

UNIVERSIDADE DO MINHO
MESTRADO EM ENGENHARIA INFORMÁTICA



LABORATÓRIO EM ENGENHARIA INFORMÁTICA

Análise Química na Web

Repositório de Datasets

Autores:

Frederico Pinto - 73639

Sara Pereira - 73700

Rui Vieira - a74658

25 de Junho de 2019

Conteúdo

1	Introdução ao Repositório de Datasets	2
2	Requisitos e Funcionalidades	2
3	Arquitetura da Aplicação	2
4	Backend	3
4.1	Modelos de Dados	3
4.2	Controladores	4
4.3	Rotas	5
4.4	Rota de Utilizadores	5
4.5	Rota de Ficheiros	5
4.6	Rota de Administrador	6
5	Frontend	6
5.1	Views	6
5.2	Rotas	6
5.3	Aplicação	8
5.3.1	/	8
5.3.2	/register	8
5.3.3	/user	9
5.3.4	/dashboards	9
5.3.5	/dashboards/import	10
5.3.6	/file/:id	10
5.3.7	*	11
6	Conclusão	11
7	Referências	12

1 Introdução ao Repositório de Datasets

A investigação no ramo da química requer equipamento de elevado custo, que apesar da sua extrema utilidade, muitas vezes não possibilita a obtenção de dados legíveis ou com acesso em qualquer plataforma. No Departamento de Química da Universidade do Minho, tal como em muitas outras, os seus investigadores têm de trabalhar com equipamentos que não sofrem qualquer tipo de atualização de software há vários anos. No âmbito desta disciplina foi-nos proposto o desenvolvimento de uma aplicação web, capaz de suprimir carências resultantes do problema referido anteriormente.

O equipamento em questão permite a extração de dados apenas num formato *RAW*, ilegível sem conversão para um formato conhecido, como por exemplo *csv*.

Para entender as complexidades enfrentadas pelos investigadores, o ficheiro *RAW* corresponde a uma matriz tridimensional com 900 colunas e em média de 12 000 linhas, que era transcrita à mão pelos investigadores para obtenção de gráficos que permitiam a análise de resultados, tornando-se necessário, assim, o desenvolvimento de um software de tradução desse formato para um com o qual possam trabalhar nos seus computadores pessoais de uma forma simples e eficaz.

Neste relatório será descrito todo o desenvolvimento da aplicação proposta.

2 Requisitos e Funcionalidades

Após varias reuniões com nosso orientador de projeto e cliente ficou definido que a aplicação devia ter os seguintes requisitos:

- Conversão do ficheiro *RAW* para formato *mzML*
- Extração da matriz a partir do ficheiro *mzML*
- Importação de ficheiros *RAW*
- Exportação de ficheiros *csv* e *RAW*
- Remoção de ficheiros *RAW*
- Listagem dos ficheiros previamente inseridos
- Visualização gráfica dos resultados
- Persistência dos dados numa base de dados
- Gestão de utilizadores e armazenamento de datasets

Quanto à visualização dos resultados, esta é feita através de gráficos gerados pelo processamento da matriz. As linhas da matriz representam os *scans* efetuados, as colunas a massa e o valor de cada célula corresponde à intensidade de cada molécula.

3 Arquitetura da Aplicação

Para conseguirmos atingir os requisitos propostos de uma maneira mais eficiente e atual, decidimos separar a aplicação em dois servidores que vão comunicar com pedidos *http*. Um servidor *backend* que será responsável por guardar e processar os dados que a aplicação tem que lidar, ficheiros e utilizadores. E outro servidor de *frontend* que correrá do lado do cliente e é responsável por gerir o que o cliente quer fazer na aplicação, comunicando o desejado ao *backend* para obter informação.

Para construir ambos os servidores utilizamos *NodeJS* com auxílio de várias *frameworks*, usando o *Express* para construir o *backend* e o *VueJS* para construir o *frontend*.

O servidor de *backend* depende de várias ferramentas externas para funcionar, utiliza o *mongoDB* como base de dados para persistência dos dados e o *Docker* para correr um *container* que faz a conversão de um ficheiro *RAW* para *mzML*.

