



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA (DEI)

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Engenharia Reversa e Configuração de Aplicações com Payara e Node.js

Fábio Vilela (46294)

Orientadores

Professor Doutor Carlos Gonçalves

Professora Doutora Valeria Pequeno

Junho, 2025

Resumo

Este projeto surge da necessidade de recuperar o controlo técnico sobre a aplicação **Circuitos Oceânicos**, um sistema legado sem documentação e cujo autor original já não se encontra contactável. A sua relevância para a investigação histórica exigia uma solução que garantisse a sua continuidade e capacidade de manutenção.

Recorreu-se a uma abordagem de engenharia reversa para compreender a arquitetura do sistema, identificar os seus principais componentes e construir uma versão modular e funcional. A solução desenvolvida baseia-se em contentores **Docker**/**Podman**, permitindo a replicação automatizada da aplicação original, bem como a sua portabilidade e manutenção futura em diferentes ambientes.

Entre os principais contributos, destacam-se a contentorização do sistema, a produção de documentação técnica e a criação de um guia de utilização acessível para os administradores. O sistema foi validado com sucesso, demonstrando equivalência funcional à versão original.

Palavras-chave: Engenharia Reversa, Sistema Legado, Contentorização, Docker, Podman, Documentação Técnica, Manutenção, Portabilidade, Aplicação, Replicação.

Abstract

This project stems from the need to regain technical control over the **Circuitos Oceânicos** application, a legacy system lacking documentation and whose original author is no longer available. Its relevance to historical research required a solution that would ensure its continuity and maintainability.

A reverse engineering approach was adopted to understand the system's architecture, identify its main components, and build a modular and functional version. The solution developed is based on **Docker**/**Podman** containers, enabling automated replication of the original application, as well as its portability and future maintainability across different environments.

The main contributions include the containerization of the system, the production of comprehensive technical documentation, and the creation of an accessible user guide for administrators. The system was successfully validated, demonstrating functional equivalence with the original version.

Keywords: Reverse Engineering, Legacy System, Containerization, Docker, Podman, Technical Documentation, Maintainability, Portability, Application, Replication.

Agradecimentos

Agradeço aos meus orientadores, Professor Doutor Carlos Gonçalves e Professora Doutora Valéria Pequeno, pela orientação, disponibilidade e contributos fundamentais ao longo do desenvolvimento deste projeto.

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Tabelas	ix
Lista de Figuras	xi
Lista de Códigos	xiii
1 Introdução	1
2 <i>Background</i>	5
2.1 Arquitetura do Sistema Legado	5
2.2 HTML/JavaScript	6
2.3 Node.js	7
2.4 Payara	7
2.5 PostgreSQL	8
2.6 Docker e Podman	8
3 Modelo Proposto	11
3.1 Requisitos Funcionais e Não Funcionais	11
3.2 Abordagem	12

4	Implementação do Modelo	15
4.1	Implementação da Arquitetura	15
4.1.1	Configurações do Node.js	17
4.1.2	Configurações do Payara	18
4.1.3	Configurações do PostgreSQL	23
4.2	Orquestração com o Ficheiro <i>Compose</i>	24
4.3	Passos para Replicar o Sistema	25
5	Validação e Testes	29
6	Conclusões e Trabalho Futuro	33
A	<i>Deploy</i> Manual da <i>WebApp</i> Java	37
B	Criação Manual de uma JDBC Connection Pool	39
C	Criação Manual de um JDBC Resource	43
D	Criação Manual de um JNDI Custom Resource	45
E	CircPetitionario-Compose.yaml	47
F	Dockerfile (Node.js)	49
G	.env	51
H	Dockerfile (Payara)	53
	Bibliografia	55

Lista de Tabelas

B.1	Propriedades de Ligação à Base de Dados PostgreSQL	41
-----	--	----

Lista de Figuras

2.1	Arquitetura do Sistema Existente	5
3.1	Diagrama de Abordagem do Projeto	13
4.1	Arquitetura Proposta	16
4.2	<i>Login</i> na Página de administração do Payara	19
4.3	Diagrama do <code>CircPeticionario-Compose.yaml</code>	24
4.4	Iniciar Contentores no Docker Desktop/Podman Desktop	26
4.5	Página Inicial da aplicação Circuitos Oceânicos	27
5.1	Criação e População da base de dados PostgreSQL	29
5.2	Teste de <i>Login</i> na Aplicação	30
5.3	Validação do <i>Login</i> na Aplicação	30
6.1	Erro na página <code>registro.html</code>	34
6.2	Erro na página <code>palchave.html</code>	35
6.3	Arquitetura Real do Sistema	36
A.1	<i>Applications</i> no Menu vertical Payara	37
A.2	<i>Deploy</i> na página <i>Applications</i> do Payara	38
A.3	<i>Choose File</i> na página <i>Applications</i> do Payara	38
B.1	JDBC Connection Pools no Menu vertical Payara	39
B.2	<i>New JDBC Connection Pools</i> no Payara	40
B.3	Configuração JDBC Connection Pools 1	40
B.4	Configuração JDBC Connection Pools 2	41
C.1	JDBC Resources no Menu vertical Payara	43
C.2	<i>New JDBC Resource</i> no Payara	44

C.3	Configuração JDBC Resource	44
D.1	JNDI Custom Resources no Menu vertical Payara	45
D.2	<i>New</i> JNDI Custom Resource no Payara	46
D.3	Configuração JNDI Custom Resource	46

Lista de Códigos

1	Ficheiro <code>go.bat</code> para compilar a <i>WebApp</i> em ambientes Windows	20
2	Método <code>getConnectionStock()</code> da <i>WebApp</i>	21
3	Ficheiro <code>go.bat</code> principal	25

Capítulo 1

Introdução

Num contexto organizacional cada vez mais dependente de soluções tecnológicas específicas, torna-se crítico garantir a continuidade e a manutenibilidade dos sistemas em funcionamento. Este projeto incide sobre o sistema denominado **Circuitos Oceânicos**, e surge da necessidade de mitigar o risco associado à ausência de documentação técnica de um sistema funcional existente, cujo autor já não se encontra presente na instituição. Com o sistema a operar como uma "caixa negra", qualquer falha comprometeria gravemente a sua reposição, pois não existe conhecimento técnico interno que permita compreendê-lo, replicá-lo ou repará-lo com rapidez e eficácia.

A aplicação **Circuitos Oceânicos** "tem por finalidade o mapeamento das demandas enviadas da América portuguesa ao Reino de Portugal, bem como dos seus caminhos institucionais (trâmites) e das respostas e ordens provocadas a partir delas em ambos os lados do Atlântico, entre os anos de 1736 a 1807"¹. Assume, assim, um papel central no apoio à investigação histórica, funcionando como uma infraestrutura digital que facilita o acesso, organização e análise de fontes documentais relevantes para o estudo do período colonial luso-brasileiro. A sua utilização por historiadores reforça a importância do sistema como ferramenta de investigação científica e preservação do património histórico.

Este trabalho visou, por isso, aplicar engenharia reversa ao sistema existente, com o objetivo de compreender o seu funcionamento interno e pro-

¹Texto apresentado na página inicial da aplicação *Circuitos Oceânicos*

por uma solução, suportada sobre contentores, que possa ser facilmente replicável. A abordagem adotada permite não só restaurar o controlo técnico sobre o sistema como também estabelecer uma base sólida para futuras evoluções ou manutenções.

Da análise do sistema legado, verificou-se que o mesmo era composto por quatro componentes principais: o *frontend* em **HTML/JavaScript**, uma componente intermédia em **Node.js**, que faz a ligação entre o **frontend** e o **backend**, em **Java**, em execução no servidor de aplicações **Payara**, e a base de dados em **PostgreSQL**.

Os objetivos deste projeto foram então:

- A análise e compreensão do sistema existente;
- O desenvolvimento de uma solução funcional, com base no conhecimento obtido;
- A criação de documentação completa detalhada que descreve todo o processo necessário para a replicação do sistema;
- A elaboração de um guia prático de utilização acessível para os administradores finais do sistema;
- A contentorização dos diversos componentes do sistema.

O projeto foi desenvolvido em etapas bem definidas, começando pela análise do sistema original, passando pela identificação dos seus componentes essenciais e funcionamento interno, e culminando na construção de uma solução funcional equivalente. Foram realizados testes de validação para garantir a equivalência funcional entre o sistema original e a solução proposta. O repositório do projeto está disponível em [rep, 2025].

Este documento está estruturado da seguinte forma: após este Capítulo 1 de introdução, o Capítulo 2 apresenta o enquadramento teórico e tecnológico do projeto. O Capítulo 3 descreve em detalhe os requisitos que o sistema deve conter, bem como o seu sustento teórico e a abordagem adotada para a solução. No Capítulo 4 é apresentada a nova solução desenvolvida, incluindo

aspectos de implementação e arquitetura. O Capítulo 5 discute os testes realizados, os resultados obtidos e uma análise crítica dos mesmos. Por fim, no Capítulo 6, são apresentadas as conclusões, possíveis limitações do trabalho e propostas de desenvolvimento futuro.

Capítulo 2

Background

Este capítulo servirá como base teórica para as tecnologias já presentes na aplicação, bem como para outras utilizadas na solução final, de modo a explicar o funcionamento do sistema. A Seção 2.1 inclui a explicação da arquitetura do sistema legado. As seções seguintes (Seção 2.2, Seção 2.3, Seção 2.4, Seção 2.5) são focadas individualmente em cada componente da aplicação. Por fim, a Seção 2.6, descreve o que são o **Docker** e o **Podman** e quais as suas divergências, bem como estes se comparam com máquinas virtuais.

2.1 Arquitetura do Sistema Legado

A arquitetura do sistema legado está representada na Figura 2.1. Esta aplicação *web* pré-existente, é composta por três camadas principais: o *frontend*, o *middleware*, o *backend* e a base de dados.

