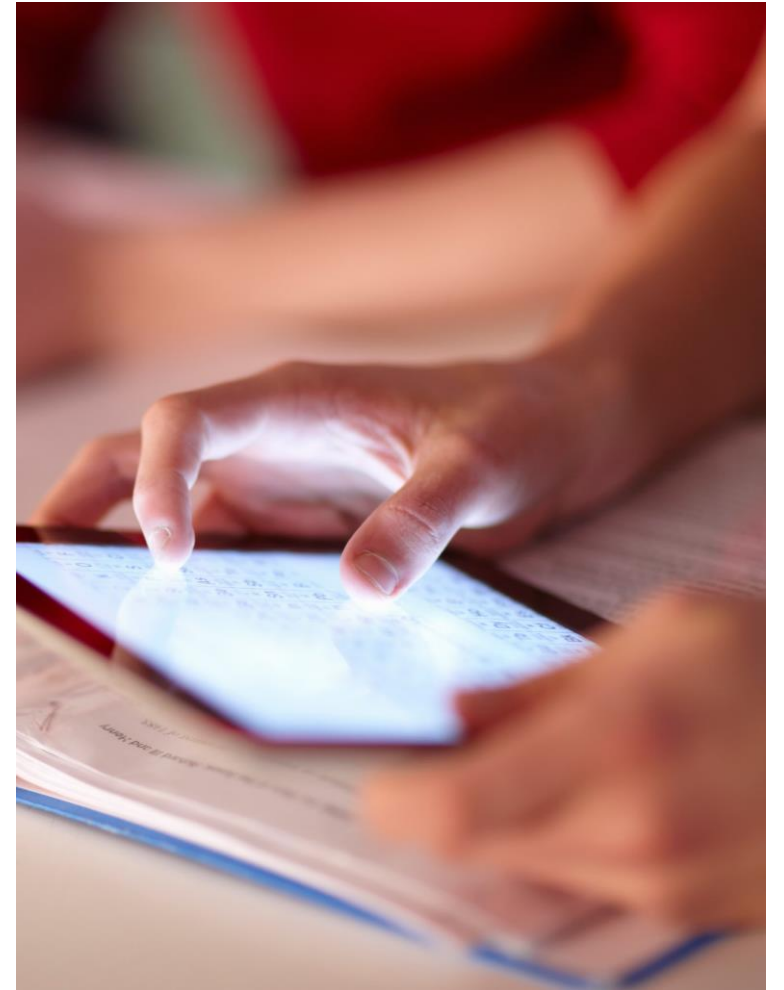
- Goal:
 - Test the CAP app in productive environment
 - Test the security handling with real users



What you've learned in this unit

In this unit, you've learned:

- Basic principles of *authentication, identification, authorization*
- How to define annotations in CDS model
`@requires`
`@restrict`
- How to access security information in custom code
- How to configure security settings in *SAP Cloud Platform*



Authorization and Authentication

Further reading



Additional Material

SAP Cloud Platform:
Sign up for trial account: see course week 1 unit 2

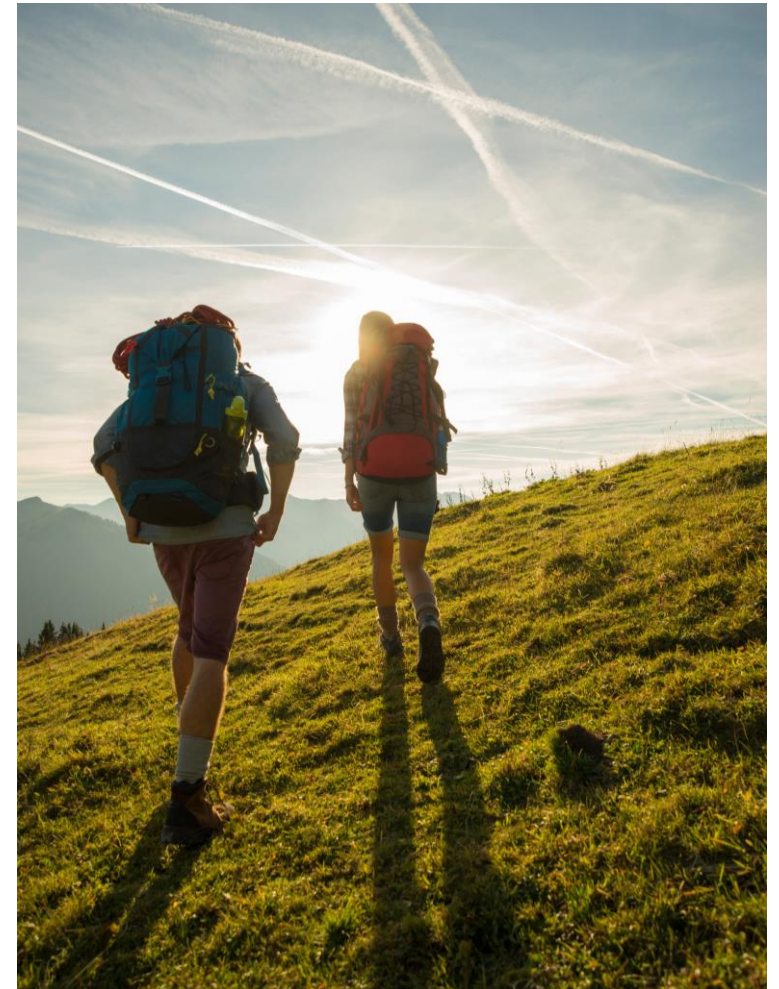
Documentation:

<https://cap.cloud.sap/docs/node.js/authentication>

<https://cap.cloud.sap/docs/guides/authorization>

SAP Help Portal:

[User Account and Authentication Service](#)



Thank you.

Contact information:

open@sap.com

Follow all of SAP



www.sap.com/contactsap

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platforms, directions, and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See www.sap.com/copyright for additional trademark information and notices.