



Visão Geral

Fundamentos

Configurando o serviço

Implementando Autenticação

Implementando Autorização

VISÃO GERAL

Nessa aula extra, vamos integrar nossa aplicação com o serviço de autenticação do Google. Você já deve ter visto vários aplicativos que permitem autenticar usando suas credenciais do Google. O nome disso é *Single Sign-on* (SSO). Existe muita teoria fundamentando isso, mas nosso propósito nessa aula é entender o básico e usar esse conceito para integrar nosso aplicativo.

Começamos entendendo alguns princípios fundamentais do SSO. Em seguida, vemos como configurar a API do Google usando o Google Cloud. Depois, é a hora de configurar a autenticação na aplicação. Por fim, configuramos a autorização.

Como de costume, os detalhes são apresentados nas seções seguintes, mas você pode assistir o vídeo abaixo para ter uma visão geral do processo.

Você também pode fazer o **download do PDF** do material dessa aula.

Diferente das demais aulas, esta aula **NÃO** tem *Verificação de Aprendizado*.

FUNDAMENTOS

Nesse momento, nossa aplicação implementa um mecanismo de autenticação baseado em sessão, implementado pelo Spring Security. Não tem nada de errado com isso, mas esse mecanismo exige que cada usuário seja cadastrado na aplicação. Apesar desse processo fazer sentido em vários cenários, hoje em dia é comum que uma mesma empresa tenha vários aplicativos. Se cada aplicativo usar um mecanismo individual de autenticação, cada usuário terá que gerenciar múltiplas senhas. Além disso, existe o trabalho de ter que cadastrar um novo usuário em cada aplicativo. Para minimizar esse trabalho, aplicativos podem usar uma *Identidade Federada* (IF).

Um sistema de gerenciamento de IF permite que um usuário use sua credencial de um *provedor de identidade* para ter acesso a outros serviços dentro de uma federação. O Google é um exemplo de *provedor de identidade*. Nesse contexto, uma federação é um conjunto de sites/serviços/aplicativos que aceitam a autenticação do *provedor de identidade*. O uso de IF permite um único login (*Single Sign-on* - SSO). Isso evita que um mesmo usuário precise se autenticar repetidas vezes.

Apesar de sistemas de IF já existirem há bastante tempo, SSO ganhou popularidade nos últimos anos graças a proliferação de serviços disponíveis na Internet. Exemplos de sistemas de IF são Kerberos e Shibboleth. Atualmente, SSO não apenas se tornou um requisito de usuários da Internet, como também é chave para uma experiência agradável para o usuário de aplicações em nuvem.

Os serviços/sites/aplicativos em uma federação devem usar um mesmo protocolo. Atualmente, o [OpenID Connect](#) é um protocolo de SSO usado por grandes empresas, como Google e Yahoo, e se tornou o padrão do mercado. O protocolo OpenID Connect considera os seguintes papéis:

- *Usuário final*, que representa a pessoa ou serviço que quer acessar um serviço ou aplicação. No nosso caso, esse é o usuário do aplicativo de gerenciamento de cidades;
- *Provedor de identidade*, que é um serviço externo (terceiro) responsável por autenticar o *usuário final*. No nosso caso, o Google é o *provedor de identidade*.
- *Parceiro (Relying party)*, que é o sistema de segurança implementado pela aplicação ou serviço que recebe a resposta da autenticação vinda do *provedor de identidade*. No nosso caso, esse papel é assumido pelo Spring Security, implementado na nossa aplicação.

Para habilitar SSO no nosso aplicativo, precisamos começar configurando o *provedor de identidade*, no nosso caso, o Google. Estamos usando o Google nesse exemplo porque a maioria das pessoas hoje em dia tem uma conta no Google. Contudo, é importante entender que o uso desse serviço é parte do portfólio de serviços do Google Cloud e, portanto, pode incorrer em cobrança.