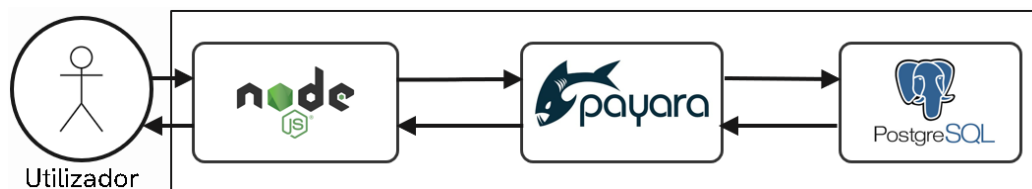


Figura 2.1: Arquitetura do Sistema Existente

O *frontend*, desenvolvido em HTML/JavaScript (representado na Figura 2.1 pelo "Utilizador"), é responsável pela apresentação dos dados ao utilizador

e pela recolha das suas interações. Este comunica diretamente com um *middleware*.

O *middleware* desenvolvido em `Node.js`, atua como intermediário entre o *frontend* e o *backend*, processando os pedidos e redirecionando chamadas HTTP para os serviços disponibilizados pelo servidor Payara.

O *backend*, desenvolvido em `Java` e executado no servidor Payara (visível no centro da Figura 2.1), disponibiliza uma API RESTful que permite a execução de diversas operações sobre uma base de dados PostgreSQL, como inserções, atualizações, validações de credenciais e remoções de dados.

Este tipo de arquitetura é composta por camadas que interagem entre si de forma sequencial. Promove uma separação clara de responsabilidades, facilita a manutenção e modernização do sistema, e permite a reutilização de componentes em diferentes contextos ou interfaces.

2.2 HTML/JavaScript

HTML (do inglês HyperText Markup Language) [htm, 2025] é uma linguagem de marcação utilizada para definir a estrutura e o conteúdo de páginas *web*, permitindo a incorporação de elementos, tais como, textos, imagens, botões e formulários.

JavaScript [jav, 2025] é uma linguagem de programação interpretada, orientada a objetos, que permite manipular dinamicamente os elementos de uma página *web* e reagir a eventos do utilizador.

Resumidamente, o HTML é responsável pela organização estrutural da interface do utilizador, enquanto o JavaScript acrescenta interatividade e dinamismo às páginas. A combinação destas tecnologias possibilita uma interação fluida entre o utilizador e a aplicação, com resposta visual quase imediata às suas ações.

2.3 Node.js

O Node.js [nod, 2025] é um ambiente de execução de JavaScript multi-plataforma, baseado no motor V8 do Google Chrome. Permite a execução de código JavaScript no lado do servidor, tornando possível o desenvolvimento de aplicações *backend* com a mesma linguagem utilizada no *frontend*.

A capacidade do Node.js para lidar com muitas operações de entrada/saída intensivas, deve-se à sua arquitetura orientada a eventos e ao modelo de I/O não bloqueante. Esta abordagem torna o Node.js especialmente adequado para aplicações como servidores *web*, APIs e sistemas em tempo real.

No contexto deste projeto, o Node.js é utilizado como componente intermédio entre o *frontend* (HTML/JavaScript) e o *backend* (Java), funcionando como um *middleware* responsável pelo encaminhamento das requisições e pelo tratamento das respostas entre as camadas da aplicação.

2.4 Payara

O Payara Server [pay, 2025] é um servidor de aplicações Java de código aberto, compatível com a especificação Jakarta EE (anteriormente conhecida como Java EE). Trata-se de uma evolução do GlassFish Server, desenvolvido para oferecer suporte contínuo e melhorias após o término do suporte comercial ao mesmo por parte da Oracle.

O Payara tem como principal objetivo fornecer uma plataforma robusta para a execução de aplicações Java. A administração do servidor pode ser realizada tanto através de linha de comandos quanto por meio de uma interface *web* intuitiva, facilitando a configuração e gestão do ambiente.

No contexto deste projeto, o Payara foi utilizado como servidor de aplicações para o *backend* desenvolvido em Java, suportando a execução e disponibilização da API RESTful [res, 2025] da aplicação.

2.5 PostgreSQL

O PostgreSQL [pos, 2025], é um sistema de gestão de bases de dados relacional de código aberto que utiliza e estende a linguagem SQL. Destaca-se pela sua arquitetura comprovada, fiabilidade, integridade dos dados e extensibilidade. Além disso, é possível definir tipos de dados personalizados, funções próprias e extensões adicionais que expandem as capacidades do sistema.

No contexto deste projeto, o PostgreSQL foi utilizado como sistema de base de dados do *backend* Java, armazenando todos os dados necessários ao funcionamento da aplicação.

2.6 Docker e Podman

Docker [doc, 2025] e Podman [pod, 2025] são plataformas de código aberto que permitem aos desenvolvedores construir, implementar, executar, atualizar e gerir contentores. Os contentores são componentes executáveis padronizados que combinam o código fonte da aplicação com as bibliotecas e dependências do sistema operativo necessárias para executar esse código em qualquer ambiente.

Apesar de apresentarem funcionalidades semelhantes, existem diferenças relevantes entre as duas plataformas. O Docker utiliza uma arquitetura baseada num *daemon* centralizado (*dockerd*), que corre com privilégios de *root*, enquanto o Podman opera de forma *daemonless*, sem necessidade de um processo em *background*, o que melhora a auditabilidade e reduz a superfície de ataque. Além disso, o Podman permite a execução de contentores em modo *rootless*, promovendo maior segurança em ambientes multiutilizador.

Ao contrário das máquinas virtuais, que emulam todo um sistema operativo incluindo o *kernel*, os contentores partilham o *kernel* da máquina *host* e isolam apenas as aplicações e respetivas dependências. Isto torna os contentores significativamente mais leves, rápidos a iniciar e mais eficientes em termos de recursos. Enquanto as máquinas virtuais necessitam de hipervisores e imagens completas de sistemas operativos, os contentores permitem encapsular apenas o essencial para a execução da aplicação. Esta abordagem

revelou-se particularmente vantajosa para o projeto em questão, ao facilitar a distribuição e a replicação do sistema de forma portátil e eficiente.

No âmbito deste projeto, estas tecnologias foram utilizadas para facilitar a replicação e distribuição do sistema. Permitiram encapsular todas as dependências e configurações necessárias para executar cada componente, garantindo que o sistema funciona de forma consistente em diferentes máquinas.

Capítulo 3

Modelo Proposto

Neste capítulo apresenta-se o modelo conceptual e funcional que serve de base à solução desenvolvida. Iniciamos pelos requisitos identificados na Seção 3.1 e procuramos construir uma descrição clara e estruturada da abordagem proposta na Seção 3.2.

3.1 Requisitos Funcionais e Não Funcionais

Os requisitos de um sistema podem ser classificados em dois tipos: requisitos funcionais e requisitos não funcionais.

Os requisitos funcionais descrevem as funcionalidades que o sistema deve oferecer, ou seja, as ações que o utilizador pode realizar e os comportamentos esperados do sistema em resposta a essas ações.

Por outro lado, os requisitos não funcionais referem-se a atributos de qualidade do sistema, como desempenho, segurança, usabilidade, escalabilidade ou fiabilidade. Estes requisitos não dizem respeito a funcionalidades específicas, mas são essenciais para garantir que o sistema funcione de forma eficiente, segura e robusta.

O projeto em questão tem os seguintes requisitos funcionais:

RF01. Contentorizar cada componente do sistema.

O projeto tem os seguintes requisitos não funcionais:

RNF01. O projeto deve poder ser replicado em qualquer máquina que tenha o Docker/Podman instalado.

3.2 Abordagem

A execução deste projeto decorreu em várias fases sucessivas, orientadas por uma abordagem prática de engenharia reversa, com o objetivo de recuperar o controlo técnico sobre o sistema **Circuitos Oceânicos** e garantir a sua continuidade operacional.

A primeira etapa consistiu na análise do sistema legado. Esta fase teve como objetivo compreender a estrutura da aplicação original, partindo do código disponibilizado. Uma vez que não existia qualquer documentação, foi necessário investigar manualmente os ficheiros, configurações e fluxos de execução, identificando as tecnologias utilizadas e os pontos de interligação entre os seus componentes.

Numa fase inicial, foram feitas tentativas de colocar a aplicação a funcionar localmente, sem recorrer à contentorização. Isto permitiu confirmar o comportamento esperado do sistema, mas evidenciou limitações sérias ao nível da portabilidade, reprodutibilidade e manutenção.

Com base no conhecimento obtido, passou-se à contentorização do sistema, através da criação de imagens e definição de serviços em ficheiros `Dockerfile` e `docker-compose`. Cada componente do sistema foi isolado em contentores independentes, garantindo a modularidade e a replicação automática do ambiente original. Esta solução também foi testada com o `Podman`, de modo a assegurar compatibilidade com diferentes ferramentas de contentorização.

Paralelamente, foi produzida documentação técnica detalhada, descrevendo cada etapa da implementação, a estrutura dos serviços, as dependências envolvidas e o processo completo de replicação do sistema. Adicionalmente, foi criado um guia prático de replicação, direcionado a administradores, com instruções simplificadas para instalação e arranque do sistema.

A última fase consistiu na validação da solução desenvolvida, através da

comparação funcional entre o sistema original e a nova versão contentorizada. Esta validação confirmou a equivalência entre ambas as versões e demonstrou a robustez e flexibilidade da nova abordagem (ver Figura 3.1).



Figura 3.1: Diagrama de Abordagem do Projeto

Capítulo 4

Implementação do Modelo

Este capítulo servirá para descrever a implementação do sistema e também irá conter os passos necessários para replicar a aplicação. Vamos começar por descrever como foi implementada a arquitetura (Seção 4.1) e descrever as configurações de cada componente. Também explicaremos como funciona o ficheiro `compose` e como este se relaciona com os ficheiros `Dockerfile` (Seção 4.2). Por fim iremos apresentar uma lista de passos que permitem realizar a replicação do sistema num ambiente `Docker`/`Podman` (Seção 4.3).

