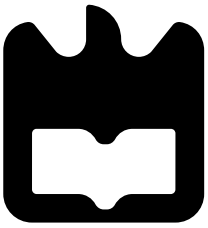
Universidade de Aveiro

UAluno

Diogo Ferreira, Luís Leira, Miguel Silva



1.0

UAluno

Departamento de Electrónica, Telecomunicações e Informática

Engenharia de Dados e Conhecimento

Diogo Ferreira, Luís Leira, Miguel Silva
76504, 76514, 76450
diogodanielsoaresferreira@ua.pt, luisleira@ua.pt, maps@ua.pt

27 de Outubro de 2017

Conteúdo

Capítulo 1

Introdução

O relatório descreve uma aplicação web efetuada no âmbito da unidade curricular de Engenharia de Dados e Conhecimento[1]. Neste projeto tivemos como objetivo construir uma aplicação web com recurso às tecnologias apresentadas na cadeira, com principal foco em *Django* e *XML*.

A ideia para o projeto teve origem na necessidade de reunir diversos dados úteis para um aluno da Universidade de Aveiro numa só plataforma online, tendo como inspiração a aplicação mobile existente *UAMobile*. Assim, pretende mostrar informações tais como a previsão de uma semana da meteorologia na cidade de Aveiro, as ementas da próxima semana das cantinas da Universidade de Aveiro e as notícias publicadas pelo Jornal Online da Universidade de Aveiro. Também foi incluída a possibilidade de inserir e manipular dados relativamente às cadeiras e respetivas notas de cada utilizador, tal como a comunicação entre utilizadores da plataforma através de um chat comum. Para atingir esse objetivo, foi utilizado o sistema de autenticação do *Django*.

O presente relatório visa demonstrar como foi desenvolvida a aplicação, referindo como foram recolhidos os dados e como foram aplicadas as várias funcionalidades, bem como as principais dificuldades na construção da aplicação.

Capítulo 2

Estrutura do projeto

Neste capítulo é descrita a estrutura geral do projeto com a indicação da localização dos ficheiros e pastas mais importantes.

No diretório do projeto podemos encontrar duas pastas relevantes. A pasta *UAluno* contém informação relativamente ao projeto e à sua configuração. Dentro da pasta *app* temos acesso a vários ficheiros importantes.

- O ficheiro *BaseXClient* é a interface de comunicação com o motor de base de dados utilizado na aplicação.
- O ficheiro *data_sources* tem as funções necessárias para o acesso às fontes de dados utilizadas.
- O ficheiro *data_logic* contém a normalização dos dados recebidos das fontes em árvores XML para serem utilizados pela interface.
- O ficheiro *xmldatabaseinterface* contém todas as operações efetuadas sobre a base de dados.
- O ficheiro *views* irá receber os pedidos html e irá efetuar o processamento necessário para devolver uma página html para o utilizador.
- O ficheiro *forms* contêm os formulários necessários para a introdução de dados por parte do utilizador.

Existem ainda três pastas relevantes para a leitura do relatório:

- A pasta *static* contém os conteúdos estáticos para a página web, como imagens, ícones, ficheiros de *javascript*, ficheiros de estilo *css*, entre outros.

- A pasta *templates* contêm os ficheiros *html* para cada página que será apresentada.
- A pasta *xml_files* contêm todos os ficheiros *XML* utilizados: ficheiros *XSLT*, ficheiros *XML Schema* e ficheiros *XML* de backup. Se não existirem as bases de dados necessárias, serão automaticamente criadas e preenchidas com os dados presentes nos ficheiros de backup.

Listing 2.1: Estrutura do projeto em árvore com as pastas e ficheiros mais relevantes.

```
UAluno
|-- UAluno
|-- app
|   |-- BaseXClient
|   |-- data_sources
|   |-- data_logic
|   |-- xmldatabaseinterface
|   |-- views
|   |-- forms
|   `-- static
|   `-- templates
|   `-- xml_files
```

Capítulo 3

Dados e suas fontes

Neste capítulo é feita a descrição dos dados recebidos e as suas fontes. Os dados externos que são recolhidos provêm de APIs públicas, nomeadamente do Academic Playground & Innovation[2] e do OpenWeatherMap[3].

