André Oliveira (107637), Alexandre Cotorobai (107849)
Grupo 06, 2025-12-18

RELATÓRIO

Trabalho Prático

Introdução

O presente relatório descreve o desenvolvimento do trabalho prático realizado no âmbito da unidade curricular de Programação Web, que teve como principal objetivo a evolução de um *website* anteriormente estático, composto apenas por HTML e CSS, para uma aplicação *web* dinâmica e interativa, recorrendo de forma extensiva a JavaScript. Partindo do trabalho anterior, centrado exclusivamente na estrutura e na apresentação visual, esta etapa procurou enriquecer significativamente a plataforma através da introdução de lógica de programação, gestão de dados e interação com o utilizador.

Foram adicionadas diversas funcionalidades dinâmicas que transformam o *website* numa plataforma funcional de apoio ao estudo da Matemática, incluindo um sistema de autenticação com gestão de sessão, áreas reservadas ao utilizador, catálogo dinâmico de recursos educativos, gerenciamento de planos de estudo, sistema de favoritos, calculadora com histórico persistente, alternância de temas visuais (claro, escuro e sistema), formulários com validação em tempo real e integração com serviços externos para envio de emails de contacto. Adicionalmente, foram implementados mecanismos de persistência de dados no **localStorage**, permitindo manter o estado da aplicação entre sessões e páginas.

Este relatório tem como finalidade apresentar as soluções técnicas adotadas, detalhar o funcionamento das diferentes componentes dinâmicas do sistema e justificar as decisões de implementação tomadas ao longo do desenvolvimento. Pretende-se, assim, demonstrar a aplicação prática dos conceitos abordados na unidade curricular, nomeadamente, o domínio do JavaScript para a manipulação do DOM, a programação assíncrona e a modularização do código, evidenciando a transição de um *website* estático para uma aplicação *web* dinâmica.

Elementos/Partes Dinâmicas

Ficheiro “utils.js”

A camada de suporte transversal à aplicação é assegurada pelo ficheiro **utils.js**, que centraliza funções reutilizáveis para evitar a duplicação de código. Um dos componentes centrais é a função assíncrona **loadData(file)**, que permite o carregamento simplificado e padronizado de ficheiros JSON, como, por exemplo, listas de utilizadores e recursos, em qualquer parte do sistema.

Para a manipulação da interface, foi implementada a função auxiliar **setTextById(id, text)**. Esta função encapsula a lógica de atualização de texto no DOM, garantindo a robustez do código ao validar a existência do elemento alvo antes de tentar alterar o seu conteúdo, prevenindo assim erros de execução caso o ID não exista na página atual.

```
async function loadData(file) {
    let response = await fetch(file);
    let data = await response.json();
    return data;
}
```

```
function setTextById(id, text) {
    let element = document.getElementById(id);
    if (element != null) {
        element.innerText = text;
    }
}
```

Ficheiro “theme.js”

A gestão da aparência visual (Modo Claro/Escuro/Sistema) é controlada pelo ficheiro **theme.js**, mas inicia-se preventivamente através de um *script* de execução imediatamente inserido no **<head>** de todas as páginas HTML. Esta estratégia de inicialização antecipada lê a preferência guardada no **localStorage** e aplica o atributo **data-theme** ao documento antes do conteúdo visual ser exibido, prevenindo assim a exibição momentânea de cores incorretas durante o carregamento.

```
<script>
  const theme = localStorage.getItem("mathpath-theme");
  if (theme && ["auto", "light", "dark"].includes(theme)) {
    document.documentElement.setAttribute("data-theme",
    theme);
  }
</script>
```

A lógica principal é centralizada na função **applyTheme(value, options),** disparada pelo evento **onclick**, que manipula o DOM para atualizar o atributo da raiz, o estado dos inputs radio e o ícone da interface (através de **updateToggleIcon(value)**). O sistema destaca-se pelo suporte ao modo “Sistema”, utilizando a API **window.matchMedia** para detetar a preferência do sistema operativo (**prefers-color-scheme**). Adicionalmente, foi implementado um **event listener** que dispara a função **handleMediaChange()**, permitindo que a aplicação reaja e se adapte instantaneamente caso o utilizador altere o tema do seu dispositivo enquanto navega no site.

```
function applyTheme(value, options)
```

```
function updateToggleIcon(value)
```

```
function handleMediaChange()
```

```
<input
  type="radio"
  class="theme-radio"
  name="site-theme"
  id="theme-auto"
  checked
  onclick="applyTheme('auto')"
/>
<input
  type="radio"
  class="theme-radio"
  name="site-theme"
  id="theme-light"
  onclick="applyTheme('light')"
/>
<input
  type="radio"
  class="theme-radio"
  name="site-theme"
  id="theme-dark"
  onclick="applyTheme('dark')"
/>
```

