University of Essex | Online

Secure Software Development

README file

Team 4 **Coding Output: ISS Project**

24 October 2021

How to execute the code

Please access the project environment via the link below:

https://codio.co.uk/tedgell/ssd-development-team-project

bash

Ensure the current working directory is set to:

```
codio@charlie-sample:~/workspace
```

Run the startup script below *:

```
python3 setup.py
```

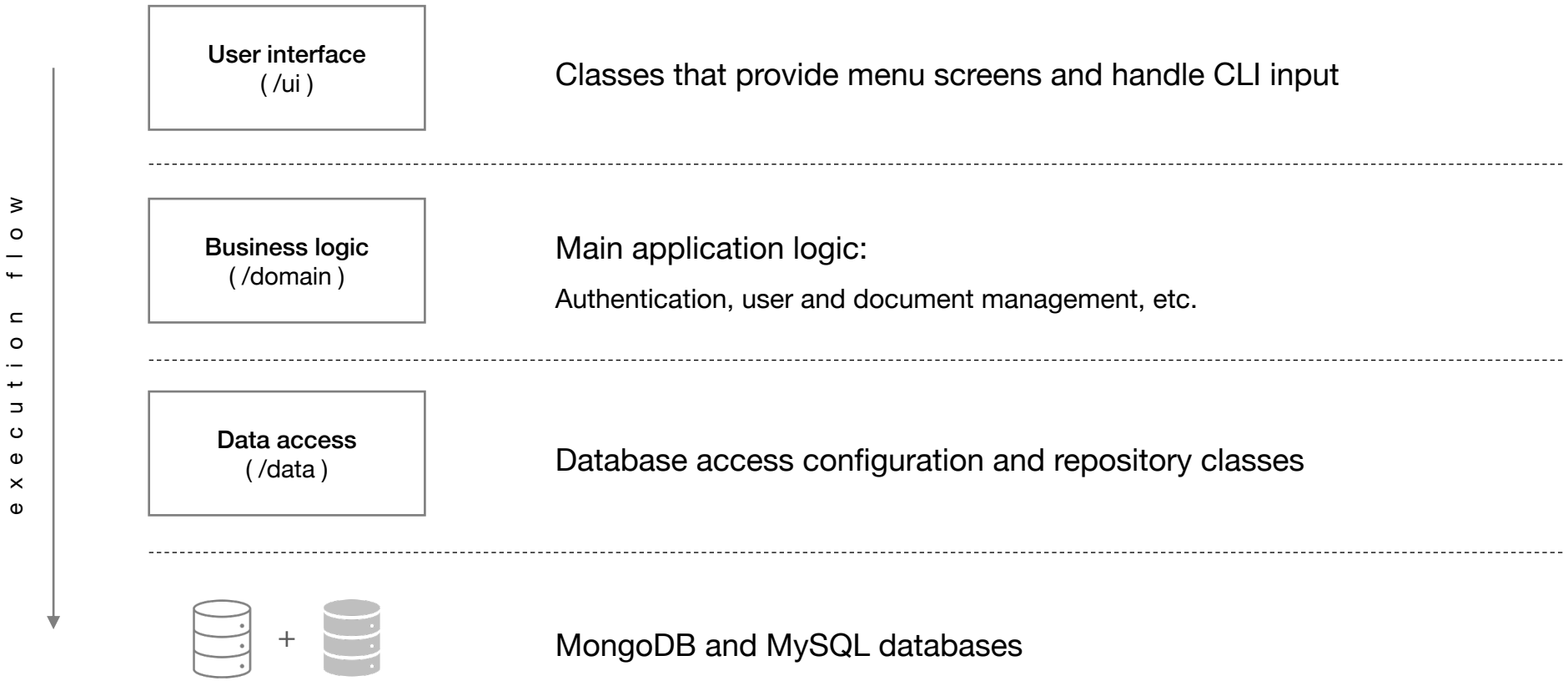
Start the ISS application:

```
python3 main.py
```

* This script will reset the user and document databases to their default states

Application structure

The main application code can be found in the `/src` folder. The monolithic implementation consists of the following layers:



Storing passwords

Below is a description of the SQL table structure used for storing user information:

Field	Type	Null	Key	Default	Extra
id	int(11) unsigned	NO	PRI	NULL	auto_increment
first_name	varchar(35)	NO		NULL	
last_name	varchar(35)	NO		NULL	
position	varchar(50)	NO		NULL	
email	varchar(255)	NO		NULL	
username	varchar(128)	NO		NULL	
password	char(60)	NO		NULL	
access_lvl	tinyint(4)	NO		NULL	

The `password` field holds a generated hash value of the user's password, along with a unique salt that is prepended to this value. Random salts increase the complexity of passwords and safeguard them in storage, making brute-force attempts more difficult by increasing the attackers' search space (Naiakshina et al., 2017).

The creation of the password hash value is done using the bcrypt library.