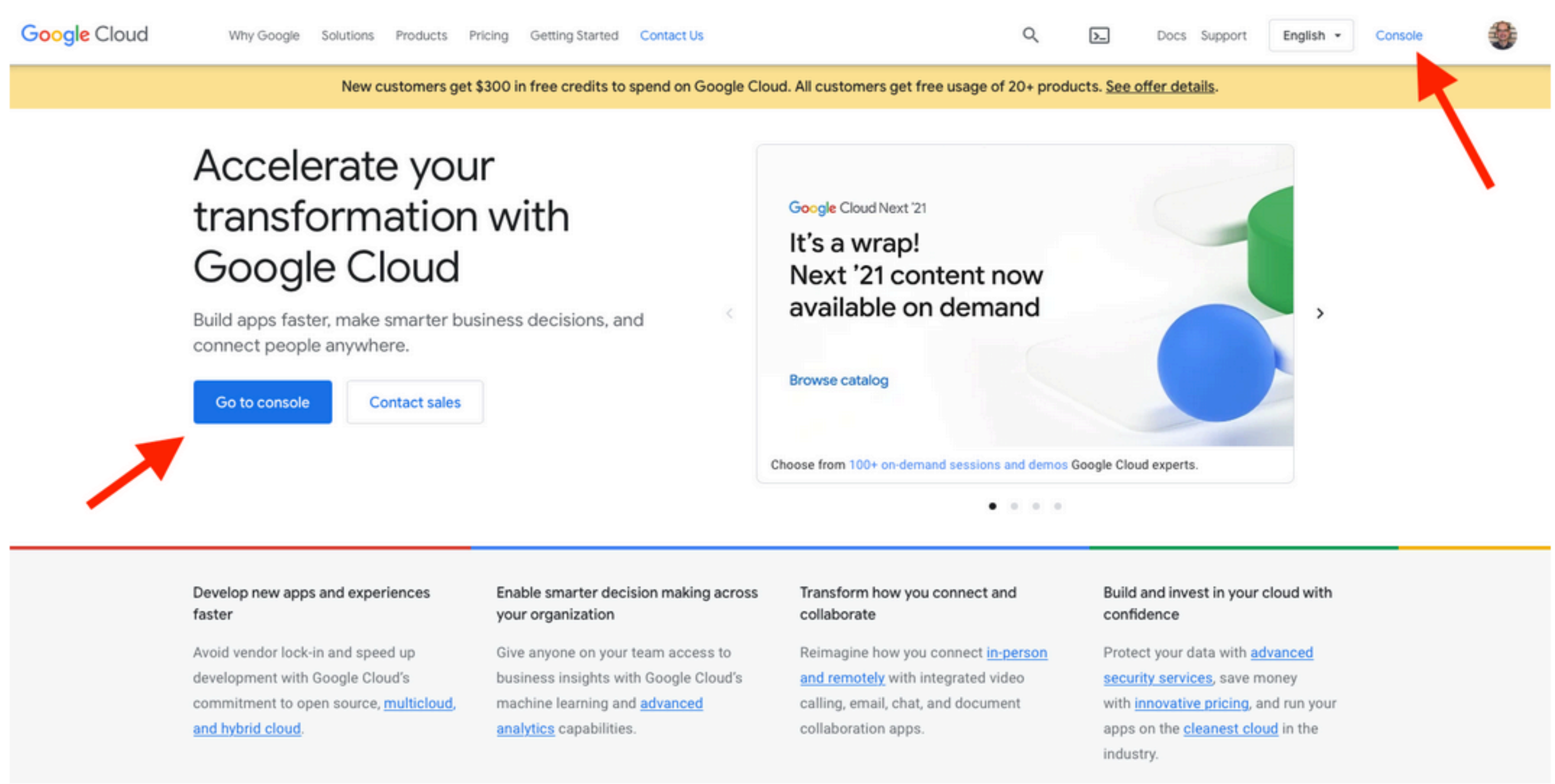
Nem o professor dessa disciplina, nem a coordenação dessa especialização se responsabilizam por cobranças realizadas pela implementação dessa aula. Por isso, se decidir implementar os conceitos dessa aula, siga por sua conta e risco.