Página “index.html”

A página de entrada da plataforma foi desenhada para ser totalmente configurável através de dados externos, permitindo a atualização de destaques e conteúdos sem necessidade de intervenção direta na estrutura HTML. Toda a lógica de apresentação é orquestrada pelo ficheiro **index.js**, que inicia o ciclo de vida da página através da função assíncrona **loadIndexData()**. Esta função é responsável por carregar o ficheiro JSON **index-data.json** e distribuir a informação recolhida por três funções distintas que correspondem a locais distintos no HTML (coleções organizadas por ano, destaques da semana e testemunhos).

```
async function loadIndexData()
```

A construção visual começa com a função **renderFeatures(features)**, que gera a tabela de funcionalidades principais ao mapear os ícones e descrições para os respetivos cartões na interface. De seguida, a função **renderFeaturedResources(resources)** constrói a secção de novidades semanais, destacando-se tecnicamente pela implementação de uma lógica de iteração *nested* que processa a lista de pontos-chave de cada recurso antes de os inserir na estrutura do cartão. Por fim, a função **renderTestimonials(testimonials)** povoa a secção de testemunhos.

```
function renderFeatures(features)
```

```
<div class="row gy-4" id="features-container"></div>
```

```
function renderFeaturedResources(resources)
```

```
<div class="row gy-4" id="featured-resources-container"></div>
```

```
80    function renderTestimonials(testimonials)
402           <div class="row gy-4" id="testimonials-container"></div>
```

Página “login.html”

O processo de início de sessão é controlado integralmente pelo ficheiro **login.js**. A verificação do estado do utilizador começa logo no carregamento da página com a função **checkIfAlreadyLoggedIn()**, que redireciona para a página inicial caso detete uma sessão com a opção “Lembrar-me” ativa.

```
91    function checkIfAlreadyLoggedIn()
```

A interação principal ocorre quando o utilizador submete o formulário, acionando a função **handleLogin()** através do evento **onsubmit**. Esta função orquestra todo o fluxo de entrada, pois valida o preenchimento dos campos através da função **validateForm(username, password)** e verifica a correspondência das credenciais consultando o ficheiro **users.json** (carregado via **loadUsers()**).

```
43    async function handleLogin()
```

```
19    function validateForm(username, password)
```

```
1     async function loadUsers()
```

```
<form
  id="login-form"
  novalidate
  onsubmit="handleLogin(); return false;"
>
```

Para garantir uma boa experiência de utilização, o sistema fornece *feedback* visual imediato através da função **showAlert(message, type)**, que injeta dinamicamente mensagens de erro ou de sucesso no contentor de alertas definido no HTML, gerindo o redirecionamento final para a área pessoal apenas se a autenticação for bem-sucedida.

```
5     function showAlert(message, type)
```

```
202           <div id="alert-container"></div>
```

Gestão da Sessão de Autenticação

A gestão de autenticação encontra-se centralizada no ficheiro **auth.js**. O sistema baseia-se na função **getSession()**, que recupera os dados do **localStorage**, e na **isLoggedIn()**, que valida a existência de sessão ativa.

```
1     function getSession()
```

```
9     function isLoggedIn()
```

A interação direta com o utilizador é controlada pela função **handleUserIconClick()**, invocada pelo evento **onclick** no botão de perfil da navbar (exemplo em **index.html**). Esta função verifica a autenticação e força o redirecionamento para a página de login caso o utilizador não esteja autenticado. O encerramento da sessão é tratado pela função **logout()**, acionada ao clicar na opção “Terminar sessão” através do evento **onclick**, que limpa os dados locais e reencaminha para a página inicial.

```
21    function handleUserIconClick()
```

```
<button
  class="btn btn-outline-light rounded-pill d-flex
  align-items-center px-3 py-2"
  type="button"
  data-bs-toggle="dropdown"
  aria-expanded="false"
  aria-label="Abrir menu do perfil"
  onclick="handleUserIconClick()"
>
```

```
16    function logout()
```

```
<a class="dropdown-item text-danger" href="#"
onclick="logout()"
  ><i class="bi bi-box-arrow-right me-2"></i>Terminar
  sessão</a
>
```