4.1 Implementação da Arquitetura

Tendo em consideração a necessidade de disponibilizar uma solução facilmente replicável e independente do ambiente da máquina hospedeira, foi adotada uma abordagem baseada em contentores.

A arquitetura existente inicialmente, representada na Figura 2.1, era composta por um *frontend* desenvolvido em `HTML` e `JavaScript`, uma camada intermédia baseada em `Node.js`, que funcionava como servidor *proxy* reverso, encaminhando os pedidos `HTTP` para o *backend*, implementado em `Java`. Este *backend* era executado no servidor de aplicações `Payara`, que comunicava diretamente com a base de dados `PostgreSQL`. Apesar da base de dados estar alojada numa máquina distinta das outras componentes, a arquitetura do sistema apresentava forte dependência de uma configuração centralizada e pouco modular, o que dificultava a sua manutenção, escalabilidade e, principalmente, a portabilidade.

Como alternativa, foi proposta uma arquitetura modular, suportada pela contentorização individual de cada um dos componentes do sistema: *frontend* e camada intermédia, *backend* e base de dados. Esta modularização foi conseguida através da utilização de ferramentas como **Docker** e **Podman**, que permitem isolar cada componente num contentor autónomo, com as respetivas dependências e configurações encapsuladas. Esta abordagem promove uma separação clara de responsabilidades, facilita a gestão e o ciclo de vida de cada módulo, e torna o sistema mais robusto perante falhas ou atualizações. Além disso, permite uma replicação simples do ambiente de desenvolvimento e produção, contribuindo para a portabilidade e escalabilidade da aplicação. O esquema da arquitetura proposta encontra-se representado na Figura 4.1, ilustrando a divisão funcional entre os módulos e a forma como interagem entre si.

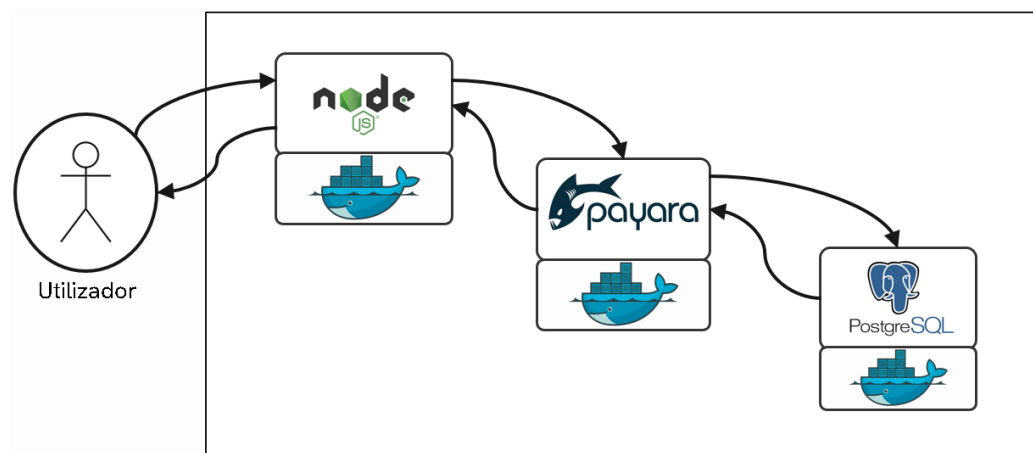


Figura 4.1: Arquitetura Proposta

A aplicação foi dividida em três componentes principais: *middleware* (Node.js), servidor de aplicações (Payara) e base de dados (PostgreSQL). Cada um destes componentes foi isolado num contentor distinto (exceto o *frontend*), permitindo um maior controlo sobre as dependências, versões e configurações específicas de cada serviço. A comunicação entre os contentores é feita através de uma rede interna que estabelece um **hostname** para cada serviço, permitindo que eles se comuniquem entre si.

As subsecções seguintes (Subsecção 4.1.1, Subsecção 4.1.2 e Subsecção 4.1.3) vão descrever quais as configurações necessárias, em cada contentor, para que se mantenha a funcionalidade esperada da aplicação.

4.1.1 Configurações do Node.js

O serviço correspondente ao Node.js foi contentorizado a partir de uma imagem personalizada, baseada numa versão mais leve que a padrão, a `node:alpine`. Esta imagem é construída com recurso a um `Dockerfile` que define o ambiente necessário para a execução da aplicação, incluindo a instalação de dependências e a definição da estrutura de diretorias e de portos.

O `Dockerfile` inclui ainda a instalação de utilitários úteis em contexto de desenvolvimento, bem como a definição da diretoria de trabalho e o comando de arranque da aplicação. Outro passo fundamental, contido no `Dockerfile` do Node.js, é a cópia dos ficheiros `package.json` e `package-lock.json` para dentro do contentor, pois estes contêm as dependências da aplicação. Toda a configuração da imagem encontra-se descrita diretamente no ficheiro `Dockerfile`, incluído em anexo no Apêndice F.

O ficheiro `CircPetitionario-Compose.yml` (ver em Apêndice E) é responsável pela orquestração deste serviço. O `Compose` contém a localização do `Dockerfile`, o mapeamento de portos e a integração em rede com os restantes serviços da aplicação. As variáveis de configuração necessárias são também definidas nesse contexto, permitindo a comunicação com a base de dados e garantindo o arranque sequencial dos serviços dependentes.

Para uma gestão centralizada da configuração, foi incluído um ficheiro `.env`, disponível em anexo no Apêndice G, responsável por armazenar variáveis sensíveis e rotas da aplicação. Este ficheiro permite atualizar *tokens* ou *URLs* sem necessidade de alterar diretamente o código-fonte, promovendo boas práticas de segurança e manutenção.

4.1.2 Configurações do Payara

O serviço Payara, correspondente ao servidor de aplicações Java, foi contentorizado com base numa imagem personalizada derivada da oficial `payara/server-full:5.2022.5`. A construção da imagem personalizada é feita com recurso a um `Dockerfile`, que inclui as definições específicas necessárias para o arranque e configuração inicial do servidor. Toda a configuração encontra-se detalhada no ficheiro `Dockerfile`, incluído no anexo Apêndice H.

Durante a criação do contentor, a aplicação *Web*, empacotada num ficheiro `.war`, é automaticamente implantada no servidor. Este processo é garantido pela criação de um volume persistente, para onde o ficheiro `.war` é copiado, permitindo que o Payara o detete e realize o *deploy* automaticamente.

O contentor expõe os portos 8080 e 4848, que correspondem, respetivamente, ao acesso à aplicação *Web* e à interface administrativa do Payara. Esta página de administração, permite fazer diversas configurações importantes no servidor de aplicações Payara. As configurações incluem, entre outros aspetos, a definição de *connection pools* JDBC, recursos JNDI e variáveis de ambiente específicas, essenciais para o correto funcionamento da aplicação. Inicialmente, estas configurações foram feitas manualmente através da interface administrativa do Payara, e posteriormente, exportadas no ficheiro `domain.xml`. Esse ficheiro foi incluído na imagem Docker personalizada do Payara, permitindo replicar automaticamente o ambiente configurado, sempre que um novo contentor é criado. Desta forma, evita-se a configuração manual a cada nova replicação do sistema.

De qualquer forma, para o caso de se pretender fazer as configurações manualmente na interface de administração, vamos descrevê-las de seguida. Para qualquer uma destas configurações, é primeiramente necessário aceder à página de administração do Payara (`http://localhost:4848`.) e fazer o *login* com as credenciais User Name: `admin` e Password: `admin`, como demonstrado na Figura 4.2.

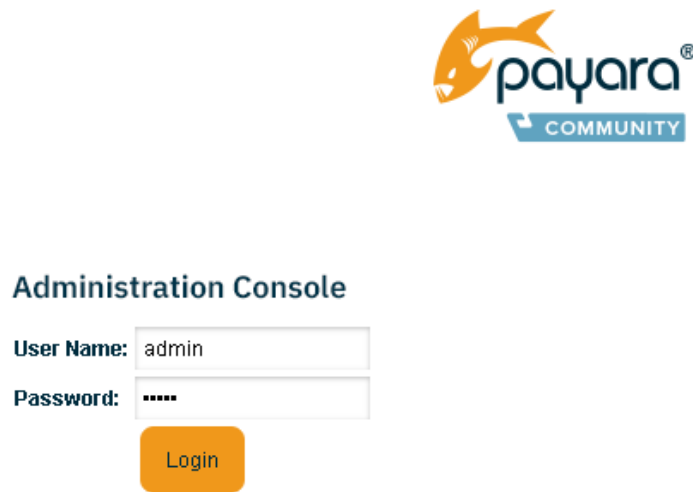


Figura 4.2: *Login* na Página de administração do Payara

Depois de concluída a autenticação, foi feito um conjunto de configurações, apresentadas de seguida.

Deploy da WebApp Java

A aplicação `CircPeticionario` foi configurada para ser implantada automaticamente com o contexto `/CircPeticionario`, e associada ao servidor virtual `server`. Esta associação permite o seu acesso externo através do porto 8080.

Consultar os passos descritos no Apêndice A para fazer o *deploy* manualmente.

A diretoria `./CircPeticionario-WebApp/src`, que contém o código-fonte da aplicação, pode ser acedida sempre que se pretenda editar o código e gerar um novo ficheiro `.war` da *WebApp*. Quando as alterações estiverem concluídas, deve-se voltar à diretoria `./CircPeticionario-WebApp` e executar

o ficheiro `go.bat` (basta fazer *double click*), que é específico para máquinas com sistema operativo **Windows**. No caso de se pretender executar num sistema **Linux**, basta remover o comando `call` e alterar a extensão do ficheiro para `.sh` (`go.sh`). Este ficheiro existe apenas para facilitar a execução dos comandos. Este conjunto de instruções compila a aplicação e gera o ficheiro `CircPeticonario.war` em `./CircPeticonario-WebApp/target`.