CONFIGURANDO O SERVIÇO DE SSO NO GOOGLE

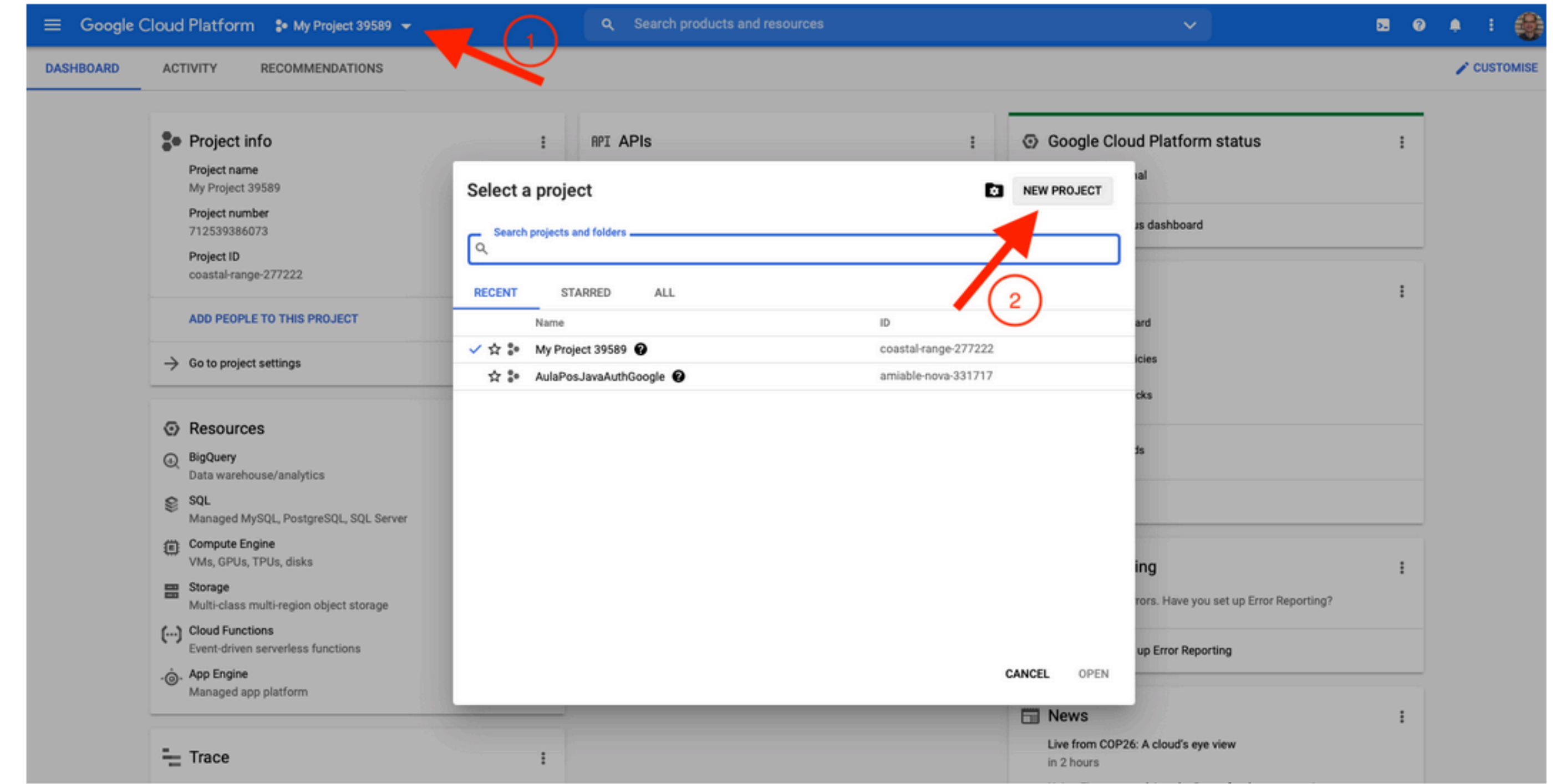
O tutorial nessa seção detalha o passo-a-passo do processo. Isso pode ser moroso demais para alguns. Se tiver pressa, consulte o vídeo no início dessa aula. Ele mostra esse processo de forma completa e rápida.

Comece acessando a página do [Google Cloud](#). Em seguida, acesse a console Web do Google Cloud clicando no botão no canto superior direito da tela, ou no botão que aparece na área central.


Se você nunca usou o Google Cloud, talvez precise se registrar e aceitar os termos de uso antes de começar a usar a console do sistema.




Se você já tem algum projeto criado no Google Cloud, você será direcionado para esse projeto. Se não tiver qualquer projeto, terá a opção de criar um novo. Vamos começar criando um novo projeto. Para isso, clique no botão à direita do logo, e selecione a opção *NEW PROJECT*.





Observe que a tela que você deve ver pode variar dependendo se você já tem um projeto no Google Cloud.


Google Cloud Platform



Search





Manage resources

 CREATE PROJECT

 CREATE FOLDER

MOVE

 Filter
Filter


<input type="checkbox"/>	Name	ID
<input type="checkbox"/>	<div>   No organization </div>	
<input type="checkbox"/>	<div>  My Project 39589 </div>	coastal-range-
<input type="checkbox"/>	<div>  AulaPosJavaAuthGoogle </div>	amiable-nova-

RESOURCES PENDING DELETION

Informe um nome para seu projeto e clique no botão *CREATE*.

Google Cloud Platform

New Project



You have 23 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)


Project name *

AulaPosJavaAuthGoogle

?