Por fim, a interface é adaptada pela função **updateHeader()**. Esta função manipula diretamente o DOM para alternar entre a exibição do nome do utilizador (extraído da sessão) e os ícones de login.

```
27    function updateHeader()
```

Página “account.html”

A personalização e gestão da área do utilizador são orquestradas pelo ficheiro **account.js**. O ciclo de vida da página inicia-se com a função **initializeAccountPage()**, que valida primeiramente a segurança através de **checkAuthentication()**, forçando o redirecionamento imediato para o login caso a sessão seja inválida.

```
180   function initializeAccountPage()
```

```
1     function checkAuthentication()
```

A apresentação inicial dos dados baseia-se na informação do utilizador guardada na sessão do **localStorage** (perfil, dados de escola, último acesso formatado, etc.), sendo injetada no DOM pelas funções **updateProfileSummary()**, **updateAccountDetails()** e **updateSkills()**. Para garantir a interatividade, foi implementada a função **savePreferences()**. Esta recolhe os dados editados pelo utilizador no modal de preferências, atualiza o objeto de sessão local e invoca novamente as funções de exibição para refletir as alterações na interface em tempo real, sem necessidade de recarregar a página.

```
37  function updateProfileSummary(userData)
```

```
282             <h4 class="mb-1" id="profile-name"></h4>
283             <p
284               class="text-secondary mb-1"
285               id="profile-year-course"
286             ></p>
287             <p
288               class="text-muted font-size-sm mb-3"
289               id="profile-school"
290             ></p>
```

```
305             <span class="text-secondary" id="profile-email"></span>
```

```
314             <span class="text-secondary" id="profile-goal"></span>
```

```
310             <img
311               id="profile-picture"
312               src=""
313               alt="Foto de perfil de estudante MathPath"
314               class="rounded-circle p-1 bg-primary"
315               width="110"
316             />
```

```
51  function updateAccountDetails(userData)
```

```
351             </div>
352             <span
353               class="badge text-bg-secondary"
354               id="account-year"
355             ></span>
356             <p class="mb-0 fw-semibold" id="account-email"></p>
```

```
365             <p class="mb-0" id="account-school"></p>
```

```
371             <span
378               class="badge rounded-pill text-bg-info text-white"
379               id="account-course"
380             ></span>
381
```

```
68  function updateSkills(userData)
```

```
388             <div class="row g-2" id="account-skills"></div>
```

```
464  function savePreferences()
```

```
704             <button
705               type="button"
706               class="btn btn-primary"
707               onclick="savePreferences()"
708             >
709               Guardar preferências
710             </button>
```

Para os conteúdos mais complexos, o sistema utiliza a função assíncrona **renderStudyPlan()**. Esta conjuga os dados da sessão com o ficheiro externo **users.json** para construir a tabela do plano semanal, permitindo ainda a sua atualização dinâmica através de, por exemplo, **savePlan()** e **addPlanRow()** com o evento **onclick**.

```
137  async function renderStudyPlan()
```

```
435             <tbody id="study-plan-body">
436               <tr>
437                 <td colspan="4" class="text-center p-3">
438                   <div
439                     class="spinner-border text-primary"
440                     role="status"
441                   >
442                     <span class="visually-hidden">A carregar...</span>
443                   </div>
444                 </td>
445               </tr>
446             </tbody>
```

```
355  function savePlan()
```

```
644             <button type="button" class="btn btn-primary" onclick="savePlan()">
645               Guardar alterações
646             </button>
```

```
302  function addPlanRow()
```

```
632             <button class="btn btn-success btn-sm mt-3" onclick="addPlanRow()">
633               <i class="bi bi-plus-circle me-1"></i>Adicionar linha
634             </button>
```