Código 1: Ficheiro `go.bat` para compilar a *WebApp* em ambientes **Windows**

```
call mvn clean install
```

Este *script* (Código 1) tem como função automatizar o processo de compilação da *WebApp*. O comando executa o ciclo padrão do **Maven**, que inclui a limpeza dos artefactos de *build* anteriores e a gera um novo ficheiro `.war` a partir do código-fonte presente em `./CircPeticonario-WebApp/src`.

Criação de uma JDBC Connection Pool

Foi criado a **JDBC Connection Pool** [jdb, 2025] `CircuitoPeticonario`, com as propriedades associadas ao acesso à base de dados **PostgreSQL**. Uma **JDBC Connection Pool** tem como principal objetivo otimizar o desempenho da aplicação na comunicação com a base de dados, mantendo um conjunto de ligações abertas que podem ser reutilizadas por múltiplas operações. Em vez de criar e fechar uma ligação a cada pedido, o que seria ineficiente e dispendioso, a aplicação reutiliza ligações existentes da *pool*, reduzindo a latência e o consumo de recursos.

Para fazer a configuração manualmente, consultar os passos descritos no Apêndice B.

Criação de um JDBC Resource

Antes de a aplicação **Java** poder aceder aos dados persistentes, é necessário estabelecer uma ligação com a base de dados. Em ambientes **Java EE**, como é o caso do servidor **Payara** presente nesta arquitetura, é comum utilizar recursos geridos pelo servidor, como fontes de dados (`DataSource`),

para abstrair e facilitar esta ligação. Este processo é realizado por meio do serviço JNDI (Java Naming and Directory Interface), que permite à aplicação localizar e utilizar recursos configurados no servidor.

O recurso `jdbc/app_cpet` foi configurado para se ligar ao *JDBC Connection Pool* denominado `CircuitoPeticionario`, tornando-o acessível à aplicação Java através de uma pesquisa via JNDI lookup (por exemplo, usando `@Resource(lookup="jdbc/app_cpet")`). Este recurso representa um `DataSource` que fornece conexões à base de dados de forma eficiente, reutilizando conexões abertas geridas pelo servidor, através da invocação do método `getConnection()`.

No método apresentado no Código 2, é ilustrado este processo: o recurso `jdbc/app_cpet` é obtido via JNDI lookup e a conexão à base de dados é estabelecida para uso na aplicação.

Código 2: Método `getConnectionStock()` da *WebApp*

```
public static Connection getConnectionStock() throws
    Exception {

    Context c = new InitialContext();

    try {

        DataSource d=(DataSource)c.lookup("jdbc/app_cpet");
        return d.getConnection();

    } catch (Exception ex) {

        throw new Exception("Unable to connect database
            CircuitoPeticionario");

    }

}
```

Para fazer a configuração manualmente, consultar os passos descritos no Apêndice C.

Configuração de um JNDI Custom Resource

Foi adicionado o recurso `TOKEN_SERVICE`, do tipo `java.lang.String`, contendo um *token* JWT como valor. Esta configuração foi necessária uma vez que, embora esse *token* esteja definido no ficheiro `.env` do serviço `Node.js`, não está acessível diretamente no ambiente do servidor `Payara`. Ao criar este recurso como uma entrada JNDI, torna-se possível disponibilizar o valor do *token* de forma segura e centralizada à aplicação `Java`, permitindo que este seja utilizado internamente para operações de validação ou autenticação.

Para fazer a configuração manualmente, consultar os passos descritos no Apêndice D.

Para permitir comunicação entre os contentores e facilitar o desenvolvimento e a manutenção, o serviço `Payara` foi integrado na rede interna `network_circ_peticonario`, sendo-lhe atribuído um `hostname`.

A construção da imagem personalizada, que está presente no `Dockerfile` (Apêndice H), inclui várias configurações importantes. Em primeiro lugar, o ficheiro de configuração `domain.xml` é substituído por uma versão customizada, colocada em `/opt/payara/appserver/glassfish/domains/domain1/config/`, que contém definições específicas como os recursos JNDI e JDBC necessários à aplicação (configurações essas, que foram explicadas anteriormente). De forma a possibilitar a comunicação entre o `Payara` e a base de dados PostgreSQL, o *driver* JDBC foi copiado para a diretoria `/opt/payara/appserver/glassfish/lib/`. O ficheiro `CircPeticonario.war` é copiado para a diretoria `/opt/payara/appserver/glassfish/domains/domain1/autodeploy/`, permitindo o seu *auto-deployment* na inicialização do servidor de aplicações. Por fim, o contentor é iniciado com o comando `asadmin start-domain --debug --verbose`, que permite executar o domínio principal do `Payara` em modo de *debug* e com mensagens detalhadas de arranque, facilitando a análise e correção de problemas durante o desenvolvimento.

4.1.3 Configurações do PostgreSQL

O serviço de base de dados foi containerizado com recurso à imagem oficial do **PostgreSQL**, versão 14, garantindo compatibilidade com a estrutura dos dados presentes nos *scripts* de *backup* de dados, exportados da aplicação em produção. O serviço foi configurado com credenciais definidas por variáveis de ambiente e inclui a criação automática de uma base de dados inicial para o sistema.

Para garantir a persistência dos dados entre reinicializações de contentores, foi definido um volume persistente montado na diretoria interna do contentor onde o **PostgreSQL** armazena os dados. Este volume assegura que os dados permanecem inalterados mesmo quando o contentor do serviço não está a correr.

Além disso, foi configurado um mecanismo de inicialização automática que permite executar *scripts* SQL aquando da primeira criação do contentor. Este processo consiste na montagem de uma pasta local, contendo ficheiros **.sql**, numa diretoria específica do contentor, reconhecida pelo **PostgreSQL**. Quando o contentor é iniciado pela primeira vez, esses *scripts* são automaticamente executados, possibilitando a criação das tabelas necessárias, definição de índices e inserção de dados iniciais. Desta forma, toda a estrutura inicial da base de dados é automaticamente construída sem necessidade de intervenção manual, facilitando a replicação e automação do ambiente. Apenas é necessário colocar os scripts que se pretende que executem inicialmente na diretoria do projeto **./DataBase/schema**.

O serviço encontra-se integrado numa rede virtual privada, onde lhe foi atribuído um **hostname**, permitindo a comunicação segura com os restantes componentes da aplicação. A porta padrão de acesso à base de dados foi exposta, possibilitando também o acesso externo.

As configurações completas encontram-se disponíveis no Apêndice E, onde está incluído o ficheiro **CircPetitionario-Compose.yml**. Esse ficheiro contém as configurações que permitem a orquestração de todos os contentores utilizados neste projeto.

4.2 Orquestração com o Ficheiro *Compose*

O ficheiro `CircPeticonario-Compose.yaml` (ver Apêndice E) tem um papel central na orquestração dos diferentes serviços que compõem o sistema. Este ficheiro permite definir, de forma estruturada, todos os contentores necessários, a rede interna, os volumes persistentes, e as respetivas interligações. A Figura 4.3 apresenta o diagrama correspondente à estrutura do `compose`.

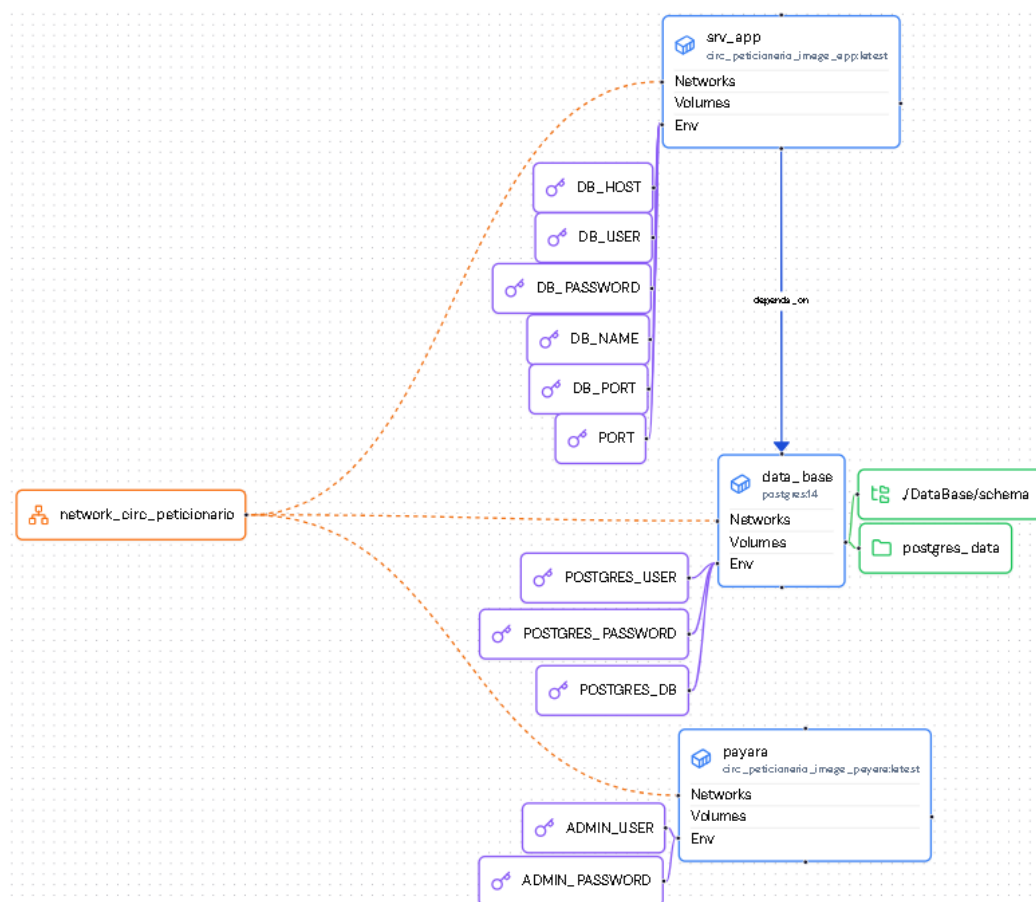


Figura 4.3: Diagrama do `CircPeticonario-Compose.yaml`

No caso do projeto desenvolvido, o `compose` define três serviços principais: a aplicação `Node.js` (`srv_app`), o servidor `Payara` (`payara`) e a base de dados `PostgreSQL` (`data_base`). Cada serviço é baseado numa imagem `Docker` que pode ser oficial (como no caso do `postgres:14`) ou personalizada, construída

a partir de um `Dockerfile`.