Project ID: amiable-nova-331717. It cannot be changed later. [EDIT](#)

Location *

 No organisation

[BROWSE](#)

Parent organisation or folder

CREATE

CANCEL

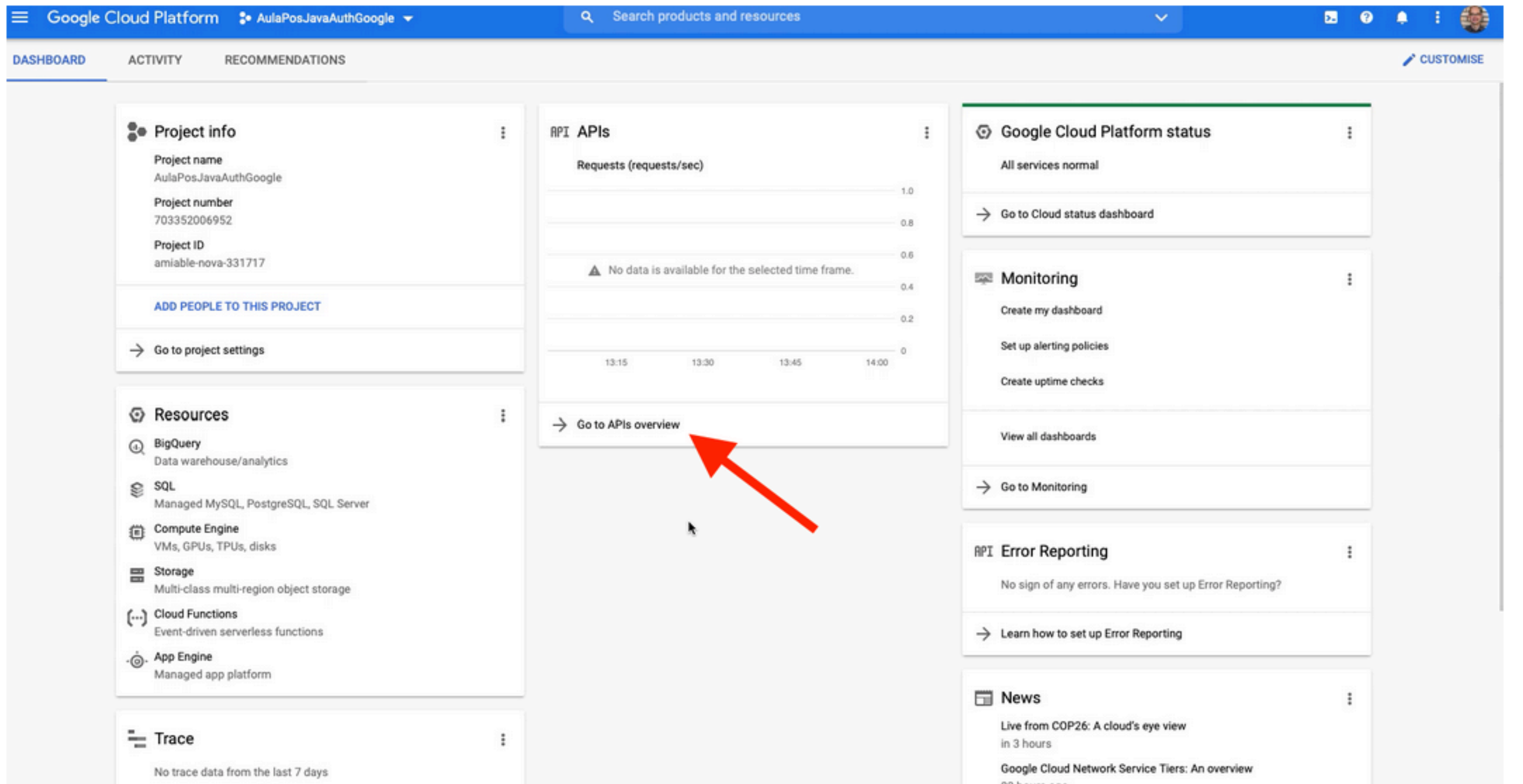
Aguarde até que o projeto seja criado. Em seguida, você será direcionado para a página do projeto. No lado superior esquerdo da tela, você pode ver o nome do seu projeto e alternar entre projetos. Na área central da tela, você pode observar várias opções e o componentes de monitoramento.

