Secure-Software Development: International Space Station Project

Team 1: The Ambassadors

## System Brief

Our system features two locations, earth and the ISS. Our system has two types of users: ground control staff and astronauts. The ISS has an inventory of resources – such as oxygen and food supplies - required to keep the space station running. There are two types of flights between earth and the ISS: manned and unmanned (NASA, 2010). Interfaces between the space station, ground control and the server will use secure facilities. The project will be developed as a distributed client-server web application, and will also run as a monolithic project (Al-Debagy & Martinek, 2018).

## System Assumptions

In reality, the location of the ISS during its orbit has an effect upon which system options are available, such as when a shuttle can be launched (NASA, 2007). In our system, we will assume the ISS is always at a convenient location for shuttle launch and communication.

## Functional Requirements

* Both users will have a different interface to access the features specific to them.
* Ground control can create astronaut accounts. Ground control oversees shuttle flights (shuttle location, travel percentage, etc.).
* Ground control can add astronauts to space ships and send them to and from the ISS.
* The astronaut's interface allows them to access the exercise regimes which monitors their performance.
* The astronauts can see their performance metrics and the ground control users can oversee all the performance metrics of the astronauts.
* The ISS automatically orders resources that are running out. When a resource has been ordered, it is added to the next flight to the space station.
* Ground control initiates the manned flights, the unmanned flights are scheduled periodically to transfer any pending orders of resources.
* There are a fixed number of docks on the ISS and the flight/docking system ensures that flights are not organized to dock at the same dock.

## Non-functional requirements

* The system will be available for use 24 hours a day, 7 days a week.
* Regulatory – The project will comply with the GDPR (Appendix A)
* Security - Mitigations will be taken to protect the attack surfaces and sensitive data (Section 6, Appendix E)
* Recoverability –The system mustn't crash. We will use error handling to catch exceptions and handle them appropriately. We will make our database transactions ATOMIC so that if any part of the transactions fails the whole transaction is cancelled and no fragmented data is added.
* Maintainability – We will use good OOP practices to keep the codebase maintainable. We will provide some documentation to help other developers understand the system. (Section 9)
* Domain-specific requirements - Data transfer will be affected by latency, as the ISS is a considerable distance from the control office. In addition, there will need to be a backup internet connectivity service in the event the main internet service fails. We do not want to lose communication between the ISS at any stage.

## Protocols

We will use the HTTPS secure protocol for communications to the server because this protocol enables secure communications (Krishna Madasu & Eltaeib, 2015).

## Attack surfaces

Web applications that are not properly secured threaten the data and privacy of users. We have made use of the OWASP Top Ten security risks (OWASP, 2017) to help us identify security threats relevant to our system.

|  |  |  |
| --- | --- | --- |
| **Surface** | **Vulnerability** | **Mitigation(s)** |
| Database | We plan to use an SQL database; this would be vulnerable to injection attacks. The ‘principle of least privilege’ will be applied to ensure all code modules and users only have access to what is necessary. | We will encrypt data so it is useless to anyone without the key. We will make use of a database layer that enforces correct usage. Additionally, we plan to use the ‘salting’ (hashing with additional characters) technique for password storage (Bambang Sugiantoro, 2019). |
| REST API | We plan to use a Flask Rest API. This REST API would be vulnerable to denial-of-service attacks | We will encrypt the communications between the client and server. We will use **monitoring** to monitor the request frequency of API calls to try and identify DOS (denial of service) attacks (Prakash et al, 2016). |
| Authentication | The users of our system have a lot of responsibility when organizing flights and resources. It is of paramount importance that our system has adequate restrictions for authenticated users. | Different levels of access control will be provided to different user types. We will use multi-factor authentication to avoid broken authentication. |
| Command-line interface | Our system will allow users to share data entered through the interface. This opens a vulnerability to cross-site scripting where malicious users can inject malicious code through the client interfaces. | We will use input validation on any user input. We will also be careful that our input validation itself is not susceptible to attacks such as a Regular expression Denial of Service attack (Kirrage et al, 2013). |
| Python and additional library's | Python has known vulnerabilities such as its dangerous ‘pickling’ features. We will make use of third-party libraries to develop our system. These libraries can have their own vulnerability's. | We will research any vulnerabilities of libraries and python features we use. |

Logging - We will use logging at various layers to provide insight into any problems or attacks when analyzed.