As instruções `build.context` e `build.dockerfile` de cada serviço indicam ao `compose` onde encontrar os ficheiros necessários para construir a imagem personalizada. Por exemplo, o serviço `srv_app` é construído a partir do `Dockerfile` localizado em `./FrontEnd`, e o serviço `payara` a partir de `./Payara`. Esses `Dockerfiles` (consultar Apêndice F e Apêndice H) definem o ambiente de execução e as configurações específicas de cada aplicação (como instalações de pacotes, cópia de ficheiros e comandos de arranque).

Além disso, o ficheiro `CircPeticionario-Compose.yml` define uma rede virtual interna (`network_circ_peticionario`) que garante a comunicação entre os serviços, atribuindo-lhes `hostnames`. Esta abordagem permite que os serviços comuniquem entre si de forma previsível e segura. Também são criados volumes para persistência de dados, como por exemplo, o volume `payara_deploy`, utilizado para disponibilizar o ficheiro `.war` à instância do Payara.

Ao utilizar o ficheiro `compose`, torna-se possível replicar todo o sistema com um único comando, garantindo que todos os serviços são lançados na ordem correta (graças à diretiva `depends_on`) e com as configurações adequadas. Esta abordagem promove a portabilidade e facilita implantação da aplicação em diferentes ambientes.

4.3 Passos para Replicar o Sistema

Nesta secção vamos descrever a sequência de passos necessários para replicar o sistema, de forma mais automatizada possível. Os mesmos que estão descritos em seguida:

1. Correr o ficheiro (*double click*) `go.bat` (Código 3 mostra o conteúdo desse ficheiro) localizado na raíz do projeto;

Código 3: Ficheiro `go.bat` principal

```
cd CircPeticionario-WebApp
call go.bat
```

```
cd ..
docker network rm network_circ_peticonario
docker network create --subnet=192.168.101.0/24 --
gateway=192.168.101.1 network_circ_peticonario
docker compose -f CircPeticonario-Compose.yaml -p
circ_peticonario create
```

Este script executa os seguintes passos:

- Acede à diretoria `./CircPeticonario-WebApp` e executa o `go.bat`, que compila a aplicação Java e gera um novo ficheiro `.war`;
 - Remove a rede Docker existente `network_circ_peticonario`, caso exista;
 - Cria uma nova rede com o mesmo nome, com o `subnet` e `gateway` definidos explicitamente;
 - Executa o comando `docker compose create`, que cria os contentores definidos no ficheiro `CircPeticonario-Compose.yaml` (ver Apêndice E), sem os iniciar;
2. No Docker Desktop/Podman Desktop, iniciar os três contentores criados. Depois esperar alguns minutos até que os dados sejam todos inseridos na base de dados;

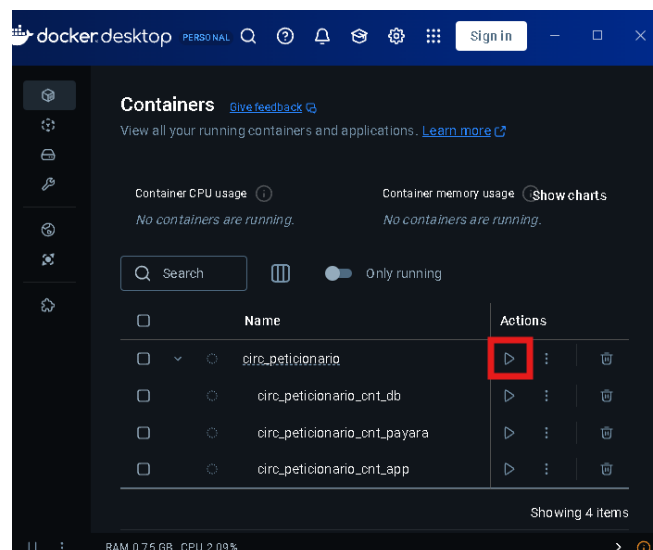
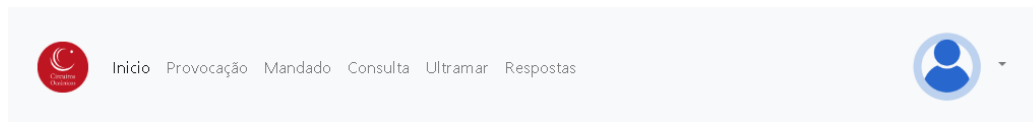


Figura 4.4: Iniciar Contentores no Docker Desktop/Podman Desktop

3. Finalmente, a aplicação fica disponível em: <http://localhost> (Figura 4.5);



Página Principal

A Base de Dados “Circuitos Oceânicos” tem por finalidade o mapeamento das demandas enviadas da América portuguesa ao Reino de Portugal, bem como dos seus caminhos institucionais (trâmites) e das respostas e ordens provocadas a partir delas em ambos os lados do Atlântico, entre os anos de 1736 a 1807. Foi desenvolvida a partir da documentação do Conselho Ultramarino e da Secretária de Estado da Marinha e do Ultramar, sob a guarda do Arquivo Ultramarino de Lisboa (AHU) e, em grande parte, disponibilizada online pelo Projeto Resgate (BNDigital). Ele é fruto de um projeto coletivo entre pesquisadoras da área de História e de Dados, coordenado por Andréa Slemian (UNIFESP), Renata Silva Fernandes (UFG, Goiás), Roberta Stumpf (Universidade Autónoma de Lisboa, UAL) e Valéria Pequeno (Escola Superior Náutica Infante D. Henrique, Lisboa). Trata-se de um projeto em andamento que, até o momento, trabalha com dados das capitanias de **Minas Gerais, São Paulo, Goiás e Mato Grosso**. A aplicação Web está sendo desenvolvida por Paulo Ricardo Rodrigues Bastos (mestrando na UAL) sob a orientação de Laércio Cruvinel (UAL, Lisboa). O projeto possui financiamento da FAPESP (processo número **2021/09104-0**).

Figura 4.5: Página Inicial da aplicação Circuitos Oceânicos

Capítulo 5

Validação e Testes

De modo a testar o funcionamento da aplicação, procedeu-se à sua replicação, conforme descrito na Seção 4.3, seguida de um teste de autenticação com credenciais válidas.

Primeiramente, foi executado o ficheiro `go.bat` (Código 3), localizado na raiz do projeto. Em seguida, iniciaram-se todos os contentores criados, aguardando alguns minutos para que a criação e população das tabelas da base de dados fossem concluídos. Este processo pode ser observado na Figura 5.1.

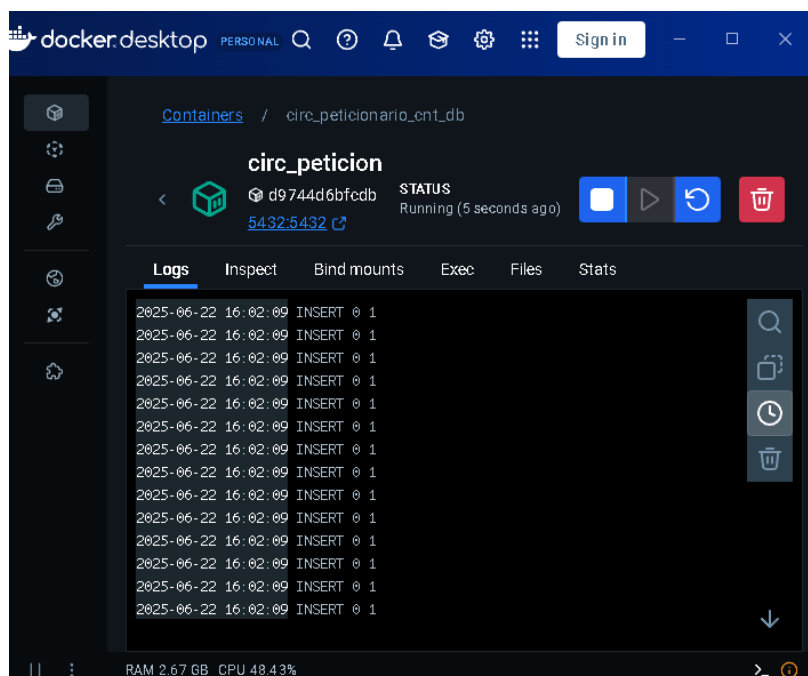
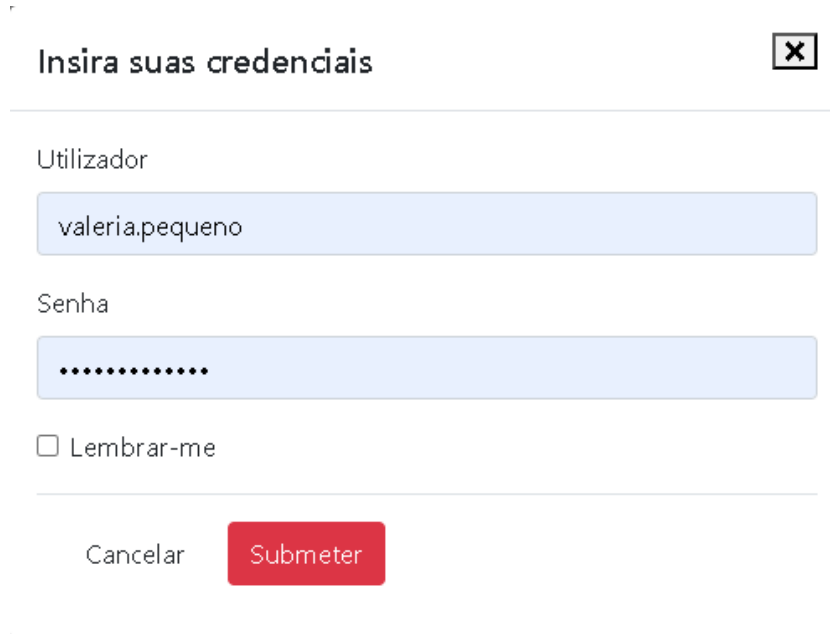


Figura 5.1: Criação e População da base de dados PostgreSQL

Posteriormente, acedeu-se à aplicação através do endereço `http://localhost` no *browser*, onde foi realizado o processo de autenticação. A interface de *login* encontra-se representada na Figura 5.2.