No primeiro caso (Academic Playground & Innovation[2]) são recolhidos grupos de dados sob o formato *XML* relativamente à Universidade de Aveiro. O primeiro grupo de dados é recolhido através do serviço de *Ementas*, que fornece informação sobre as ementas dos refeitórios, snack-bar e restaurante da Universidade de Aveiro. O segundo grupo de dados é recolhido através do serviço do *Jornal Online* que fornece informação sobre notícias presentes no Jornal Online da Universidade de Aveiro e também a listagem de departamentos que é usada para filtrar as notícias associadas aos mesmos.

Listing 3.1: Exemplo de recolha de dados (*RSS*). Neste caso, esta função devolve as notícias presentes no Jornal Online, sendo validadas pelo respetivo *XML Schema*. Caso não seja possível efetuar a ligação à *url*, a função irá devolver os últimos dados válidos armazenados numa base de dados XML.

```
# Get XML for UA news highlights
def get_newsXml(dt, d):
    url =
        "http://services.sapo.pt/UA/Online/contents_xml?dt="+str(dt)+"&d="+str(d)

    newData = False

    try:
        r = requests.get(url)
        r.encoding = 'utf-8'
        content = r.text
```

```
        parser = etree.XMLParser(encoding='utf-8')
        news = etree.fromstring(content.encode('utf-8'),
            parser=parser)

        if r.status_code != 200:
            # If it could not get from the web, get the data from the
                database
            print("Could not fetch the news. Getting news from
                database...")
            content = xmldatabaseinterface.getAllDatabase("news",
                settings.XML_FILES + "/news.xsd")
            parser = etree.XMLParser(encoding='utf-8')
            news = etree.fromstring(content, parser=parser)
        else:
            # There is new data to be updated
            newData = True
    except:
        # If it could not get from the web, get the data from the
            database
        print("Could not fetch the news. Getting news from
            database...")
        content = xmldatabaseinterface.getAllDatabase("news",
            settings.XML_FILES + "/news.xsd")
        parser = etree.XMLParser(encoding='utf-8')
        news = etree.fromstring(content, parser=parser)

    # Validate xml with schema
    try:
        with open(settings.XML_FILES + '/news.xsd') as ifile:
            xmlschema_doc = etree.parse(ifile)
            xmlschema = etree.XMLSchema(xmlschema_doc)

            if xmlschema.validate(news):

                # Update the database with new content
                if(newData):
                    xmldatabaseinterface.updateDatabase('news',
                        settings.XML_FILES + '/news.xsd', content)
                return news
            else:
                return None
    except:
        raise FileNotFoundError("Could not validate the news schema")
```

No segundo caso, são recolhidos dados sob o formato de *XML* relativamente às condições meteorológicas da cidade de Aveiro através da API do OpenWeatherMap[3].

Caso as APIs se encontrem inacessíveis através da web, será fornecida a última informação válida recebida e guardada em base de dados *XML*. Assim, por cada pedido efetuado, é necessário armazenar os dados na base de dados.

Os dados relativamente às notas (e respetivas cadeiras) do utilizador e ao chat são fornecidos pelos utilizadores e armazenados também em base de dados *XML*, não havendo qualquer intervenção de fontes externas.

Capítulo 4

Esquemas dos dados *(XML Schema)*

Para definir os esquemas de dados *XML* foi utilizada a norma *XML Schema*. Foram criados esquemas de dados para os dados da meteorologia, ementas, notícias, listagem de departamentos, notas e chat.

Os esquemas de dados são utilizados na criação das bases de dados, para verificar se os dados inseridos inicialmente estão de acordo com o especificado. Quando se efetua uma operação de *GET* sobre uma base de dados integral também são utilizados para verificar a sua integridade. Para além destes casos, os esquemas são utilizados nos pedidos *RSS* para obter informação atualizada sobre as notícias, bem como sobre as ementas e a meteorologia.

Listing 4.1: Exemplo de esquema de dados para os conteúdos do chat que serão posteriormente armazenados na base de dados.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="chat" type="chatType"/>
    <xs:complexType name="chatType">
        <xs:sequence>
            <xs:element name="message" minOccurs="0"
                maxOccurs="unbounded" type="messageType" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="messageType">
        <xs:simpleContent>
```

```
            <xs:extension base="xs:string">
                <xs:attribute name="id" type="xs:string"/>
                <xs:attribute name="name" type="xs:string"/>
                <xs:attribute name="timestamp" type="xs:string"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

No exemplo do esquema anterior para os conteúdos do chat pode ser visto que o elemento pai "chat"pode conter um ou mais elementos do tipo "message", os quais são identificados pelo *id* do utilizador, pelo seu nome e pela data de envio.