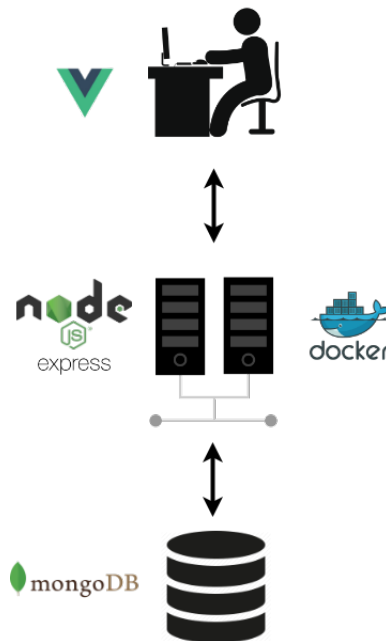


Figura 1: Arquitetura da Aplicação.

4 Backend

O *backend* é um servidor que é responsável pelo armazenamento e processamento de informação, aqui está presente toda a informação sobre os ficheiros e utilizadores. Vamos de seguida mostrar toda a nossa ideologia na resolução dos problemas que surgiram na construção deste servidor, começando primeiro por especificar os modelos por nós criados para retratar os dados que estamos a tratar.

4.1 Modelos de Dados

Como dados temos os ficheiros e os utilizadores, logo faz total sentido criar um modelo que especifique cada um, guardando informações relevantes que, de alguma maneira nos facilitem na resolução dos requisitos propostos.

Listing 1: file.js.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema

var FileSchema = new Schema({
  name: String,
  dateInput: {type: Date, default: Date.now},
  dateTest: Date,
  description: String,
  nScans: Number,
  idUser: String,
  sumIntensities: [mongoose.Decimal128],
  sumIntensitiesPerMass: [mongoose.Decimal128]
}, {
  versionKey: false
})
module.exports = mongoose.model('File', FileSchema, 'files')
```

Como mostrado acima, no modelo do ficheiro decidimos guardar para além dos parâmetros normais, o *nScans*, o *sumIntensities* e *sumIntensitiesPerMass* que têm por objetivo nos ajudar,

guardando os valores necessários para devolver de uma maneira mais eficiente quando os gráficos resultantes do ficheiro são pedidos.

Se seguida, apresentamos o modelo do utilizador:

Listing 2: user.js.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema

var UserSchema = new Schema({
  name:{ type:String , required:true },
  email:{ type:String , required:true },
  password:{ type:String , required:true },
  userType:{ type:Number, required: true }
}, {
  versionKey: false
})

module.exports = mongoose.model('User', UserSchema, 'users')
```

Aqui para além de guardarmos o nome, email e password guardamos também o *userType* que corresponde ao tipo de utilizador será, se tiver o valor *2* é um utilizador normal, caso seja *1*, é um utilizador do tipo administrador em que terá acesso a mais rotas fornecidas pelo *backend*.

4.2 Controladores

De seguida, iremos falar sobre os controladores que operam sobre cada um dos modelos que acabamos de especificar acima, estes operam sobre o *MongoDB* que contém os registos guardados.

Para lidar com os ficheiros definimos os seguintes controladores:

- **allFiles(id)** - retorna todos os ficheiros de determinado utilizador
- **adminAllFiles** - retorna todos os ficheiros disponíveis;
- **addFile** - adiciona um ficheiro à base de dados;
- **getIntMass(id)** - retorna os valores para construir o gráfico de intensidade por massa;
- **getSumIntensity(id)** - retorna os valores para construir o gráfico de intensidade por scan;
- **getInfoFile(id)** - retorna todos os valores exceto o dos gráficos para determinado ficheiro;
- **deleteFile(id)** - elimina um ficheiro da base de dados;

Já para lidar com o modelo de utilizadores, construímos os controladores:

- **allUsers** - retorna todos os utilizadores disponíveis;
- **getUser(email)** - retorna o utilizador com determinado email;
- **getUserInfo(id)** - retorna toda a informação com dado id, exceto tipo de utilizador e password;
- **deleteUser(id)** - elimina um utilizador da base de dados;
- **addUser(user)** - adiciona um utilizador à base de dados;
- **changeUser(id, name, email)** - altera o nome e email de um utilizador;

4.3 Rotas

Neste servidor de *backend* possuímos três rotas possíveis, sendo uma para tratar de pedidos sobre ficheiros, outra para lidar com os pedidos sobre utilizadores e por fim uma destinada apenas a pedidos sobre administração. Contudo, para protegermos essas rotas de acessos não desejados, implementamos autenticação por *token*, usando o *JSON Web Token*. Sendo assim, quando um utilizador efetua o *login* é fornecido um *token* de acesso que terá que vir no cabeçalho de qualquer outro pedido *http* para qualquer outra rota fornecida pelo servidor.