The screenshot shows the Google Cloud Platform console interface. At the top, the navigation bar includes the Google Cloud Platform logo, the project name 'AulaPos.JavaAuthGoogle' (highlighted with a red arrow), a search bar, and user profile icons. Below the navigation bar, the main content area is divided into several sections:

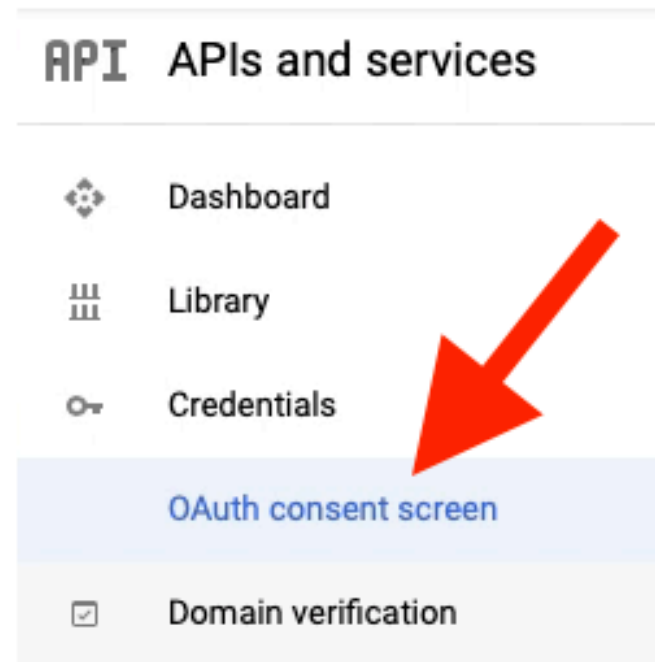
- Project info:** Displays project details such as Project name (AulaPos.JavaAuthGoogle), Project number (703352006952), and Project ID (amiable-nova-331717). It includes a link to 'ADD PEOPLE TO THIS PROJECT' and a 'Go to project settings' button.
- Resources:** Lists various Google Cloud services available to the project, including BigQuery, SQL, Compute Engine, Storage, Cloud Functions, and App Engine.
- Trace:** Shows a message indicating 'No trace data from the last 7 days' and a link to 'Get started with Trace'.
- APIs:** Displays a graph of 'Requests (requests/sec)' over time, with a message stating 'No data is available for the selected time frame.' and a link to 'Go to APIs overview'.
- Notifications:** A floating notification box on the right side of the screen, titled 'Notifications', contains three items:
 - 'Create Project: AulaPos.JavaAuthGoogle' with links 'SELECT PROJECT' and 'ADD PRINCIPALS'.
 - 'Create Project: GoogleAuthProject' with links 'SELECT PROJECT' and 'ADD PRINCIPALS'.
 - 'Protect your account from attackers' with a message about account security and links 'SECURE NOW' and 'DISMISS'.

At the bottom of the screen, a status bar indicates 'Now viewing project 'AulaPos.JavaAuthGoogle' in organisation 'No organisation''.

Na área central da tela, clique em *Go to APIs overview*.



No menu lateral esquerdo, clique em *OAuth consent screen*. O [OAuth2](#) é um outro protocolo, no qual o Open ID Connect está baseado. A diferença é que o OpenID Connect é um protocolo de autenticação, enquanto o OAuth2 é um protocolo de autorização. Vamos falar mais sobre isso na seção sobre autorização.



Você vai perceber que a tela de configuração tem um painel com informações à direita. Use esse painel se quiser mais detalhes sobre o processo.

As telas seguintes possuem várias opções que vão definir como o processo de autenticação deve se comportar para nossa aplicação. O Google facilita bastante as coisas explicando em um nível de detalhe muito bom cada etapa do processo.

Se você se sentir confuso com a quantidade de opções e telas, pode consultar o vídeo no início dessa aula. Ele resume o processo e pode servir como guia.

Na tela *OAuth consent screen*, selecione a opção *External* e, em seguida, clique em *CREATE*. Na tela seguinte, *Edit app registration / 1 - OAuth consent screen*, informe:

- *App name*: um nome para representar nosso aplicativo;
- *User support email*: normalmente seu e-mail, para que usuários possam entrar em contato caso tenham problemas;
- *Developer contact information*: seu e-mail para que o Google possa entrar em contato caso necessário.

Em seguida, clique em *SAVE AND CONTINUE*. Isso te leva para a próxima tela, *Scopes*. Um escopo representa um conjunto de informações que o *parceiro (relying party)* terá acesso. Nesse exemplo, vamos selecionar apenas o e-mail. Para isso, clique em *ADD OR REMOVE SCOPES*, selecione a opção */auth/userinfo.email* e, em seguida, clique em *UPDATE*. Você verá o escopo listado em *Your non-sensitive scopes*. Agora, siga com o processo clicando em *SAVE AND CONTINUE*.