Capítulo 5

Transformações sobre os dados *(XLST)*

A transformações sobre dados são aplicadas sobre o conteúdo das bases de dados *XML* que contêm as notas do utilizador e o chat.

No caso da transformação das notas, a informação sobre as cadeiras (com nome, ano, semestre, área, ects e nota final) é transformada numa tabela. Essa tabela apresenta a quadrícula da nota com a cor verde caso o valor seja maior ou igual a 10 e vermelho caso contrário. Após a tabela, é calculada a média do aluno com base nas notas obtidas nas suas cadeiras. Também é oferecida a possibilidade de remover uma cadeira através de um botão na linha associada da tabela.

Para a transformação do chat, cada mensagem é apresentada na interface de maneira diferente, ou seja, caso o emissor da mensagem seja o próprio utilizador, o aspeto é diferente de uma mensagem de outro utilizador.

Listing 5.1: Exemplo de transformação *XSLT* para o chat. Para cada mensagem é verificado se o id é igual ao do utilizador, para possibilitar a distinção na interface de mensagens enviadas e recebidas.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:param name="uid" />
    <xsl:template match="/">
        <xsl:for-each select="chat/message">
            <xsl:choose>
                <xsl:when test="\$uid=@id">
                    <div class="row msg_container base_sent">
```

```
                    <div class="col-md-10 col-xs-10 ">
                        <div class="messages msg_sent">
                            <p><xsl:value-of select="."/></p>
                            <time><xsl:value-of select="@name"/> -
                                <xsl:value-of
                                select="@timestamp"/></time>
                        </div>
                    </div>
                </div>
            </xsl:when>
            <xsl:otherwise>
                <div class="row msg_container base_receive">
                    <div class="col-md-10 col-xs-10 ">
                        <div class="messages msg_receive">
                            <p><xsl:value-of select="."/></p>
                            <time><xsl:value-of select="@name"/> -
                                <xsl:value-of
                                select="@timestamp"/></time>
                        </div>
                    </div>
                </div>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

No exemplo de transformação anterior para o chat, é feita uma iteração sobre as mensagens do chat, definindo como devem aparecer ao utilizador. Para cada mensagem é verificado o id para analisar se corresponde ao id do utilizador que se encontra a visualizar a informação. Caso se verifique, a mensagem é apresentada de uma forma e, caso não se verifique, é apresentada de outra forma (definindo uma diferença visível entre mensagem recebida e mensagem enviada).

Capítulo 6

Operações sobre os dados *(XQuery e XQuery Update)*

Neste capítulo são descritas as bases de dados *XML* existentes na aplicação e os vários queries que são efetuados sobre estas. Existem seis bases de dados com funções bem definidas:

- Base de dados para as notas dos estudantes;
- Base de dados para o chat dos utilizadores;
- Base de dados para a meteorologia;
- Base de dados para as ementas;
- Base de dados para as notícias;
- Base de dados para os departamentos.

6.1 Base de dados para notas dos estudantes

A base de dados para as notas dos estudantes segue a seguinte estrutura abaixo representada em que tem um grupo de elementos estudantes que é composto por múltiplos elementos estudante, os quais possuem como único atributo o seu identificador interno. Cada estudante é composto pelo conjunto das suas disciplinas, estas que têm como atributos o seu nome, ects, área, ano e semestre. Cada disciplina é composta por apenas uma nota. Esta descrição é sempre validada pelo Schema correspondente.

Listing 6.1: Base de dados para notas dos estudantes, com dois estudantes e respetivos cursos e notas.

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
    <student id="1">
        <course name="IHC" ects="6" area="informática" year="3"
            semester="2">
            <grade>18</grade>
        </course>

        <course name="LFA" ects="6" area="informática" year="2"
            semester="1">
            <grade>17</grade>
        </course>
    </student>
    <student id="2">
        <course name="Programação I" ects="6" area="informática"
            year="1" semester="1">
            <grade>19</grade>
        </course>

        <course name="Programação II" ects="8" area="informática"
            year="1" semester="2">
            <grade>18</grade>
        </course>
    </student>
</students>
```

Na base de dados com as notas dos utilizadores são utilizadas diversas *queries*. Uma das *queries* é usada para devolver todas as notas de um utilizador, notas estas que serão usadas posteriormente para efetuar uma transformação de dados através de *XSLT*.

Listing 6.2: *Query* usada para ir buscar todas as notas de um determinado aluno baseado no seu identificador.