## Tools and library's

* Python3
* SQLite database package
* Flask (Rest API library)
* cryptography (A python package for cryptography)
* PyCharm IDE (which features a linter)

## High-level design UML diagrams

We have created a static structural description of the system, detailing the classes required (Appendix C).

We have produced a behavioral activity diagram (Appendix D) to detail the main activities of the logistics operation in the space station.

## Architecture and patterns

Our system will be designed to work in a client-server architecture as well as a monolithic architecture. Both designs will make use of a three-tier architecture for the separation of concerns (IBM Cloud Education, 2020). The three tiers will be, interface, business and data storage. Monolith designs require the whole system to be updated at once, the client-server architecture, however, will allow either of the clients or the server to be updated individually.

We will avoid ‘anti-patterns’ in our code (Jaafar et al, 2015). We will use design patterns to allow us to create modular code that is extensible and easily understood by other developers. We will use suitable design patterns for our project as detailed in (Appendix B).

## References

* Al-Debagy, O., Martinek, P. (2018). A Comparative Review of Microservices and Monolithic Architectures. Available from: <https://arxiv.org/ftp/arxiv/papers/1905/1905.07997.pdf> [Accessed 06 June 2021]
* Bambang Sugiantoro, S. (2019). Analysis of Password and Salt Combination Scheme To Improve Hash Algorithm Security: <https://thesai.org/Publications/ViewPaper?Volume=10&Issue=11&Code=IJACSA&SerialNo=58> [Accessed 06 June 2021]
* Gholami, A., Laure, E. (2016). SECURITY AND PRIVACY OF SENSITIVE DATA IN CLOUD COMPUTING: A SURVEY OF RECENT DEVELOPMENTS. Available from: <https://arxiv.org/ftp/arxiv/papers/1601/1601.01498.pdf> [Accessed 06 June 2021]
* IBM Cloud Education. (2020). Three-Tier Architecture. Available from: <https://www.ibm.com/uk-en/cloud/learn/three-tier-architecture> [Accessed 07 June 2021]
* Hoofnagle, C. J., Sloot, B. van der and Borgesius, F. Z. (2019) ‘The European Union general data protection regulation: What it is and what it means’, *Information and Communications Technology Law*, 28(1), pp. 65–98. DOI: <https://doi.org/10.1080/13600834.2019.1573501> [Accessed 07 June 2021]
* Jaafar, M. et al. (2015) Evaluating the impact of design pattern and anti-pattern dependencies on changes and fault. Available from: <https://www.researchgate.net/publication/273898577_Evaluating_the_impact_of_design_pattern_and_anti-pattern_dependencies_on_changes_and_faults> [Accessed 07 June 2021]
* Kirrage, J. et al. (2013) Static Analysis for Regular Expression Denial-of-Service Attacks. Available from: <https://www.cs.bham.ac.uk/~hxt/research/reg-exp-sec.pdf> [Accessed 07 June 2021]
* Knott, N. (2018) ‘The General Data Protection Regulation’, *Faculty Dental Journal*, 9(2), pp. 54–57. DOI: <https://doi.org/10.1308/rcsfdj.2018.54> [Accessed 06 June 2021]
* Krishna Madasu, V., Eltaeib, T. (2015). Web authentication and authorization and Role of HTTP, HTTPS Protocol in Networking. Available from: <https://www.researchgate.net/publication/274074810_Web_authentication_and_authorization_and_Role_of_HTTP_HTTPS_Protocol_in_Networking> [Accessed 06 June 2021]
* Lambrinoudakis, C. (2018) The general data protection regulation (GDPR) Era: Ten steps for compliance of data processors and data controllers, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer International Publishing. DOI: <https://doi.org/10.1007/978-3-319-98385-1_1> [Accessed 07 June 2021]
* Lou, C. et al. (2019). Comprehensive and Efficient Runtime Checking in System Software through Watchdogs. Available from: <https://www.cs.jhu.edu/~chlou/paper/watchdog-hotos19-preprint.pdf> [Accessed 06 June 2021]
* Martin, Y. S. and Kung, A. (2018) Methods and Tools for GDPR Compliance Through Privacy and Data Protection Engineering, *Proceedings - 3rd IEEE European Symposium on Security and Privacy Workshops, EURO S and PW 2018*, pp. 108–111. DOI: <https://doi.org/10.1109/EuroSPW.2018.00021> [Accessed 07 June 2021]
* Murphy, J. F. A. (2018) The General Data Protection Regulation (GDPR), *Irish Medical Journal*, 111(5), p. 747. DOI: <https://doi.org/10.1007/978-3-319-57959-7> [Accessed 07 June 2021]
* NASA. (2010). Reference guide to the international space station. Available from: <https://www.nasa.gov/pdf/508318main_ISS_ref_guide_nov2010.pdf> [Accessed 06 June 2021]
* NASA. (2007). Final report of the international space station independent safety task force. Available from: <https://www.nasa.gov/pdf/170368main_IIST_%20Final%20Report.pdf> [Accessed 06 June 2021]
* OWASP. (2017) OWASP Top Ten. Available from: <https://owasp.org/www-project-top-ten/> [Accessed 07 June 2021]
* Prakash, A. (2019). Detection and Mitigation of Denial of Service Attacks Using Stratified Architecture: <https://www.sciencedirect.com/science/article/pii/S1877050916305014> [Accessed 06 June 2021]

