

Síntese das Atividades de Desenvolvimento no Projeto ***Registo de Tratamentos***

# Tarefas Técnicas Realizadas com Auxílio do Agente ***Cursor***

* Adição de dependências e configuração de ambiente: Foi adicionada a biblioteca cross-env às dependências de desenvolvimento para assegurar a correta definição de variáveis de ambiente em diferentes plataformas durante a inicialização da aplicação. Com o *cross-env* incluído no *package.json*, a execução de **pnpm install** passa a instalar automaticamente essa dependência, permitindo que comandos como **pnpm run start** funcionem em qualquer ambiente com as variáveis de ambiente apropriadas.
* Resolução de erros de build após atualizações de código: Vários erros de compilação foram resolvidos após a integração de alterações significativas (e.g., *pull requests* complexos). Destaca- se a correção de um erro ao importar o CSS do Swagger UI – passou-se a usar o pacote

**swagger-ui-dist**

adequado ( ) em substituição de um caminho inexistente no

**swagger-ui-**

, resolvendo o problema de módulo não encontrado . Adicionalmente, enfrentaram-se incompatibilidades de versões do React: determinadas bibliotecas de terceiros não suportavam a versão 19 do React, o que gerou avisos e falhas. A solução consistiu em fazer *downgrade* para React 18, alinhando-o com as expectativas dessas dependências . Após limpar e reinstalar módulos, e remover configurações obsoletas de Babel (permitindo o uso do compilador nativo SWC do Next.js), o projeto voltou a compilar com sucesso, eliminando os erros de build identificados .

**react**



1



2



3



4

* Refatoração de código e padronização de tipos em TypeScript: Conduziu-se um extenso

**any**

**@ts-**

esforço de refatoração para remover usos inadequados de e eliminar diretivas

dispersas no código. Foram criadas interfaces TypeScript centralizadas que modelam as entidades de domínio da aplicação (por exemplo, interfaces Tratamento, ArtigoTratamento, Especialidade, Serviço, entre outras) e estruturas de apoio como respostas de API padronizadas e resultados paginados . Essas definições foram concentradas no diretório comum de

**ignore**

**/types**



5



6

tipos ( ), promovendo a reutilização e consistência de tipos em todos os módulos. Com

**any**

as novas tipagens, substituíram-se variáveis antes declaradas como por tipos específicos

ou genéricos seguros, e removeu-se a necessidade de ignorar erros de compilação via

**@ts-**

**ignore**

. Esta padronização melhorou a segurança de tipos e a manutenibilidade do código, tornando-o mais fiável e fácil de compreender.

* Otimizações de desempenho (cache e paginação no cliente): Para mitigar problemas de lentidão no carregamento de grandes volumes de dados, implementaram-se otimizações tanto no lado do servidor quanto no cliente. Num caso concreto, o componente “Princípios Ativos – Informação Terapêutica” apresentava elevado tempo de carregamento devido a uma consulta complexa e ao retorno de milhares de registos de uma só vez. A solução adotada consistiu em introduzir um cache em memória no servidor para armazenar os resultados da primeira consulta dispendiosa, servindo respostas imediatas nos pedidos subsequentes . Em complemento, no *frontend* implementou-se paginação client-side: embora todos os dados sejam obtidos da API (agora de forma eficiente graças ao cache), o componente passa a exibir



7



8

apenas um subconjunto por página, evitando sobrecarregar o navegador ao renderizar muitos elementos de uma vez . Com esta abordagem combinada, resolveu-se o problema de desempenho sem necessidade de paginação a nível de base de dados, obtendo uma interface mais fluida e responsiva tanto no carregamento inicial quanto na navegação entre páginas de resultados.



