Introdução a Arquitetura REST

Representational State Transfer, ou simplesmente REST, é um estilo de arquitetura para sistemas distribuídos com base em princípios da web, como HTTP, URI e formato de dados como JSON ou XML, para permitir a comunicação entre sistemas distintos de forma simples, escalável e padronizada. Essa arquitetura é amplamente utilizada para a criação de APIs (Application Programming Interface) e serviços web, permitindo que diferentes aplicativos se comuniquem e compartilhem recursos entre si. O REST enfatiza a interoperabilidade entre diferentes sistemas e a separação de preocupações entre servidor e cliente, tornando os sistemas mais flexíveis, fáceis de manter e evoluir.

A arquitetura REST foi criada pelo cientista da computação Roy Fielding durante seu doutorado pela Universidade da Califórnia. Diferentemente de web services SOA (Simple Object Access Protocol), que é um protocolo, REST não é um protocolo e sim uma arquitetura que faz uso do protocolo HTTP como solução para requisições e respostas. É importante destacar que API REST não é a única nomenclatura aceita, embora seja a mais utilizada, podemos ver referências citando API RESTful ou Web Service REST.

O que é API REST?

Uma API REST é um tipo de API baseado na arquitetura REST que permite a comunicação entre diferentes sistemas de softwares. Ela define um conjunto de operações (métodos HTTP) que podem ser usadas para criar, ler, atualizar e excluir recursos (dados) em um servidor web. Uma API REST usa URIs (Identificadores Uniformes de Recursos) para identificar os recursos e o formato JSON ou XML para representar dados transferidos entre os sistemas. Com uma API REST, diferentes aplicativos podem se comunicar e compartilhar recursos entre si de forma padronizada e escalável. Isso permite a criação de soluções de softwares mais flexíveis e integráveis.

– Clientes

Quando você acessa uma API como a do Youtube ou Google Maps, o acesso será realizado por meio de uma aplicação cliente. Essa aplicação pode ser o navegador web do seu laptop, um aplicativo em seu smartphone, tablet ou até mesmo smart tv. As APIs REST podem ser classificadas em três tipos com base em seu nível de acesso: privadas, públicas e híbridas.

API REST privada: é uma API que só pode ser acessada por aplicativos e sistemas específicos, geralmente dentro da mesma organização ou rede. Isso garante que somente aplicativos autorizados possam usar a API e acessar os recursos protegidos.

API REST pública: é uma API que pode ser acessada por qualquer aplicativo por meio da internet. Essas APIs são frequentemente usadas por desenvolvedores para integrar seus aplicativos com serviços populares de terceiros, como Google Maps, Facebook, Twitter, etc.

API REST híbrida: é uma combinação de API pública e privada, onde a API pode ser acessada por qualquer aplicativo, mas alguns recursos e funcionalidades são protegidos e só podem ser acessados por aplicativos autorizados.

A escolha de qual tipo de API REST usar depende do nível de segurança necessário e dos recursos que se deseja disponibilizar para os aplicativos clientes. As APIs privadas são geralmente usadas para proteger dados sensíveis e garantir a segurança dos aplicativos internos. As APIs públicas são usadas para disponibilizar serviços para o público em geral e para promover a integração de aplicativos de terceiros. As APIs híbridas são usadas quando é necessário equilibrar a segurança dos dados com a acessibilidade pública dos recursos.

Uma aplicação cliente poderá ser desenvolvida em qualquer linguagem de programação que faça uso do protocolo HTTP. Essa é uma vantagem para os desenvolvedores de aplicações clientes, já que não ficam obrigados a utilizar uma linguagem de programação que não estão habituados ou mesmo que não tenham o domínio necessário.

– O que são Recursos

Os recursos são os registros que as APIs REST fornecem a seus clientes. Eles podem ser de diversos tipos como textos, imagens, vídeos, etc. Os recursos são representados por URIs (Uniform Resource Identifiers), que são endereços únicos que identificam o recurso a ser acessado. Cada recurso pode ser acessado e manuseado usando um conjunto de métodos HTTP (como GET, POST, PUT, DELETE) que são definidas pela API. Os recursos são a base da arquitetura REST e permitem que os clientes manipulem os dados da API de forma padronizada e intuitiva.

– Documentação

A documentação de uma API REST é formada por um conjunto de informações que descrevem a funcionalidade, os recursos e os endpoints disponíveis na API. Ela é destinada aos desenvolvedores que desejam consumir a API por meio de aplicações clientes e fornece detalhes sobre como acessar e usar seus recursos.

A documentação da API REST também pode incluir exemplos de código que mostram como usar a API em diferentes linguagens de programação e plataformas, bem como informações de autenticação e segurança, limites de taxa de solicitação e outras informações relevantes. Uma boa documentação da API REST é importante para garantir que os desenvolvedores de aplicações cliente possam integrar facilmente a API em seus aplicativos, entender como usar seus recursos e resolver problemas quando eles surgem.

Benefícios de APIs REST

O modelo arquitetural REST se tornou preferível em relação ao SOAP com o passar dos anos. Isso ocorreu por várias razões, algumas delas citadas a seguir:

Escalabilidade: As APIs REST são altamente escaláveis e podem lidar com grandes volumes de solicitações sem comprometer o desempenho ou a disponibilidade. Isso é possível devido ao fato da arquitetura REST ser Stateless, ou seja, não é mantido um estado ou contexto entre a aplicação cliente e o servidor.

Flexibilidade: As APIs REST permitem que os desenvolvedores criem soluções de software flexíveis e modulares, que podem ser facilmente adaptadas para atender às necessidades em constante mudança dos usuários.

Padronização: As APIs REST são baseadas em padrões abertos e amplamente adotados, como HTTP, URIs e JSON ou XML, o que torna mais fácil para os desenvolvedores integrar diferentes sistemas de software.

Reutilização: As APIs REST permitem que os recursos e serviços sejam compartilhados e reutilizados em diferentes aplicativos, o que pode economizar tempo e recursos de desenvolvimento.

Integração: As APIs REST permitem a integração fácil de diferentes aplicativos e sistemas de software, permitindo que as organizações compartilhem dados e informações entre si.

Acessibilidade: As APIs REST podem ser acessadas por diferentes plataformas e dispositivos, incluindo navegadores da web, aplicativos móveis e dispositivos IoT, o que amplia o alcance da solução de software.

Segurança: As APIs REST oferecem recursos de segurança, como autenticação e autorização, para garantir que apenas usuários autorizados possam acessar os recursos protegidos.

Esses benefícios tornam as APIs REST uma solução popular e eficiente para a construção de aplicativos e sistemas de software modernos e escaláveis.

Segurança em APIs REST

Existem diferentes tipos de segurança que podem vir a ser utilizados em APIs REST e assim, proteger os dados e garantir a integridade das solicitações e respostas. Alguns dos tipos de segurança mais aplicados em API REST são:

Autenticação: verificar a identidade do usuário que está solicitando acesso à API. Os métodos de autenticação incluem senhas, tokens de acesso, certificados digitais e autenticação de dois fatores.

Autorização: verificar se o usuário tem permissão para acessar o recurso solicitado na API. Isso é geralmente feito com base em regras de acesso predefinidas, como as funções do usuário.

Criptografia: protege os dados em trânsito, tornando-os inelegíveis para qualquer pessoa que possa interpretá-los. A criptografia pode ser aplicada a diferentes níveis, como TLS/SSL no nível de transporte ou criptografia ponta a ponta para dados sensíveis.

Monitoramento de atividades: rastreia as atividades do usuário na API para detectar comportamentos suspeitos e garantir a conformidade com as políticas de segurança. Isso pode incluir registros de auditoria, monitoramento de eventos e alertas de segurança.

Gerenciamento de tokens: protege os tokens de acesso usados na autenticação, garantindo que apenas usuários autorizados possam acessar os recursos da API. Isso pode incluir a validação de tokens, a expiração de tokens e a revogação de tokens em caso de comprometimento de segurança.

Os diferentes tipos de segurança são de grande valia para garantir que as APIs REST se mantenham seguras e protejam os dados confidenciais dos usuários e das organizações.

API REST - Fundamentos

API REST - Fundamentos

A arquitetura REST trabalha sobre o protocolo HTTP e cada URI funciona como um recurso da API. Portanto, devemos usar substantivos para determinar recursos em vez de verbos.

URI - Uniform Resource Identifier, ou Identificador Uniforme de Recursos

Exemplo: https://mballem.com/cursos/

A URI une o Protocolo (https://);

A localização do recurso (URL - mballem.com);

E o nome do recurso (URN - /cursos/)

Endpoints estilo RPC usam verbos, por exemplo, api/v1/getPersons. Em contrapartida, em REST, esse endpoint deve ser escrito como api/v1/persons. Então, você deve estar se perguntando como poderemos diferenciar as ações executadas em um recurso REST? Pois bem, é agora que faremos uso dos chamados Métodos HTTP (HTTP METHOD). A ideia é fazer com que os métodos HTTP trabalhem como um verbo, por exemplo, GET (recuperar), DELETE (remover), POST (criar), PUT (modificar) e PATCH (atualização parcial), entre outros.

Endpoint

O endpoint REST é um URI exclusivo que representa um recurso REST.

Por exemplo, http://app/api/v1/persons é um endpoint REST. Além disso, /api/v1/persons é o caminho do endpoint e as pessoas são recurso REST.

Métodos HTTP

O protocolo HTTP define um conjunto de métodos ou verbos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. Em APIs REST alguns desses verbos são mais comuns que outros de serem utilizados. Isso acontece não por preferência, mas por conta dos tipos de ações sobre os recursos. Por exemplo, o método GET é utilizado para operações que recuperam um recurso e provavelmente uma API REST terá muito mais ações do tipo GET do que do tipo DELETE (usado para exclusão de recursos).

Um conceito muito falado quando começamos a estudar REST é quais métodos são ou não são idempotentes. Na verdade, esse não seria um conceito específico da arquitetura REST, mas sim, do protocolo HTTP. Porém, por algum motivo, sempre foi muito citado em estudos sobre REST.

Um método HTTP idempotente é um método HTTP que pode ser chamado várias vezes sem alterar os resultados. Não importa se o método é chamado apenas uma vez ou dezenas de vezes. O resultado deve ser o mesmo. Importante destacar, idempotência se aplica ao resultado, não ao recurso. Entretanto, esse assunto pode gerar controvérsias, assim como vemos na postagem O que é ser idempotente em REST? O debate continua do site Infoq.

GET

O GET é usado para recuperar um recurso do servidor. É uma operação segura e idempotente, ou seja, várias solicitações para o mesmo recurso retornarão o mesmo resultado e não modificará o estado do recurso no servidor.

Alguns casos de uso do GET:

Recuperar uma página da web

Buscar a lista de postagens de um blog

Download de arquivo de um servidor

POST

POST é usado para criar um novo recurso no servidor. Não é considerado um método idempotente, o que significa que várias solicitações para criar o mesmo recurso resultará em vários recursos sendo criados. As solicitações POST também podem modificar o estado do recurso no servidor.

Alguns casos de uso do POST:

Criação de uma nova conta de usuário

Carregar uma nova imagem ou arquivo

Enviar um formulário a um servidor para criar um novo recurso

PUT

PUT é usado para atualizar um recurso existente no servidor. É considerado idempotente, o que significa que várias solicitações para atualizar o mesmo recurso resultarão em o mesmo estado do recurso no servidor. O PUT também pode criar um novo recurso se o recurso que está sendo atualizado não existe.

Casos de uso para PUT:

Atualizar o conteúdo de uma postagem de blog

Modificar os detalhes de uma conta de usuário

Substituir um arquivo ou imagem em um servidor por um nova versão

Atualizar todo o endereço de um contato

PATCH

O método PATCH é usado para fazer uma atualização parcial em um recurso existente no servidor. O objetivo é realizar atualizações em campos ou propriedades específicas de um recurso, sem exigir que todo o recurso seja enviado ao servidor.

Casos em que o PATCH é usado:

Atualizar o título ou descrição de uma postagem de blog

Alterar a senha ou endereço de e-mail associado a uma conta de usuário

Fazer uma atualização parcial de um arquivo ou imagem

DELETE

O verbo DELETE é usado para excluir um recurso do servidor. É considerado idempotente, o que significa que várias solicitações para excluir o mesmo recurso resultarão no mesmo estado do servidor. Isso parece estranho e gera muitas dúvidas. Mas na verdade, ao executar a primeira solicitações de DELETE para o recurso, ele será excluído e a resposta será 200 (OK) ou 204 (No Content). As próximas solicitações para o mesmo recurso (já excluído) retornarão 404 (não encontrado). Claramente, a resposta é diferente da primeira solicitação, mas não há mudança de estado para nenhum recurso do lado do servidor porque o recurso original já foi excluído.

Usar casos para DELETE incluem:

Remover uma postagem de blog

Excluir uma conta de usuário

Remover um arquivo ou imagem

Podemos destacar que uma operação de exclusão pode ser física ou lógica. A física é quando o recurso é realmente excluído da base de dados e a lógica quando uma coluna na tabela do recurso indica que ele foi excluído, mesmo ainda estando presente na tabela. A exclusão lógica é muito usada para manter dados históricos.

Demais métodos HTTP

O protocolo HTTP tem mais alguns métodos além dos citados. Para algumas APIs eles poderão vir a ser úteis, mas talvez para outras, nem tanto. Portanto, nossa abordagem será sobre o GET, POST, PUT, PATCH e DELETE. Caso queira conhecer outros métodos, como HEAD, OPTIONS, TRACE, etc, acesse - Métodos de requisição HTTP.

Status HTTP

Toda vez que uma solicitação HTTP é enviada a uma API REST, algum tipo de resposta será retornada de volta ao solicitante. Independentemente se a solicitação foi aceita, gerou erro, se o recurso não foi encontrado ou se teve sucesso na execução.

Essa resposta é chamada de HTTP Status Code e existem cinco categorias de Status Code no protocolo HTTP.

Informational responses (100 – 199)

Successful responses (200 – 299)

Redirects (300 – 399)

Client errors (400 – 499)

Server errors (500 – 599)

Alguns dos mais comuns códigos em respostas REST

200 - OK - Padrão de resposta para solicitações HTTP sucesso. A resposta real dependerá do método de solicitação usado. Em uma solicitação GET, a resposta conterá uma entidade que corresponde ao recurso solicitado. Em uma solicitação POST a resposta conterá a descrição de uma entidade, ou contendo o resultado da ação.