MySQL user account

As shown below, we have created a dedicated MySQL user account that can perform operations on the `iss_auth` database. This account is set up in such a way that it does not have access to other databases in the MySQL environment.

mysql

```
mysql> SELECT CURRENT_USER();

+--------------------+
| CURRENT_USER()     |
+--------------------+
| iss_user@localhost |
+--------------------+
1 row in set (0.00 sec)


mysql> SHOW GRANTS FOR 'iss_user'@'localhost’;

+-----------------------------------------------------------------+
| Grants for iss_user@localhost                                   |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'iss_user'@'localhost'                    |
| GRANT ALL PRIVILEGES ON `iss_auth`.* TO 'iss_user'@'localhost' |
+-----------------------------------------------------------------+
2 rows in set (0.00 sec)
```

MySQL user account (cont.)

As can be seen in the configuration section of the `mysql_db.py` module, we have moved the username and password information of the MySQL user out of the main application code. This configuration information can be reviewed below:

```
 GNU nano 2.9.3                        .my.cnf

[client]
user=iss_user
password=n-v9Ke2W;uC}]7u%

[mysql]
no-auto-rehash
connect_timeout=2
```

This file is located in the `/home/codio` folder and is only accessible by the `codio` user, or under the system root account. Pushing the database credentials out of the client application is generally considered a good security practice (Regateiro et al., 2017).

```
root@charlie-sample:~# ll .my.cnf
-rw-r----- 1 codio codio 91 Sep 26 20:03 .my.cnf
root@charlie-sample:~#
```

MongoDB configuration

We have leveraged MongoDB for the management of safety entries. MongoDB is a document-oriented database that allows for greater flexibility when storing data and processes queries faster when compared to MySQL (Palanisamy & Suvithavani, 2020). Another reason behind choosing a NoSQL solution was to eliminate the need in performing object-relational mapping.

In order to enable search functionality, we have created text indexes for the `title`, `summary` and `keywords` fields.

```
{
  _id: ObjectId("61742468f96fa51def4d6d8c"),
  created: '23-09-2021',
  creator: 'Michael Justus',
  title: 'Pressure vessel TBR 4-3',
  summary: 'When the MDP is greater than 100 psia, it is considered a high pressure component and high pressure system safety requirements are
applicable. End items that are comprised of more than one pressurized component are considered a pressure system.',
  keywords: [ 'pressure', 'system', 'container', 'hazard' ],
  related: [ '614cb684792646c22da171a6', '614cb7ad792646c22da171a7' ],
  documents: {},
  sensitivity: 3
}
```

The next slide demonstrates the process of authenticating into the `issSafetyDB` database with dedicated user credentials using the MongoDB shell, also known as `mongosh`.

MongoDB configuration (cont.)

mongosh

```
issSafetyDB> db.getUsers()
{
  users: [
    {
      _id: 'issSafetyDB.issUser',
      userId: UUID("ff86c101-d938-4590-9355-e14dac20697c"),
      user: 'issUser',
      db: 'issSafetyDB',
      roles: [ { role: 'readWrite', db: 'issSafetyDB' } ],
      mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
    }
  ],
  ok: 1
}


issSafetyDB> db.auth( 'issUser', 's=M^$VC_jpyL_6xF' )
{ ok: 1 }


issSafetyDB> db.runCommand( {connectionStatus : 1} )
{
  authInfo: {
    authenticatedUsers: [ { user: 'issUser', db: 'issSafetyDB' } ],
    authenticatedUserRoles: [ { role: 'readWrite', db: 'issSafetyDB' } ]
  },
  ok: 1
}
```

API implementation

The API is created with Flask using the Flask-RESTful extension, and implements the following functions:

Function	CRUD Operation	Endpoint	Example Command
Generate session token	CREATE / POST	/tokens	curl --user *name:password* http://127.0.0.1:5000/tokens
Search DB for entry	READ / GET	/search/<string:search_string>	curl -H "Authorization: Bearer *token*" * http://127.0.0.1:5000/search/*search_string*
Show entry	READ / GET	/entry/<string:entry_id>	curl -H "Authorization: Bearer *token*" http://127.0.0.1:5000/entry/download/*entry_id*
Create entry	CREATE / POST	/entry/new	curl -H "Authorization: Bearer *token*" http://localhost:5000/entry/new -d "title=*title*" -d "summary=*summary*" -d "keywords=*keyword*" -d "sensitivity=*sensitivity*" -X POST

* The token authentication approach is based on Flask Mega Tutorial by Miguel Grinberg (2017)

API implementation (cont.)

