Secure Software Development

Team 4 **ISS Design Document** 0.12

22 September 2021

Team Members

Name	Email	ePortfolio
Andrey Smirnov	soundmaven@gmail.com	https://soundmaven.github.io/e-portfolio/
Michael Justus	micjustus@gmail.com	https://micjustus.github.io/essex-eport2/
Taylor Edgell	tayloredgell@googlemail.com	https://tedgell.github.io/MSc-ComputerScience/

Document Version History

Version	Date	Author	Details
0.1	06/09/21	Taylor Edgell	Document creation
0.2	07/09/21	Andrey Smirnov	Draft list of assumptions
0.3	10/09/21	Michael Justus	Revision and minor updates
0.4	10/09/21	Andrey Smirnov	List of technical challenges
0.5	10/09/21	Taylor Edgell	Addition of UML diagrams
0.6	12/09/21	Andrey Smirnov	Design patterns and tools sections
0.7	13/09/21	Taylor Edgell	OWASP table and UML explanation
0.8	14/09/21	Michael Justus	System requirements and revision
0.9	15/09/21	Andrey Smirnov	Updated patterns & methodology
0.10	17/09/21	Taylor Edgell	GDPR, minor edits, additional references
0.11	18/09/21	Andrey Smirnov	Section updated on APIs and microservices
0.12	19/09/21	Michael Justus	Content revision.
1.0	19/09/21	Taylor Edgell	Submission

Definitions and Abbreviations

Acronym	Description
API	Application Programming Interface
ISS	International Space Station
MVC	Model View Controller
RBAC	Role-Based Access Control
REST	Representational State Transfer
MVP	Model View Presenter
MVVM	Model View ViewModel
UML	Unified Modelling Language

Contents

1. Overview

The following technical design document describes the proposed solution design for International Space Station's requirements for a Safety Entry Database. The rationale of the solution design is taken from the recommendations of an independent task forces analysis of the ISS (NASA, 2017). Assessing risks is a key functionality of the ISS (Vitali & Lutomski, 2004), so a cohesive central database would be invaluable.

Security is a top priority of the solution's design. Therefore, there is a need to prevent cybersecurity attacks across the entire attack surface of the solution.

This technical design document lists requirements and assumptions concerning the solution and describes the proposed design using UML diagrams. The report also details the technology, patterns, and tools used to deliver the solution. Lastly, all academic reference materials are listed at the end of this design report.

2. System Requirements

The following table lists requirements elicited from relevant stakeholders and the solution brief:

Table 1 - System requirements

ID	Name	Rationale
REQ01	Upload reports	Authenticated user uploads report to the system
REQ02	View reports	Authenticated users can request and view report data
REQ03	Download reports	Authenticated users can download report data
REQ04	Manage users	Admins can manage external users within the system
REQ05	Crew management	Capture information of crew members aboard the ISS
REQ06	Safety management	Capture issues related to safety aboard the ISS

3. Assumptions

The following assumptions concern the solution design:

3.1 User interface

- A fully-fledged web-based user interface is not provided. The solution features a command-line interface and REST APIs to interact with the underlying system. Therefore, implementation of user interface design patterns (i.e., MVC, MVP, MVVM) is not essential.

3.2 Architecture

- The core of the proposed solution is business logic, implemented in Python modules using the object-oriented paradigm.
- The solution's class structure allows for a smooth transition to a microservices architecture.
- Clear separation of concerns between the business logic and data access layers (and modularisation) will help decrease future migration efforts (Kuryazov et al., 2020).

3.3 Scalability

- There is no support for horizontal scalability, data or event streaming functionality (i.e., Kafka, RabbitMQ).

3.4 Testing approach

- Testing uses white-box techniques (e.g., unit and integration tests). Black box testing is not applicable because the team has visibility into the software code (Umar, 2019).
- Performance testing is not considered.

3.5 Authentication and authorisation

- Authorisation uses the scalable and flexible Role-Based Access Control (RBAC) approach, which can be expanded upon when transitioning to microservices architecture (Triartono et al., 2019).
- Infrastructure security (i.e., Firewall, Load balancers), distributed access delegation techniques (i.e., OAuth, Authorisation server) and patterns such as API gateway are out of scope.