201 - Create - O pedido foi cumprido e resultou em um novo recurso que está sendo criado. Usado em solicitação POST e PUT quando criar um novo recurso.

204 - No Content - O servidor processou a solicitação com sucesso, mas não é necessária nenhuma resposta. Resposta comum na solicitação de um DELETE.

400 - Bad Request - O pedido não pôde ser entregue devido à sintaxe incorreta.

401 - Unauthorized - Basicamente usado quando a solicitação exige que o cliente esteja autenticado, entretanto, o cliente ainda não realizou a autenticação. A resposta deve incluir um cabeçalho do campo www-authenticate contendo um desafio aplicável ao recurso solicitado.

403 - Forbidden - O pedido é reconhecido pelo servidor, mas este recusa-se a executá-lo. Ao contrário da resposta “401”, aqui a autenticação não fará diferença e o pedido não deve ser requisitado novamente.

404 - Not Found - O recurso solicitado não foi encontrado, mas pode ser disponibilizado novamente no futuro. As solicitações subsequentes pelo cliente são permitidas.

405 - Method Not Allowed - Foi feita uma solicitação de um recurso usando um método de pedido que não é compatível com esse recurso, por exemplo, usando GET em um formulário, que exige que os dados sejam apresentados via POST, PUT; ou usados em um recurso somente de leitura.

409 - Conflict - Indica que a solicitação não pôde ser processada por causa do conflito no pedido, como um conflito de edição.

422 - Unprocessable Entity - O pedido foi bem formatado, mas foi incapaz de ser seguido devido a erros de semântica.

500 - Internal Server Error - Indica um erro do servidor ao processar a solicitação.

501 - Not Implemented - O servidor ainda não suporta a funcionalidade ativada.

503 - Service Unavailable - O servidor está em manutenção ou não consegue dar conta dos processamentos de recursos devido à sobrecarga do sistema. Isto deve ser uma condição temporária.

É muito importante que todas as respostas de solicitações possuam um Status Code definido para caso da solicitação ter sido processada ou ter gerado algum tipo de erro. O Status Code é a forma como a aplicação cliente vai saber tratar a resposta da requisição. Por exemplo, se a resposta for 200 o cliente poderá listar os dados recebidos, mas se for 403 poderá informar ao usuário que ele não tem permissão de acesso àquele recurso.

Para saber mais saber mais sobre status code do HTTP acesse - [HTTP response status codes](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

A arquitetura REST trabalha sobre o protocolo HTTP e cada URI funciona como um recurso da API. Portanto, devemos usar substantivos para determinar recursos em vez de verbos.

URI - Uniform Resource Identifier, ou Identificador Uniforme de Recursos

Exemplo: https://mballem.com/cursos/

A URI une o Protocolo (https://);

A localização do recurso (URL - mballem.com);

E o nome do recurso (URN - /cursos/)

correção no metodo post para patch???

Introdução a JWT

Stateful vs Stateless

As diferenças entre os termos “stateless” e “stateful” estão relacionadas à maneira como os sistemas ou protocolos lidam com o armazenamento e o gerenciamento de informações de estado.

Stateful (Com estado):

Em um sistema stateful, o servidor mantém informações de estado do cliente e usa essas informações para processar solicitações subsequentes. O estado do cliente é armazenado no servidor e é usado para acompanhar a interação contínua com o cliente ao longo do tempo. O servidor pode armazenar informações em uma sessão, como dados de autenticação, preferências do usuário, histórico de transações etc. Essas informações são usadas para personalizar a experiência do cliente e processar as solicitações de forma contextual. Por exemplo, um aplicativo de carrinho de compras online mantém um estado de sessão para cada cliente, rastreando os itens no carrinho, o histórico de compras etc.

Stateless (Sem estado):

Em um sistema stateless, não há armazenamento de informações de estado do cliente no servidor. Cada solicitação é tratada de forma independente e isolada, sem levar em consideração solicitações anteriores. O servidor não mantém informações sobre o estado do cliente entre as solicitações. Cada solicitação contém todas as informações necessárias para ser processada de forma completa. Por exemplo, o protocolo HTTP é considerado stateless, pois cada solicitação HTTP é tratada de forma independente, sem que o servidor mantenha informações de sessão do cliente.

Com base nestes dois tipos de estados podemos definir uma aplicação REST como sendo Stateless. Isso porque, uma aplicação REST é construída independentemente do cliente que se conecta a ela. Poderá até mesmo, ter vários clientes conectados como navegadores, aplicativos de smartphones, tablets, smart tvs, etc.

Um dos principais desafios no uso de stateless é o processo de autenticação. Como não existe um estado, ou seja, uma sessão para o servidor controlar o cliente que está conectado ao servidor, não é possível saber se duas requisições seguidas ou intercaladas pertencem ao mesmo cliente. Por isso, diferentes técnicas de autenticação foram criadas como as baseadas em token (Token-based Authentication), chave de API (API Key Authentication), certificado de autenticação (Certificate Authentication), autenticação por assinatura (Signature Authentication)e autenticação por tempo (Time-based Authentication).

Entre os métodos de autenticação o mais popular talvez seja a autenticação por token. Ele envolve a emissão de um token de autenticação após um processo bem-sucedido do pedido de autenticação pelo cliente. O token é então gerado pelo servidor e respondido ao cliente que deverá incluí-lo nas solicitações subsequentes como parte do cabeçalho “Authorization”. O servidor valida e verifica a assinatura do token para autenticar o cliente e realizar as operações solicitadas. Para esse tipo de operação o uso mais frequente é JSON Web Token (JWT).

JSON Web Token

O JSON Web Token (JWT) é um padrão aberto (RFC 7519) para representação compacta e segura de informações entre duas partes. Ele é frequentemente usado como um mecanismo de autenticação em aplicativos da web.

O JWT é composto por três partes:

Header: O cabeçalho especifica o algoritmo usado para assinar o token

Payload: O payload contém as informações que são codificadas no token, como dados do usuário, data de criação e expiração do token, permissões de acesso, etc.

Signature: A assinatura é gerada usando uma chave secreta, de escolha do desenvolvedor, com o objetivo de verificar se o token é válido. Essa é a única parte do token que não poderá ser visualizada pelo cliente, justamente por ser a parte criptografada.

Após a geração de um token ele terá o seguinte formato:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

Observando com atenção podemos ver entre a sequência de caracteres a separação das três partes pelo caractere ponto.

A primeira parte é o cabeçalho:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

A segunda parte é o payload:

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ

E a terceira parte é a assinatura:

SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

Uma das principais vantagens do JWT é ele ser autocontido, ou seja, todas as informações necessárias para verificar a autenticidade do token estão incluídas no próprio token. Isso significa que não é necessário armazenar informações de autenticação em um servidor, o que ajuda a simplificar a implementação de sistemas distribuídos.

Outra vantagem do JWT é a capacidade dele ser usado com qualquer linguagem de programação e plataforma, tornando-o uma opção popular para aplicativos da web e móveis.

No entanto, é importante notar que a segurança do JWT depende da implementação correta. É fundamental manter a chave secreta segura e proteger o token de ataques de força bruta ou de interceptação de rede. Alguns desenvolvedores consideram uma boa prática realizar a troca da chave de tempos em tempos para dificultar as possíveis tentativas de quebra da chave.

Além disso, os tokens devem ser configurados com tempo de vida adequado e expirados após o tempo limite definido para evitar o uso indevido. Mas é importante salientar que, não existe um tempo de vida do token considerado ideal. Esse tempo vai depender das necessidades específicas da aplicação e dos riscos de segurança associados a ela.

Mas lembre-se que, o token deve ter um tempo mínimo para o cliente autenticado possa realizar suas atividades sem a necessidade de a todo instante ser desligado e precisar autenticar novamente.

O cabeçalho de um Token

O cabeçalho de um JSON Web Token (JWT) é a primeira parte do token e contém informações sobre o tipo de token e o algoritmo de criptografia usado para gerar a assinatura do token. O cabeçalho é uma parte do token que é codificada em base64 e decodificada para revelar seus conteúdos.

O cabeçalho de um JWT contém dois campos obrigatórios:

“alg” (algorithm): indica qual algoritmo de criptografia é usado para gerar a assinatura do token. Os valores possíveis para esse campo incluem HMAC com SHA-256, RSA com SHA-256 e ECDSA com SHA-256, entre outros.

“typ” (type): indica o tipo de token, que é sempre JWT (JSON Web Token).

Além desses dois campos obrigatórios, o cabeçalho de um JWT pode incluir quaisquer outros campos personalizados que a aplicação precisa transmitir para a entidade que está usando o token. Esses campos personalizados são adicionados ao cabeçalho do token como pares chave/valor e são separados por vírgulas.

Por exemplo, um cabeçalho JWT pode ser representado em formato JSON da seguinte forma:

{

"alg": "HS256",

"typ": "JWT"

}

Nesse exemplo, o campo “alg” indica que a assinatura do token foi gerada usando o algoritmo HMAC com SHA-256 e o campo “typ” indica que é um token do tipo JWT.

Os ítens de um Payload

O payload de um token JWT (JSON Web Token) contém informações sobre o usuário ou objeto para o qual o token foi emitido e quaisquer metadados adicionais que são relevantes para a aplicação que está usando o token. O payload é uma parte do token que é codificada em base64 e decodificada para revelar seus conteúdos.

Entre algumas das informações de um payload as mais comuns são:

“iss” (issuer): o emissor do token

“sub” (subject): o assunto do token, geralmente o identificador exclusivo do usuário ou objeto (ex.: id ou username)

“aud” (audience): a audiência para a qual o token foi emitido

“exp” (expiration time): o tempo de expiração do token

“nbf” (not before): a hora a partir da qual o token é válido

“iat” (issued at): o momento em que o token foi emitido

“jti” (JWT ID): um identificador exclusivo para o token

Além desses itens, o payload também pode incluir quaisquer outros dados relevantes que a aplicação precisa transmitir para a entidade que está usando o token. Um item que poderá ser necessário em aplicações que possuem restrições por perfil é justamente o perfil do usuário autenticado. Isso servirá para o cliente conhecer e criar as regras necessárias para diferentes perfis no lado cliente.

Por exemplo, um payload JWT pode ser representado em formato JSON da seguinte forma:

{

"sub": "[email protected]",

"iat": 1682193714,

"exp": 1682194614,

"role": "ADMIN"

}

A Assinatura de um Token

A assinatura de um JSON Web Token (JWT) é uma etapa importante no processo de segurança do JWT. Ela é formada por uma sequência de caracteres anexada ao final do token para verificar sua integridade e autenticidade. A assinatura é baseada em um algoritmo de criptografia de chave compartilhada (symmetric key cryptography) ou chave pública/privada (asymmetric key cryptography).

O processo de assinatura funciona da seguinte maneira:

O cabeçalho e o payload do JWT são codificados em base64.

Os valores codificados do cabeçalho e do payload são combinados usando um ponto (.) como separador para criar uma sequência de caracteres.

Essa sequência de caracteres é criptografada usando um algoritmo de criptografia, juntamente com uma chave secreta ou chave pública/privada, dependendo do tipo de algoritmo de criptografia escolhido.

O resultado da criptografia é a assinatura do token JWT.

O token JWT final é criado concatenando o cabeçalho, o payload e a assinatura, separados por pontos.

Geração de Token

Existem várias bibliotecas de suporte para geração de tokens. É possível ver uma lista delas na página oficial jwt.io. A seguinte temos um breve exemplo da geração de um token usando a biblioteca jjwt:

Artefato Maven

<!-- JWT -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

Código fonte Java

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.security.Keys;

import io.jsonwebtoken.SignatureAlgorithm;

import java.security.Key;

import java.util.Date;

public class TokenGenerator {

public static void main(String[] args) {

// Define a chave secreta para assinatura do token

Key key = Keys

.secretKeyFor(SignatureAlgorithm.HS256);

// Define a data de expiração do token

// 1 dia em milissegundos (86400000)

Date expiration =

new Date(System.currentTimeMillis() + 86400000);

// Gera o token com os dados desejados

String token = Jwts.builder()

// Define o assunto do token

.setSubject("usuario")

// Define a data de expiração do token

.setExpiration(expiration)

// Assina o token com a chave secreta

.signWith(key)

.compact();

// Exibe o token gerado

System.out.println("Token gerado: " + token);

}

}

Em resumo, o JWT é um padrão aberto para representação compacta e segura de informações entre duas partes, amplamente utilizado como mecanismo de autenticação em aplicativos da web e móveis. No entanto, é importante implementar corretamente para garantir a segurança dos dados.

O Cliente e Autenticação

Após a autenticação do usuário, o servidor geralmente envia o JSON Web Token (JWT) para o cliente como uma resposta à solicitação de autenticação bem-sucedida. O token pode ser enviado em uma resposta HTTP ou em um cookie. Há também que responda com o token em um cabeçalho da resposta http.

A maneira mais comum de enviar o token para o cliente é incluí-lo no corpo da resposta HTTP como um objeto JSON. Por exemplo:

HTTP/1.1 200 OK

Content-Type: application/json

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c"

}

Nesse exemplo, o token JWT é incluído no campo “token” do objeto JSON retornado pelo servidor. O cliente pode armazenar o token em um cookie ou em outro local seguro para usá-lo em solicitações futuras.

Alternativamente, o servidor também pode enviar o token em um cookie HTTP seguro, definido com a opção “HttpOnly” e “Secure” para evitar ataques de roubo de cookies. Isso pode ser útil para aplicativos que usam autenticação baseada em sessão.

Independentemente de como o token é enviado ao cliente, é importante lembrar que o token JWT contém informações confidenciais do usuário e deve ser tratado com cuidado para evitar vazamento de informações ou adulteração. O cliente deve armazenar o token em um local seguro e protegê-lo contra ataques mal-intencionados.

O Cliente autenticado

Após a autenticação o cliente terá o token de acesso aos recursos protegidos da aplicação. Para enviar solicitações protegidas usando o JWT, o cliente deve incluir o token no cabeçalho Authorization da solicitação HTTP. O valor do cabeçalho de autorização é formado pela palavra “Bearer” seguida de um espaço e o valor do token. Por exemplo:

GET /api/users/123 HTTP/1.1

Host: api.example.com

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

Neste exemplo, o token JWT é incluído no cabeçalho “Authorization” da solicitação GET. O valor do cabeçalho é formado pela palavra “Bearer” seguida de um espaço e o valor do token.

É importante lembrar que o token JWT é sensível a letras maiúsculas e minúsculas e deve ser transmitido exatamente como foi recebido do servidor. O cliente deve armazenar o token em um local seguro e protegê-lo contra ataques mal-intencionados.