The image shows a login form titled "Insira suas credenciais" with a close button (X) in the top right corner. Below the title, there are two input fields: "Utilizador" (Username) containing the text "valeria.pequeno" and "Senha" (Password) containing a series of dots. Below these fields is a checkbox labeled "Lembrar-me". At the bottom, there are two buttons: "Cancelar" (Cancel) and "Submeter" (Submit).

Figura 5.2: Teste de *Login* na Aplicação



Figura 5.3: Validação do *Login* na Aplicação

Como é possível verificar pela observação da Figura 5.3, a autenticação foi concluída com sucesso. Após este passo navegou-se pela aplicação, não tendo sido detetados erros.

Durante o desenvolvimento do projeto, verificou-se um cenário real que permitiu validar a utilidade prática da solução proposta. O sistema original sofreu uma falha técnica inesperada, que impossibilitou o seu funcionamento. Este incidente destacou a fragilidade do sistema legado, especialmente devido à ausência de documentação e à dificuldade de recuperação.

A solução desenvolvida neste projeto foi então utilizada para restaurar rapidamente a aplicação, recorrendo à versão contentorizada da aplicação. Graças à arquitetura modular e à documentação técnica criada, foi possível replicar o ambiente original e colocar o sistema novamente em funcionamento com mínima intervenção. Este episódio funcionou como uma validação concreta da eficácia da abordagem adotada, demonstrando não só a equivalência funcional com a versão original, mas também a sua robustez, portabilidade e facilidade de manutenção.

Capítulo 6

Conclusões e Trabalho Futuro

O presente projeto teve como principal objetivo devolver o controlo técnico sobre o sistema legado **Circuitos Oceânicos**, cuja ausência de documentação e de conhecimento interno representava um risco significativo para a sua continuidade. Através da aplicação de técnicas de engenharia reversa e de uma análise minuciosa das suas componentes, foi possível reconstruir a arquitetura funcional da aplicação, compreendê-la em profundidade e propor uma solução modular, replicável, documentada e com todas as dependências auto-contidas necessárias ao seu correto funcionamento.

Durante o desenvolvimento deste projeto, constatou-se a importância de uma abordagem estruturada, assente na identificação de cada componente do sistema e respetiva interação. Esta análise permitiu a contentorização dos serviços e a orquestração dos mesmos através do ficheiro `compose`. Paralelamente, foram implementados artefactos técnicos essenciais como os `Dockerfile`, ficheiros de configuração e *scripts* auxiliares bem como documentação para garantir a replicação fiável do sistema.

A validação da solução desenvolvida demonstrou que a aplicação foi replicada com sucesso e está funcionalmente equivalente ao sistema original. Isto significa que investigadores e utilizadores finais continuarão a dispor de uma ferramenta valiosa para análise histórica, agora sustentada por uma infraestrutura moderna, portátil e mais facilmente gerível.

Trabalho Futuro

Apesar dos objetivos iniciais terem sido cumpridos, este trabalho pode servir de base para futuras melhorias e evoluções. Foram encontrados alguns *bugs* silenciosos que não comprometem, pelo menos até então, as funcionalidades da aplicação, mas que devem ser corrigidos futuramente:

Na página `resgistro.html`, quando autenticado com uma conta de administrador e se cria, altera ou elimina algum registro, surge um erro no *browser*, apresentado na Figura 6.1.

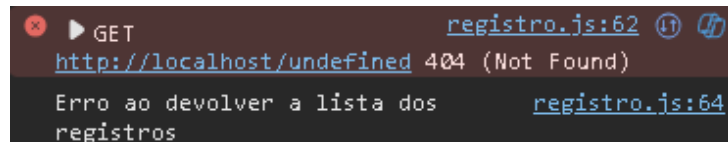


Figura 6.1: Erro na página `registro.html`

Ainda utilizando uma conta de administrador, na página `termo.html` (acessível através do menu no campo **Toponímia**), quando se cria um novo termo, a página não é atualizada automaticamente após se pressionar o botão **Criar**, e o formulário de criação continua no centro da página. Este comportamento obriga o utilizador a atualizar a página manualmente para ver a tabela com a nova entrada criada.

Na página `ultramar.html`, quando é criada uma nova entrada ultramar e não se seleciona nenhuma referência documental, dá origem a um `NullPointerException` nos *Logs* do contentor do **Payara**. Acontece exatamente o mesmo na página `consulta.html`, na criação de uma nova entrada quando não é atribuída nenhuma referência documental.

Também foram detetados erros relacionados com `eventListeners` em várias páginas, nomeadamente:

- `palChave.html`
- `tema.html`
- `agregadorTema.html`

- `secretario.html`
- `agregadorSecretario.html`
- `ofiTitulo.html`
- `agregador.html`
- `ecoOcup.html`
- `socJur.html`
- `referencia.html`
- `registro.html`
- `termo.html`
- `freguesia.html`
- `comarca.html`
- `capitania.html`

Este erro (Figura 6.2) é o mesmo em todas estas páginas e ocorre pois estão a tentar ser adicionados `eventListeners` a elementos HTML (aos elementos `VSearch`) que não existem;

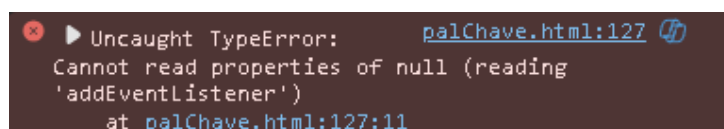


Figura 6.2: Erro na página `palchave.html`

Com o acesso ao código da aplicação *Web* do Payara, torna-se possível dar continuidade às melhorias desejadas pelos utilizadores finais, permitindo uma evolução alinhada com as suas necessidades.

Durante a fase de testes, verificou-se que a arquitetura inicialmente assumida para o sistema não correspondia integralmente à realidade. Embora

se considerasse que toda a comunicação entre o *browser* (*frontend*) e o servidor de aplicações **Payara** fosse mediada pela camada intermédia **Node.js**, constatou-se que, em determinadas situações, o navegador comunica diretamente com o servidor **Payara**, contornando essa camada. A Figura 6.3 apresenta o esquema corrigido da arquitetura do sistema.

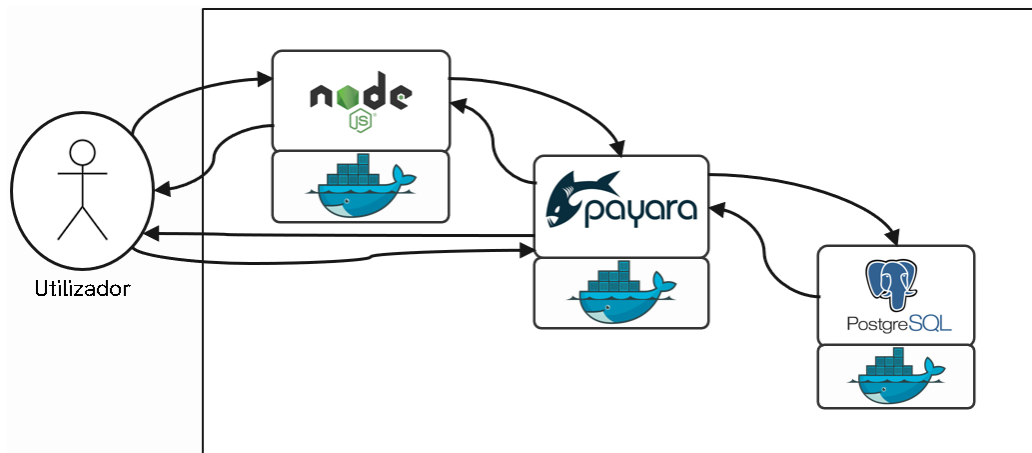


Figura 6.3: Arquitetura Real do Sistema

Como trabalho futuro, recomenda-se a reformulação das operações que atualmente ocorrem diretamente entre o *browser* e o servidor **Payara**, de modo a que todas as comunicações passem exclusivamente pela camada intermédia em **Node.js**. Esta alteração permitirá manter uma arquitetura mais coesa, consistente e segura.

Este projeto demonstrou que é possível recuperar, compreender e preservar sistemas legados através de uma abordagem prática e bem documentada. Espera-se que este trabalho possa servir de referência para iniciativas semelhantes noutras instituições, promovendo a sustentabilidade tecnológica de aplicações críticas, mesmo em cenários de perda de conhecimento técnico institucional.