3.6 GDPR

- The solution will meet the data protection regulations as set out by the European GDPR standard (GDPR, 2019).

Table 2 - GDRP requirements and provisions

GDPR requirement	Provision
Evaluation of data processing risks, activities and appropriate controls, such as encryption, should be in place.	An appropriate method on encryption will be applied to security sensitive data such as passwords.
Collected data will be from consenting parties and kept no longer than is necessary for the purposes for which the personal data are processed. (Not withstanding legal requirements such as HIPAA health record requirements).	Theoretical data policies will be put in place, and provisions made for data and sensitive information to be removed from the system. Only legally required personal information will be maintained within the system.
There must be a reasonable level of data protection and privacy.	Secure processes and paradigms will be used within the creation of the system. Appropriate testing will also take place to confirm data security.

4. Security Considerations

4.1 OWASP Top 10

The solution considers OWASP Top Ten (Mitre, 2017) points, ensuring sufficient, planned provision to combat common security weaknesses.

Table 3 - Solution security considerations

Security risk	Applicability to solution	Mitigation
Injection	Applicable – Various sections of the proposed solution require user input, which can be manipulated.	• Input sanitisation, e.g., Regular expressions.
Broken Authentication	Applicable – Our system will apply account security with various levels of access.	• Implement weak password checks. • Monitor and log all failed log-in attempts. • Establish a log-in attempt limit with soft and hard locks.
Sensitive Data Exposure	Applicable - Passwords and sensitive user input is a fundamental part of our system.	• Implementation of an encryption system to protect vulnerable data (e.g., passwords).
XML External entities	Not Applicable	
Broken Access Control	Applicable - The functionality of the system will require access privileges.	• Implementation of an access control system, including user privilege levels.

Security Misconfiguration	Minor applicability – relating to the removal of user accounts.	• Strict security policies will be in place to monitor for redundant accounts.
Cross-Site Scripting XSS	Not Applicable	
Insecure Deserialisation	Not Applicable	
Using components with known vulnerabilities	Applicable – The solution will leverage various libraries and other components.	• Using the current version of all required components. • Checking for known vulnerabilities. • Developing a theoretical patching process.
Insufficient logging and monitoring	Applicable – Various suspicious behaviours should be monitored.	• Failed access attempts will be logged. • Failed input validation will be logged. • Other potential suspicious activity will be considered and potentially logged.

5. High level System design

Appropriate UML guidelines and stylistic choices, such as those provided in 'Elements of UML Style (Ambler, 2003), have been referenced when creating all UML diagrams.