Além disso, é importante observar que o token JWT pode ter um tempo de expiração. Se o token expirar, o servidor não aceitará mais solicitações enviadas com esse token e o cliente deverá obter um novo token de autenticação para continuar a acessar recursos protegidos.

Um passo a passo na autenticação com JWT

Os passos de uma autenticação entre o cliente e o servidor usando JSON Web Tokens (JWT) geralmente segue:

1- O cliente envia as credenciais de autenticação (geralmente nome de usuário e senha) para o servidor por meio de uma solicitação de login.

2- O servidor verifica as credenciais do cliente em relação aos dados armazenados, como um banco de dados de usuários ou serviço de autenticação.

3- Se as credenciais forem válidas, o servidor gera um JWT contendo informações relevantes (payload) sobre o usuário e quaisquer permissões associadas.

4- O servidor assina o JWT usando uma chave secreta ou chave privada, dependendo do algoritmo de criptografia selecionado.

5- O servidor retorna o JWT para o cliente, geralmente como parte da resposta de autenticação bem-sucedida.

6- O cliente armazena o JWT de forma segura, como em um cookie seguro ou em armazenamento local (como Local Storage ou Session Storage) do navegador.

7- Em solicitações subsequentes a recursos protegidos, o cliente inclui o JWT no cabeçalho “Authorization” da solicitação HTTP, usando o formato “Bearer <token>”. Isso permite que o servidor verifique a autenticidade e autorização do cliente com base no JWT recebido.

8- O servidor valida a assinatura do JWT usando a chave secreta ou chave pública correspondente. Se a assinatura for válida, o servidor confia nas informações contidas no payload do JWT para autorizar ou negar o acesso ao recurso solicitado.

9- Se a autenticação for bem-sucedida e o cliente tiver permissão para acessar o recurso solicitado, o servidor processa a solicitação e retorna a resposta apropriada para o cliente.

Esse processo permite que o cliente autentique-se de forma segura e eficiente em recursos protegidos do servidor usando um JWT, sem a necessidade de armazenar informações de autenticação do usuário no servidor ou em sessões. O JWT contém todas as informações necessárias para autenticar e autorizar o cliente.

Referências

Introcução ao JSON Web Tokens

Documentação biblioteca io.jsonwebtoken

**Introdução ao uso do JasperSoft Studio**

JasperReports

A JasperReports é uma biblioteca Java, de código aberto, usada para geração de relatórios em Java. É considerada a biblioteca de relatórios mais utilizada no mundo.

Características

Fornece a opção de conexão com diferentes tipos de fontes de dados, entre elas, podemos citar algumas como:

* Bancos de dados relacionais (MySQL, PostGreSql, Oracle, …)
* MongoDB (NoSQL)
* Arquivos JSON
* Arquivos XML
* Arquivos CSV
* Objetos Java do tipo Map e Collection

Possibilita gerar relatórios em diversos formatos de arquivos como PDF, HTML, XLS, ODT, DOC, entre outros.

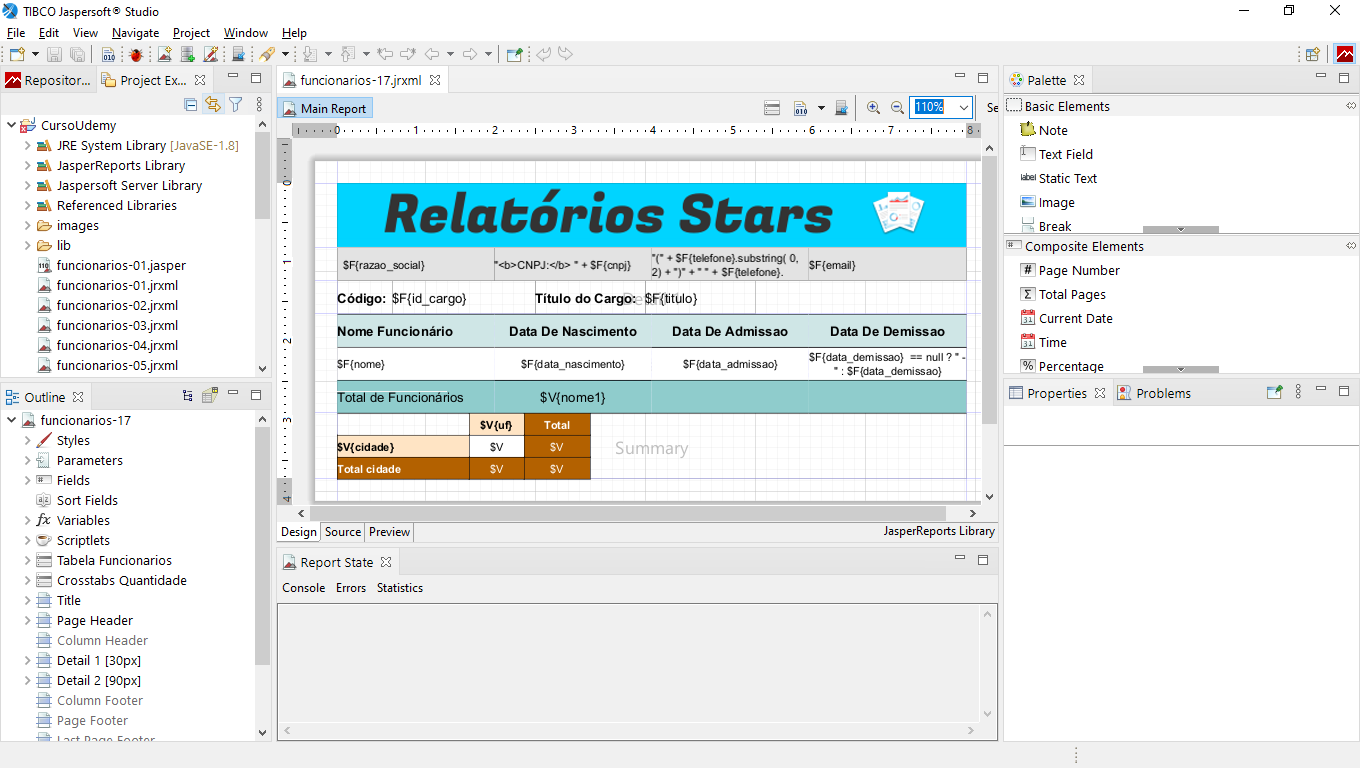
A biblioteca oferece diferentes tipos de elementos para exibição de dados nos relatórios. Com esses elementos podemos exibir textos estáticos e dinâmicos, gráficos (pizza, linha, barras, …), listas de dados, sub-relatórios, códigos de barras, qrCode, imagens, tabelas, crosstabs, entre outros.

JasperSoft Studio

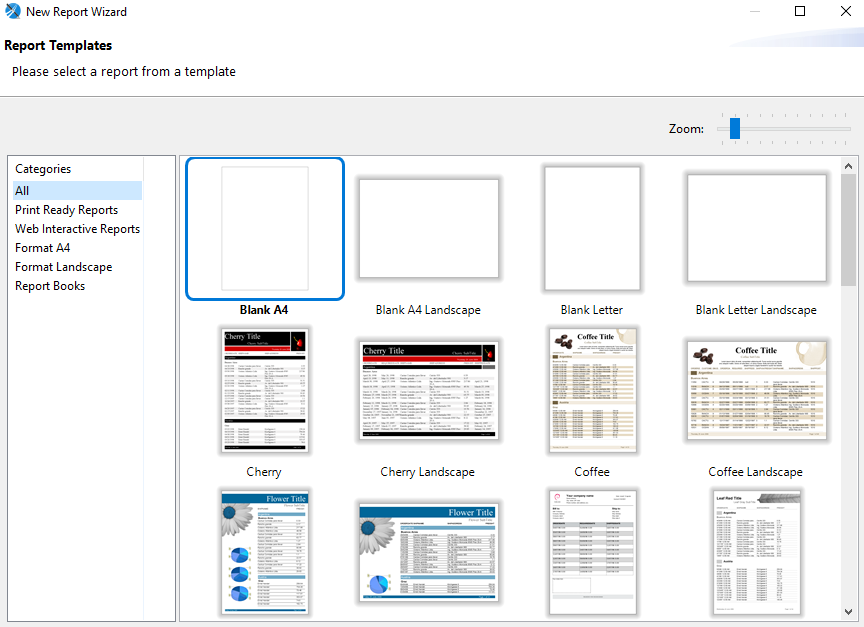
JasperSoft Studio é uma plataforma de desenvolvimento de relatórios do tipo JasperReports. Existem dois tipos de versões, a gratuita e a comercial. A gratuita é bastante completa e fornece uma grande quantidade de recursos para o desenvolvimento de relatórios. Já a versão comercial possui alguns componentes a mais na área de gráficos, mapas e outros recurso, além de suporte aos assinantes.  
No curso, vamos trabalhar com a versão gratuita e você poderá ver que é extremamente completa para o dia a dia.

Características

Uma característica bem visível no JasperSoft Studio é a aparencia a qual se assemelha muito com a IDE Eclipse. Isso para alguns é um ponto positivo, porque torna mais fácil o uso da plataforma para aqueles que já trabalharam com Eclipse.  
Existe também um plugin para ser instalado no Eclipse, assim, você pode ter em uma única plataforma a IDE Eclipse e a JasperSoft. Entretando, particularmente, não recomendo usar o plugin. Ele deixa a IDE extremamente pesada e pequenos bugs começam a aparecer com tempo.

Plataforma JasperSoft Studio  
A JasperSoft Studio é uma plataforma para se desenvolver visualmente os relatórios Jasper com a biblioteca JasperReports. Assim como no desenvolvimento Java Swing, você poderia desenvolver seus relatórios diretamente dentro de uma aplicação Java, usando as classes e métodos da biblioteca JasperReports, mas seria algo muito trabalhoso. Por conta disso, a plataforma se torna uma ferramenta essencial, já que todo trabalho é feito visualmente, com componentes selecionaveis e do tipo drag and drop.

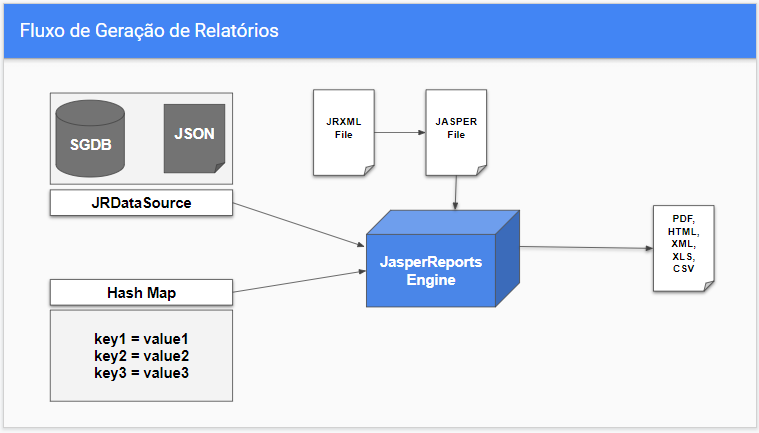
É possível selecionar diferentes tamanhos de páginas, como as mais comuns A4 e Letter e também personalizá-los. Além disso, existem modelos já estilazados que você pode usar em seus relatórios ou então se preferir, selecionar páginas vazias para começar do zero.

Janela de Seleção de Templates  
Ao selecionar um modelo de página você estará selecionando um modelo de template o qual vai gerar no seu projeto um arquivo do tipo .jrxml. Essa é a extensão dos templates JasperReports (JR) que utiliza o formato de código fonte baseado em XML. Quando geramos os relatórios, eles devem ser compilados e transformados assim, em arquivos binários do tipo .jasper.