8

* Melhorias de acessibilidade na interface: Foram identificados e corrigidos problemas de

**<select>**

acessibilidade em formulários, nomeadamente em elementos que careciam de

nome acessível. A ferramenta de auditoria acusava erros do tipo *“select element must have an*

**aria-label**

*accessible name”*. Para resolver, acrescentaram-se atributos descritivos e títulos

( **title** ) diretamente nos elementos nativos desses componentes, através da prop **inputProps** (no caso de componentes Material-UI). Com esta alteração, cada campo de seleção passou a possuir um nome acessível programaticamente e um texto de apoio visível em forma de tooltip, sem alterar a funcionalidade existente. Estas medidas alinham a aplicação com as diretrizes WCAG 2.1 de nível A, assegurando que os formulários são percepcionáveis por tecnologias de apoio e eliminando os erros de acessibilidade apontados pelas ferramentas de teste .



9

* Testes funcionais e validação visual de componentes (ex.: Princípios Ativos): Procedeu-se à criação e melhoria de testes, assim como à verificação manual do comportamento de componentes críticos. No caso do componente Princípios Ativos, após as otimizações de desempenho implementadas, testou-se o fluxo completo para garantir a correta apresentação das informações terapêuticas. Por exemplo, consultaram-se entradas reais na base de dados (via queries de teste) e verificou-se na interface que, ao expandir o item de um princípio ativo específico (e.g. Adenosina), os dados detalhados antes ausentes passaram a ser exibidos quando disponíveis, ou então surge uma mensagem adequada indicando a sua indisponibilidade . Além dos testes funcionais (como os unitários e de integração mencionados adiante), validou-se visualmente a interface reformulada, garantindo que elementos adicionados (como indicadores de carregamento, mensagens de “nenhum resultado”, paginação) estão corretamente alinhados e realçam a boa experiência do utilizador após as mudanças.



10

# Casos de Integração Técnica com a Infraestrutura Hospitalar

* Exportação de dados para o sistema SONHO: Implementou-se a integração de exportação de registos de tratamento para o sistema hospitalar SONHO, o que implicou inserir dados em tabelas específicas de interface. Durante o desenvolvimento, verificaram-se dificuldades iniciais: por um lado, uma comparação lógica falhava porque um campo indicador de exportação (e.g. **EXPORTADO\_SONHO** ) vinha da base de dados como string em vez de número, impedindo a condição **=== 0** de funcionar corretamente. Por outro lado, um identificador gerado ( **CDU\_IDEPENTIDADEACTOS** ) não estava presente em certos contextos, exigindo um mecanismo de *fallback*. A solução envolveu ajustar o código para comparar o valor exportado como string ( **String(row.EXPORTADO\_SONHO) === '0'** ) e usar um campo alternativo ( **CDU\_CSU\_ID** ) sempre que o identificador principal viesse indefinido. Para validar a abordagem sem afetar dados reais, a exportação foi primeiramente testada em tabelas de preparação com prefixo TESTE\_, passando depois para as tabelas definitivas em produção. Após as correções, confirmou-se que cada tratamento e os respetivos artigos associados são exportados corretamente: é inserido um registo de “ato” (tratamento) na tabela SONHO de faturação de atos, obtendo-se um ID que em seguida é utilizado para inserir os vários “gastos” (artigos/ medicamentos utilizados) na tabela de faturação de gastos. O processo foi dotado de logging detalhado em cada passo e tolerância a falhas individuais (um erro ao exportar um artigo não

interrompe a exportação dos restantes), tornando a integração mais robusta. Em suma, a exportação para o SONHO encontra-se agora funcional e fiável, com os registos a serem inseridos nas estruturas de destino adequadas e sinalizando o estado exportado apenas quando todo o fluxo é bem-sucedido.

* Correção do fluxo de alteração de palavra-passe e autenticação JWT: Melhorou-se o mecanismo de alteração de password dos utilizadores, garantindo a sua correção funcional e alinhamento com as políticas de autenticação baseadas em JWT. O endpoint responsável

**POST /api/utilizadores/changePassword**

( ) foi refatorado para: (1) validar os dados de

entrada (exigindo o preenchimento de todos os campos e impondo critérios de robustez à nova palavra-passe, como comprimento mínimo e presença de certos caracteres); (2) autenticar o pedido via JWT, extraindo o utilizador atualmente logado a partir do *token* de sessão