5.1 Use Cases

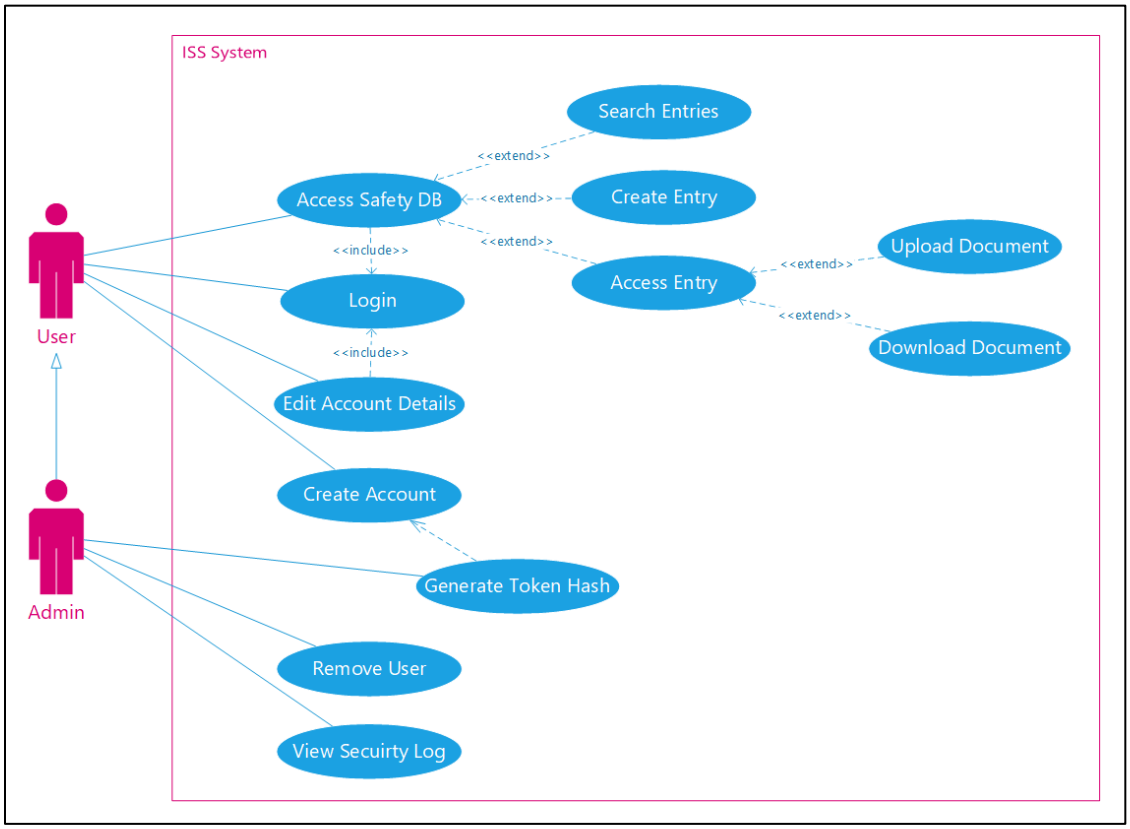

Figure 1 - Use cases for the proposed solution

The system's primary purpose is to provide users with easy access to a database containing safety entries. After creating a user account, they can search, access, and upload/download documents associated with safety entries. A user may also log into the system and edit their account details (provided they have appropriate levels of access control).

A specialised user type exists (“Admin”) that contains all the capabilities of a standard user. In addition, Admin may also view security logs, provide new log-in tokens for user accounts, and remove existing user accounts.