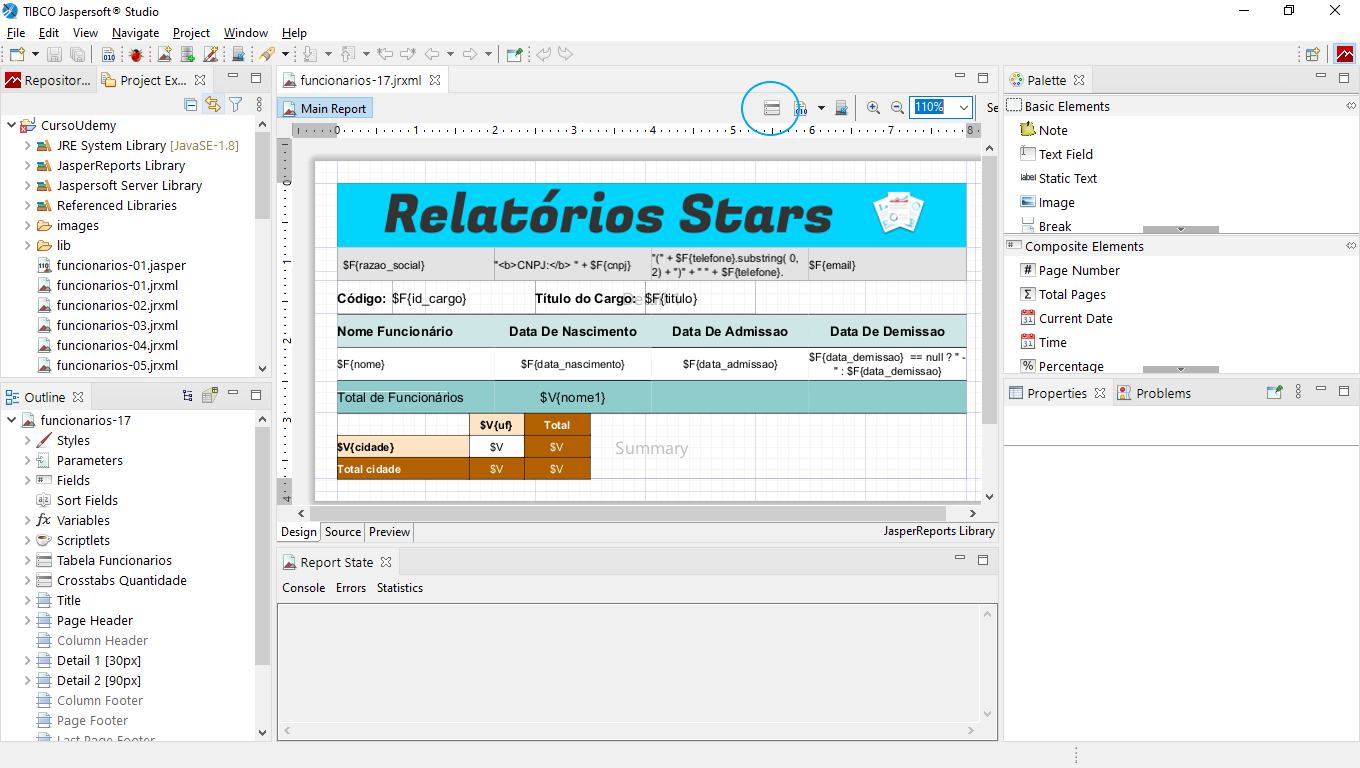
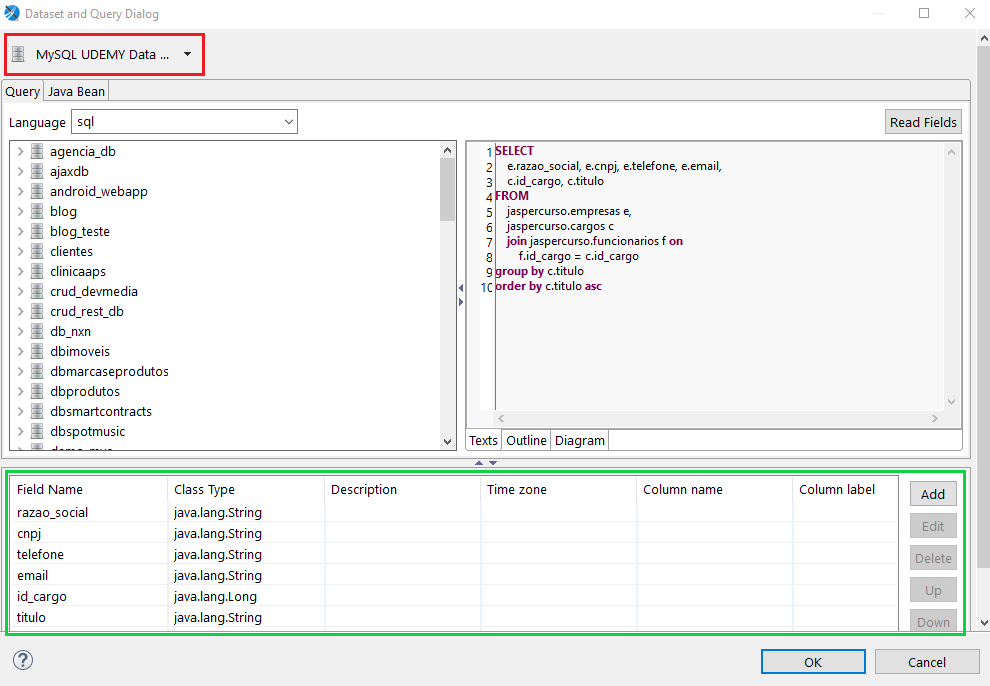
São os arquivos .jasper que serão executados, interpretados e vão gerar como saída um relatório em PDF, HTML ou outro formato qualquer.

Todo relatório Jasper precisa de pelo menos uma fonte de dados. Essa fonte de dados pode ser um banco de dados MySQL, MongoDB ou mesmo arquivos, como JSON ou XML e também coleções de objetos Java.

O mais comum é se trabalhar com conexão JDBC e assim, usar um banco de dados como fonte de dados via consultas SQL. As consultas são incluídas em um recurso da plataforma chamado Dataset e todo relatório vai precisar ter pelo menos um dataset. Confira a imagem a seguir e, observe o fluxo de geração de um relatório Jasper:

Geração de Relatórios  
Observando o fluxo veja que existe um objeto central chamado *JasperReports Engine*, esse é o responsável por gerar os relatórios e apresenta-los ao usuário como um PDF, HTML ou outro formato selecionado.  
Para que o J.R. Engine gere o relatório ele precisa de uma fonte de dados e de um template já compilado. Assim, ele vai passar a fonte de dados para o arquivo .jasper ser populado com os dados vindos de uma consulta e gerar as páginas necessárias, para então, exportá-lo no formato desejado pelo usuário.

Esse processo é executado tanto na plataforma JasperSoft Studio quanto em uma aplicação Java. Na plataforma, teremos acesso ao Dataset por meio do botão Dataset And Query Editor, posicionado no topo da janela central da plataforma, conforme mostra imagem abaixo:

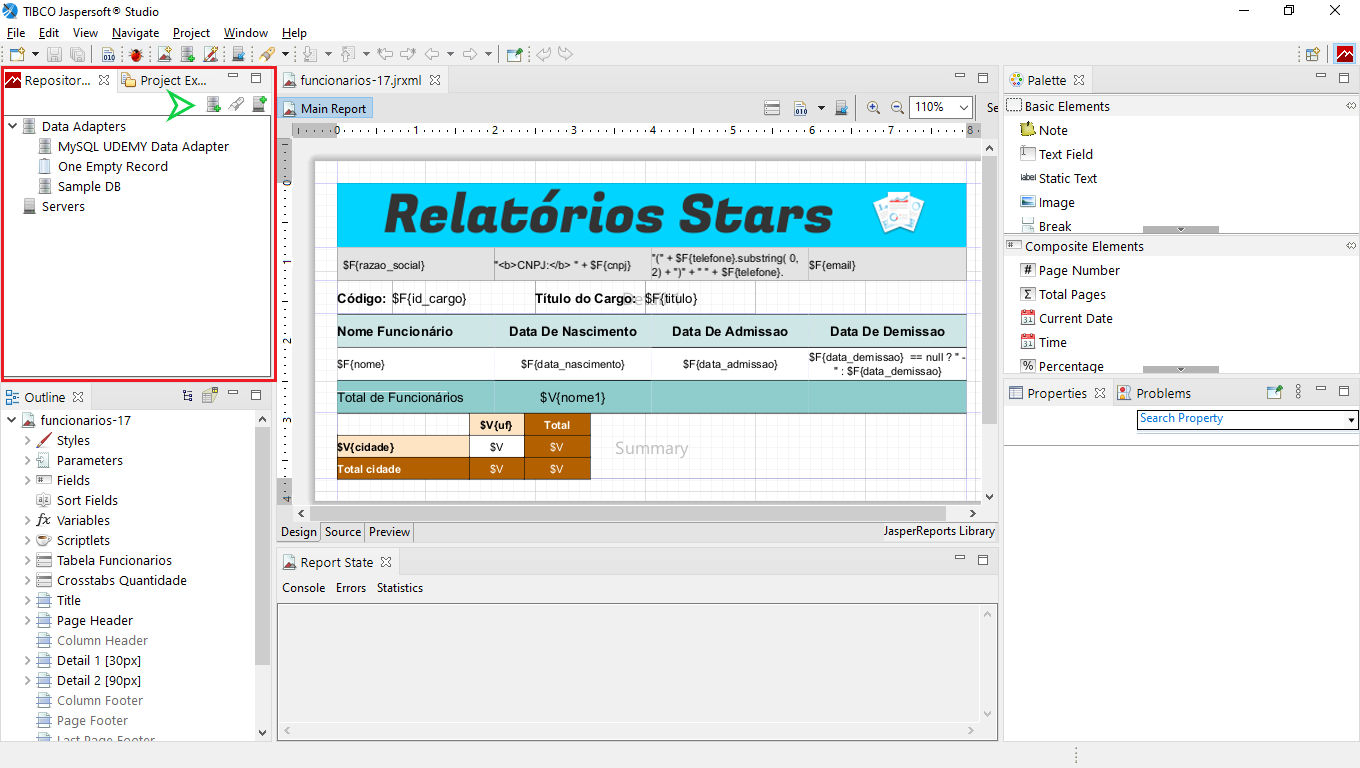
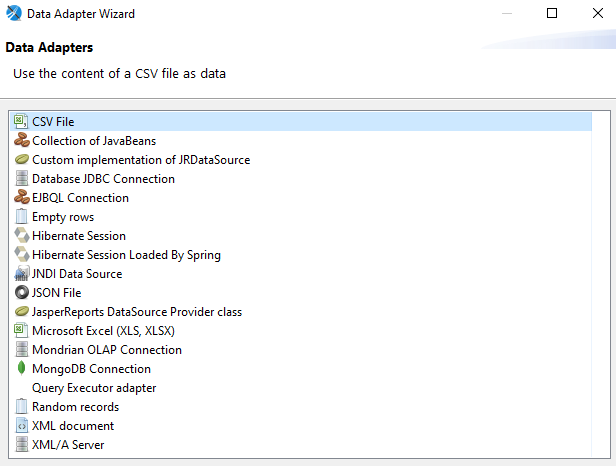
Acesso ao Dataset  
Ao clicar sobre o botão será aberta uma nova janela contendo um editor para o código SQL da consulta.  
Dataset e Query Editor  
Alguns outros ítens estão dispostos nessa janela, como o seletor do Data Adapter (marcado em vermelho) que representa a configuração de acesso a fonte de dados deste Dataset.

Ainda temos uma tabela na parte de baixo da janela. Nesta tabela, destacada em verde, teremos a coluna Field Name, com os nomes das colunas retornadas pela consulta SQL. E ao lado, a coluna Class Type, com o tipo de dado em Java que cada coluna retornada pela consulta vai ser representada dentro do relatório. Por exemplo, um varchar será java.lang.String, um BigInt será java.lang.Long, um Date será java.sql.Date, entre outras representações.

Como dito, a fonte de dados é o local de armazenamento de dados que o relatório vai consumir. Para configurar essa fonte de dado usamos o recurso nomeado como Data Adapter. Ele fica disposto na janela superior a esquerda do JasperSoft Studio, localizada na aba Repository Explorer.

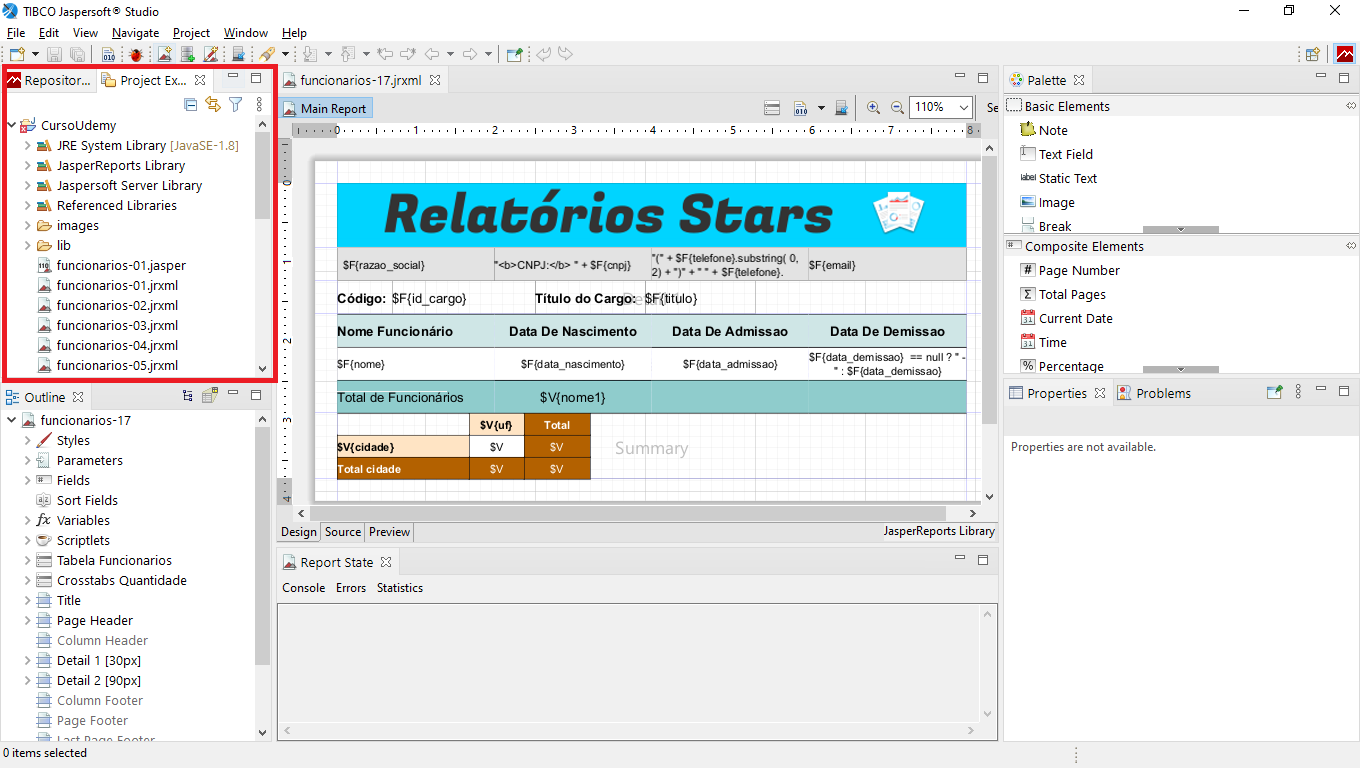
Janela Repository Explorer

Na imagem abaixo, destacado em vermelho, temos a aba Repository Explorer selecionada. Nela você terá uma lista com todos os Data Adapters cadastrados no seu projeto. E no botão indicado pela seta verde, é onde você clica para criar um novo Data Adapter.

Janela Repository Explorer  
Ao clicar no botão, uma janela será aberta com a lista dos tipos suportados pelo JasperReports para fonte de dados. Nessa lista há fontes como arquivo **CSV**, **JSON** e **XML**. Para bancos de dados temos os tipos **Database JDBC Connection**, **EJBQL**, **Hibernate Session** e **MongoDB Connection**. Além de alguns outros tipos de fontes de dados.  
Lista de Data Adpters  
Particularmente considero a conexão por JDBC a mais indicada no caso de uso de um banco de dados relacional. EJBQL e Hibernate Session necessitam das bibliotecas referentes ao Hibernate, se usará a implementação SessionFactory do framework e da JPA para uso de EJBQL. É possível também trabalhar com o Eclipse Link junto a EJBQL. O problema dessas conexões é a quantidade de erros que você pode vir a ter devido a conflitos de bibliotecas e suas versões. Também são indicadas apenas se seu projeto vai trabalhar com JPA ou Hibernate, para que na sua aplicação Java você pode fornecer essa fonte de dados. Mesmo assim, usando JPA ou Hibernate na aplicação, é possível fornecer uma conexão JDBC para um relatório desenvolvido com Database JDBC Connection.  
O SQL de uma consulta, quando adicionado no Dataset do relatório, vai fazer parte do código fonte do relatório, então, você não vai precisar se preocupar com ele na sua aplicação Java, vai apenas direcionar para o relatório um objeto de conexão do tipo java.sql.Connection.