**getUserFromRequest**

(armazenado em cookie) – recorrendo a uma função auxiliar que

decifra o JWT e obtém os dados do utilizador na base de dados; (3) verificar se a password atual fornecida coincide com a registada (utilizando comparações de *hash* bcrypt); e (4) em caso de sucesso, atualizar a palavra-passe no Oracle (armazenando o hash da nova password) através da função **updateUserPassword** , bem como invalidar as sessões JWT ativas desse utilizador ( **invalidateUserSessions** ), forçando uma renovação de autenticação. Este fluxo corrigido assegura que apenas utilizadores autenticados e com credenciais válidas conseguem alterar a sua password, prevenindo também reutilização indevida de *tokens* antigos. Foram implementados testes automatizados cobrindo cenários críticos: por exemplo, um teste que simula um pedido válido de alteração espera uma resposta *HTTP 200 OK* e verifica que a nova palavra-passe foi efetivamente guardada com *hash* seguro; já cenários de erro, como uma password nova fraca (menos de 8 caracteres) ou a password atual incorreta, resultam respetivamente em códigos *400 Bad Request* e *401 Unauthorized*, com as mensagens de erro adequadas. Esses testes confirmam a robustez da solução e asseguram que o fluxo de autenticação JWT (incluindo a validade e renovação de tokens) permanece consistente após a mudança de credenciais. Em suma, a aplicação passou a oferecer um mecanismo seguro de alteração de palavra-passe, alinhado com as melhores práticas de autenticação ( JWT) e proteção de dados do utilizador.

# Melhorias na Interface do Utilizador (UI) e Experiência de Utilizador (UX)

* Reformulação da UI na página de Farmácia – “Prescrições Ativas”: A interface da secção de farmácia foi reorganizada para simplificar a visualização dos estados das prescrições e melhorar a usabilidade. Anteriormente, os estados “aguarda validação” e “validados” eram filtrados separadamente; agora, foram agrupados sob uma nova categoria unificada denominada “Prescrições Ativas”, exibida por omissão quando o utilizador acede à página . Esta mudança implicou remover os controlos individuais para “aguarda” e “validado”, passando a considerar ambos os estados como parte de um único filtro lógico e visual. A contagem apresentada ao lado do rótulo “Prescrições Ativas” reflete a soma de todos os registos nestes dois estados , fornecendo ao utilizador uma noção imediata do total de prescrições em curso. Adicionalmente, o estado “alta” foi renomeado para “Prescrições de Alta”, tornando a nomenclatura mais clara sobre tratar-se de prescrições associadas a alta de pacientes . Importa realçar que a lógica subjacente de filtragem por estado manteve-se inalterada – por exemplo, no nível de consulta aos dados backend ainda se distinguem os estados “aguarda” e “validado” –, mas a camada de apresentação/persistência de filtros foi adaptada para tratar “prescrições ativas” como um alias representativo dos dois estados. Com esta reformulação, a interface ficou mais limpa e orientada às necessidades do utilizador, evitando redundância de



11



12



13



14



15



16

opções. A UX beneficia de uma interação mais direta: ao marcar/desmarcar “Prescrições Ativas”, o utilizador filtra simultaneamente ambos os estados relevantes, e a aplicação indica esse filtro combinado de forma destacada (utilizando, por exemplo, rótulos coloridos com ícones junto às *checkboxes* correspondentes) . Em suma, a página de farmácia ganhou em simplicidade e clareza, alinhando-se com princípios de design centrado no utilizador e facilitando o acompanhamento das prescrições em diferentes etapas do fluxo clínico.