Apêndice A

Deploy Manual da *WebApp* Java

- No menu vertical da página de administração do Payara, selecionar *Applications* (Figura A.1);

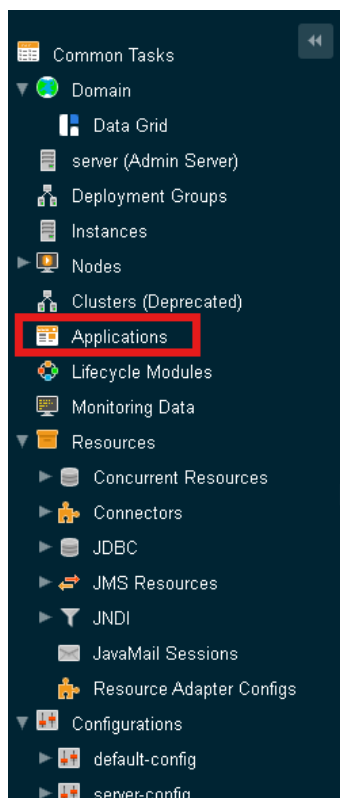


Figura A.1: *Applications* no Menu vertical Payara

- Na tabela central, selecionar *Deploy* (Figura A.2);

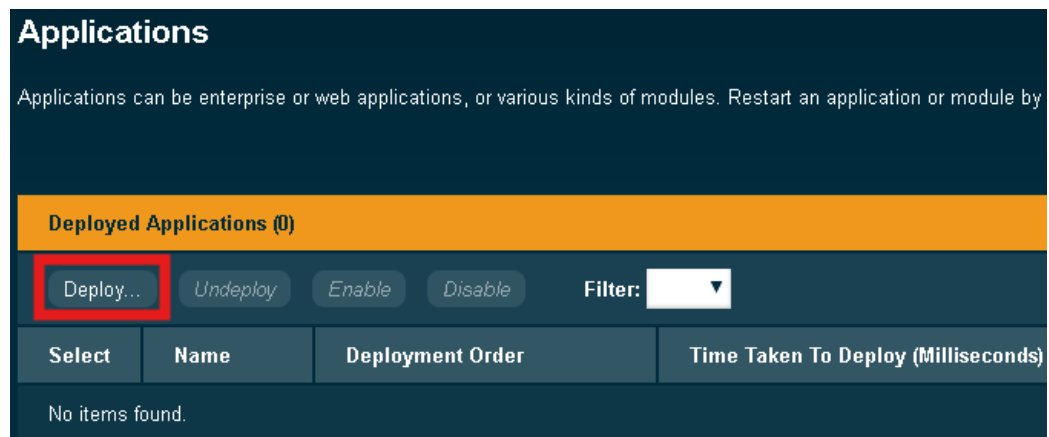


Figura A.2: *Deploy* na página *Applications* do Payara

- Selecionar *Choose File* e escolher o ficheiro *.war* pretendido (Figura A.3). Depois clicar em *OK* (localizado no canto superior direito da página) para finalizar o processo.

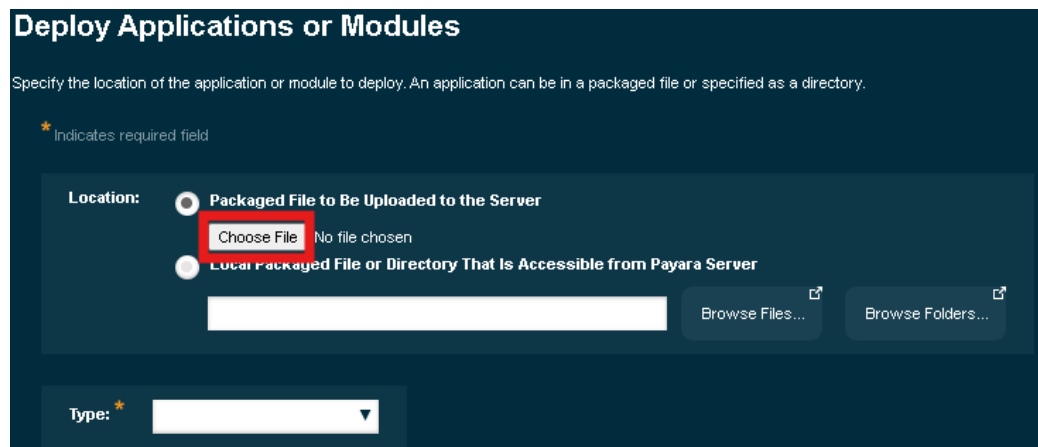


Figura A.3: *Choose File* na página *Applications* do Payara

Apêndice B

Criação Manual de uma JDBC Connection Pool

- No menu vertical da página de administração do Payara, selecionar JDBC → JDBC Connection Pools (ver Figura B.1);

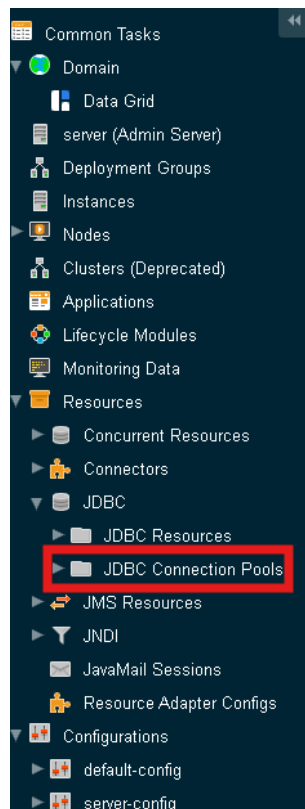


Figura B.1: JDBC Connection Pools no Menu vertical Payara

- Depois selecionar *New* (ver Figura B.2);

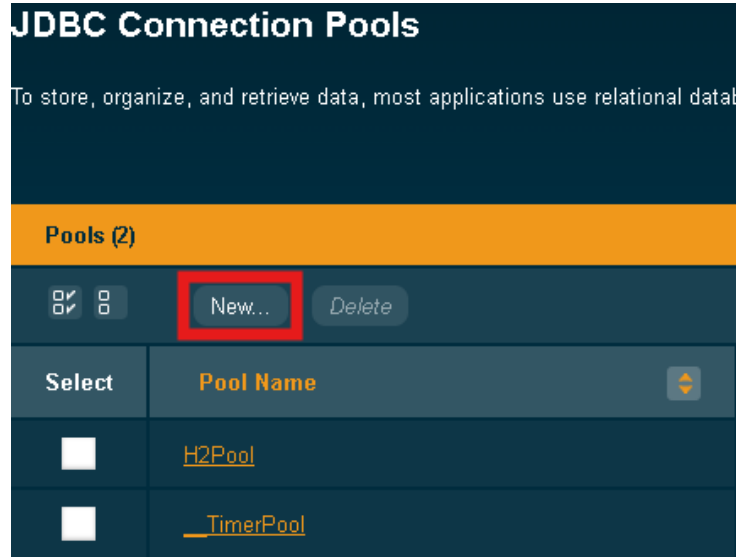


Figura B.2: *New* JDBC Connection Pools no Payara

- De seguida, preencher os campos como na Figura B.3 e selecionar *Next* (canto superior direito da página);

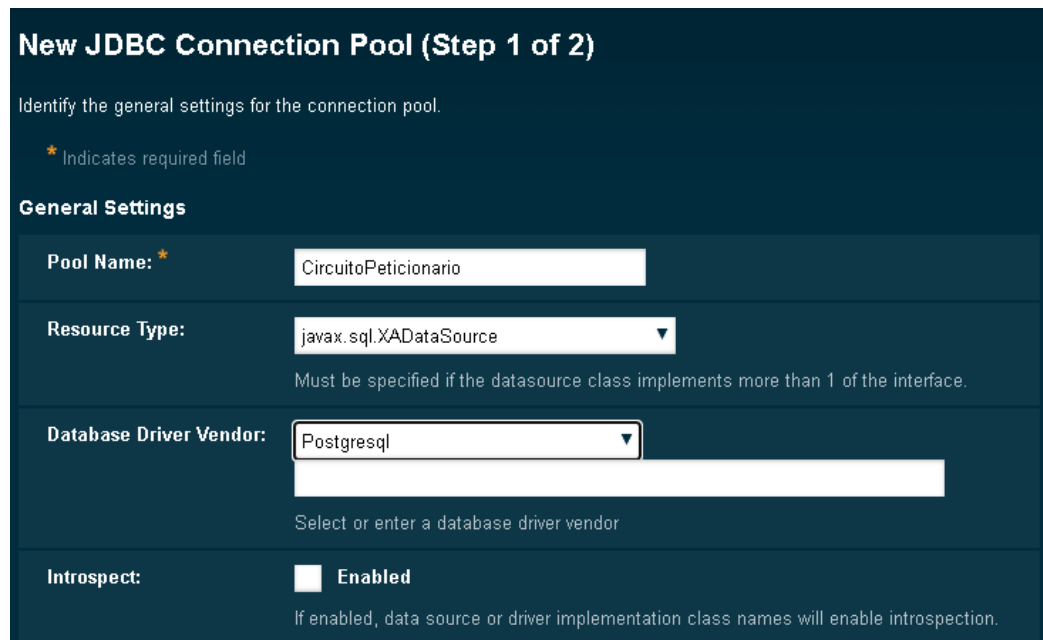


Figura B.3: Configuração JDBC Connection Pools 1

- Na página seguinte, fazer *scroll* até ao campo ***Additional Properties*** e completar com as seguintes propriedades (Tabela B.1):

Tabela B.1: Propriedades de Ligação à Base de Dados PostgreSQL

Name	Value
portNumber	5432
serverName	circ.peticionario.db
password	postgres
databaseName	proj_fapesp
user	postgres

Caso estas propriedades não estejam presentes na tabela, basta clicar em ***Add Property*** para adicionar uma nova (Figura B.4);

Select	Name	Value
<input type="checkbox"/>	user	postgres
<input type="checkbox"/>	databaseName	proj_fapesp
<input type="checkbox"/>	password	postgres
<input type="checkbox"/>	serverName	circ.peticionario.db
<input type="checkbox"/>	portNumber	5432

Figura B.4: Configuração JDBC Connection Pools 2

- Para finalizar, seleccionar ***Finish*** (localizado no canto inferior direito da página).