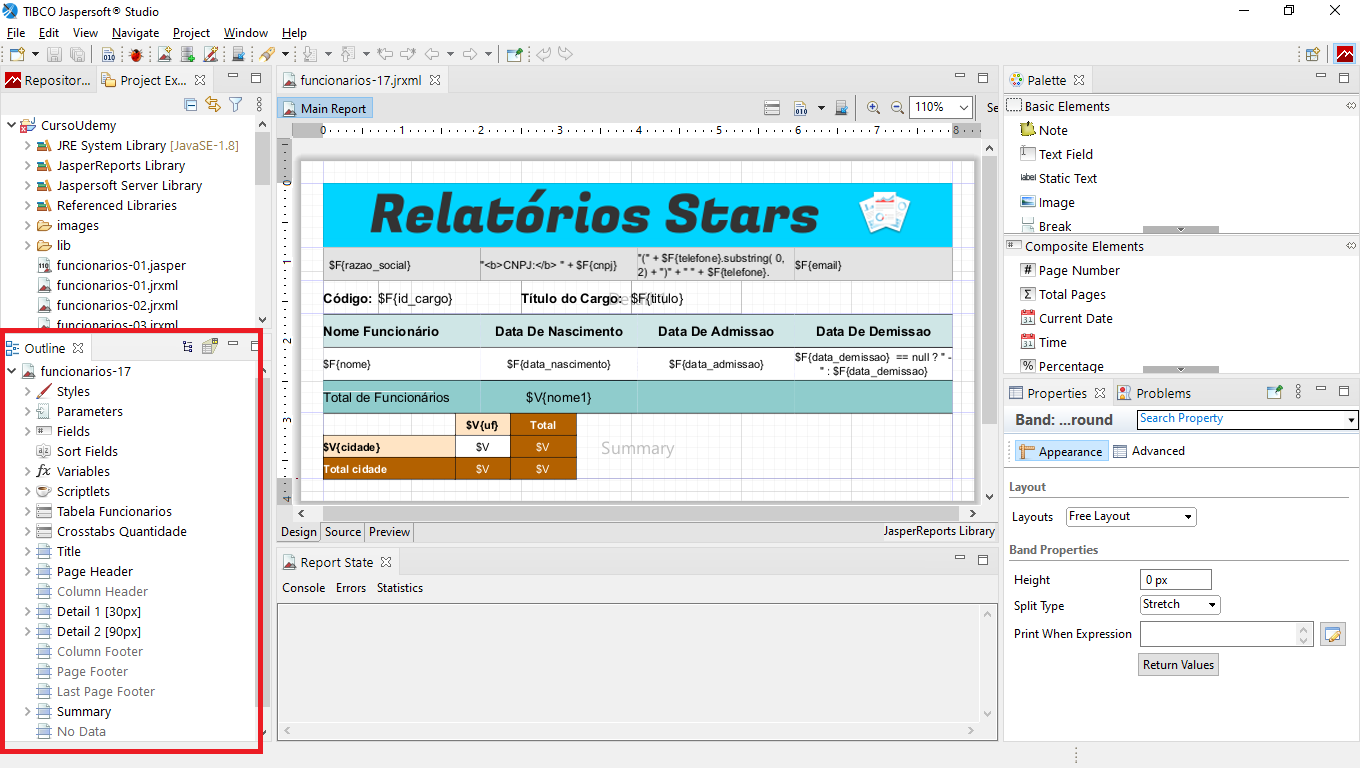
Janela Project Explorer

Outra aba importante é a Project Explorer, que fica ao lado de Repository Explorer. Essa aba vai listar para você cada um dos seus projetos adicionadas na plataforma JasperSoft Studio. Os projetos são dispostos em pastas, da mesma forma que na IDE Eclipse e, dentro de cada projeto teremos as bibliotecas do Jasper e também do Java (JRE).

Janela Project Explorer  
Além disso, você terá seus arquivos de templates, do tipo .jrxml, seus arquivos compilados do tipo .jasper, e outros arquivos externos como imagens e bibliotecas que estejam incluídas no projeto. Cada projeto poderá ter inumeros relatórios, então, crie um projeto para cada uma de suas aplicações para manter uma organização e assim, facilitar o seu dia a dia.

Janela Outline

Ainda no lado esquerdo da plataforma temos a janela Outline. Nela, ficam dispostos o acesso rápido aos componentes adicionados no template do relatório.

Janela Outline

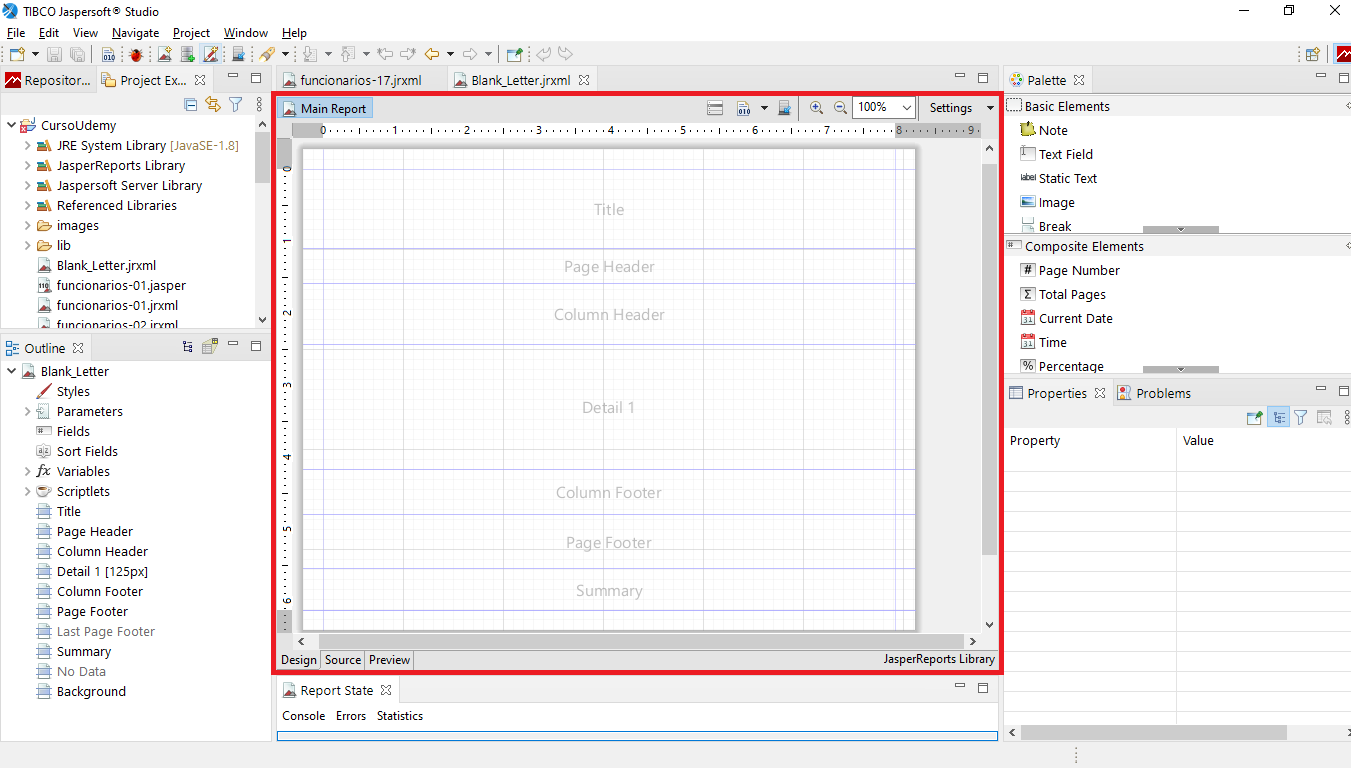
O primeiro ítem de Outline é o próprio template e por isso, todos demais estão organizados dentro dele. Vamos analisar alguns destes ítens:

* **Fields:**
  + Representação no relatório pela expressão: $F{nome}
  + Os fields ou campos são usados para exibir os dados resultantes de um Dataset, como exemplo, o retorno de uma query. Cada campo vai representar uma coluna retornada pela consulta e seu tipo será uma classe Java correspondente.
  + Os campos são preenchidos dinamicamente no momento da geração do relatório.
* **Variables:**
  + Representação no relatório pela expressão: $V{nome}
  + Armazena valores como totalizadores, médias, maior ou menor, …
* **Parameters:**
  + Representação no relatório pela expressão: $P{nome}
  + Parâmetros são usados para passar valores entre Dataset primários e secundários, enviar valores para preencher os critérios de uma consulta ou mesmo para passar alguma informação da aplicação para o relatório no momento de sua geração pelo Jasper Reports Engine.
* **Styles:**
  + Armazena formatações para campos de textos como cor a ser utilizada, tipo de fonte, alinhamento do texto, tamanho da fonte, entre outras opções. Outros componentes também pode trabalhar com estilos. Ao usar estilos, você apenas seta no componente o estilo desejado e assim, não precisa configurar uma a uma as configurações de layout daquele campo.

Em Outline, há também cada uma das bandas do template e cada elemento que for adicionado na banda poderá ser selecionado de forma rápida e simples por esse caminho. Verermos mais sobre as bandas logo a frente.

Janela Main Report / Design

Todo template Jasper é dividido em bandas e cada uma delas tem uma forma de ser apresentada ou apresentar os dados. As bandas estão “fisicamente” presentes na aba Design, da janela central, nomeada como Main Report.

Janela Main Report / Design  
É sobre as bandas que os elementos do relatório serão dispostos e quando você estiver criando seu templates deverá pensar bem em qual banda deseja apresentar tal informação. Vejamos algumas características de cada banda:

* **Title:** A primeira banda do relatório e tem como princípio ser exibida apenas na primeira página. Não importa se seu relatório tem duas ou mais páginas, o conteúdo adicionado nessa banda irá aparecer apenas na primeira página.
* **Page Header:** Esta banda é um tipo de cabeçalho da página e irá aparecer em todas as páginas do relatório. Adicione nela informações que queira que seja exibidas de forma repetida em todas as páginas, por exemplo, a data de geração do relatório, a página atual mais o total de páginas, o CNPJ da empresa ou qualquer outro informação. Ela também serve para exibir variaveis de média, totalizadores e outras funções matemáticas.
* **Column Header:** é o cabeçalho da banda Detail, que fica logo abaixo. Você pode adicionar o nome da cada coluna dos registros que serão apresentados na banda Detail, ou mesmo adicionar informações do tipo médias, totalizadores e outras operações matemáticas. Irá aparecer em todas as páginas do realatório, em que a banda Detail também esteja presente.
* **Detail:** essa banda é aquela que você vai usar para exibir a lista de resultados de um consulta ao banco de dados. Por exemplo, se a consulta retornar 15 linhas, a banda vai exibir as 15 linhas. Essa é a principal banda do relatório e muitos relatórios podem ser construídos apenas com o uso da Detail. É possível ter mais de uma banda Detail no mesmo relatório e esse recurso é útil para separar a exibição de dados e assim, organizar o relatório.
* **Column Footer:** segue os mesmo principios da Column Header, a diferença é que será o rodapé da banda Detail.
* **Page Footer:** segue os mesmo principios da Page Header, a diferença é que será o rodapé da página.
* **Summary:** é uma banda de resumo, irá aparecer sempre ao final do relatório, logo após a banda Detail. Mesmo que visualmente no template existam duas bandas entre a Detail e a Summary, quando o relatório for gerado ela salta para baixo da Detail. A função dessa banda é exibir principalmente totalizadores gerais, sejam eles de soma, média ou qualquer outra função matemática. Enquanto Page Footer e Column Footer exibem totalizadores por página e coluna.

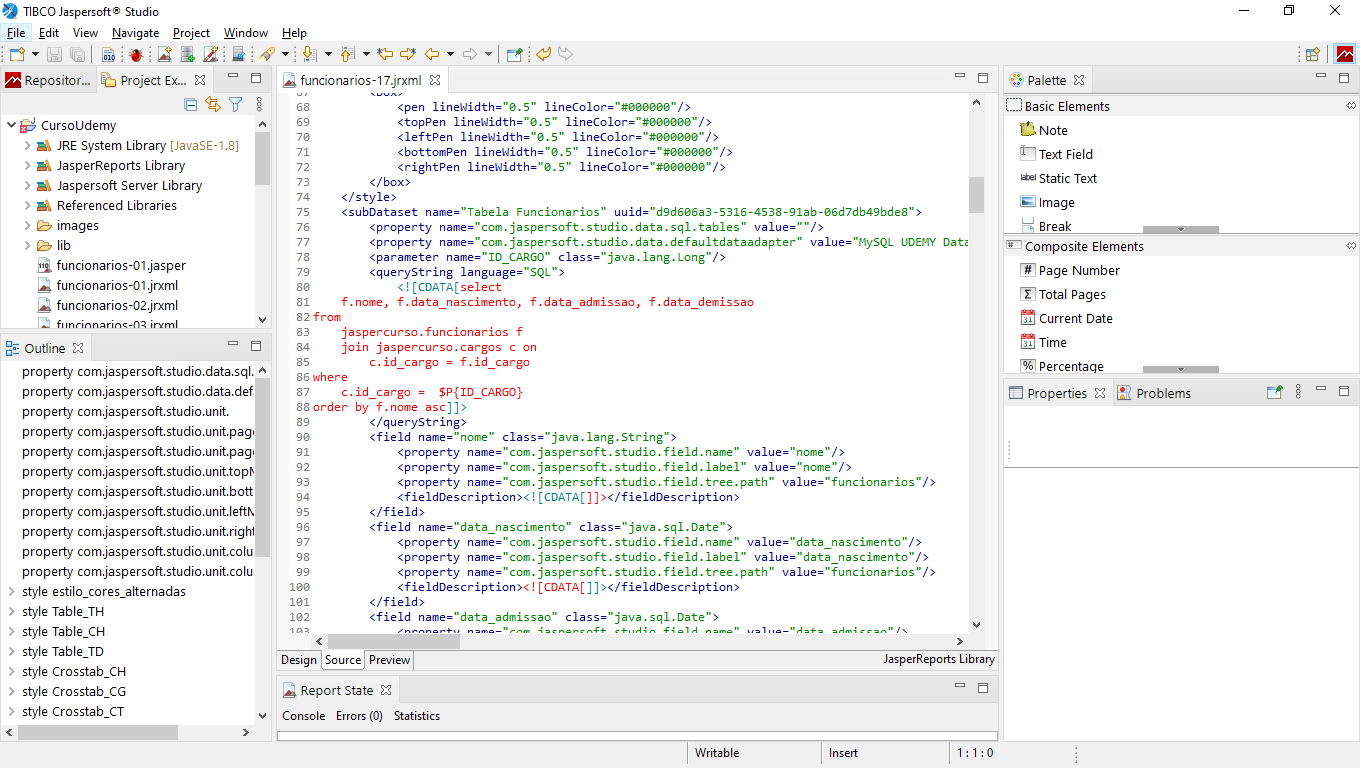
Embora não esteja no template padrão, as bandas **No Data** e **Last Page Footer** podem ser adicionadas. A primeira, tem como função exibir uma mensagem qualquer quando o relatório não retorna nenhum registro para ser exibido.

