

CRUD Dinâmico

Luís Oliveira
201707229

Miguel Romariz
201708809

Abstract—O objectivo deste projeto é o desenvolvimento de um sistema de informação que pode ser personalizado pelo utilizador, podendo este criar uma base de dados conforme as suas necessidades e editá-la. Pode ainda adicionar e remover dados da base de dados e fazer diferentes listagens dos mesmos.

Este relatório serve como um guia para implementação sugerida pelo grupo.

I. INTRODUÇÃO

Um sistema de informação normalmente permite inserir, ler, atualizar e apagar dados de uma base de dados. O nosso objectivo é elevar este conceito dando a possibilidade ao utilizador de alterar não só os dados mas também a estrutura da própria base de dados.

Para este efeito o projeto foi dividido em 3 partes: criação de um ficheiro YAML com a especificação da base de dados, geração da base de dados e da aplicação a partir do ficheiro criado e manipulação dos dados da base de dados.

Cada uma destas tarefas está explicada em detalhe nas secções seguintes.

II. GERAÇÃO DINÂMICA DAS PÁGINAS HTML

De modo a permitir a geração dinâmica das diferentes páginas da plataforma, existem templates de páginas HTML que são populados conforme as necessidades do utilizador.

Se o objectivo do utilizador for criar uma base de dados nova ou adicionar tabelas a uma já existente, a aplicação irá adicionar á pagina inicial os componentes da página de criação de tabelas.

```
function generateIndex2(){
  let file_content = `<input type="file" id="loadTables" multiple>+
  <button type="button" id="addTable">Add Table</button> <div id="createTables"></div>+
  <form action="/" method="post" id="form">+
  <input id="createButton" type="submit" value="Submit DB"></input></form>`
  generateHTMLFile("Create Table", file_content, "src/frontend/index.html")
}

//file generation
function generateHTMLFile(title, content, dest_dir) {
  let replacementDictionary = {
    title: title,
    content: content
  }
  generateFileFromTemplate(replacementDictionary, "src/generation/templates/pages/index.html", dest_dir)
}
```

Fig. 1. Código de criação da página de criar tabelas

Se o objectivo do utilizador for adicionar dados á base de dados, a aplicação irá ler as tabelas que existem na base de dados e cria uma rota para cada uma delas. Depois o utilizador pode escolher a tabela em que pretende inserir dados. Neste ponto se a tabela escolhida já conter dados estes são listados na página e o utilizador ainda tem a opção de adicionar novos dados á tabela actual. Para fazer isto apenas é necessário preencher o formulário e carregar no botão "Submit".

```
async function generateListingPage(table_rows, table_name, tables_model) {
  let file_content = `<ul>`
  for (let row in table_rows) {
    const field = table_rows[row]
    const id = field['id']
    let replacementDictionary = {
      href: `/${table_name}/${id}`,
      text: id
    }
    const tableListingHTMLElem = await generateListingHTMLElement(replacementDictionary,
      'listingelement.html')
    file_content += `<li>${tableListingHTMLElem}</li>`
    // < a href = "${table_name}/${id}" > ${ id }</a >
  }
  file_content += `</ul>`
  let form_content = await generateCreateFormHTML(table_name, tables_model)
  let replacementDictionary = {
    title: table_name + " listing",
    content: file_content,
    form: form_content
  }
  return generateHTMLFileContent(replacementDictionary, 'tableListing.html')
}
```

Fig. 2. Código de criação da página de listar tabelas

```
async function generateTableRowPage(row, table_name, tables_model) {
  let file_content = `<ul>`
  for (let field in row) {
    let value = row[field]
    value = addAnchorToForeignKeyValue(value, table_name, tables_model, field, row['id'])

    let replacementDictionary = {
      field: field,
      value: value
    }
    const tableListingHTMLElem = await generateListingHTMLElement(replacementDictionary,
      'tableRow.html')
    file_content += `<li>${tableListingHTMLElem}</li>`
  }
  file_content += `</ul>`
  file_content += await generateRelatedTablesHTML(tables_model, table_name, row.id)
  let form_content = await generateUpdateFormHTML(table_name, tables_model, row)
  let replacementDictionary = {
    title: table_name,
    content: file_content,
    form: form_content
  }
  return generateHTMLFileContent(replacementDictionary, 'tableRow.html')
}
```

Fig. 3. Código de criação da página de listar entradas de uma tabela

III. ESCRITA DO YAML

Ao correr o comando "node gen.js" no terminal irá ser gerada a página de criação de tabelas. Nesta página o utilizador tem duas opções: criar tabelas diretamente na plataforma ou carregar um ficheiro yaml que represente uma tabela previamente gerada.

Para o utilizador criar as suas próprias tabelas, tem de carregar no botão "Add Table", revelando uma caixa de input onde introduz o nome da tabela. Posteriormente, o utilizador pode adicionar tantos campos quantos forem necessários, carregando no botão "Add Field", tendo de especificar, para cada campo, o nome e tipo de dados. Quando o utilizador acaba de criar a tabela pode carregar no botão "Save", bloqueando-a, bastando repetir este processo para adicionar várias tabelas. No final, quando todas as tabelas estiverem criadas, o utilizador carrega no botão "Submit DB", gerando um ficheiro YAML, que representa a estrutura da base de dados definida pelo utilizador.