17



18

# Adoção de Boas Práticas no Projeto

* Centralização e consistência de tipos em TypeScript: Como mencionado, foi implementada uma rigorosa centralização das definições de tipos da aplicação. Todas as interfaces e tipos

**app/types**

comuns foram movidos para um diretório dedicado (tipicamente ), assegurando

que módulos distintos (frontend, API, componentes) utilizem as mesmas definições de dados. Por exemplo, a estrutura de um objeto Tratamento ou Paciente é agora descrita num único local e importada conforme necessário, evitando duplicação de definições e possíveis divergências. Esta prática melhora não só a consistência, mas também facilita futuras extensões do modelo de dados – quaisquer alterações de estrutura precisam ser feitas num único ficheiro

**any**

de tipos. Além disso, a eliminação de e a preferência por tipos específicos ou genéricos

seguros reforçaram a segurança em tempo de compilação, reduzindo erros em tempo de execução. Em termos académicos e de engenharia de software, esta abordagem vai ao encontro dos princípios de tipagem estática forte e de *DRY (Don't Repeat Yourself)*, contribuindo para um código-base mais robusto e autodocumentado.

* Melhorias no logging e capacidade de depuração: Introduziu-se um sistema de registo de logs mais completo e padronizado, crucial para monitorização e depuração, sobretudo em ambiente de produção. Foram definidas diferentes categorias de log (p.ex., níveis *debug*, *info*, *warn*, *error*) e estruturas uniformes para as mensagens, facilitando a filtragem e leitura dos registos de execução. Em diversas partes críticas do código adicionaram-se mensagens de

**console.log**

depuração ( ou utilizações de um objeto logger dedicado) que descrevem o

estado interno e as ações executadas – por exemplo, ao efetuar operações de filtragem na UI (indicando quais filtros estão ativos, quando são limpos, etc.) , ou ao realizar operações de base de dados (indicando sucesso/falha no fecho de conexões, parâmetros de queries, etc.). No novo fluxo de exportação de tratamentos, cada etapa (chamada à API externa, inserção em SONHO, conclusão ou erro) é logada com detalhe , permitindo rastrear facilmente o progresso e diagnosticar problemas. Também se identificaram e corrigiram pontos onde logs existentes podiam expor informações sensíveis ou não seguiam padrões – por exemplo, removendo passwords em claro nos ficheiros de configuração e evitando imprimir dados confidenciais em ambientes de produção. Com estas melhorias, a aplicação tornou-se mais transparente durante a execução: os desenvolvedores dispõem de informação granular para diagnosticar comportamentos anómalos, e a operação do sistema em produção beneficia de auditoria e troubleshooting simplificados. Em suma, a adoção de boas práticas de logging e debug aumentou a confiabilidade do sistema e a facilidade de manutenção, alinhando o projeto com padrões profissionais de engenharia de software.



19



20



21

Nota: Todas as atividades acima descritas foram conduzidas de forma a preservar a segurança institucional e a confidencialidade dos dados. Evitou-se expor qualquer informação sensível do hospital nos exemplos apresentados, limitando os detalhes ao necessário para ilustrar as melhorias técnicas e funcionais realizadas no âmbito deste projeto. Cada intervenção foi feita seguindo princípios de boas práticas e com validação rigorosa (via testes automáticos ou manuais), garantindo que a evolução do sistema ocorre de forma controlada e preparada para a utilização em ambiente real hospitalar.

cursor\_fixing\_dependency\_issues\_and\_bui.md



1 2 3 4

file://file-2Zr2pbij23eQutf5J7tpWt

cursor\_refactor\_type\_safety\_in\_typescri.md



5 6

file://file-Snug1TjdRfnQv19jDs8VE6

cursor\_an\_lise\_de\_lentid\_o\_no\_carregame.md



7 8

file://file-Ktu55kD4hXmYpfKcMcuVyn

cursor\_resolver\_problemas\_de\_acessibili.md



9

file://file-X9SNcmbYQDPAbKah6EXtNp

cursor\_teste\_do\_componente\_princ\_pios\_a.md



10

file://file-KrYvu3W9qi4ztzvvfwZ9jx

cursor\_atualiza\_o\_da\_ui\_da\_p\_gina\_de\_fa.md



11 12 13 14 15 16 17 18 19

file://file-Xmgep6NwnwVLjLHzoMn17Z

cursor\_compara\_o\_de\_routes\_ts\_diferen\_a.md



20 21

file://file-56Jgtqnvfr7FdYcXs7ZrQ6