5.2 Class Diagram

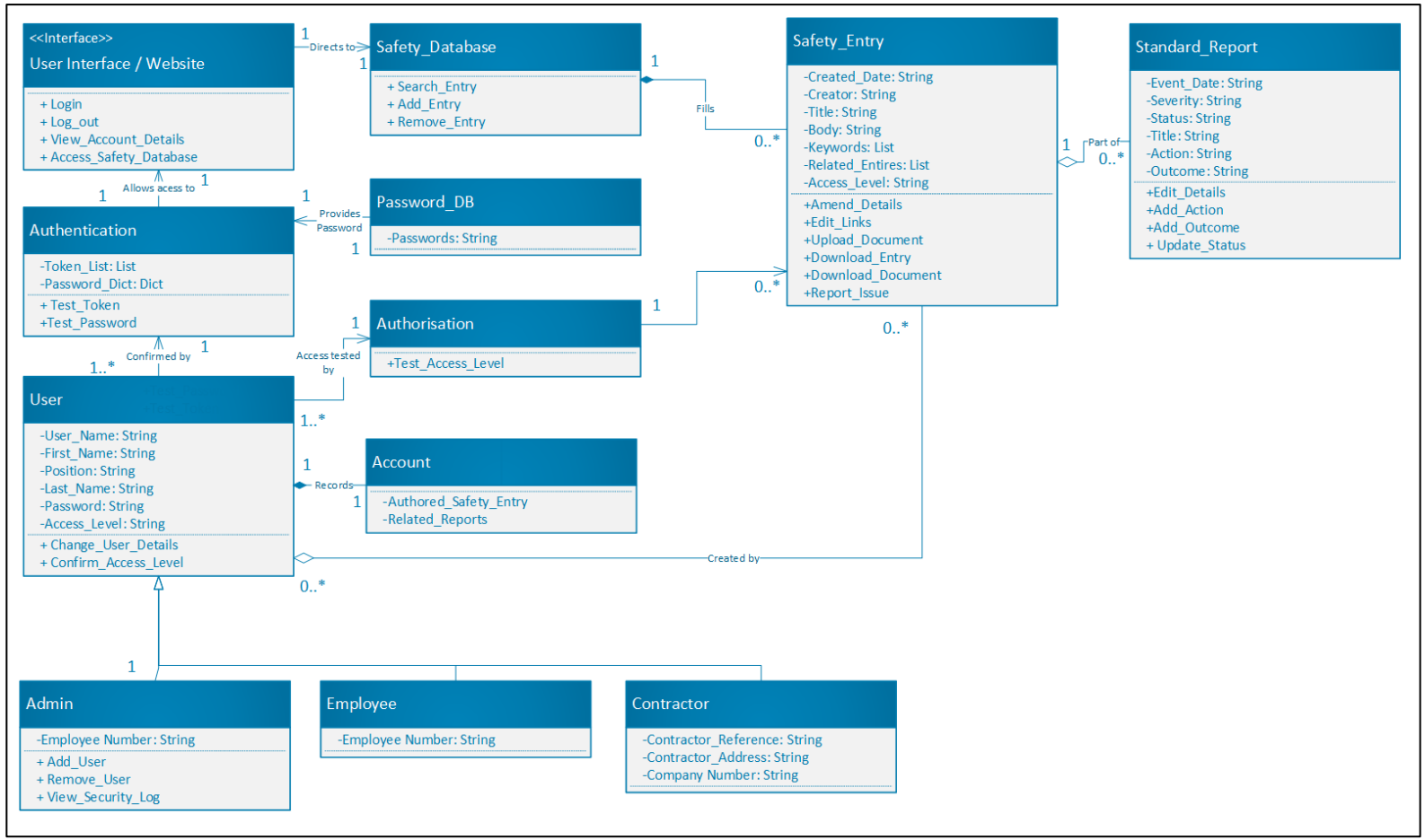

Figure 2 - Solution class diagram

The solution design consists of several modules, as shown in Figure 2. Users access the system, leverage the front-end user interface to navigate around, and perform several actions related to safety data. The system includes provisions for factors such as authentication and authorisation and user types.