Destaca-se ainda a gestão de favoritos, onde a função **displayFavorites()** orquestra a interface visual. Para tal, implementa a lógica de clonagem de *templates* existentes no HTML, enquanto delega as operações de dados (como **getFavorites()** e **removeFromFavorites(resourceId)**) para as funções auxiliares do ficheiro **favorites.js**, garantindo assim uma clara separação entre a visualização e a manipulação de dados.

```
96  function displayFavorites()
```

```
525             <template id="template-favorite-item">
```

```
51  function removeFromFavorites(resourceId)
```

```
10  function getFavorites()
```

Página “calculator.html”

A funcionalidade de cálculo matemático é assegurada pelo ficheiro **calculator.js**. A lógica aritmética central reside na função **calculate()**, acionada pelo botão de igualdade através do evento **onclick**, que processa as operações básicas e gere exceções de erros, como a divisão por zero. O sistema inclui ainda funções científicas dedicadas, como **calculateSquareRoot()** e **calculatePower()**, invocadas também diretamente pelos eventos **onclick** da interface.

```
31  function calculate()
```

```
371             <button
372               class="btn btn-primary w-100 h-100"
373               onclick="calculate()"
374             >
375               =
376             </button>
```

```
104  function calculateSquareRoot()
```

```
402             <button
403               class="btn btn-info w-100"
404               onclick="calculateSquareRoot()"
405             >
```

```
138  function calculatePower()
```

```
410             <button
411               class="btn btn-info w-100"
412               onclick="calculatePower()"
413             >
```

```
11  function appendNumber(number) {
```

```
<button
  class="btn btn-light w-100"
  onclick="appendNumber('3')"
>
  3
</button>
```

Um componente distintivo deste módulo é a persistência do histórico de operações. A função **addToHistory(expression, result)** não só exibe visualmente os cálculos anteriores na interface através de **displayHistory()**, como também os salvaguarda no armazenamento local do navegador (**localStorage**) via **saveHistory()**. Isto garante que o registo de atividade do aluno é preservado.

```
134  function displayHistory()
```

```
123  function addToHistory(expression, result)
```

```
457  <div id="history" class="history-list"></div>
```

```
168  function saveHistory()
```

Página "catalog.html"

A biblioteca de conteúdos é gerada de forma totalmente dinâmica pelo ficheiro **catalog.js**. O processo inicia-se com a função assíncrona **loadAllResources()**, que agrega os dados provenientes de múltiplos ficheiros JSON (**recursos-10.json**, **recursos-11.json** e **recursos-12.json**). Para exibir esta informação, o sistema implementa uma hierarquia de três *templates* HTML definidos na estrutura da página. Primeiro utiliza o **template-accordion-item** para criar as categorias temáticas, depois usa **o template-resource-table** para estruturar a tabela de dados responsiva dentro de cada secção e, por fim, recorre ao **template-resource-row** para povoar as linhas individuais de cada recurso.

```
109  async function loadAllResources()
```

```
572  <template id="template-resource-row">
```

```
603  <template id="template-resource-table">
```

```
619  <template id="template-accordion-item">
```

A funcionalidade de pesquisa é assegurada pela função **applyFilters()**, que executa uma filtragem em tempo real no DOM. Esta função não se limita a ocultar linhas da tabela que não correspondam aos critérios de texto ou tipo, mas implementa também uma lógica de limpeza visual que oculta automaticamente categorias (acordeões) e secções de ano inteiras caso fiquem vazias após a filtragem, atualizando o contador de resultados instantaneamente.

```
178  function applyFilters()
```

Por fim, a experiência de visualização é gerida pela função **openPreview(button)**. Esta distingue automaticamente entre vídeos e documentos PDF, injetando o conteúdo no modal apropriado (**#videoPreviewModal** ou **#pdfPreviewModal**), permitindo ao aluno consultar o material sem nunca abandonar a página de catálogo.

```
294  function openPreview(button)
```

```
<div
  class="modal fade"
  id="videoPreviewModal"
  tabindex="-1"
  aria-labelledby="videoPreviewModalLabel"
  aria-hidden="true"
>
```

```
<div
  class="modal fade"
  id="pdfPreviewModal"
  tabindex="-1"
  aria-labelledby="pdfPreviewModalLabel"
  aria-hidden="true"
>
```