Apêndice C

Criação Manual de um JDBC Resource

- No menu vertical da página de administração do Payara, selecionar JDBC → JDBC Resources (Figura C.1);

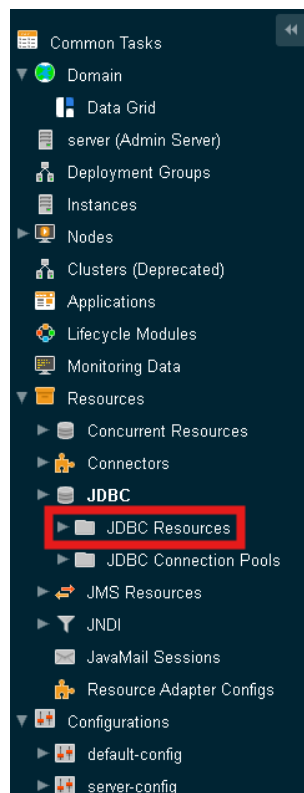


Figura C.1: JDBC Resources no Menu vertical Payara

- Na página seguinte, selecionar **New** (Figura C.2);

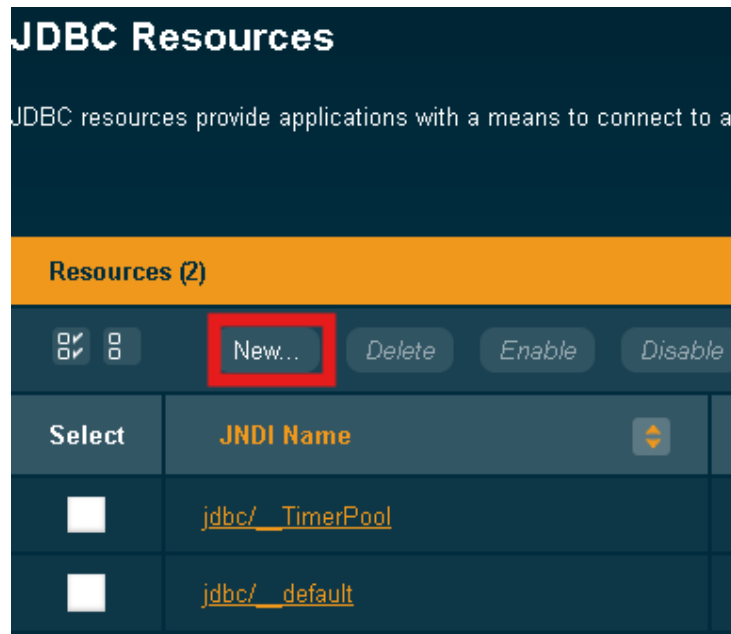


Figura C.2: *New* JDBC Resource no Payara

- Depois, preencher os campos como na Figura C.3 e, para concluir, selecionar **OK** (localizado no canto superior direito da página);

JNDI Name: *	<input type="text" value="jdbc/app_cpet"/>
Pool Name:	<input type="text" value="CircuitoPeticionario"/>
Use the JDBC Connection Pools page to create new pools	
Description:	<input type="text"/>
Status:	<input checked="" type="checkbox"/> Enabled

Figura C.3: Configuração JDBC Resource

Apêndice D

Criação Manual de um JNDI Custom Resource

- No menu vertical da página de administração do Payara, selecionar JNDI → Custom Resources (Figura D.1);

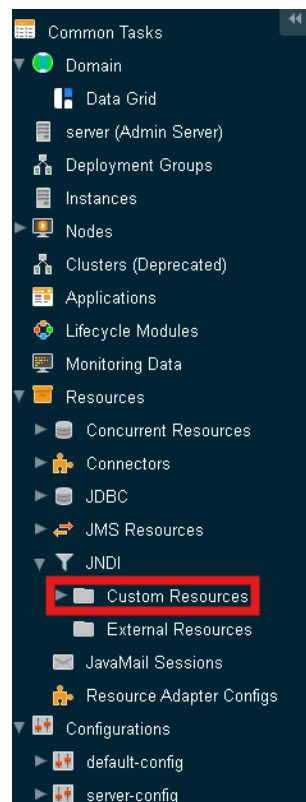


Figura D.1: JNDI Custom Resources no Menu vertical Payara

- Clicar em *New* (Figura D.2);



Figura D.2: *New* JNDI Custom Resource no Payara

- Preencher os campos conforme ilustrado na Figura D.3. No campo *Value*, em *Additional Properties*, deve ser introduzido o valor da variável `TOKEN_SERVICE`, definida no ficheiro `.env`, localizado na diretoria `./FrontEnd/` do projeto. Para concluir o processo, selecionar *OK*, no canto superior direito da página.

Figura D.3: Configuração JNDI Custom Resource

Apêndice E

CircPeticonario-Compose.yaml

```
networks:
  network_circ_peticonario:
    external: true

volumes:
  postgres_data:
  node_modules:
  payara_deploy: # Volume para o WAR

services:

  # PostgreSQL Database
  data_base:
    image: postgres:14
    container_name: circ_peticonario_cnt_db
    hostname: circ.peticonario.db
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./DataBase/schema:/docker-entrypoint-initdb.d
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=proj_fapesp
    ports:
      - '5432:5432'      # Data base access
    networks:
      network_circ_peticonario:
        ipv4_address: 192.168.101.10
```

```
# Node.js Application
srv_app:
  build:
    context: ./FrontEnd
    dockerfile: Dockerfile
  image: circ_peticionario_image_app
  container_name: circ_peticionario_cnt_app
  hostname: circ.peticionario.app
  working_dir: /app
  environment:
    - DB_HOST=db.eng.rev
    - DB_USER=postgres
    - DB_PASSWORD=postgres
    - DB_NAME=proj_fapesp
    - DB_PORT=5432
    - PORT=80
  ports:
    - '80:80'          # Web Server - User entry point
  depends_on:
    - data_base
  command: ["node", "index.js"]
  networks:
    network_circ_peticionario:
      ipv4_address: 192.168.101.20

# Payara Server
payara:
  build:
    context: ./Payara
    dockerfile: Dockerfile
  image: circ_peticionario_image_payara
  container_name: circ_peticionario_cnt_payara
  hostname: circ.peticionario.payara
  ports:
    - '8080:8080'      # Web Server
    - '4848:4848'      # Web consola de administracao
                      do Payara
  environment:
    - ADMIN_USER=admin
    - ADMIN_PASSWORD=admin
  networks:
    network_circ_peticionario:
      ipv4_address: 192.168.101.30
```


Apêndice F

Dockerfile (Node.js)

```
# Uses the official, lightweight Node.js image based on
  Alpine Linux
FROM node:alpine

# Updates the Alpine package index
RUN apk update

# Installs the vim text editor (useful for manual
  debugging)
RUN apk add vim

# Installs the nano text editor (alternative to vim)
RUN apk add nano

# Set working directory inside the container
WORKDIR /app

# Copies dependency configuration files to the working
  directory
COPY package.json package-lock.json ./

# Installs application dependencies defined in package.
  json
RUN npm install

# Copies all remaining application files into the
  container
COPY . .
```

```
# Exposes port 80 for external access to the application
EXPOSE 80

# Starts the application by running index.js with Node.
js
CMD ["node", "index.js"]
```

Apêndice G

.env

```
ACCESS_TOKEN_SECRET=***  
REFRESH_TOKEN_SECRET=***  
URL_PAYARA_PROXY=http://circ.peticionario.payara:8080  
URL_PAYARA_DIRETO=http://localhost:8080  
URL_NODE=http://localhost  
TOKEN_SERVICE=***
```


Apêndice H

Dockerfile (Payara)

```
# Base image with full Payara Server
FROM payara/server-full:5.2022.5

# Set the working directory inside the container
WORKDIR /opt/payara

# Replace the default domain.xml with a preconfigured
  one (includes JNDI, JDBC, and other server settings)
COPY domain.xml /opt/payara/appserver/glassfish/domains/
  domain1/config/domain.xml

# Add the PostgreSQL JDBC driver to the server's
  classpath
COPY drivers/postgresql-42.7.5.jar /opt/payara/appserver
  /glassfish/lib/

# Deploy the application by copying the WAR file to the
  autodeploy directory
# This ensures automatic deployment during container
  startup
COPY deploy/CircPeticionario.war /opt/payara/appserver/
  glassfish/domains/domain1/autodeploy/

# Expose the default HTTP port and the admin console
  port
EXPOSE 8080 4848

# Start Payara using asadmin with debug and verbose
  output enabled
```

```
ENTRYPOINT ["/opt/payara/appserver/bin/asadmin"]  
CMD ["start-domain", "--debug", "--verbose"]
```

Bibliografia

- [doc, 2025] (2025). Docker. <https://docs.docker.com/get-started/overview/>.
- [htm, 2025] (2025). HTML. <https://developer.mozilla.org/pt-BR/docs/Web/HTML>.
- [jav, 2025] (2025). Javascript. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.
- [jdb, 2025] (2025). Jdbc connection pool. <https://docs.payara.fish/community/docs/5.184/documentation/user-guides/connection-pools/connection-pools.html>.
- [nod, 2025] (2025). Node.js. <https://nodejs.org/pt/learn/getting-started/introduction-to-nodejs>.
- [pay, 2025] (2025). Payara. <https://www.payara.fish/>.
- [pod, 2025] (2025). Podman. <https://podman.io/>.
- [pos, 2025] (2025). Postgresql. <https://www.postgresql.org/about/>.
- [rep, 2025] (2025). Repositório do Projeto. <https://github.com/justFV/PRJ0346294>.
- [res, 2025] (2025). Restful api - introdução com node.js. <https://www.geeksforgeeks.org/node-js/rest-api-introduction/>.