5.3 Process: Application Access

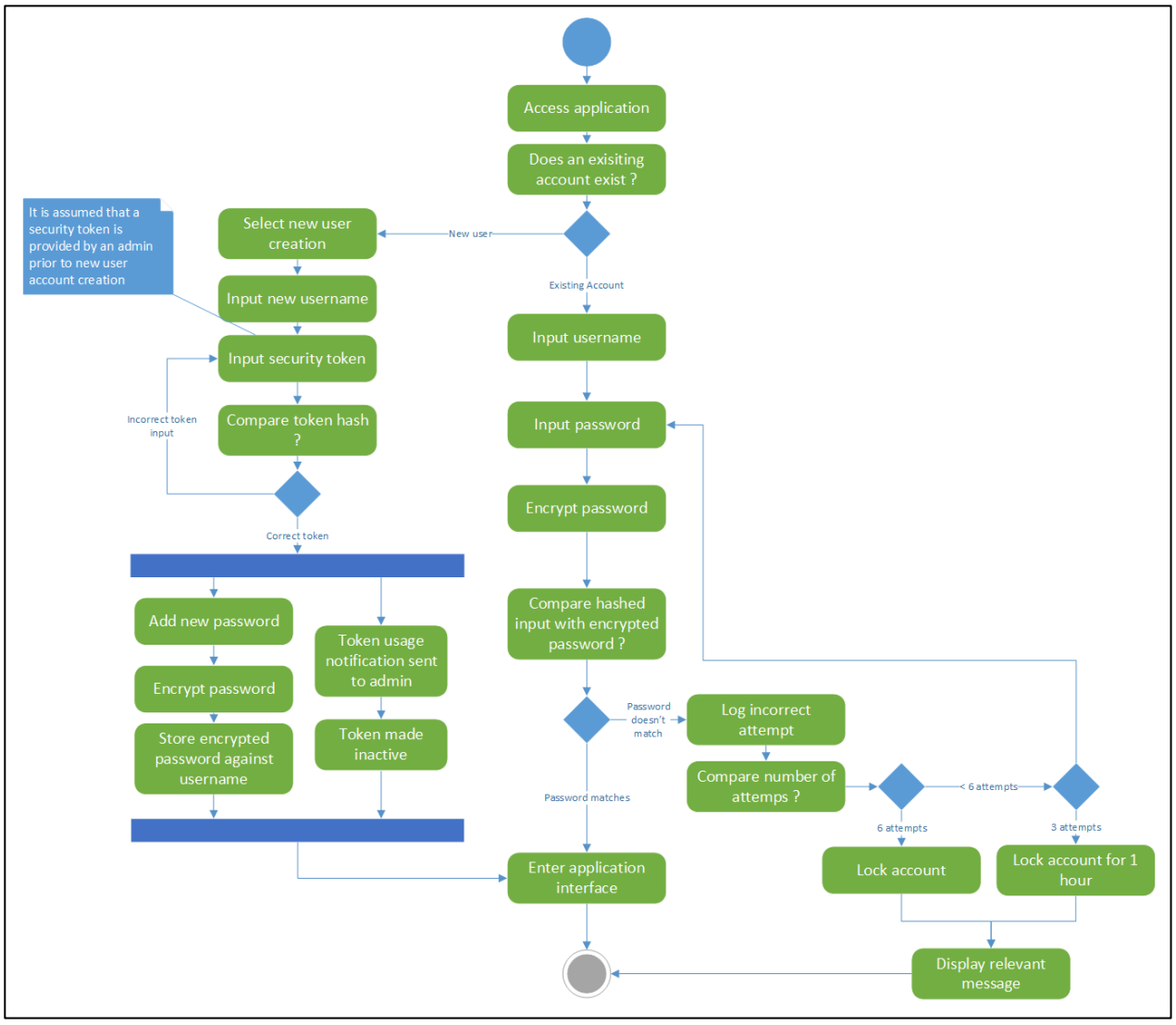

Figure 3 - Process for account creation

The system implements authentication that applies to each user account. User accounts are accessed by relevant username and password combinations. Passwords must meet specified criteria—yet to be determined. All incorrect log-in attempts will be logged. The system will utilise soft and hard locks depending on the number of incorrect log-in attempts. New user accounts can also be created, but only by an admin provided token. Once a “live” token is used, the token will be deactivated. The Admin will then be informed that the token has been activated.