A gestão transversal dos favoritos é centralizada no ficheiro **favorites.js**, como já referido. Este módulo garante a integridade e privacidade dos dados através da função **getFavoritesKey()**, que gera uma chave de armazenamento única baseada no ID da sessão do utilizador, assegurando que múltiplos utilizadores no mesmo dispositivo mantêm listas de favoritos independentes no **localStorage**.

```
1  function getFavoritesKey()
```

Relativamente à interação, a função **toggleFavorite(resource, button)** atua como o controlador principal. Ao ser acionada, verifica o estado atual do recurso e decide entre a adição e a remoção, invocando de imediato a função **updateFavoriteButton(button, isFav)**. Esta encarrega-se de fornecer *feedback* visual instantâneo, alternando as classes CSS do ícone (entre **bi-heart** e **bi-heart-fill**), permitindo uma experiência de utilização fluida sem necessidade de recarregamentos de página. Estas duas funções são chamadas através da função **toggleFavoriteResource(button)** em **account.js** com o evento **onclick**.

```
75   function toggleFavorite(resource, button)
90   function updateFavoriteButton(button, isFav)
```

```
152  function toggleFavoriteResource(button)
```

```
<td>
  <button
    class="btn btn-sm btn-outline-danger favorite-btn"
    onclick="toggleFavoriteResource(this)"
    title="Adicionar aos favoritos"
  >
    <i class="bi bi-heart"></i>
  </button>
</td>
```

Página "contact.html"

A secção de comunicação e suporte é dinamizada pelo ficheiro **contact.js**. A integridade dos dados inseridos pelo utilizador é assegurada por um sistema de validação em tempo real, onde funções como **onInputEmail(input)** (que aplica expressões regulares) e **onInputMessage(input)** são acionadas pelos eventos **oninput** definidos no HTML, fornecendo *feedback* visual imediato através da manipulação das classes de estilo (**is-valid** / **is-invalid**) geridas pela função auxiliar **showFieldError(fieldId)**.

```
88   function onInputName(input)
92   function onInputEmail(input)
96   function onChangeSubject(input)
100  function onInputMessage(input)
43   function showFieldError(fieldId, errorMessage)
```

```
<input
  type="text"
  name="name"
  id="name-field"
  class="form-control"
  required=""
  oninput="onInputName(this)"
/>
```

O processamento do formulário é centralizado na função **handleFormSubmit()**. Esta função não só previne o envio padrão do formulário, como orquestra a integração com o serviço externo **EmailJS** (inicializado no arranque do *script*) para o envio efetivo da mensagem via API.

```
105  function handleFormSubmit()
```

```
<form
  id="contact-form"
  data-aos="fade-up"
  data-aos-delay="200"
  onsubmit="handleFormSubmit(); return false;"
  onreset="onFormReset()"
>
```

O ciclo encerra-se com a exibição de um modal de sucesso **(#success-modal**) através da função **showSuccessModal()**, confirmando a operação ao utilizador.

```
194  function showSuccessModal()
```

```
<div class="modal" id="success-modal" tabindex="-1">
```

Página "resource.html"

A visualização detalhada de conteúdos é gerida pelo ficheiro **resource.js**. A inicialização da página depende da função **getResourceIdFromUrl()**, que extrai o identificador único presente no URL para localizar o recurso correto do ficheiro JSON. Com base nesses dados, a função **updateIframes(resource)** configura dinamicamente os elementos **<iframe>** do HTML, alternando a visualização entre a ficha de trabalho e as respetivas soluções conforme a interação do utilizador com as abas de navegação.

```
17   function getResourceIdFromUrl()
44   function updateIframes(resource)
```

```
<iframe
  id="worksheet-iframe"
  src=""
  title="Ficha de trabalho"
  loading="lazy"
  allowfullscreen
></iframe>
```

Além da consulta de material, este módulo implementa um sistema de discussão local através da função **saveMessage(resourceId, message)**. Esta funcionalidade permite aos alunos publicar dúvidas ou comentários, que são persistidos no **localStorage** com uma chave associada ao ID do recurso

(**getChatKey(resourceId)**). A exibição das mensagens é feita pela função **displayMessages(resourceId)**, que injeta todas as mensagens no DOM e calcula o tempo decorrido (ex: 'há 5 min') através da função auxiliar **formatTimeAgo(timestamp)**, simulando uma experiência de chat em tempo real.

```
90   function saveMessage(resourceId, message)
74   function getChatKey(resourceId)
96   function formatTimeAgo(timestamp)
```

```
123   function displayMessages(resourceId)
367       <div class="chat-thread" id="chat-thread"></div>
```

Alimentação de Informação

1. **Obtenção de Dados**

A comunicação entre a aplicação e as fontes de dados estáticas (JSON) é mediada exclusivamente pelo ficheiro **utils.js**. Foi implementada a função genérica **loadData(file)**, como já referido.