Na tela seguinte, você precisa informar quais usuários poderam acessar sua aplicação. É claro que em um cenário normal não tem como saber antecipadamente quem vai usar a aplicação. Mas, nesse exemplo, estamos criando um teste, dentro de um escopo de desenvolvimento. Por isso, o Google não permite extrair informações do usuário ao menos que ele esteja registrado aqui.

Para isso, basta clicar em *ADD USERS* e informar os e-mails dos usuários do Google que poderão acessar seu aplicativo. Agora, siga com o processo clicando em *SAVE AND CONTINUE*. A tela que segue é a última tela do processo e apresenta informações para confirmação. Clique em *BACK TO DASHBOARD* para finalizar o processo.

Registrar e-mails é necessário apenas caso você queira extrair informações do usuário que se autenticou na sua aplicação. Não vamos fazer isso nessa aula. Por isso, se quiser, não precisa preencher qualquer e-mail nesse passo.

De volta ao *DASHBOARD*, selecione *Credentials*, no menu lateral. Na tela seguinte, selecione *CREATE CREDENTIALS* no menu superior, e escolha a opção *OAuth client ID*.

Google Cloud Platform

AulaPosJavaAuthGoogle

Search products and resources

APIs and services

Dashboard

Library

Credentials

OAuth consent screen

Domain verification

Page usage agreements

Credentials

+ CREATE CREDENTIALS

DELETE

Create credentials to access your data

API keys

OAuth 2.0 Client IDs

Service Accounts

API key

Identifies your project using a simple API key to check quota and access

OAuth client ID

Requests user consent so that your app can access the user's data

Service account

Enables server-to-server, app-level authentication using robot accounts

Help me choose

Asks a few questions to help you decide which type of credential to use

No API keys to display

No OAuth clients to display

No service accounts to display

Quando um *usuário final* quer se autenticar, o *provedor de identidade* precisa saber para onde ele deve enviar o resultado da autenticação. É justamente isso que vamos fazer nesse processo.

←

Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type *

Web application

Name *

GerenciamentoCidades

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

❗

The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorised domains](#).

Authorised JavaScript origins ⓘ

For use with requests from a browser

+ ADD URI

Authorised redirect URIs ⓘ

For use with requests from a web server

URIs *

http://localhost:8080/login/oauth2/code/google

+ ADD URI

CREATE

CANCEL

Na tela *CREATE OAuth client ID*, informe o tipo da aplicação (*Web application*) e um nome descritivo (*Name*) nos campos apropriados.

Na seção *Authorised redirect URIs*, informe a url `http://localhost:8080/login/oauth2/code/google`. Com exceção do `localhost`, o restante da URL é um endereço padrão definido no próprio Spring Security, e não deve ser alterado. Esse endereço diz ao Google para entregar o *token* de acesso nesse endereço.

Por padrão, a URL deve ser protegida por SSL/TLS. A única exceção quando a URL inicia com `localhost`, que normalmente é usado para fins de teste e/ou desenvolvimento.

Após inserir a URL, clique no botão *CREATE*. Se tudo deu certo, o Google irá apresentar a tela *OAuth client created*. Nessa tela, o Google vai apresentar seu **Cliente ID** e **Client Secret**. Essas são as credenciais de acesso que sua aplicação vai usar para iniciar o processo de autenticação. Observe que a tela oferece a opção de fazer o download dessas credenciais.

Você deve fazer o download das credenciais ou salvá-las em lugar seguro.

OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

❗

OAuth access is restricted to the [test users](#) listed on your [OAuth consent screen](#)

Your Client ID

7[REDACTED]c

Your Client Secret

G[REDACTED]o

⬇️

DOWNLOAD JSON

OK

IMPLEMENTANDO A AUTENTICAÇÃO

O [Spring Security](#) implementa nativamente os protocolos usados para SSO no Google. Dessa forma, basta adicionar a dependência `spring-boot-starter-oauth2-client` no `pom.xml` para que o Spring Boot entenda e pré-configure o SSO.

Apesar do Spring Boot configurar automaticamente as opções básicas do SSO, algumas informações precisam ser adicionadas. Isso inclui o `client-id` e o `client-secret`, gerados pelo Google no processo de criação das credenciais.