5.4 Process: Using the Safety Database

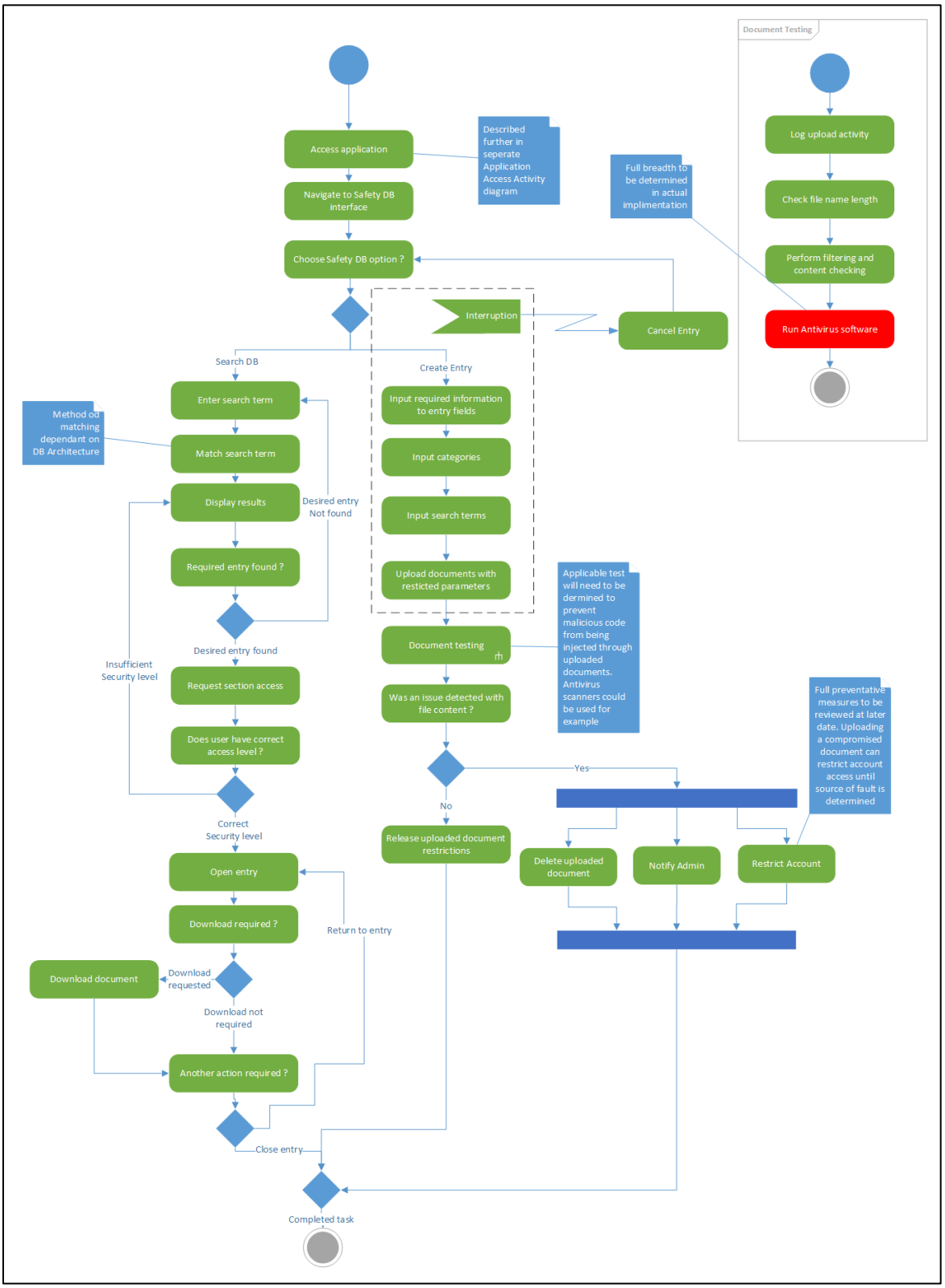

Figure 4 - Process for accessing safety information

The safety database will be created in such a way to allow a user to search, interact and create new entries within the database. The software functionality to access database content is implemented using a CRUD approach. The underlying database tables will contain appropriate access levels to limit access to data.

Users can search any entry within a specified category or set of keywords. Within a safety entry, a user can download any related documents and an overview of an entry. Access levels will also be considered before accessing a safety entry.

When creating an entry, correct required information must be input. Any uploaded documents will be analysed for any potential security vulnerabilities.

6. Technical Overview

6.1 Patterns and Methodology

As mentioned in section 3.2 Architecture, the solution's core is a monolithic application created in Python. Its implementation loosely follows a layered architecture approach without adhering to formal design patterns such as MVC. Overall, the application is divided into three primary layers, as shown below:

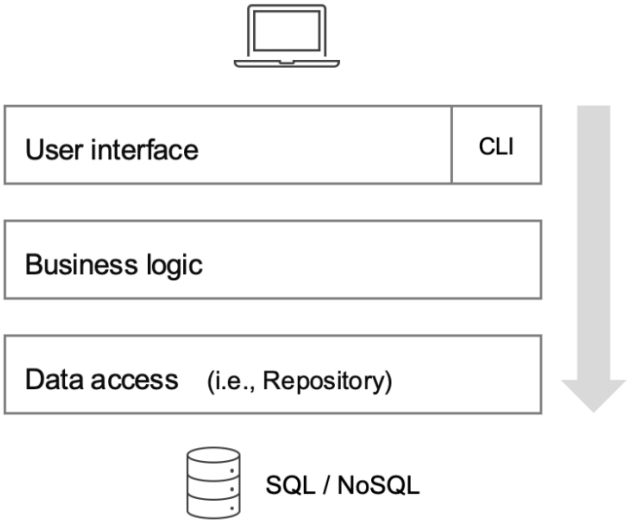

Figure 5 - Monolithic application architecture

User Interface. Processes user input through a command-line interface (CLI) or an application programming interface (API). The API exposes only a subset of the complete functionality (See Appendix 8.2).

Business Logic Implements several object-oriented concepts such as *encapsulation*, *inheritance*, *composition*, and *polymorphism*. Increasing the application's maintainability is achieved by focusing on the Single Responsibility Principle (SRP). SRP facilitates high cohesion and low coupling of code (Ampatzoglou et al., 2019). Due to time constraints, however, advanced techniques such as dependency injection are not applied.

Data Access layer abstract the details of the data storage solutions away from the business logic, allowing for a more loosely coupled application architecture.