Function	CRUD Operation	Endpoint	Example Command
Download entry	CREATE / POST	/entry/download/<string:entry_id>	w3m -header "Authorization: Bearer *token*" http://127.0.0.1:5000/entry/download/*entry_id*
Delete entry	DELETE	/entry/<string:entry_id>	curl -H "Authorization: Bearer *token* http://localhost:5000/entry/*entry_id* -X DELETE
Check username/password	READ / GET	Called with /tokens	N/A
Error handling password check	N/A	N/A	N/A
Error handling token check	N/A	N/A	N/A
Token checker	READ / GET	Called with all endpoints (except /tokens)	N/A
Revoke token (when past expiration date)	DELETE	Access as required by token checker	N/A

Overview of security risks

OWASP Category *	Mitigation
A01:2021-Broken Access Control	We have implemented the principle of least priviledge, which can be seen by examining the system access definitions in the Authentication class. It is not be possible to view administrator content as a standard user, or to elevate the priviledges. We have made sure that an attacker cannot easily bypass the access controls checks.
A02:2021-Cryptographic Failures	We have implemented strong protection of user passwords by using the latest version of a highly reliable cryptographic library and leveraged satls to increase the complexity of passwords. When selecting the hashing solution, we have made sure to avoid weak or deprecated cryptographic algorithms such as MD5 or SHA1.
A03:2021-Injection	We have made sure to validate, filter, or sanitize user-supplied data. This can be seen by examining the user interface classes handling command line interface input. We have implemented repository classes providing data access operations, and ensured that no hostile or malicious data is allowed in dynamic queries or commands.
A04:2021-Insecure Design	*This category includes high-level considerations such as business risk profiling, threat modelling, and implementation of secure SDLC. Proper prevention is out the scope.*

* Referenced from the OWASP Top 10 awareness document (OWASP Foundation, 2021)

Overview of security risks (cont.)

OWASP Category	Mitigation
A05:2021-Security Misconfiguration	We have set up dedicated accounts for accessing the MongoDB and MySQL databases and made sure that no unnecessary features, such as priviledges or grants, are enabled for these accounts. There are no default accounts and passwords used in the client application, and there are also no unused ports are open in Flask.
A06:2021-Vulnerable and Outdated Components	We have made sure that we regularly update the components and libraries that are used for this project using the `apt-get upgrade` command. We have decided that automating this procedure is outside of scope of this assignment.
A07:2021-Identification and Authentication Failures	We do not permit weak or well-known passwords, or store plain text or weakly hashed passwords in the user database. Other prevention mechanisms such as multi-factor authentication or single sign-on (SSO) tokens are more complicated and were not implemented due to the time constraints.
A08:2021-Software and Data Integrity Failures	We have ensured that the libraries and dependencies that we consume come from trusted repositories. Other mitigations such as implementation of secure CI/CD pipelines is outside of scope of this assignment.

Overview of security risks (cont.)

OWASP Category	Mitigation
A09:2021-Security Logging and Monitoring Failures	We have created basic functionality that logs unsuccessful user login attempts. Complete prevention of this vulnerability goes beyond the scope of this assignment, and would include penetration testing and dynamic security testing, setting up a proper response management process, configuration of alerting thresholds, etc. It should also be noted that the authors of the OWASP Top 10 consider locally stored logs a bad practice, however addressing this would require setting up a dedicated server outside of the primary project envrionment.
A10:2021-Server-Side Request Forgery	Not applicable to this implementation.

Summary of imports

Name	Version	Description
brcypt	3.2.0	Password-hashing library exposing a function designed by N. Provos and D. Mazieres. This adaptive function is based on the Blowfish cipher and was designed to be resistant to brute-force search attacks (Sriramya & Karthika, 2015)
flake8	3.9.2	Wrapper around several style guide enforcement tools
Flask	2.0.2	Micro web framework that simplifies web server creation
Flask-HTTPAuth	4.4.0	Flask extension that provides basic HTTP authentication via Flask routes
Flask-RESTful	0.3.9	Flask extension that enables REST API creation
mccabe	0.6.1	McCabe complexity checker
mysql-connector-python	8.0.26	Standardized MySQL database driver for Python development
pylint	1.9.5	Static code analysis tool that helps to enforce a coding standard
PyMongo	3.12.0	Official Python driver for interacting with MongoDB databases
stdiomask	0.0.6	Module that masks passwords entered in a stdio terminal

References

Grinberg, M. (2017) The Flask Mega-Tutorial. Available from: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world [Accessed 24 October 2021].

Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S. & Smith, M. (2017) 'Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study', *2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, 30 October - 7 November. New York: Association for Computing Machinery. 311-328.

OWASP Foundation (2021) OWASP Top Ten. Available from: https://owasp.org/www-project-top-ten/ [Accessed 24 October 2021].

Palanisamy, S. & Suvithavani, P. (2020) 'A survey on RDBMS and NoSQL Databases MySQL vs MongoDB', *2020 International Conference on Computer Communication and Informatics (ICCCI)*. Coimbatore, 22-24 January. New York: IEEE. 1-7.

Regateiro, D., Pereira, O. & Aguiar, R. (2017) 'Server-Side Database Credentials: A Security Enhancing Approach for Database Access', *International Conference on Data Management Technologies and Applications*. Madrid, 24-26 July. New York: Springer. 215-236.

Sriramya, P. & Karthika, R. (2015) Providing password security by salted password hashing using bcrypt algorithm. *ARPN Journal of Engineering and Applied Sciences*. 10(13): 5551-5556.