Abra o arquivo `src/main/resources/application.properties` e insira as seguintes linhas:

- `spring.security.oauth2.client.registration.google.client-id=SEU-CLIENT-ID-GERADO-PELO-GOOOGLE`
- `spring.security.oauth2.client.registration.google.client-secret=SEU-CLIENT-SECRET-GERADO-PELO-GOOOGLE`

Agora, o próximo passo é instruir o Spring Security para usar a autenticação do Google, em vez do banco de dados usado anteriormente. Para isso, abra a classe `br.edu.utfpr.cp.esjava.crudcidades.SecurityConfig` e apague as seguintes linhas:

- `.formLogin()`
- `.loginPage("/login.html").permitAll()`
- `.defaultSuccessUrl("/", false)`
- `.and()`
- `.logout().permitAll();`

Essas linhas de código acima definem o acesso à página de login, à página padrão quando o login é bem sucedido, e à página de logout. Como o login agora será terceirizado, precisamos substituir as linhas deletadas acima pela seguinte linha: `.oauth2Login().permitAll();`. Essa linha define o acesso à página de login do Google.

As linhas 36 a 44, que definem os métodos `cifrador` e `printSenhas` também não são mais necessárias e podem ser deletadas.

Um último ajuste que precisa ser realizado nesse arquivo é que estamos definindo a autenticação, e não a autorização. Dessa forma, para permitir que o usuário, após a autenticação, tenha acesso à página de listagem de cidades, precisamos fazer um pequeno ajuste.

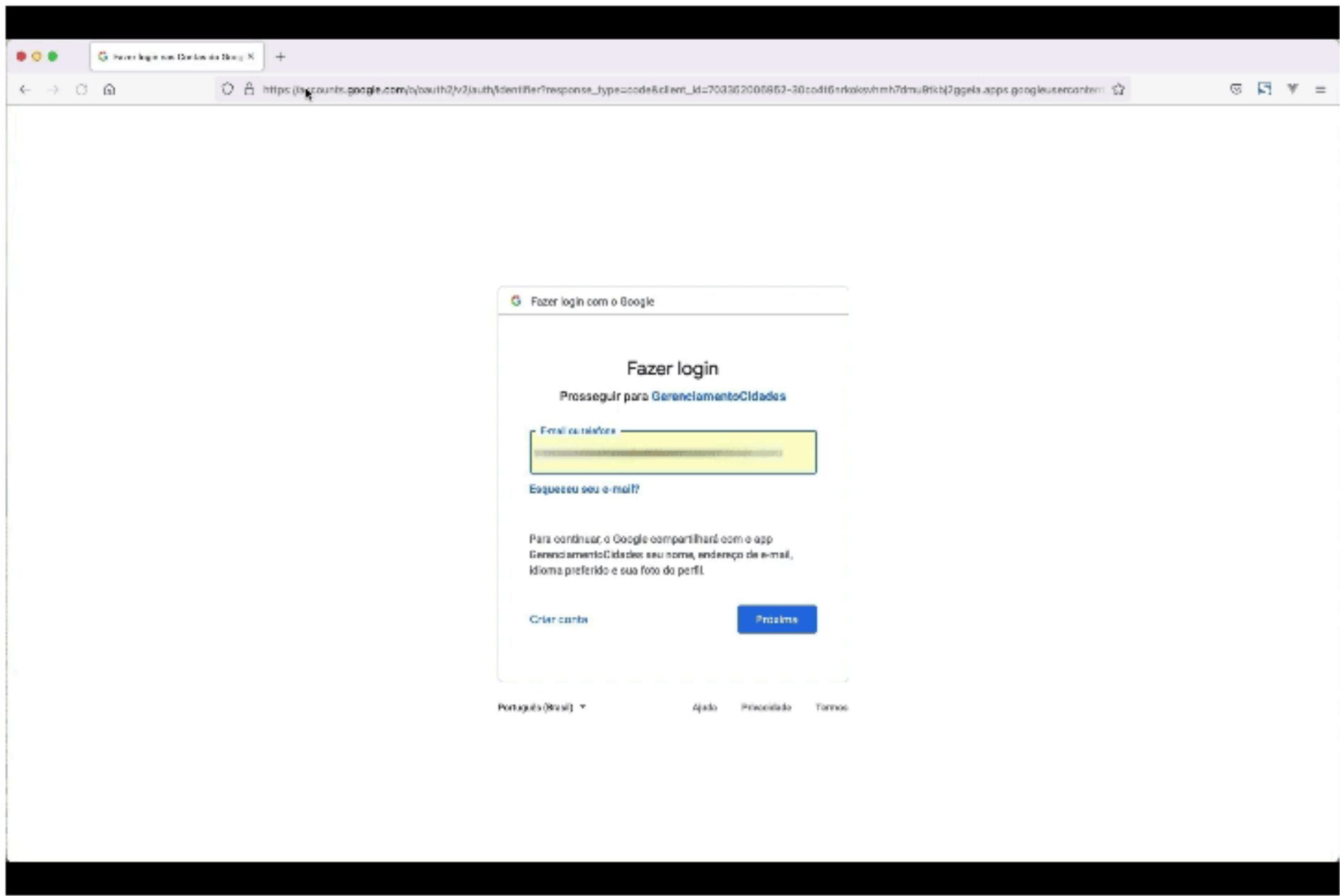
Comente a linha `.antMatchers("/").hasAnyAuthority("listar", "admin")`, e substitua essa linha por: `.antMatchers("/").authenticated()`. Essa nova regra permite que qualquer usuário autenticado possa acessar a URL "/", que leva à página de listagem de cidades.