Esta função opera de forma assíncrona (**async/await**). Ao invocar **await response.json()**, a função converte automaticamente o fluxo de texto recebido em objetos JavaScript utilizáveis, permitindo que qualquer módulo do sistema (seja o catálogo ou a área pessoal) solicite dados apenas fornecendo o caminho do ficheiro, promovendo a reutilização de código e a modularidade.

2. **Estrutura e Tipologia dos Dados**

A integridade do sistema depende de uma estruturação rigorosa dos ficheiros JSON, onde são utilizados diferentes tipos de dados primitivos e compostos para modelar a informação.

a. **Dados de Utilizador (users.json)**
   i. **Lista – Raiz (cada utilizador é um Objeto)**
      - id: Inteiro - identificador único do utilizador
      - username: String – nome de login
      - password: String – palavra-passe
      - name: String – nome completo
      - email: String – email do utilizado
      - profilePicture: String – caminho/URL da imagem de perfil
      - year: Inteiro – ano de escolaridade
      - school: String – escola
      - course: String – curso/área
      - goal: String – objetivo do aluno
      - activePlan: String – nome do plano ativo
      - **Lista – skills (cada skill é uma String)**
      - **Lista – studyPlan (cada entrada do plano é um Objeto)**
        - day: String – dia da semana
        - theme: String – tema da sessão
        - objective: String – objetivo/tarefa a cumprir
        - time: Inteiro – duração (em minutos)

b. **Catálogo de Recursos (recursos-10.json, recursos-11.json, recursos-12.json)**
   i. **Lista – Raiz (cada categoria é um Objeto)**
      - id: Inteiro – identificador único da categoria
      - category: String – nome da categoria (ex: “Estatística”, “Funções”)
      - description: String – descrição/resumo da categoria
      - **Lista – resources (cada recurso é um Objeto)**
        - id: Inteiro – identificador único do recurso
        - title: String – título do recurso
        - type: String – tipo de recurso

- url: String – link para abrir o recurso (normalmente existe quando o url é externo)
- worksheetUrl: String – URL do PDF da ficha (normalmente existe quando type é "Ficha de trabalho")
- solutionsUrl: String – URL do PDF das soluções (normalmente existe quando type é "Ficha de trabalho")

c. Configuração da Homepage (index-data.json)

i. Objeto – Raiz

- **Lista – features (cada destaque é um Objeto)**
  - icon: String – nome/classe do ícone
  - title: String – título do destaque
  - description: String – descrição do destaque.
- **Lista – featuredResources (cada recurso é um Objeto)**
  - image: String – caminho/URL da imagem
  - imageAlt: String – texto alternativo da imagem
  - badge: String – texto do badge
  - badgeClass: String – classe CSS do badge
  - title: String – título do recurso em destaque
  - description: String – descrição do recurso em destaque
  - **Lista – highlights (cada ponto-chave é uma String)**
  - link: String – link do recurso no *site*
  - buttonText: String – texto do botão
- **Lista - testimonials (cada testemunho é um Objeto)**
  - image: String – caminho/URL da imagem
  - imageAlt: String – texto alternativo da imagem
  - quote: String – citação do testemunho
  - author: String – autor do testemunho
  - year: Inteiro – ano do aluno

Organização dos ficheiros

O projeto, apesar de sabermos que ainda há alguma repetição de código que poderia ser minimizada, apresenta uma organização de ficheiros modular e bem estruturada, seguindo as boas práticas do desenvolvimento *web*, com uma clara separação entre páginas HTML, estilos CSS, *scripts* JavaScript e dados em formato JSON. Cada ficheiro JavaScript assume uma responsabilidade específica, os estilos encontram-se organizados entre regras globais e estilos próprios de cada página, e os dados estão isolados em ficheiros JSON, o que facilita a manutenção e a evolução do projeto. Esta estrutura contribui igualmente para a *performance* da aplicação, por exemplo, permitindo que os ficheiros JavaScript permaneçam em *cache* no *browser* por mais tempo quando estão separados, e garante a facilidade de mudanças sem necessidade de reestruturações profundas.