A biblioteca *JSON Web Token* permite-nos enviar informação no *payload* do *token* de acesso, então achamos por bem enviar o *_id* do *mongoDB* referente ao utilizador e o tipo de utilizador que este é. Sendo assim, protegemos todas as rotas criando um função, *VerifyToken*, que é chamada a cada pedido recebido pelo servidor, que para além de verificar a validade do *token*, retira também esses valores do *payload* para o podermos utilizar na resposta ao pedido. Para além disso, é implementada uma outra função para proteger as rotas do administrador, *verifyAdmin*, que para além de verificar o *token*, verifica também se o tipo de utilizador que efetuou o pedido é *1*, ou seja, um administrador.

Iremos de seguida explicitar as rotas suportadas pelo servidor *backend* criado.

4.4 Rota de Utilizadores

- **POST /users/login** - efetuar *login* de um utilizador fornecendo email e password, se for efetuado com sucesso um *token* de acesso é fornecido como descrito acima;
- **POST /users/register** - registar um utilizador no sistema para isso é necessário inserir nome, email e password;
- **GET /users/** - retorna informação sobre o utilizador cujo *id* vai no *token* de acesso;
- **POST /users/update** - recebe o nome e o email e atualiza os dados do utilizador cujo *id* vai no *token* de acesso;

4.5 Rota de Ficheiros

- **GET /file/graph/sumMass?file=ID** - devolve os valores do gráfico do somatório das intensidades por massa;
- **GET /file/graph/sumIntensity?file=ID** - devolve os valores do gráfico do somatório das intensidades por scan;
- **GET /file/info?file=ID** - devolve informações sobre um ficheiro menos os valores dos gráficos;
- **DELETE /file/delete/:id/:name** - elimina determinado ficheiro do servidor, removendo-o da base de dados e os seus ficheiros do *system file*.
- **POST /file/import** - insere um ficheiro na aplicação, para isso converte o ficheiro *RAW* em *mzML* utilizando um *container* no *Docker*. Através do ficheiro *mzML* conseguimos obter uma matriz que representa a análise efetuada pela máquina, com ela conseguimos calcular os valores para os gráficos necessários como requisito, por fim, é gerado o *csv* e guardada toda informação calculada e obtida no *mongoDB*, para além disso o ficheiro *mzML* que é 3 vezes maior que o *RAW* é eliminado;
- **GET /file/csv/:id** - devolve o *csv* para download;
- **GET /file/raw/:id** - devolve o *RAW* para download;
- **GET /file/all** - devolve informação sobre todos os ficheiros, excluindo os valores dos gráficos;
- **GET /file/plot/:id** - devolve toda a informação para representar um *plot 3D* da análise efetuada, para isso, tem que devolver a matriz toda que se encontra guardada fisicamente no *file system* no ficheiro *csv*;

4.6 Rota de Administrador

- *GET* /**admin/users** - Devolve todos os utilizadores da aplicação;
- *DELETE* /**admin/users/:id** - Elimina um utilizador da aplicação;
- *GET* /**admin/files** - Devolve todos os ficheiros presentes na aplicação contendo toda a sua informação, exceto os valores dos gráficos.

5 Frontend

Como dito anteriormente, utilizamos o Vue como *framework* para desenvolver este servidor. Para além disso utilizamos ferramentas para o Vue como o *Vuetify* que nos ajudou a construir o *html* necessário para os *templates* das *views* e dos componentes.

Tal como o *backend*, também o *frontend* possui rotas. Cada uma dessas rotas corresponde a uma *view* que possui vários componentes, cada componente é responsável por mostrar ou não, determinada informação sobre os ficheiros/utilizadores que o *backend* guarda, que é obtida através de pedidos ao mesmo.