Já a Last Page Footer irá aparecer sempre na última página do relatório e substituirá nessa página a banda Page Footer. Assim, se você tiver alguma informação que precisa ser exibida na última página e é mais importante que as informações da Page Footer, é a hora de usá-la.

Todas as bandas são opcionais, você pode remover e incluir a qualquer momento qualquer uma das bandas apresentadas.

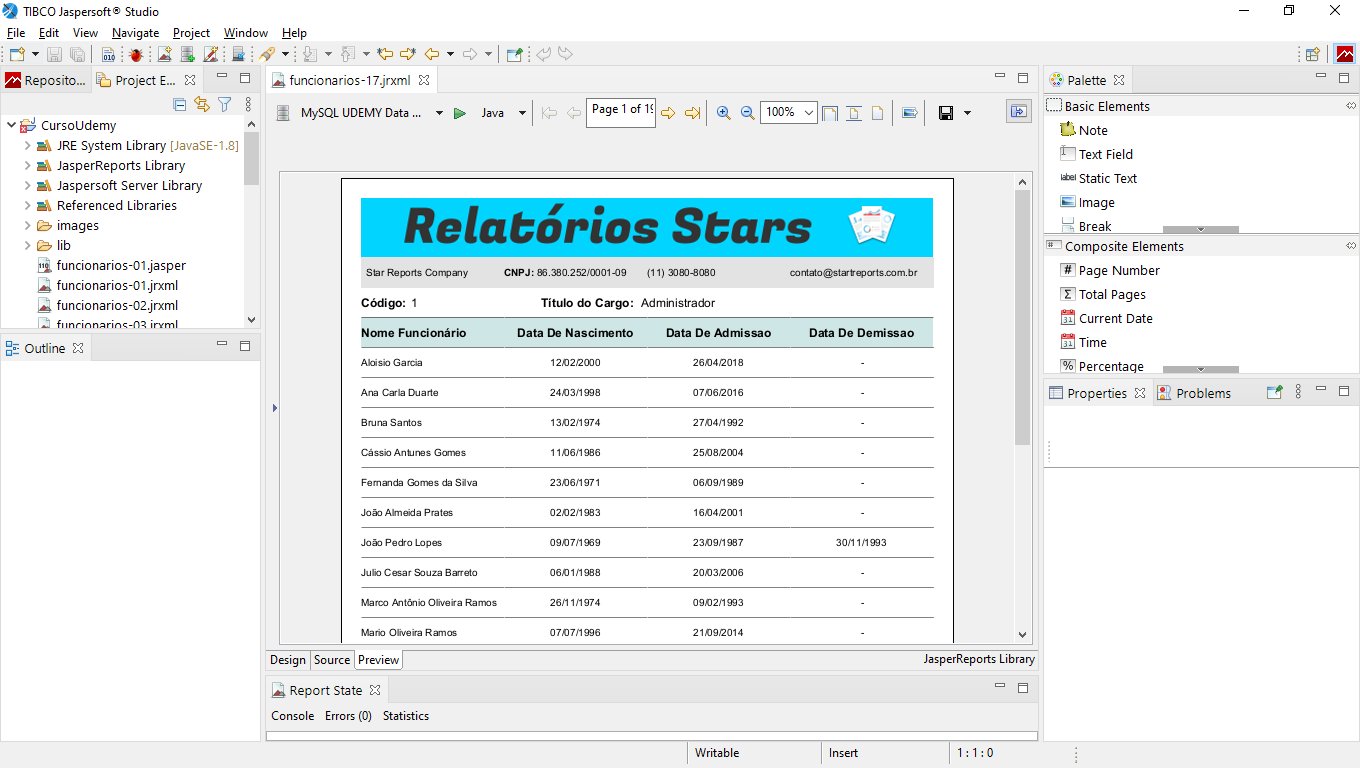
Janela Main Report / Source

Em Main Report temos também a aba Source. Ao clicar nessa aba você terá acesso ao código fonte do relatório, descrito em XML.

Janela Main Report / Source  
O código fonte não é bloqueado, então, você pode alterá-lo manualmente se desejar. Por isso, aconselho a ter cuidado, principalmente no inicio de sua tragetória com o JasperSoft Studio, porque se alterar algo de errado irá “bugar” seu relatório e talvez até impedir a compilação.

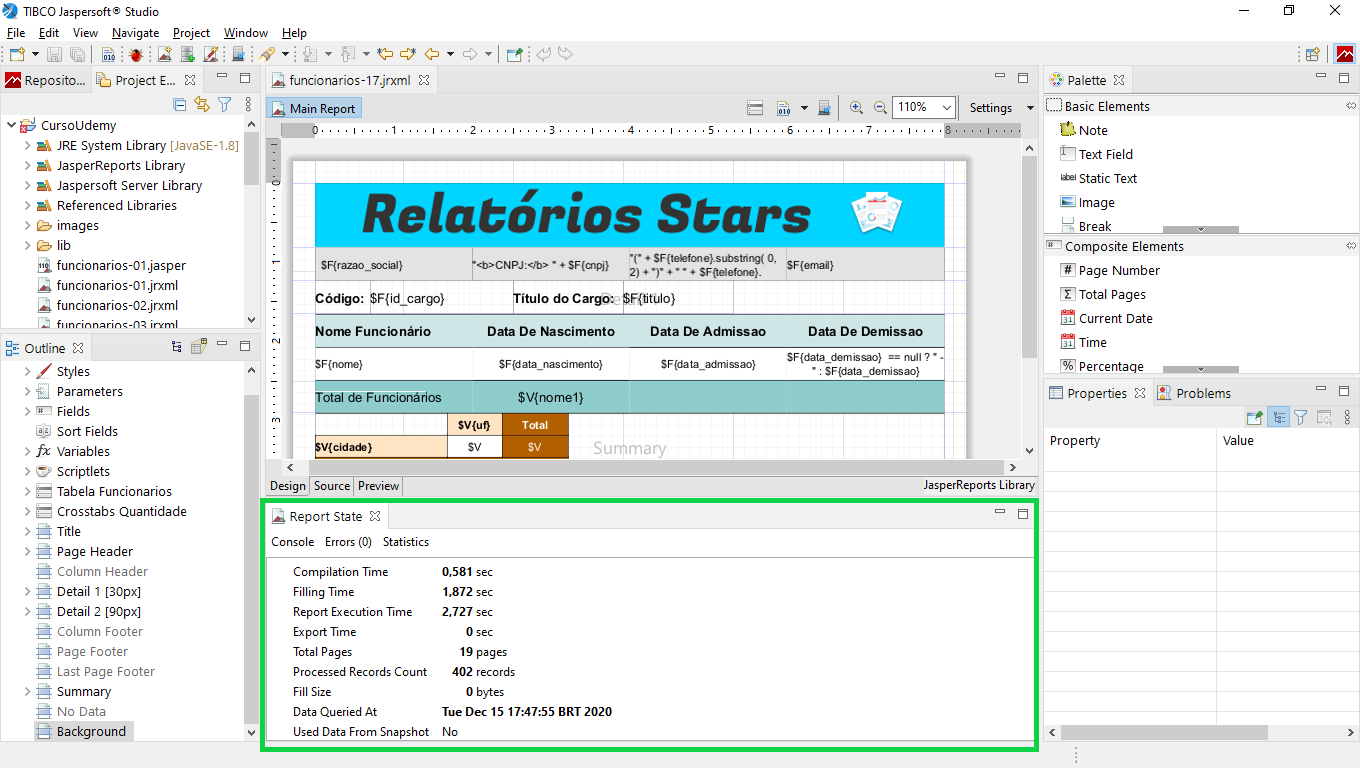
Todos os elementos como campos de textos, imagens, código de barras, gráficos, entre outros, adicionados ao template, são trasformados em XML e passam a fazer parte do código fonte. Até mesmo a consulta SQL fica registrada nele, como mostra a imagem acima entre as linhas 79 e 90.

A última aba em Main Report é a Preview. Ao clicar sobre ela você vai executar o processo de visualização do relatório e assim, poderá a qualquer momento testar o relatório enquanto o está construindo.

Janela Main Report / Preview  
Na janela que exibe o relatório você poderá paginá-lo, caso o relatório tenha mais de uma página, dar zoom, alterar a exibição da página e também exportar o realtório para qualquer formato disponível pelo JasperReports. Faça isso clicando sobre a seta ao lado do ícone de disquete.

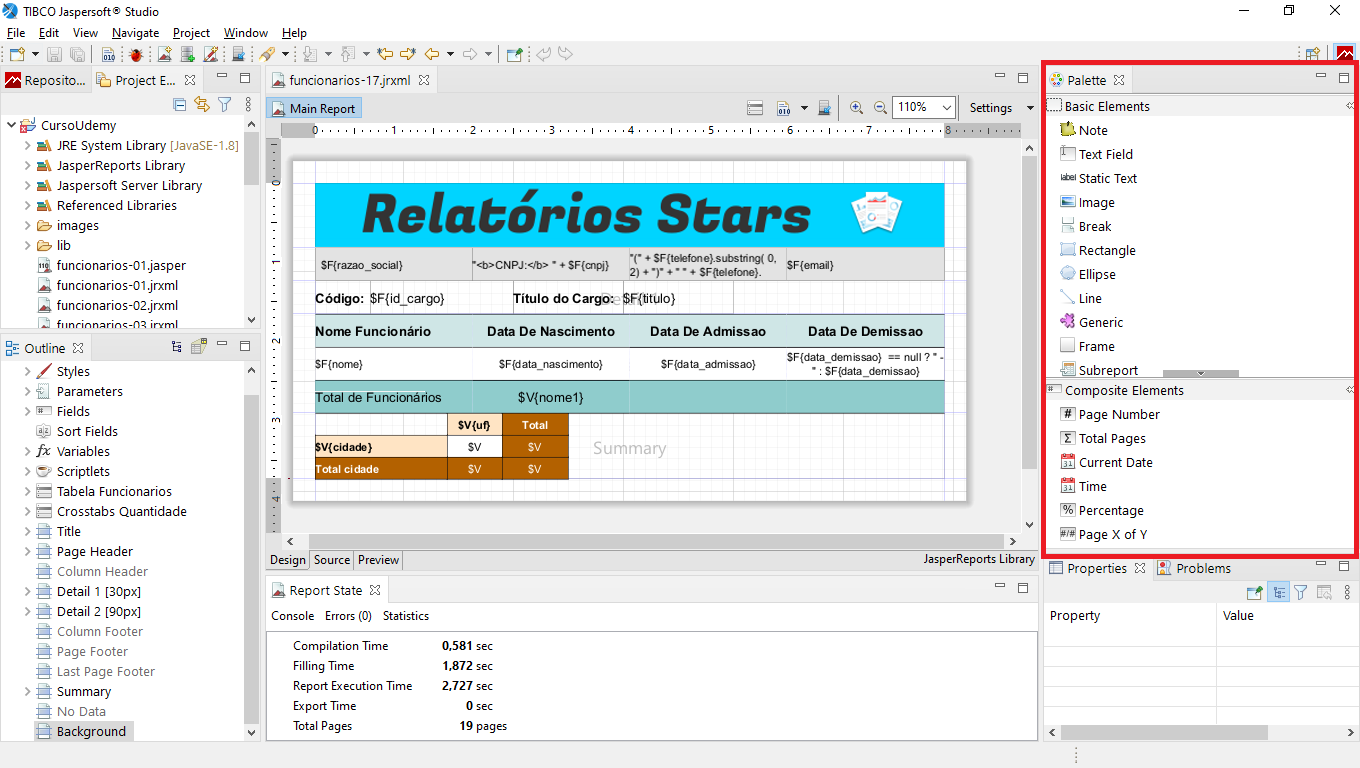
Janela Report State

Abaixo de Main Report temos a janela Report State. Esta é responsável por exibir mensagem de log de execução e compilação, erros e dados estatísticos do processo de preview.