## Appendix

Appendix A

**GDPR Considerations**

During design, implementation, and continued use of this project, we will ensure compliance with the GDPR (General Data Protection Regulation) (Hoofnagle, Sloot and Borgesius, 2019)(Knott, 2018). As such, the following protocols will be put in place:

* Designate a DPO (Data Protection Officer) (Murphy, 2018).
* Software design to focus on protection of privacy and personal information (Lambrinoudakis, 2018). This includes all medical and exercise history relating to astronauts and other employees(Voigt and von dem Bussche, 2017). (Appendix E)
* All personnel information will be protected from unauthorized access as well as be pseudo-anonymized, as well as encrypted. This includes but is not limited to names, identity numbers, date of birth, address, contact details, pay, bank information, education, and position. All former employees’ information shall be afforded the same privacy (Martin & Kung, 2018)
* Internal communications shall be kept private and secure through encryption. (The design of all communications method will follow these principles.)
* Any data breach requires individuals to be notified and an investigation to begin immediately (Lambrinoudakis, 2018).

Appendix B

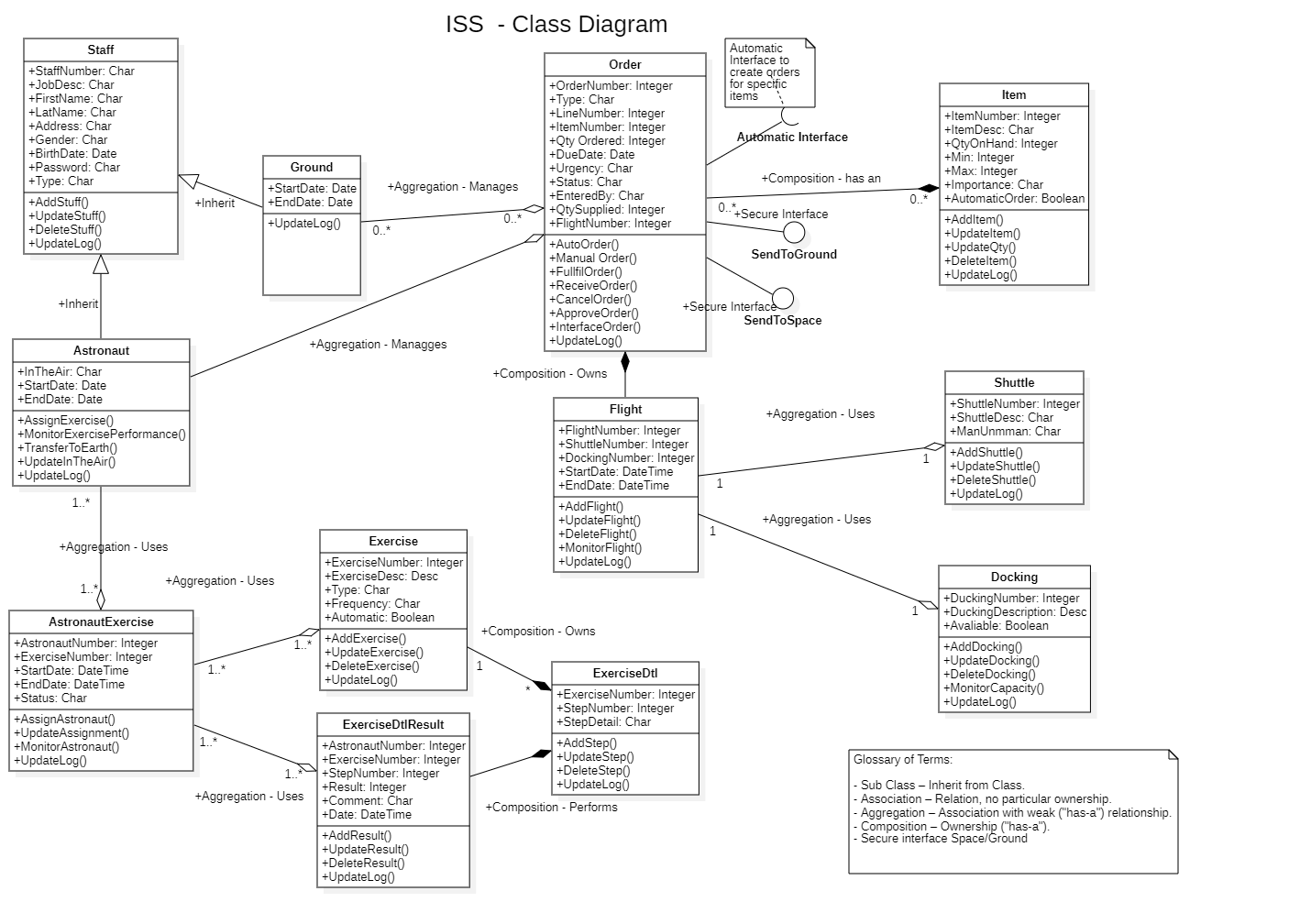
**Appropriate Design Patterns**

* Strategy design pattern – This pattern allows for the dynamic choice of an algorithm to be executed. This could be useful for our exercise system where a different algorithm may be executed based on which exercise regime is selected.
* Adapter design pattern – The adapter pattern allows two incompatible interfaces to communicate. This pattern will be very useful for converting to and from the ‘transport’ layer that the client-server communication purposes.
* Observer design pattern – The observer pattern decouples the observers of events from the subject. The observer pattern may be useful for the ISS resource management feature. For example, observers may subscribe to an ‘oxygen low’ event.
* Singleton pattern – The singleton pattern is a way of ensuring only one instance of a class exists. Considering that our system will only every have ‘one’ ISS this pattern may be useful.

The ‘anti-patterns’ we aim to avoid include the ‘god object’ pattern -where one class has too many responsibility's -, the ‘boat anchor’ pattern – where code is left in the project that is not needed (we can capture this code in revision control) - and magic numbers – where naming is not explicit (we will use explicit naming).