```
xquery for $student in doc('gradesdatabase')/students/student where
    $student/@id=student_id return $student
```

Existem ainda *queries* mais complexas usadas para verificar, quer a existência de um estudante dado o seu identificador interno, quer a existência de uma nota de um estudante dado o nome da disciplina.

Para inserir uma nova disciplina (e consequentemente a sua nota), é necessário verificar se o estudante existe na base de dados de notas. Se tal não se verificar, é necessário criar uma entrada para o novo estudante e finalmente inserir a disciplina e sua respetiva nota. No caso em que o estudante já se encontra registado na base de dados, é necessário verificar se a disciplina que está a ser adicionada já existe associada ao utilizador na base de dados. Caso não esteja, a disciplina e respetiva nota são adicionadas, ficando então associadas ao estudante em questão, e no caso em que a disciplina em questão já se encontra na base de dados, a inserção não é efetuada.

Foi sobre esta operação que surgiram as queries mais complexas, apresentadas abaixo.

Listing 6.3: *Query* para verificar a existência de um estudante

```
xquery for $student in doc('gradesdatabase')/students/student where
    $student/@id=student_id return $student
```

Como referido anteriormente, caso o estudante já estivesse registado teria de se verificar posteriormente se a disciplina que estava a ser adicionada já existia dai ter sido criado a seguinte *query*.

Listing 6.4: *Query* para verificar a existência de uma nota de um determinado estudante

```
xquery for $student in doc('gradesdatabase')/students/student where
    $student/@id=student_id return $student/course/@name=course_name
```

Caso o estudante não estivesse registado na base de dados, este teria de ser criado e adicionado à base de dados, da mesma forma e em simultâneo seria adicionada a disciplina. Nesta *query* apenas foi necessário criar um node com a informação do estudante e disciplina a ser adicionada, e por fim inserir efetivamente na base de dados.

Listing 6.5: Exemplo de *query* para adicionar um estudante e uma disciplina caso este não existisse previamente na base de dados.

```
node = <student id="id"><course name="name" ects="ects" area="area"
    year="year"
    semester="semester"><grade>grade</grade></course></student>

insert node ' + node + ' into doc("gradesdatabase")/students
```

No caso em que o estudante já estivesse registado na base de dados, apenas

teríamos de criar um node com a informação da disciplina e adicionar esse node como filho do elemento estudante em causa.

Nesta caso apenas foi necessário criar um node com a informação da disciplina a ser adicionada, pesquisar o estudante em questão e por fim adicionar o node como filho desse estudate.

Listing 6.6: *Query* para associar uma nova disciplina a um estudante já registado na base de dados.

```
node = <course name="name" ects="ects" area="area" year="year"
    semester="semester"><grade>grade</grade></course>

xquery for $student in doc("gradesdatabase")/students/student where
    $student/@id="id" return insert node 'node' into $student
```

Por fim, a *query* mais desafiante foi a desenvolvida para poder remover uma nota de um determinado estudante.

Neste caso foi necessário fazer uma pesquisa por todas as disciplinas do estudante em questão, depois foi necessário pesquisar a disciplina com o nome da disciplina a remover e remove-la.

Listing 6.7: *Query* para remover uma nota de um estudante.

```
let $courses := for $student in
    doc("gradesdatabase")/students/student where $student/@id="id"
    return $student/course

let $course := for $course in $courses where
    $course/@name="course_name" return $course