Janela Report State

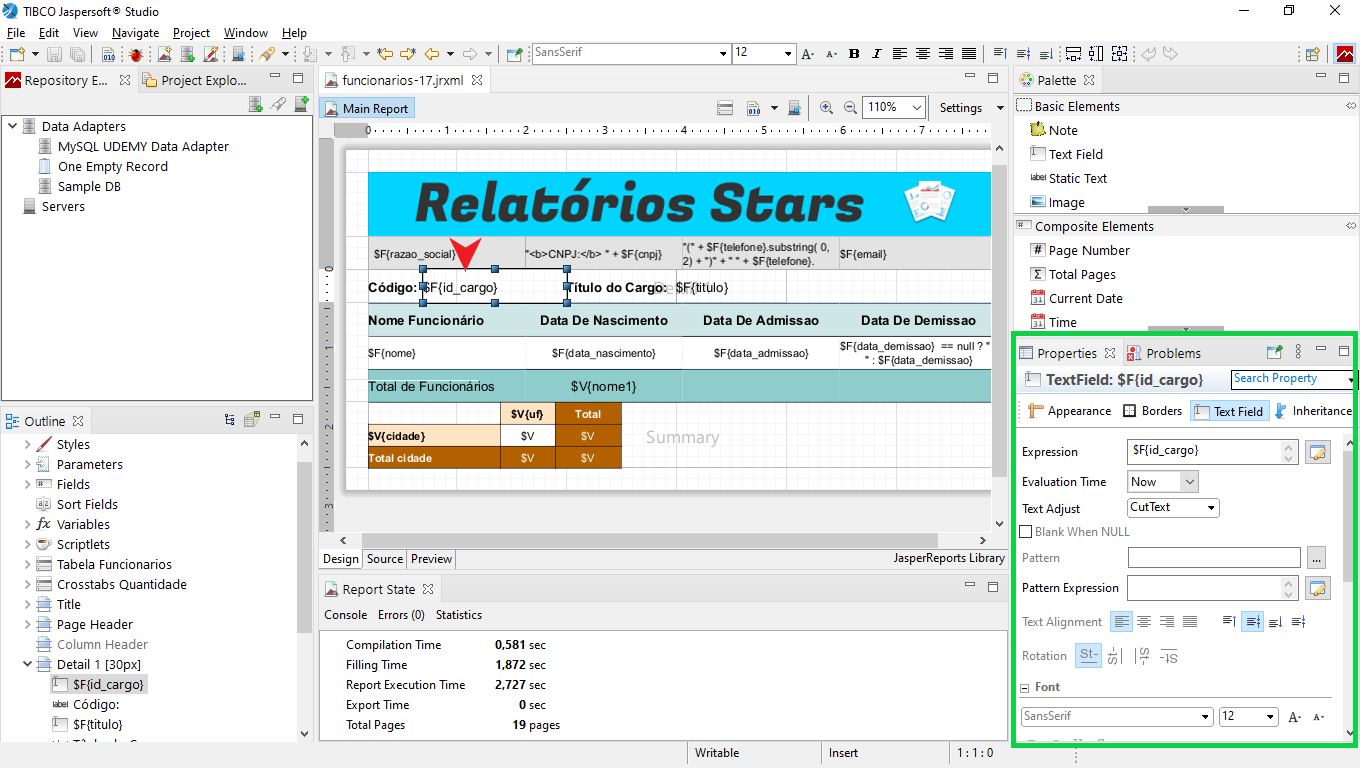
Janela Palette

Na janela Palette é onde encontramos a maioria dos componentes que são adicionados ao template do relatório. Esses componentes são divididos em duas categorias, **Basic Elements** e **Composite Elements**.  
Os elementos do tipo Basic são a grande maioria e servem para você adicionar no relatórios textos estáticos, dinâmicos, gráficos, imagens, frame, sub-relatório, entre outros. O sistema de inclusão é baseado em arrasta e solta, mais simples impossível.

Janela Palette  
Já os elementos do tipo Composite são funções pré definidas pela plataforma e com eles você pode adicionar a data e hora atual da geração do relatório, número da página, total de páginas, porcentagem e uma agregação para exibir a página atual mais o total de páginas.

Janela Properties

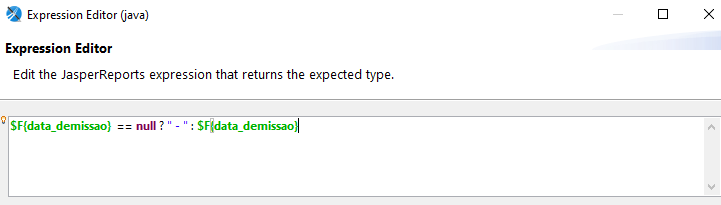
Em Properties é onde você terá acesso as propriedades de cada objeto, seja o próprio documento do relatório, bandas ou elementos dispostos no template.  
Basta selecionar o elemento com um clique do mouse que a janela vai exibir as propriedades deste elemento.

Janela Properties  
As propriedades são muito importantes durante o desenvolvimento, porque por meio delas, você poderá alterar ou reconfigurar o modo como o elemento será apresentado no relatório. Veja na imagem acima, que o campo $F{id\_cargo} está selecionado no template e as propriedades exibidas na janela Properties são desse campo.  
Observe que a aba selecionada é a Text Field e ainda há outras abas como Borders e Appearance.

Em Text Field você poderá configurar o alinhamento do texto, a rotação, fonte do texto, tamanho da fonte, negrito, itálico entre outras. Em Borders você poderá adicionar bordas em torno do campo e também espaço entre campo e bordas. Em Appearance, é onde você altera a cor da fonte e a cor de fundo do elemento.

Ainda existem outras opções de propriedades que veremos durante as aulas do curso, mas é importante ressaltar que, cada tipo de elemento poderá ter diferentes tipos de propriedades.

Em Text Field temos a propriedade Expression. É nela que o componente Text Field, de Basic Elements, recebe a expressão com o nome do campo retornado pela consulta. Ao lado direito dessa propriedade temos um botão que abre um editor de expressão e nesse editor você pode trabalhar com concatenação de campos ou strings, testes condicionais e fazer uso de classes do Java ou de bibliotecas externas para processar algo desejavel. Na imagem abaixo temos um exemplo de um teste condicional, onde é verificado se o campo $F{data\_demissão} é nulo, se for, será impresso no relatório o caractere -, caso contrário, será impresso o valor armazenado no campo $F{data\_demissão}.

Editor de Expressão

Saiba mais em:

* **Site oficial do JasperReports -** <https://community.jaspersoft.com/>
* **Jaspersoft Studio User Guide -** <https://community.jaspersoft.com/documentation?version=61056>

O guia de usuário do JasperSoft Studio (User Guide) deve ser usado por você como uma fonte importante de pesquisa.

Antes de perguntar “como eu faço isso”, pesquise no guia e tente encontrar a resposta, isso vai ajudar no seu aprendizado.

Para obter o Guia é necessário criar um cadastro no site do JasperReports, por conta disso, estou disponibilizando o PDF para download como um recurso dessa aula.

Introdução ao uso do JasperSoft Studio

JasperReports

A JasperReports é uma biblioteca Java, de código aberto, usada para geração de relatórios em Java. É considerada a biblioteca de relatórios mais utilizada no mundo.

Características

Fornece a opção de conexão com diferentes tipos de fontes de dados, entre elas, podemos citar algumas como:

Bancos de dados relacionais (MySQL, PostGreSql, Oracle, …)

MongoDB (NoSQL)

Arquivos JSON

Arquivos XML

Arquivos CSV

Objetos Java do tipo Map e Collection

Possibilita gerar relatórios em diversos formatos de arquivos como PDF, HTML, XLS, ODT, DOC, entre outros.

A biblioteca oferece diferentes tipos de elementos para exibição de dados nos relatórios. Com esses elementos podemos exibir textos estáticos e dinâmicos, gráficos (pizza, linha, barras, …), listas de dados, sub-relatórios, códigos de barras, qrCode, imagens, tabelas, crosstabs, entre outros.

JasperSoft Studio

JasperSoft Studio é uma plataforma de desenvolvimento de relatórios do tipo JasperReports. Existem dois tipos de versões, a gratuita e a comercial. A gratuita é bastante completa e fornece uma grande quantidade de recursos para o desenvolvimento de relatórios. Já a versão comercial possui alguns componentes a mais na área de gráficos, mapas e outros recurso, além de suporte aos assinantes.

No curso, vamos trabalhar com a versão gratuita e você poderá ver que é extremamente completa para o dia a dia.

Características

Uma característica bem visível no JasperSoft Studio é a aparencia a qual se assemelha muito com a IDE Eclipse. Isso para alguns é um ponto positivo, porque torna mais fácil o uso da plataforma para aqueles que já trabalharam com Eclipse.

Existe também um plugin para ser instalado no Eclipse, assim, você pode ter em uma única plataforma a IDE Eclipse e a JasperSoft. Entretando, particularmente, não recomendo usar o plugin. Ele deixa a IDE extremamente pesada e pequenos bugs começam a aparecer com tempo.

Plataforma JasperSoft StudioPlataforma JasperSoft Studio

A JasperSoft Studio é uma plataforma para se desenvolver visualmente os relatórios Jasper com a biblioteca JasperReports. Assim como no desenvolvimento Java Swing, você poderia desenvolver seus relatórios diretamente dentro de uma aplicação Java, usando as classes e métodos da biblioteca JasperReports, mas seria algo muito trabalhoso. Por conta disso, a plataforma se torna uma ferramenta essencial, já que todo trabalho é feito visualmente, com componentes selecionaveis e do tipo drag and drop.

É possível selecionar diferentes tamanhos de páginas, como as mais comuns A4 e Letter e também personalizá-los. Além disso, existem modelos já estilazados que você pode usar em seus relatórios ou então se preferir, selecionar páginas vazias para começar do zero.

Janela de Seleção de TemplatesJanela de Seleção de Templates

Ao selecionar um modelo de página você estará selecionando um modelo de template o qual vai gerar no seu projeto um arquivo do tipo .jrxml. Essa é a extensão dos templates JasperReports (JR) que utiliza o formato de código fonte baseado em XML. Quando geramos os relatórios, eles devem ser compilados e transformados assim, em arquivos binários do tipo .jasper.

São os arquivos .jasper que serão executados, interpretados e vão gerar como saída um relatório em PDF, HTML ou outro formato qualquer.

Todo relatório Jasper precisa de pelo menos uma fonte de dados. Essa fonte de dados pode ser um banco de dados MySQL, MongoDB ou mesmo arquivos, como JSON ou XML e também coleções de objetos Java.

O mais comum é se trabalhar com conexão JDBC e assim, usar um banco de dados como fonte de dados via consultas SQL. As consultas são incluídas em um recurso da plataforma chamado Dataset e todo relatório vai precisar ter pelo menos um dataset. Confira a imagem a seguir e, observe o fluxo de geração de um relatório Jasper:

Geração de RelatóriosGeração de Relatórios

Observando o fluxo veja que existe um objeto central chamado JasperReports Engine, esse é o responsável por gerar os relatórios e apresenta-los ao usuário como um PDF, HTML ou outro formato selecionado.

Para que o J.R. Engine gere o relatório ele precisa de uma fonte de dados e de um template já compilado. Assim, ele vai passar a fonte de dados para o arquivo .jasper ser populado com os dados vindos de uma consulta e gerar as páginas necessárias, para então, exportá-lo no formato desejado pelo usuário.

Esse processo é executado tanto na plataforma JasperSoft Studio quanto em uma aplicação Java. Na plataforma, teremos acesso ao Dataset por meio do botão Dataset And Query Editor, posicionado no topo da janela central da plataforma, conforme mostra imagem abaixo:

Acesso ao DatasetAcesso ao Dataset

Ao clicar sobre o botão será aberta uma nova janela contendo um editor para o código SQL da consulta.

Dataset e Query EditorDataset e Query Editor

Alguns outros ítens estão dispostos nessa janela, como o seletor do Data Adapter (marcado em vermelho) que representa a configuração de acesso a fonte de dados deste Dataset.

Ainda temos uma tabela na parte de baixo da janela. Nesta tabela, destacada em verde, teremos a coluna Field Name, com os nomes das colunas retornadas pela consulta SQL. E ao lado, a coluna Class Type, com o tipo de dado em Java que cada coluna retornada pela consulta vai ser representada dentro do relatório. Por exemplo, um varchar será java.lang.String, um BigInt será java.lang.Long, um Date será java.sql.Date, entre outras representações.

Como dito, a fonte de dados é o local de armazenamento de dados que o relatório vai consumir. Para configurar essa fonte de dado usamos o recurso nomeado como Data Adapter. Ele fica disposto na janela superior a esquerda do JasperSoft Studio, localizada na aba Repository Explorer.

Janela Repository Explorer

Na imagem abaixo, destacado em vermelho, temos a aba Repository Explorer selecionada. Nela você terá uma lista com todos os Data Adapters cadastrados no seu projeto. E no botão indicado pela seta verde, é onde você clica para criar um novo Data Adapter.

Janela Repository ExplorerJanela Repository Explorer

Ao clicar no botão, uma janela será aberta com a lista dos tipos suportados pelo JasperReports para fonte de dados. Nessa lista há fontes como arquivo CSV, JSON e XML. Para bancos de dados temos os tipos Database JDBC Connection, EJBQL, Hibernate Session e MongoDB Connection. Além de alguns outros tipos de fontes de dados.

Lista de Data AdptersLista de Data Adpters

Particularmente considero a conexão por JDBC a mais indicada no caso de uso de um banco de dados relacional. EJBQL e Hibernate Session necessitam das bibliotecas referentes ao Hibernate, se usará a implementação SessionFactory do framework e da JPA para uso de EJBQL. É possível também trabalhar com o Eclipse Link junto a EJBQL. O problema dessas conexões é a quantidade de erros que você pode vir a ter devido a conflitos de bibliotecas e suas versões. Também são indicadas apenas se seu projeto vai trabalhar com JPA ou Hibernate, para que na sua aplicação Java você pode fornecer essa fonte de dados. Mesmo assim, usando JPA ou Hibernate na aplicação, é possível fornecer uma conexão JDBC para um relatório desenvolvido com Database JDBC Connection.

O SQL de uma consulta, quando adicionado no Dataset do relatório, vai fazer parte do código fonte do relatório, então, você não vai precisar se preocupar com ele na sua aplicação Java, vai apenas direcionar para o relatório um objeto de conexão do tipo java.sql.Connection.

Janela Project Explorer

Outra aba importante é a Project Explorer, que fica ao lado de Repository Explorer. Essa aba vai listar para você cada um dos seus projetos adicionadas na plataforma JasperSoft Studio. Os projetos são dispostos em pastas, da mesma forma que na IDE Eclipse e, dentro de cada projeto teremos as bibliotecas do Jasper e também do Java (JRE).

Janela Project ExplorerJanela Project Explorer

Além disso, você terá seus arquivos de templates, do tipo .jrxml, seus arquivos compilados do tipo .jasper, e outros arquivos externos como imagens e bibliotecas que estejam incluídas no projeto. Cada projeto poderá ter inumeros relatórios, então, crie um projeto para cada uma de suas aplicações para manter uma organização e assim, facilitar o seu dia a dia.

Janela Outline

Ainda no lado esquerdo da plataforma temos a janela Outline. Nela, ficam dispostos o acesso rápido aos componentes adicionados no template do relatório.

Janela OutlineJanela Outline

O primeiro ítem de Outline é o próprio template e por isso, todos demais estão organizados dentro dele. Vamos analisar alguns destes ítens:

Fields:

Representação no relatório pela expressão: $F{nome}

Os fields ou campos são usados para exibir os dados resultantes de um Dataset, como exemplo, o retorno de uma query. Cada campo vai representar uma coluna retornada pela consulta e seu tipo será uma classe Java correspondente.

Os campos são preenchidos dinamicamente no momento da geração do relatório.

Variables:

Representação no relatório pela expressão: $V{nome}

Armazena valores como totalizadores, médias, maior ou menor, …

Parameters:

Representação no relatório pela expressão: $P{nome}

Parâmetros são usados para passar valores entre Dataset primários e secundários, enviar valores para preencher os critérios de uma consulta ou mesmo para passar alguma informação da aplicação para o relatório no momento de sua geração pelo Jasper Reports Engine.