Para garantir que está tudo funcionando como deveria, vamos também excluir as classes e arquivos que eram usados para gerenciar os usuários no banco de dados. Os arquivos que podem ser excluídos são: `br.edu.utfpr.cp.esjava.crudcidades.usuario.Usuario.java`, `br.edu.utfpr.cp.esjava.crudcidades.usuario.UsuarioDetailsService.java`, `br.edu.utfpr.cp.esjava.crudcidades.usuario.UsuarioRepository.java`, e `src/main/resources/data.sql`.

Observe que você ainda pode usar o banco de dados em conjunto com SSO para, por exemplo, ligar o usuário a um conjunto de papéis.

Execute a aplicação e você deve ver algo similar à figura abaixo. Faça sua autenticação com seu login do Google e você deve acessar a página de gerenciamento de cidades normalmente.

Note que eu tenho *Multi Factor Authentication (MFA)*. Isso significa que o Google verifica toda vez que autentico usando um dispositivo diferente. Nesse caso, ele enviou um código de verificação para meu tablet. Se você não tiver MFA ativado, você deve conseguir acessar a página logo após inserir seu login e senha.



O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana09-10-google-authentication`.

IMPLEMENTANDO A AUTORIZAÇÃO

Se você tentou criar uma cidade com o código desenvolvido na aula anterior, deve ter recebido um erro por falta de autorização. Isso acontece porque implementamos a autenticação, mas não a autorização. As regras de autorização na classe `br.edu.utfpr.cp.esjava.crudcidades.SecurityConfig` definem que apenas usuários com o papel (*authority*) `admin` podem realizar operações no sistema.

No mecanismo de autenticação em banco, padrão do Spring Security, os papéis estavam definidos no banco de dados. Mas, agora, com a autenticação sendo gerenciada pelo Google, como vamos definir os papéis?

É importante ressaltar que um não exclui o outro. Você pode ainda ter um banco de dados onde guarda os papéis que serão atribuídos aos usuários. Por exemplo, você poderia ter uma tabela que associe o e-mail do usuário e o papel que ele pode assumir. Assim, quando um novo usuário logar no sistema, o aplicativo verifica no banco qual é o papel desse usuário.

O Spring Security fornece os mecanismos que precisamos para atribuir um papel para um usuário de forma simples. Para isso, abra a classe `br.edu.utfpr.cp.esjava.crudcidades.SecurityConfig` e crie um novo método, com a seguinte assinatura: `public GrantedAuthoritiesMapper userAuthoritiesMapper()`. Decore o método com a anotação `@Bean`. Dessa forma, o Spring saberá que precisa gerenciar o retorno do método.

Esse método retorna o mapeamento de papéis para um usuário que logou no sistema. Dentro desse método você pode implementar a lógica que precisar. Por exemplo, verificando no banco de dados o e-mail do usuário e os papéis que ele pode exercer. Por fim, o retorno do método é um conjunto de mapeamentos que liga os papéis atuais, com os papéis que usuário pode exercer.

Esse exemplo é para ser simples, por isso, não vamos implementar qualquer mapeamento mais complexo aqui. Vamos apenas dizer que todo usuário que logar no sistema deve assumir o papel de `admin`. Para isso, o método `userAuthoritiesMapper()` terá somente a seguinte linha de código: `return (authorities) -> Set.of(new SimpleGrantedAuthority("admin"));`

O último passo é instruir o Spring Security para usar esse método quando alguém fizer o login usando o Google. Para isso, altere a linha `.oauth2Login().permitAll();` para `.oauth2Login().userInfoEndpoint().userAuthoritiesMapper(userAuthoritiesMapper());`

Pronto! Ao executar o código você será capaz de criar, alterar, excluir e listar cidades.

O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana09-20-google-authorisation`.