Considerações Finais

Ao longo do desenvolvimento, optou-se consistentemente por opções que estão em conformidade com o que foi abordado na unidade curricular. Por exemplo, uma sintaxe explícita em detrimento da implícita, refletida nas comparações diretas, como **if (session != null)**, prescindindo da avaliação booleana direta de variáveis que a linguagem permite, como, por exemplo, apenas **if (session)**. Seguindo a mesma filosofia, abdicou-se deliberadamente do operador de negação unário **! (NOT)**, privilegiando sempre comparações explícitas, como **if (isLoggedIn() == false)**. Em contrapartida, recorreu-se extensivamente aos operadores binários **&& (AND)** e **|| (OR)** para a composição de condições lógicas mais complexas,

essenciais, por exemplo, nas validações múltiplas dos formulários de contacto e de *login*. As estruturas condicionais **if...else** constituem a base lógica de praticamente todos os módulos funcionais, sendo utilizadas, por exemplo, na verificação de autenticação em **auth.js**, na validação de formulários em **contact.js** e **login.js**, na exibição condicional de conteúdos em **catalog.js** e **resource.js**, e ainda na gestão de temas em **theme.js**, onde o encadeamento de condições permite devolver *feedback* preciso ao utilizador.

Relativamente às estruturas de iteração, a natureza determinística dos dados em formato JSON favoreceu o uso exclusivo de ciclos **for** clássicos para percorrer listas. Este padrão foi aplicado de forma transversal, nomeadamente na exibição de categorias de recursos em **catalog.js**, na exibição de favoritos em **account.js**, na geração de mensagens de chat em **resource.js** e na população de cartões na página inicial em **index.js**. Não foi necessário recorrer a ciclos **while** ou **do...while**, uma vez que o número de iterações era sempre conhecido através da propriedade **length** das listas, evitando assim condições de paragem mais complexas e o risco de ciclos infinitos.

Um dos pilares arquiteturais do projeto reside na programação assíncrona, concretizada através da utilização de funções **async/await** para o carregamento de dados JSON via **Fetch API**. A função utilitária **loadData(file)**, definida em **utils.js**, encapsula toda a lógica de pedido e conversão de dados, sendo reutilizada de forma central por módulos como **catalog.js**, **account.js**, **index.js** e **resource.js**. Destaca-se igualmente o uso da API **URLSearchParams** para a navegação entre páginas, especificamente em **resource.js**, onde a extração do identificador do recurso através de **window.location.search** permite a passagem de parâmetros. Este método foi adotado após uma pesquisa realizada por nós, através deste *link*, uma vez que existiam alternativas para esse controlo, como, mais uma vez, o uso de **localStorage** para armazenar o identificador do recurso a abrir.

A persistência de dados é assegurada de forma extensiva através do **localStorage**, que mantém o estado da aplicação entre sessões e páginas. Este mecanismo armazena, por exemplo, a sessão do utilizador (**mathpath-session**), a lista de recursos favoritos (**mathpath-favorites**), o histórico de cálculos da calculadora (**mathpath-calculator-history**) e a preferência de tema visual (**mathpath-theme**). Todos estes dados são serializados e desserializados utilizando **JSON.stringify()** e **JSON.parse()**, garantindo a integridade de objetos complexos.

A organização do código privilegiou a criação de funções reutilizáveis, definidas no âmbito global. Esta abordagem promove a separação de responsabilidades, permitindo que funções utilitárias, como **setTextById(id, text)** ou **loadData(file)**, sejam invocadas em múltiplos contextos. Ao nível da manipulação do DOM, implementaram-se deliberadamente duas estratégias distintas, como a clonagem de *templates* através de **cloneNode()**, utilizada em **catalog.js** e **account.js** pela sua robustez, e a injeção direta de conteúdo via **innerHTML**, para demonstrar o domínio da manipulação de **strings HTML**.