return delete node $course
```

Nas restantes bases de dados as *queries* utilizadas seguiram a mesma estrutura dos queries anteriormente apresentados.

6.2 Base de dados para o chat dos utilizadores

Na base de dados com os conteúdos do chat dos utilizadores, são efetuadas as operações de inserção de uma mensagem (com identificador interno do utilizador, nome de utilizador e data de envio) e uma *query* que devolve todo

o conteúdo *XML* presente na base de dados para efetuar a sua transformação usando *XSLT*.

6.3 Bases de dados para a meteorologia, ementas e notícias

As bases de dados de meteorologia, ementas e notícias contêm os dados do último pedido efetuado às respetivas APIs. Assim sendo, servem como bases de dados de backup para quando os dados não são acessíveis por *HTTP*. Quando os dados são recebidos, se o seu conteúdo estiver de acordo com o esquema, são armazenados na base de dados.

6.4 Base de dados para os departamentos

A base de dados com os departamentos é uma base de dados estática (sem alteração de conteúdos por parte da aplicação) e contém os departamentos aceites pela API das notícias (Academic Playground & Innovation[2]). Quando a página de notícias é carregada, é feita uma *query* que devolve o nome de todos os departamentos para auxiliar a filtragem de conteúdos. Quando o utilizador decide filtrar as notícias por departamento é feita outra *query* sobre a base de dados que irá devolver o *id* que representa o nome do departamento escolhido.

Capítulo 7

Funcionalidades da aplicação *(UI)*

Neste capítulo são apresentadas as diversas funcionalidades da aplicação. Tendo como primeiro passo o registo e a entrada do utilizador na sua conta, o utilizador é encaminhado para uma página inicial. Nesta página (figura 7.1) é fornecida informação básica relativamente à meteorologia em Aveiro no dia presente e no dia seguinte, os pratos principais relativamente à ementa para a semana em que o utilizador se encontra e, por fim, as últimas notícias publicadas pelo Jornal Online da UA onde poderá ver uma descrição associada ao carregar no título de uma notícia.

Através do menu principal, o utilizador pode aceder às páginas Meteorologia, Ementas, Notícias e Notas.

A página Meteorologia (figura 7.2) permite aceder à previsão das condições meteorologias em Aveiro para os próximos 7 dias (incluindo o dia em que se encontra), sendo fornecidos dados como o estado atual do tempo, hora prevista para o nascer e pôr do sol, temperatura atual, máxima e mínima, precipitação, nuvens, pressão, humidade e vento, sendo cada dia apresentado consoante a escolha feita através do menu lateral direito.

A página Ementas (figura 7.3) permite aceder às ementas completas da semana atual nos refeitórios, snack-bar e cantinas da UA (como por exemplo o Refeitório de Santiago, Refeitório do Crasto, Snack-bar da UA) sendo cada dia apresentado consoante a escolha feita através do menu lateral direito.

A página Notícias (figura 7.4) permite aceder às notícias em destaque no Jornal Online da UA e posteriormente, através da interação com a componente de filtragem do lateral direito, poderá visualizar as notícias relevantes