Deste modo, para aceder à maioria das rotas no *backend* é necessário incorporar o *token* no pedido *http*, para tal o servidor de *frontend* tem que ser capaz de guardar o *token* de acesso referente ao utilizador para assim conseguir fazer os pedidos. Então, para resolver esse problema, utilizamos o *Vuex*, esta ferramenta para o *Vue*, permite-nos guardar um estado que é partilhado por todos os componentes da aplicação. Sendo esse estado o *token*, podemos atualizar o seu valor na rota de *login*, removê-lo na rota de *logout* e verificar o seu valor quando quisermos. Para além disso, a presença e validade desse *token* permite-nos proteger as rotas aqui no servidor de *frontend*, sendo impossível aceder a uma rota caso o *token* de acesso não exista ou caso, o *backend* confirme a invalidade do mesmo.

É de salientar também, que o *token* de acesso é guardado fisicamente no *LocalStorage* permitindo assim a persistência do mesmo, para não o perder caso, caso se feche a janela de comunicação do utilizador com o servidor de *frontend*.

5.1 Views

- **User** - View que permite alterar informação sobre o utilizador que está com *login* efetuado;
- **File** - View que permite visualizar os resultados da análise de cada ficheiro;
- **Dashboards** - View que mostra a lista de ficheiros de cada utilizador;
- **Home** - View que mostra a página de *Login*;
- **Register** - View que contém os componentes para mostrar a página de registo;
- **FileImport** - View que contém o formulário para importar um ficheiro;
- **NotFound** - View que mostra a informação de erro;

5.2 Rotas

De maneira a estruturar a comunicação entre a interface e o *backend* foram criadas algumas rotas, que dirigem o cliente para diferentes páginas. Cada uma contém diversos componentes para recriar os pedidos efetuados pelo utilizador.

- / - encaminha o utilizador para a página inicial da aplicação onde o mesmo pode efetuar o *login*;
- /**register** - encaminha o utilizador para a página de registo na aplicação;
- /**user** - encaminha o utilizador para a página que contém o componente de alteração da sua informação (nome e email);

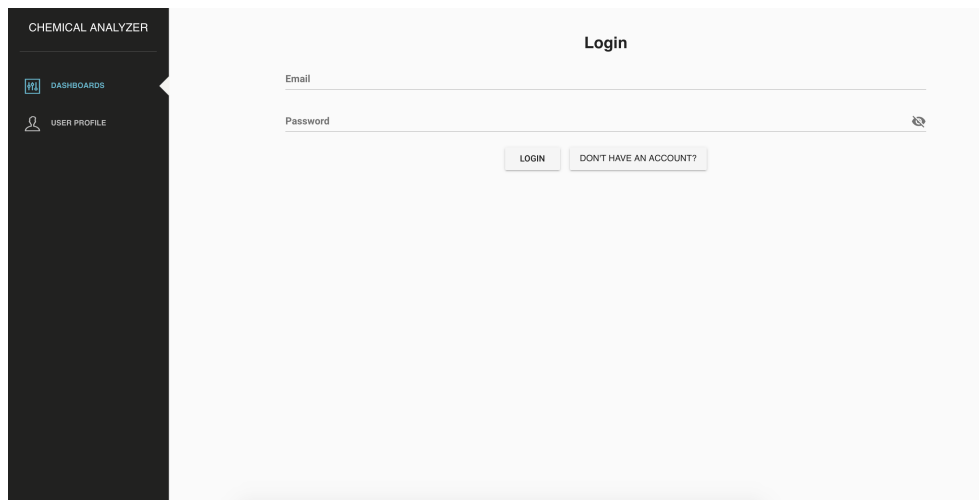
- **/dashboards** - encaminha o utilizador para a página que contém o componente de listagem de ficheiros importados, assim como o botão de importação dos ficheiros;
- **/dashboards/import** - encaminha o utilizador para a página que contém o componente que permite a importação dos ficheiros **RAW**;
- **/file/:id** - encaminha o utilizador para a página onde é efetuada a visualização gráfica do ficheiro com o respetivo **:id**;
- ***** - qualquer outra rota para além das anteriores serão encaminhadas para uma página de erro;

5.3 Aplicação

É de salientar que para além de todos os componentes e *views* implementadas e toda a lógica por trás disso, tivemos também cuidado com a estética permitindo ao utilizador um bom uso da aplicação. Para tal, tivemos vários cuidados como colocar avisos caso algum erro de pedido *backend-frontend* tenha acontecido em algumas *views*. Os gráficos que implementamos possuem *zoom* o que é uma grande vantagem para visualizar a informação do ficheiro.