Complementarmente, foram explorados diversos métodos nativos e APIs do navegador, como o método **split()** para manipulação de strings em **auth.js**, métodos matemáticos como **Math.round()** e **Math.floor()** na calculadora e na conversão de tempo, **parseInt()** para o tratamento de identificadores e a manipulação de classes CSS através de **classList**. Destaca-se ainda a utilização da API **window.matchMedia** no sistema de temas, permitindo verificar programaticamente a preferência de cor do sistema operativo.

A definição explícita do comportamento de interação diretamente nos elementos HTML foi uma opção aplicada de forma consistente ao longo do projeto, sempre que se pretendeu um controlo rigoroso sobre o fluxo de execução. Um exemplo disso encontra-se no formulário de contacto (**contact.html**), através de **onsubmit="handleFormSubmit(); return false;"**, onde o comportamento padrão de submissão (evitar o recarregamento da página) é intencionalmente suprimido para permitir a validação manual dos dados, o processamento assíncrono e a integração com serviços externos. Esta abordagem foi igualmente adotada noutros pontos da aplicação, como em interações baseadas em cliques, garantindo que ações críticas só são executadas quando todas as condições lógicas definidas são satisfeitas.

Importa ainda referir que os recursos educativos e conteúdos de apoio utilizados para compor os exemplos de materiais de estudo foram retirados do portal mat.absolutamente.net e do canal de YouTube Explicamat.

Por fim, salienta-se que a integração com serviços externos foi demonstrada através da biblioteca **EmailJS**, pesquisada por nós neste *link*, no formulário de contacto, enquanto a lógica herdada do *template* original (animações e comportamento em dispositivos móveis) foi isolada no ficheiro **main.js**, garantindo que

o desenvolvimento personalizado nos restantes módulos não comprometesse a estabilidade visual do *layout* base.

Conclusões

O desenvolvimento deste *website* permitiu aprofundar e consolidar os conhecimentos adquiridos ao longo da unidade curricular de Programação Web, evidenciando uma evolução clara relativamente ao trabalho anterior, que se encontrava limitado a HTML, CSS e Bootstrap. Nesta fase, a introdução extensiva de JavaScript possibilitou a transformação de uma base estática numa aplicação *web* dinâmica, interativa e orientada ao utilizador, reforçando a aplicação prática de conceitos fundamentais do desenvolvimento *web* moderno.

Ao longo do projeto, tornou-se evidente a importância de uma planificação cuidada da arquitetura desde as fases iniciais, garantindo uma organização modular e coerente entre páginas HTML, folhas de estilo, *scripts* JavaScript e dados em formato JSON. A separação clara de responsabilidades, aliada à reutilização de funções e componentes, revelou-se essencial para assegurar a manutenibilidade e a legibilidade do código, facilitando a evolução do sistema sem necessidade de reestruturações profundas.

A implementação de funcionalidades como autenticação de utilizadores, gestão de sessão, catálogo dinâmico de recursos, sistema de favoritos, calculadora com histórico persistente, alternância de temas e formulários com validação em tempo real demonstrou uma aplicação consistente dos conteúdos lecionados, nomeadamente a manipulação do DOM e a programação assíncrona (**Fetch API**). Estas funcionalidades permitiram enriquecer significativamente a experiência do utilizador, tornando a plataforma mais funcional e personalizada.

Em síntese, o projeto resultou numa aplicação *web* educativa completa, com uma base técnica sólida, uma organização estruturada e uma interface consistente e responsiva. O trabalho desenvolvido não só consolidou competências técnicas essenciais, como também reforçou boas práticas de desenvolvimento, constituindo uma base robusta para futuras extensões, como a integração com bases de dados reais, autenticação remota ou serviços externos adicionais, aproximando o projeto de um produto final plenamente funcional.

O website encontra-se disponível publicamente através do GitHub Pages, acessível no seguinte endereço: https://andreaoliveira9.github.io/Projeto-PW-2/.

Referências

https://mat.absolutamente.net/wp/
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
https://www.w3schools.com/js/js_htmldom_document.asp
https://getbootstrap.com/docs/5.3/getting-started/introduction/
https://www.youtube.com/watch?v=rufqTv6KeYI&t=1s
https://www.youtube.com/@Explicamat-1
https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams
https://www.emailjs.com/docs/tutorial/overview/