Styles:

Armazena formatações para campos de textos como cor a ser utilizada, tipo de fonte, alinhamento do texto, tamanho da fonte, entre outras opções. Outros componentes também pode trabalhar com estilos. Ao usar estilos, você apenas seta no componente o estilo desejado e assim, não precisa configurar uma a uma as configurações de layout daquele campo.

Em Outline, há também cada uma das bandas do template e cada elemento que for adicionado na banda poderá ser selecionado de forma rápida e simples por esse caminho. Verermos mais sobre as bandas logo a frente.

Janela Main Report / Design

Todo template Jasper é dividido em bandas e cada uma delas tem uma forma de ser apresentada ou apresentar os dados. As bandas estão “fisicamente” presentes na aba Design, da janela central, nomeada como Main Report.

Janela Main Report / DesignJanela Main Report / Design

É sobre as bandas que os elementos do relatório serão dispostos e quando você estiver criando seu templates deverá pensar bem em qual banda deseja apresentar tal informação. Vejamos algumas características de cada banda:

Title: A primeira banda do relatório e tem como princípio ser exibida apenas na primeira página. Não importa se seu relatório tem duas ou mais páginas, o conteúdo adicionado nessa banda irá aparecer apenas na primeira página.

Page Header: Esta banda é um tipo de cabeçalho da página e irá aparecer em todas as páginas do relatório. Adicione nela informações que queira que seja exibidas de forma repetida em todas as páginas, por exemplo, a data de geração do relatório, a página atual mais o total de páginas, o CNPJ da empresa ou qualquer outro informação. Ela também serve para exibir variaveis de média, totalizadores e outras funções matemáticas.

Column Header: é o cabeçalho da banda Detail, que fica logo abaixo. Você pode adicionar o nome da cada coluna dos registros que serão apresentados na banda Detail, ou mesmo adicionar informações do tipo médias, totalizadores e outras operações matemáticas. Irá aparecer em todas as páginas do realatório, em que a banda Detail também esteja presente.

Detail: essa banda é aquela que você vai usar para exibir a lista de resultados de um consulta ao banco de dados. Por exemplo, se a consulta retornar 15 linhas, a banda vai exibir as 15 linhas. Essa é a principal banda do relatório e muitos relatórios podem ser construídos apenas com o uso da Detail. É possível ter mais de uma banda Detail no mesmo relatório e esse recurso é útil para separar a exibição de dados e assim, organizar o relatório.

Column Footer: segue os mesmo principios da Column Header, a diferença é que será o rodapé da banda Detail.

Page Footer: segue os mesmo principios da Page Header, a diferença é que será o rodapé da página.

Summary: é uma banda de resumo, irá aparecer sempre ao final do relatório, logo após a banda Detail. Mesmo que visualmente no template existam duas bandas entre a Detail e a Summary, quando o relatório for gerado ela salta para baixo da Detail. A função dessa banda é exibir principalmente totalizadores gerais, sejam eles de soma, média ou qualquer outra função matemática. Enquanto Page Footer e Column Footer exibem totalizadores por página e coluna.

Embora não esteja no template padrão, as bandas No Data e Last Page Footer podem ser adicionadas. A primeira, tem como função exibir uma mensagem qualquer quando o relatório não retorna nenhum registro para ser exibido.

Já a Last Page Footer irá aparecer sempre na última página do relatório e substituirá nessa página a banda Page Footer. Assim, se você tiver alguma informação que precisa ser exibida na última página e é mais importante que as informações da Page Footer, é a hora de usá-la.

Todas as bandas são opcionais, você pode remover e incluir a qualquer momento qualquer uma das bandas apresentadas.

Janela Main Report / Source

Em Main Report temos também a aba Source. Ao clicar nessa aba você terá acesso ao código fonte do relatório, descrito em XML.

Janela Main Report / SourceJanela Main Report / Source

O código fonte não é bloqueado, então, você pode alterá-lo manualmente se desejar. Por isso, aconselho a ter cuidado, principalmente no inicio de sua tragetória com o JasperSoft Studio, porque se alterar algo de errado irá “bugar” seu relatório e talvez até impedir a compilação.

Todos os elementos como campos de textos, imagens, código de barras, gráficos, entre outros, adicionados ao template, são trasformados em XML e passam a fazer parte do código fonte. Até mesmo a consulta SQL fica registrada nele, como mostra a imagem acima entre as linhas 79 e 90.

A última aba em Main Report é a Preview. Ao clicar sobre ela você vai executar o processo de visualização do relatório e assim, poderá a qualquer momento testar o relatório enquanto o está construindo.

Janela Main Report / PreviewJanela Main Report / Preview

Na janela que exibe o relatório você poderá paginá-lo, caso o relatório tenha mais de uma página, dar zoom, alterar a exibição da página e também exportar o realtório para qualquer formato disponível pelo JasperReports. Faça isso clicando sobre a seta ao lado do ícone de disquete.

Janela Report State

Abaixo de Main Report temos a janela Report State. Esta é responsável por exibir mensagem de log de execução e compilação, erros e dados estatísticos do processo de preview.

Janela Report StateJanela Report State

Janela Palette

Na janela Palette é onde encontramos a maioria dos componentes que são adicionados ao template do relatório. Esses componentes são divididos em duas categorias, Basic Elements e Composite Elements.

Os elementos do tipo Basic são a grande maioria e servem para você adicionar no relatórios textos estáticos, dinâmicos, gráficos, imagens, frame, sub-relatório, entre outros. O sistema de inclusão é baseado em arrasta e solta, mais simples impossível.

Janela PaletteJanela Palette

Já os elementos do tipo Composite são funções pré definidas pela plataforma e com eles você pode adicionar a data e hora atual da geração do relatório, número da página, total de páginas, porcentagem e uma agregação para exibir a página atual mais o total de páginas.

Janela Properties

Em Properties é onde você terá acesso as propriedades de cada objeto, seja o próprio documento do relatório, bandas ou elementos dispostos no template.

Basta selecionar o elemento com um clique do mouse que a janela vai exibir as propriedades deste elemento.

Janela PropertiesJanela Properties

As propriedades são muito importantes durante o desenvolvimento, porque por meio delas, você poderá alterar ou reconfigurar o modo como o elemento será apresentado no relatório. Veja na imagem acima, que o campo $F{id\_cargo} está selecionado no template e as propriedades exibidas na janela Properties são desse campo.

Observe que a aba selecionada é a Text Field e ainda há outras abas como Borders e Appearance.

Em Text Field você poderá configurar o alinhamento do texto, a rotação, fonte do texto, tamanho da fonte, negrito, itálico entre outras. Em Borders você poderá adicionar bordas em torno do campo e também espaço entre campo e bordas. Em Appearance, é onde você altera a cor da fonte e a cor de fundo do elemento.

Ainda existem outras opções de propriedades que veremos durante as aulas do curso, mas é importante ressaltar que, cada tipo de elemento poderá ter diferentes tipos de propriedades.

Em Text Field temos a propriedade Expression. É nela que o componente Text Field, de Basic Elements, recebe a expressão com o nome do campo retornado pela consulta. Ao lado direito dessa propriedade temos um botão que abre um editor de expressão e nesse editor você pode trabalhar com concatenação de campos ou strings, testes condicionais e fazer uso de classes do Java ou de bibliotecas externas para processar algo desejavel. Na imagem abaixo temos um exemplo de um teste condicional, onde é verificado se o campo $F{data\_demissão} é nulo, se for, será impresso no relatório o caractere -, caso contrário, será impresso o valor armazenado no campo $F{data\_demissão}.

Editor de ExpressãoEditor de Expressão

Saiba mais em:

Site oficial do JasperReports - https://community.jaspersoft.com/

Jaspersoft Studio User Guide - https://community.jaspersoft.com/documentation?version=61056

O guia de usuário do JasperSoft Studio (User Guide) deve ser usado por você como uma fonte importante de pesquisa.

Antes de perguntar “como eu faço isso”, pesquise no guia e tente encontrar a resposta, isso vai ajudar no seu aprendizado.

Para obter o Guia é necessário criar um cadastro no site do JasperReports, por conta disso, estou disponibilizando o PDF para download como um recurso dessa aula.

CORS

Introdução

CORS, ou Cross-Origin Resource Sharing (Compartilhamento de Recursos entre Origens), é uma política de segurança implementada pelos navegadores da web para controlar as solicitações HTTP feitas por scripts em uma página web a um servidor em um domínio diferente do domínio da página web.

Em termos mais simples, quando um navegador carrega uma página de um determinado domínio (origem) e essa página tenta fazer uma requisição para um recurso (como uma API) em um domínio diferente, o navegador verifica se o servidor que hospeda o recurso permite que a origem da página acesse esse recurso. Isso é feito através de cabeçalhos HTTP.

A política CORS ajuda a proteger os usuários da web contra ataques de scripts maliciosos (como ataques de injeção de código) que tentam explorar a capacidade de um navegador de fazer solicitações para outros servidores em nome do usuário.

A CORS é configurada no lado do servidor e pode ser usada para especificar quem (quais origens) tem permissão para acessar os recursos da API. Existem várias maneiras de configurar a política CORS no servidor, como:

Permitir todas as origens: Isso permite que qualquer origem acesse a API. No entanto, isso pode ser menos seguro, pois qualquer página da web pode fazer solicitações à sua API.

Permitir origens específicas: Você pode configurar o servidor para permitir apenas determinadas origens (ou domínios) de fazer solicitações à sua API.

Configurar cabeçalhos específicos: Além de permitir ou negar origens, você pode especificar quais tipos de cabeçalhos HTTP podem ser usados durante a solicitação.

Em resumo, o CORS é uma parte importante da segurança web que ajuda a proteger os usuários contra acessos não autorizados a recursos em servidores diferentes. Ele é especialmente relevante quando se trabalha com APIs e serviços web em aplicações web modernas.

Liberando CORS em um API Spring

Para criar uma classe de configuração de CORS no Spring Boot para liberar o acesso da API a qualquer cliente, você pode seguir os seguintes passos:

Crie uma classe de configuração que estenda a classe WebMvcConfigurer.

Sobrescreva o método addCorsMappings().

Na implementação do método addCorsMappings(), adicione uma nova configuração de CORS.

Na configuração de CORS, especifique as seguintes propriedades:

pathPatterns: Os padrões de URIs que serão permitidos.

allowedOrigins: Os domínios ou endereços IP que serão permitidos.

allowedHeaders: Os cabeçalhos HTTP que serão permitidos.

allowedMethods: Os métodos HTTP que serão permitidos.

Aqui está um exemplo de classe de configuração de CORS que libera o acesso da API a qualquer cliente:

@Configuration

@EnableWebMvc

public class SpringCorsConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowedHeaders("\*")

.allowedMethods("\*");

}

}

Neste exemplo, a configuração de CORS permite acesso a todas as URIs da API, de qualquer domínio ou endereço IP. Também permite todos os cabeçalhos HTTP e todos os métodos HTTP.

Para liberar o acesso da API a um domínio ou endereço IP específico, você pode especificar a propriedade allowedOrigins com o valor do domínio ou endereço IP. Por exemplo, para liberar o acesso da API ao domínio example.com, você pode usar o seguinte código:

registry.addMapping("/\*\*") .allowedOrigins("https://example.com");

Para liberar o acesso da API a um conjunto de domínios ou endereços IP, você pode especificar a propriedade allowedOrigins com uma lista de valores. Por exemplo, para liberar o acesso da API aos domínios example.com e another-example.com, você pode usar o seguinte código:

registry.addMapping("/\*\*").allowedOrigins(

"https://example.com", "https://another-example.com"

);

Para restringir o acesso da API a determinados cabeçalhos HTTP, você pode especificar a propriedade allowedHeaders com uma lista de valores. Por exemplo, para permitir apenas os cabeçalhos Authorization e Content-Type, você pode usar o seguinte código:

registry.addMapping("/\*\*")

.allowedHeaders("Authorization", "Content-Type");

Para restringir o acesso da API a determinados métodos HTTP, você pode especificar a propriedade allowedMethods com uma lista de valores. Por exemplo, para permitir apenas os métodos HTTP GET, POST e PUT, você pode usar o seguinte código:

registry.addMapping("/\*\*").allowedMethods("GET", "POST", "PUT");

Desta forma, aplicações desenvolvidas em Angular, ReactJS, Android ou qualquer outro cliente hospedado em um servidor de origem diferente da sua API terá liberdade para consumi-la.

Liberando o CORS no Spring Security

Caso esteja trabalhando com o Spring Security em sua API REST, será necessário configurar o CORS na classe de configuração do Spring Security. A configuração não passa de uma linha, como se pode ver no exemplo abaixo:

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

return http.cors(Customizer.withDefaults())

... // Demais etapas da configuração

}

Com isso, sua API estará com o CORS configurado.

18/09/2024

Seção 14 Recursos de clientes

Contexto do spring security contem informações relacionadas ao usuario autenticado;

Teste de recursos RESTful

Injeção de dependência do WebTestClient testCliente;

Pageable / Page, permite uma paginação de recursos muito extensos, de forma a não mostrar tudo de uma vez. Por exemplo em um recurso para listar todos os usuarios cadastrados, não é uma boa prática retornar todos eles direto por um aceso de URL. O Spring gerencia o conteudo de retorno da lista, de forma a não poluir o retorno.

@JsonProperty(“page”) 🡪 sobrescreve o que será escrito no json gerado pelo objeto referente, utilizado em classes DTO.

@EntityListeners 🡪 habilita autenticação para entidade

@RequiredArgsConstructor 🡪 anotação usada para injeção de dependência

@Transactional 🡪 Define que o método ou classe será executado dentro de uma transação, garantindo a consistência dos dados.

@Transactional(readOnly = true) 🡪 Indica que a transação é apenas de leitura, otimizando consultas e prevenindo alterações no banco de dados.

Exemplo de injeção de dependência via construtor por anotação

@RequiredArgsConstructor

Public class EstacionamentoService() {

private final EstacionamentoRepository estacionamentoRepository

...

}

HttpHeaders.LOCATION 🡪 Constante usada para definir o cabeçalho "Location" em uma resposta HTTP, geralmente utilizada para especificar a URL de redirecionamento ou a localização de um recurso recém-criado (ex: após um POST bem-sucedido).

Na aula 110 seção 16, não são cobertos todos os casos de teste para o erro 422... implementar.