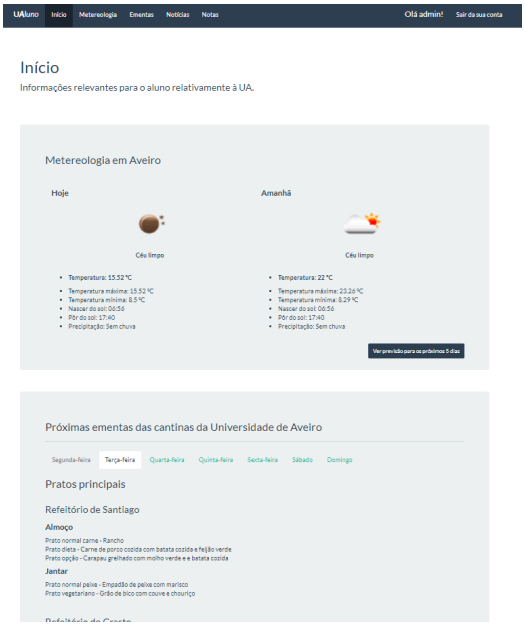


Figura 7.1: Página inicial

e em destaque por departamento. Cada notícia é apresentada através de um título, uma descrição e um link para o utilizador ler na íntegra.

A página Notas (figura 7.5) permite inserir, apagar e visualizar informações das cadeiras e respetivas notas fornecidas pelo utilizador em causa, tal como a possibilidade de saber a sua média atual com base nas informações associadas ao utilizador. Cada cadeira inserida contém vários dados tais como um nome, os ects, a área, o ano, o semestre e a respetiva nota do utilizador.

Em todas as páginas também é possível a comunicação entre utilizadores da plataforma através de um chat comum a todos os utilizadores, sendo sempre guardado e apresentado o histórico de mensagens trocadas.

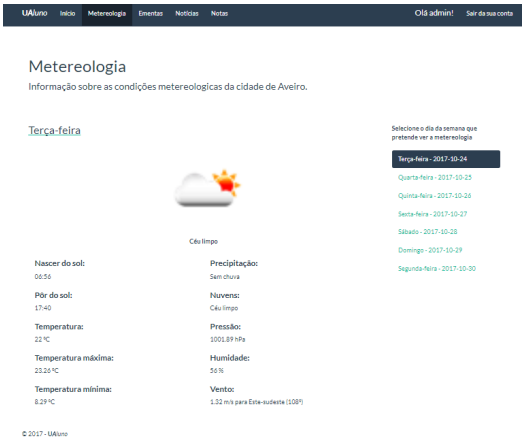


Figura 7.2: Página Meteorologia

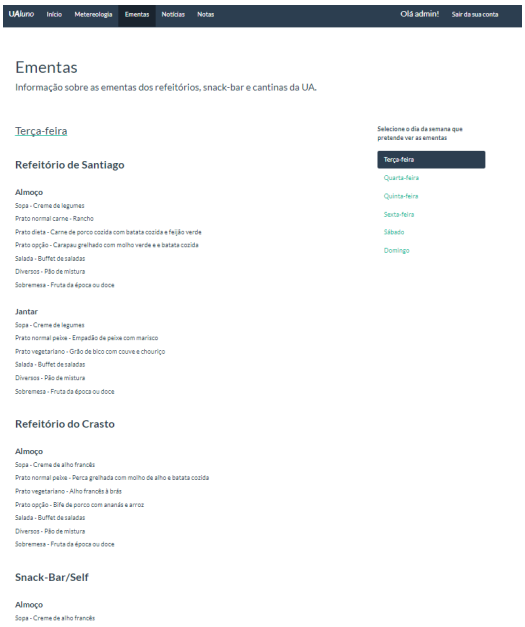


Figura 7.3: Página de Ementas

Figura 7.4: Página das notícias

Figura 7.5: Página das notas do estudante

Capítulo 8

Conclusões

Após a elaboração deste projeto, consideramos que atingimos com sucesso o objetivo proposto de criar um portal útil para os alunos da Universidade de Aveiro, com informação agregada de várias fontes e com uma interface agradável e funcional. Para além disso, foram utilizadas as tecnologias propostas como objetivos do projeto (*Django*, *XML*, *XML Schema*, *XPath*, *XSLT*, base de dados *XML*, *XQuery* e *XQuery Update* e *RSS*).

Como trabalho futuro seria possível integrar mais dados relevantes sobre a Universidade de Aveiro, como informação automática sobre as notas obtidas de um aluno. Tal não foi implementado nesta fase devido às limitadas APIs disponíveis em XML. A integração com o sistema de autenticação utilizado pelo *e-learning* também seria uma prioridade, que permitiria identificar inequívocamente o utilizador autenticado, mas não foi considerada uma prioridade neste projeto.

Capítulo 9

Configuração para executar a aplicação

O projeto foi desenvolvido utilizando a linguagem de programação *Python (3.6.2)* e a *framework Django (1.11.5)*. Para executar a aplicação são necessários os seguintes packages e as suas versões correspondentes:

- lxml==4.0.0
- pytz==2017.2
- requests==2.18.4
- urllib3==1.22

Para executar a aplicação, é necessário executar o servidor BaseX, depois deve abrir a pasta do projeto (UAluno) e executar o seguinte comando: *python3 manage.py runserver*. A aplicação fica acessível no browser via url 127.0.0.1:8000 (equivalente a localhost:8000).

Se não existirem as bases de dados necessárias para a aplicação, serão automaticamente criadas e preenchidas com os dados presentes nos ficheiros de backup.

Existem inicialmente quatro utilizadores registados na plataforma, que podem ser utilizados para verificar as funcionalidades, ou podem ser adicionados mais utilizadores. Os tuplos (*username*, *password*) dos utilizadores são:

- (admin, rootroot).

- (user1, trabalhoedc).
- (user2, rootroot).
- (user3, rootroot).

Bibliografia

[1] URL: http://www.ua.pt/deti/uc/2380.

[2] URL: http://api.web.ua.pt/.

[3] URL: https://openweathermap.org/api.