De seguida apresentamos as *views* presentes no servidor *frontend*.

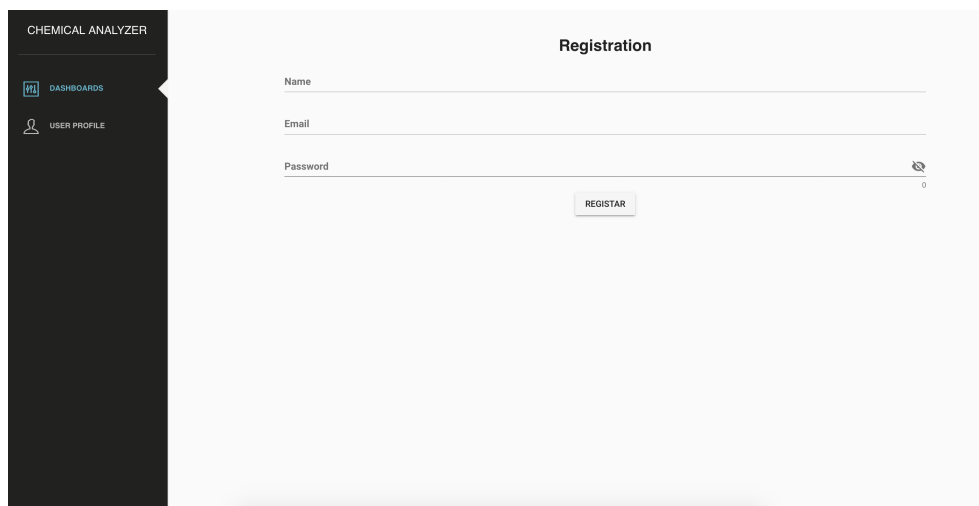
5.3.1 /



The screenshot shows the 'Login' view of the 'CHEMICAL ANALYZER' application. On the left is a dark sidebar with the title 'CHEMICAL ANALYZER' and two menu items: 'DASHBOARDS' (with a bar chart icon) and 'USER PROFILE' (with a person icon). The main content area is light gray and titled 'Login'. It contains two input fields: 'Email' and 'Password'. The 'Password' field has a toggle icon on the right. Below the fields are two buttons: 'LOGIN' and 'DON'T HAVE AN ACCOUNT?'.

Figura 2: View de *Login*.

5.3.2 /register



The screenshot shows the 'Registration' view of the 'CHEMICAL ANALYZER' application. The sidebar is identical to the previous view. The main content area is light gray and titled 'Registration'. It contains three input fields: 'Name', 'Email', and 'Password'. The 'Password' field has a toggle icon on the right. Below the fields is a single button labeled 'REGISTAR'.

Figura 3: View de *Register*.

5.3.3 /user

CHEMICAL ANALYZER

DASHBOARDS

USER PROFILE

Your Information

Name
camal

Email Address
camal

✓ SAVE CHANGES LOGOUT

Figura 4: View de *User*.

5.3.4 /dashboards

CHEMICAL ANALYZER

DASHBOARDS

USER PROFILE

Search File

Imported Files

Name	Test Date	Description	Input Date
A1F3P1R1_4_POS_BP	2019-06-02	Veremos se está bem!	2019-06-24
A2F1P1R1_4_POS_PTEA	2019-06-20	Quartooo	2019-06-24
A2F1P2R2_2_POS	2019-06-01	Vamos lá tentar fazer mais um!	2019-06-24
A2F2P1R1_2_POS_PTEA	2019-06-17	A certa!	2019-06-25
A2F2P1R1_5_POS_PTEA	2019-06-12	segundo no cp2	2019-06-25

Rows per page: 5 1-5 of 6

+

Figura 5: View de *Dashboards*.

5.3.5 /dashboards/import

The screenshot shows the 'Import RAW File' form in the Chemical Analyzer dashboard. The sidebar on the left contains the 'CHEMICAL ANALYZER' title and navigation links for 'DASHBOARDS' and 'USER PROFILE'. The main form area has a title bar 'Import RAW File' and two input fields: 'Description' and 'Exam Date'. Below these is a file selection area with a button 'Explorar...' and the text 'Nenhum ficheiro selecionado.' At the bottom right of the form is a blue button with a plus icon and the text 'IMPORT FILE'.