In terms of **software design**, the following patterns are considered: *Singleton* pattern (e.g., authentication and authorisation classes), *Factory method* pattern (e.g., user creation), and *Repository* pattern (encapsulates data access logic). Other patterns might also be adopted during application development.

6.2 Required Tools

Table 4 - Solution implementation tools

Name	Description
Flask	Popular Python micro web framework that takes care of web server creation, allowing developers to focus on business logic when creating REST APIs
Postman	Collaboration platform for API development, documenting, and testing
PyCrypto	It offers a collection of secure hash functions and encryption algorithms
Pylint	Static analysis tool that looks for programming errors in Python source code and can help with code refactoring by enforcing a good coding standard
PyMongo	Contains tools for interacting with MongoDB database using Python. This driver module is recommended for use by the creators of MongoDB
sqlite3	A library that provides a lightweight disk-based SQL database and interface that is compliant with the DB-API 2.0 specification as described by PEP 249
unittest	The standard library module for unit testing support. Originally based on JUnit, this library provides a rich set of tools that aid in test automation

Note: The development team could introduce additional libraries or tools during the implementation phase.

6.3 Technical Challenges

- There is a need to organise and control revisions of the software which is addressed by using an appropriate **version control system** such as Git (if supported in the Codio environment).
- Lack of in-depth understanding of cryptographic protocols and information **security expertise**. The solution may resort to standard/popular cryptographic libraries that might not fit for purpose.

- Existing knowledge of the Python development ecosystem could lead to suboptimal choices of external libraries/tools.
- Using Python to implement familiar software design patterns such as singleton or repository.
- Lack of experience using Flask to develop Python APIs.

7. References

Ambler, S. (2003) *Elements of UML Style.* Cambridge: Cambridge University Press

Ampatzoglou, A., Tsintzira, A., Arvanitou, E., Chatzigeorgiou, A., Stamelos, I., Moga, A., Heb, R., Matei, O., Tsiridis, N. & Kehagias, D. (2019) 'Applying the Single Responsibility Principle in Industry: Modularity Benefits and Trade-offs', *23rd International Conference on the Evaluation and Assessment in Software Engineering (EASE' 19)*. Copenhagen, 15-17 April. New York: Association for Computing Machinery. 347-352.

GDPR: European Commission. (2019) Data protection. Rules for the protection of personal data inside and outside the EU. Brussels: European Commission.

Kuryazov, D., Jabborov, D. & Khujamuratov, B. (2020) 'Towards Decomposing Monolithic Applications into Microservices', *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*. Tashkent, 7-9 October. New York: IEEE. 1-4.

Mitre (2017) Weaknesses in OWASP Top Ten.

NASA (2007) Final Report of the International Space Station Independent Safety Task Force

Triartono, Z., Negara, R. & Sussi (2019) 'Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorisation System', *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. Bandung, 18-20 September. New York: IEEE. 259-263.

Umar, M. (2019) Comprehensive study of software testing: Categories, levels, techniques, and types. *International Journal of Advance Research, Ideas and Innovations in Technology* 5(6): 32-40.

Vitali, R. and Lutomski, M.G., (2004). Derivation of failure rates and probability of failures for the international space station probabilistic risk assessment study. In Probabilistic Safety Assessment and Management (pp. 1194-1199). Springer, London.