Se o utilizador optar por carregar um ficheiro YAML com a especificação da sua base de dados, apenas necessita de carregar no botão "Choose Files" e depois escolher o ficheiro. Assim, todas as tabelas presentes no ficheiro são mostradas na plataforma. Posteriormente o utilizador pode adicionar novas tabelas ou editar as tabelas já existentes.

Fig. 4. Página de criar tabelas

Para adicionar campos a uma tabela já existente o utilizador necessita de criar uma tabela nova com o nome da tabela a que pretende adicionar campos e especificá-los. Quando uma tabela é adicionada esta é adicionada á lista de *types* válidos. Isto permite que o utilizador crie referências a outras tabelas, atribuindo um tipo com o nome da tabela referida ao campo correspondente à chave estrangeira.

```
Doctor:
  Name: text
  Age: number
  Specialization: text;
Patient:
  Name: text
  Age: number
  Doctor: Doctor
  Birthdate: Text;
```

Fig. 5. Exemplo de tabelas num ficheiro YAML

IV. GERAÇÃO DA BASE DE DADOS

Após a criação de uma base de dados na aplicação "gen.js" o programa processa o ficheiro YAML, gerando os ficheiros constituintes de uma aplicação CRUD. Para tal, cria primeiro um ficheiro SQL que define a base de dados; depois, gera um ficheiro que define o modelo da base de dados e a resposta do servidor aos pedidos CRUD para todas as tabelas, incluindo os queries necessários e, no caso de operações de leitura, utiliza também a geração dinâmica de páginas HTML mencionada previamente.

```
get: (request, response) => {
  const id = parseInt(request.params.id)
  pool.query('SELECT * FROM doctor WHERE id = $1', [id], (error, results) => {
    if (error) {
      throw error
    }
    generateTableRowPage(results.rows[0], "doctor", tables).then( html => {
      response.send(html)
    })
  })
}
```

Fig. 6. Resposta gerada para pedidos de leitura de um elemento da base de dados

V. MANIPULAÇÃO DINÂMICA DO CONTEÚDO DA BASE DE DADOS

Correndo o comando "node app.js", as páginas HTML para editar os dados da base de dados são geradas conforme

as tabelas existentes. Depois o utilizador pode aceder a cada tabela e adicionar dados carregando no botão "+" e respondendo ao formulário. AO carregar no botão "Submit", os dados são enviados para a base de dados e as páginas HTML são atualizadas com a nova informação.

Fig. 7. Lista de tabelas

Table: doctor				
id	name	age	birthdate	specialty
1	Miguel Romariz	25	20/06/1998	MSF

Fig. 8. Lista de campos de uma tabela

Quando uma tabela está relacionada com outra, como, por exemplo, uma tabela Paciente que tenha um campo Doutor que referencia a tabela Doutor, uma lista com todos os dados da tabela referenciada aparece no campo respetivo.

Fig. 9. Criar campo com referência a outra tabela

VI. DIFICULDADES

O primeiro obstáculo encontrado foi o método de criação dinâmica das páginas HTML conforme as necessidades do utilizador. De modo a superar isto, cada página da nossa aplicação é gerada conforme o caso de uso atual: se for

necessário criar ou adicionar tabelas à base de dados, a aplicação substitui nos ficheiros HTML que servem como template as tags **content*, **form* e **title* pelo código HTML correto. O mesmo se aplica no caso do utilizador querer listar o conteúdo da base de dados.

Outra dificuldade encontrada foi no upload do ficheiro YAML e display das tabelas que se encontram no mesmo. Visto que a única maneira de ler o conteúdo do ficheiro no frontend é guardar tudo numa *String* e não era possível fazer o parse da mesma pelas keys (nome das tabelas), foi necessário adicionar um carácter especial (";"), de modo a delimitar cada tabela. Assim, ao fazer parse da *String* por ";" , cada objeto do *array* resultante é uma tabela.

De modo a que a geração da base de dados não se tornasse demasiado complexa, utilizaram-se apenas três tipos de dados (número, texto e booleano), e a remoção de tabelas com referências a outras ficou apenas com o comportamento "ON DELETE SET NULL". No entanto, devido a questões de tempo, não foi possível adicionar mais opções de definição de estrutura da base de dados à aplicação de geração.

VII. CONCLUSÃO

Apesar de ter havido várias dificuldades e contratempos ao longo do desenvolvimento e não tenha sido possível implementar a modificação do sistema em run-time, acreditamos que atingimos vários dos objetivos definidos no início do projeto. Achamos que o produto final integra técnicas aprendidas ao longo da Unidade Curricular e que em termos de funcionalidades, embora não tenhamos conseguido implementar todas as que queríamos, apresentamos uma alternativa às CRUDs tradicionais.