Figura 6: View de *FileImport*.

5.3.6 /file/:id

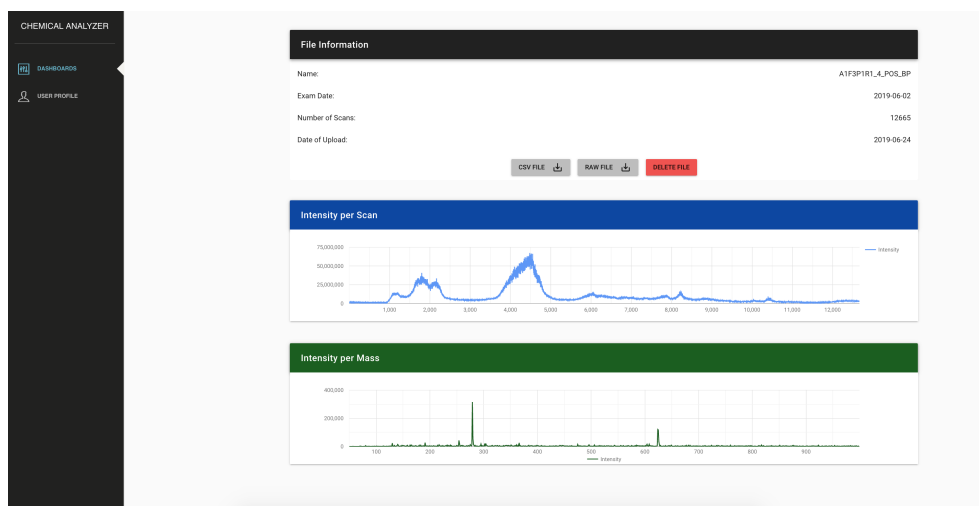


Figura 7: View de *File*.

5.3.7 *

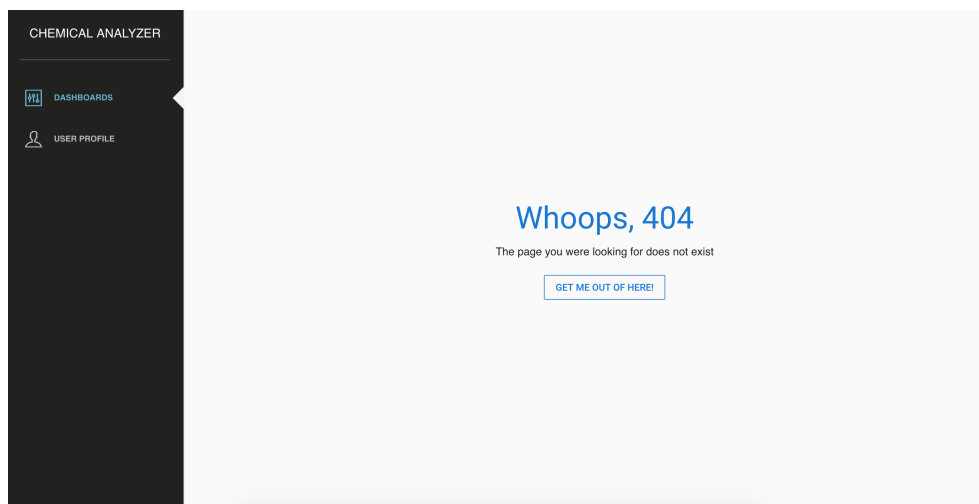


Figura 8: View de *NotFound*.

6 Conclusão

A execução deste trabalho, foi de extrema importância para nós, pois permitiu a exponenciação do nosso conhecimento em desenvolvimento *web*, introduzindo-nos a novas ferramentas e a diferentes problemas neste contexto.

No entanto, não foi possível a completa realização deste projeto na medida em que foi feita a implementação de rotas, no *backend*, para administração e fornecimento dos dados para a apresentação do *plot 3D*. Contudo, devido ao grande volume de dados, ainda não conseguimos desenvolver esta funcionalidade no servidor *frontend*.

7 Referências

Link do repositório que contém o código - <https://github.com/129Camal/LEI-19>