8. Appendix

8.1 Process: Application Access with swim lanes

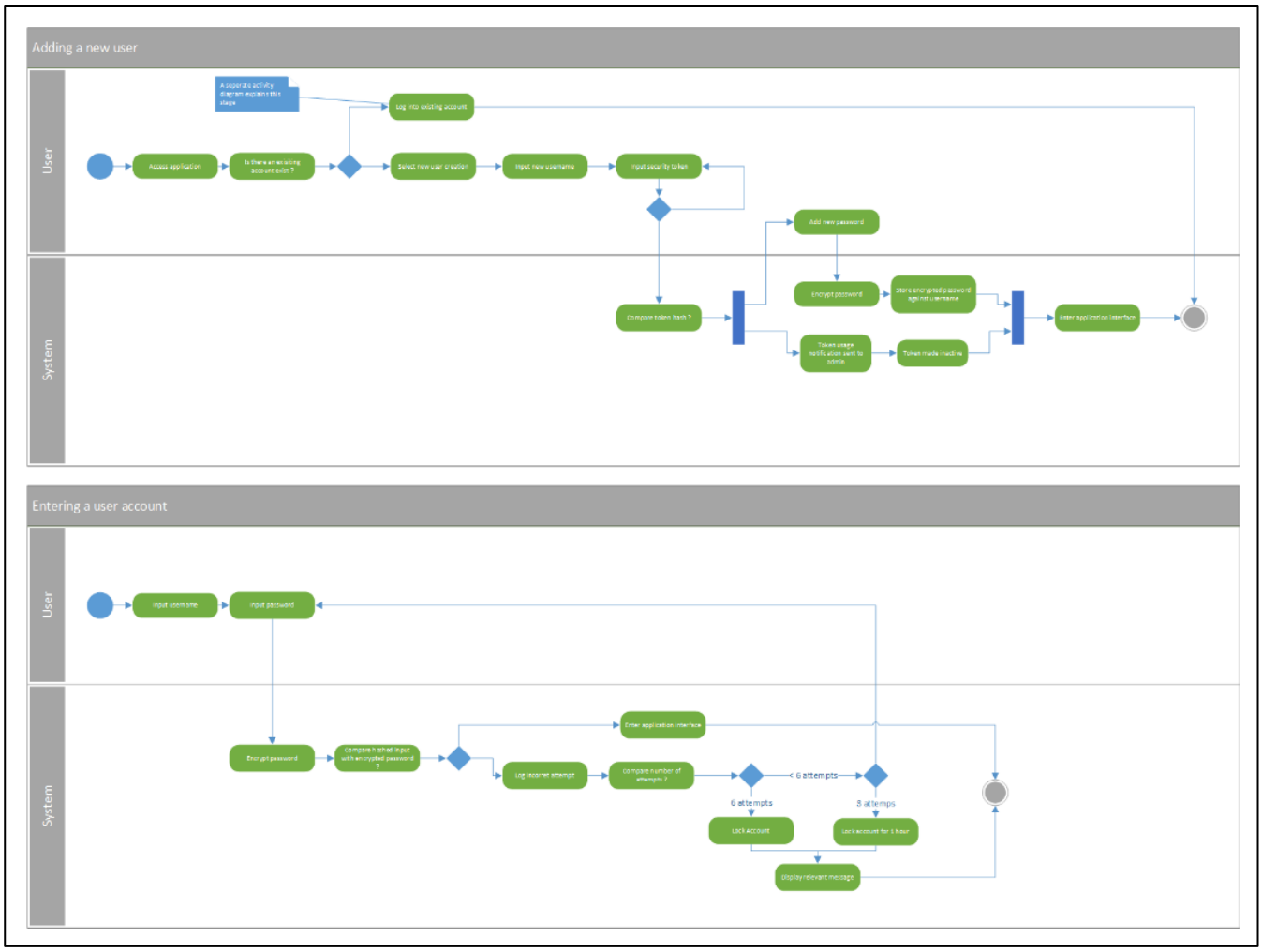

Figure 6 - Activity flows (with swim lanes)

8.2 APIs and microservices

Figure 7 - REST API implementation

The diagram above depicts how an application programming interface (API) is leveraged to handle interactions between users and an underlying information system. The API approach utilises Representational State Transfer (REST) architectural style based on HTTP verbs—GET, POST, PUT and DELETE— to operate on data. Each request is made against resources exposed via an Uniform Resource Identifier (URI) and typically result in responses formatted as JSON, although other formats such as XML are supported too.

The layered architecture presented above (as well as in Figure 5) is an example of **technical partitioning** a software system in which software components are "grouped" based on technical usage. Other examples of technical partitioning are the *microkernel architecture* approach where additional capabilities (plugins) are added on top of a minimalistic core OS. Another example is structuring components based on their business domain. This partitioning style is often referred to as *Domain-Driven-Design* (DDD), and features loosely coupled domain services that communicate with other software components across a network using a protocol called *Service-Oriented Architecture* (SOA). *Microservices Architecture* (MSA) is a variant of SOA that has gained wide adoption in the industry and tend to be finer grained than domain services than SOA.