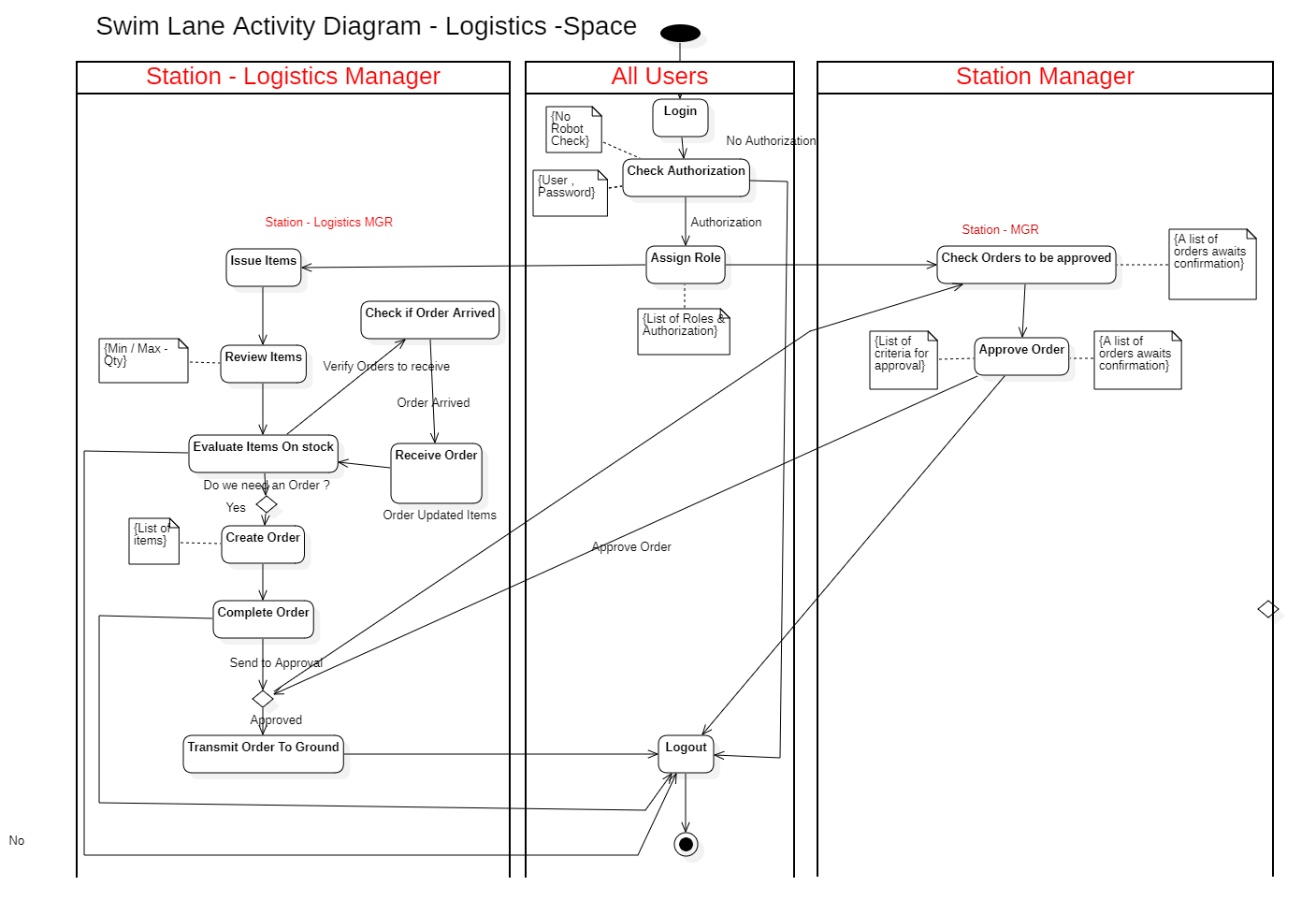
Appendix C

**Class Diagram**



Appendix D

**Activity Diagram**



Appendix E

**Sensitive Data**

We consider the users’ passwords, exercise performance metrics and the ISS inventory to be critically sensitive data (Gholami & Laure, 2016). Data such as shuttle location are considered less sensitive. We have highlighted that the data and measurement of the oxygen are mission-critical and mistakes could cause serious harm. We have decided to use a system watchdog (Lou et al, 2019) that will monitor the performance to make sure the software is working as expected. This watchdog will monitor that the data changes at the expected rate, I.e., The oxygen level reduces.