Secure Software Development

Team Meetings for Team 4

Notes:

Date	18 Aug 2021	**Meeting ID**	2
Attendees	Andrey Smirnov; Taylor Edgell; Michael Justus;		

Agenda

- Discussion of the ISS domain.
- Discuss the problem to be solved, including any challenges.
- Identify (if any) specific approaches to the challenges (academic material if necessary).
- Identify requirements of ISS problem.
- Agree on high-level solution design approach i.e., Activity diagram, Object diagram
  *Note: Design Proposal does not require **detailed** component/class diagrams.*
- Identify assumptions.
- Discuss list of tools or libraries.

Minutes of Meeting

ISS Domain.

- Discussing the domain of the ISS project, the team felt the domain requires up-front planning, but cannot cover all exceptional cases and things that could go wrong.
- The team considered the referenced document quite good to recommend the type of improvements required by NASA team. We agreed that it is safe to focus on their requirements for our project as they already have robust systems.
- Risks related to the ISS domain were considered. However, the team felt the public nature of risks is not good and a member expressed how they did not derive any requirement for public access from the referenced document.

- We felt that everything concerns encryption and secure APIs. Why would we build a public API?
    - The team considered API to sit atop a monolithic design.
    - Caution raised to not crazy on microservices
    - Focus is on authentication at public touchpoints.
    - Question is: what should we expose through the API?
- The team considered focus on access control and best practices employed by NASA. They felt this would allow the team to focus on access control skills.
    - In conjunction with this, consideration was given to *role-based* access.
- The team agreed to focus on the main functionality, which concerns a "safety database".
    - The team considered the volume of content for delivery, and the conclusion was the volume is not a requirement to achieve a good team mark.
- OWASP was considered; specifically, how many OWASP points to address?
    - Recommended is a list of top 10 and against each one we demonstrate how we can combat it, for example, using password/username and encryption.

Problem and Challenges.

- Lack of programming experience with Python.
- Unfamiliarity with Flask.
- Unfamiliarity with Pylint.

Design Approach.

Gathered basic requirements from the reference document and drafted a bare-bones UML model for the team. The reference document pulled out the need for a "safety database". This was combined with the Part 1 brief, and consideration given to how such a brief could be fulfilled from a user interface.

The team discussed UML modelling:

1. Which UML diagrams to deliver in the report.
    a. Class diagrams (most useful) shall show the overall solution design.
    b. Use cases (based on listed requirements)
    c. Activity diagram
        i. Roughly three will be delivered, dictated by time constraints, for example, user accounts, report records and issues.
        ii. Not every flow in the monolithic application shall be modelled.
    d. Sequence diagram
        i. These were considered but rejected due to complexity of size and effort.
    e. ERD diagram

        i. These were considered due to lack of value and were considered low priority because the brief is not a SQL-oriented set of data structures.
    f. Security viewpoint
        i. Considered to be valuable to add but marked as low priority. The team felt there is benefit to such visual diagram showing the touchpoints of security between classes/components.
2. README file
    a. References will be added here.

Requirements.

ID	Description	What is it?
REQ01	**Upload reports**	Authenticated user uploads report to system
REQ02	**View reports**	Authenticated user can request and view report data.
REQ03	**Download reports**	Authenticated user can download report data.
REQ04	**Manage users**	Admin can manage external users within the system
REQ05	Project management* (probably not needed)	Capture NASA project information.
REQ06	Crew Management	Capture information of crew members aboard the ISS.
REQ07	**Safety Management**	Capture issues related to safety aboard the ISS.

Solution design approach.

Q: API first? Or Monolithic first?
The team felt API-first makes sense with public consumers. As this module delivers an internal system, it was agreed to take a monolithic-first approach and add APIs atop later. Also, since everything is at national security level, it is better to start with a monolithic approach and then put API layer in place at a later stage.

The team adopted a monolithic approach that can easily be broken into modules; such an approach builds on what has come before.

Assumptions.

The teams agreed on the following assumptions:

- Assumptions are added as the project progresses.
- OWASP and Security:
    - Each OWASP Top Ten attack will be addressed in Part 2 using (vague) terms.
    - We not dealing with a subset of DDOS and so cannot emulate load balancers, traffic monitoring; though, we will show awareness of that these happen in real life.

- Testing:
    - Time constraints constrain amount of possible testing.
    - We don't do strange use case scenarios for testing. Testing remains fundamental and UI is a cherry on the top.
    - We require simple unit tests, then integration on the level of component testing, and can ignore UI.
- User Interface
    - We are demonstrating a command line application and therefore will not implement UI.
- Data/Architecture:
    - We do not support event streaming scenario using Kafka as the module is not about event-driven architecture.
    - Data replication is out of scope
    - We use SCRUM iterations to implement basics of the skeleton; it is natural that the code will be extended during implementation
    - API design is not major concern now and is delegated to the Part 2.
- Diagrams:
    - Time constraints prevent copious diagrams.

Tools & Libraries.

- Python
- The team will leverage what is recommended, want to add in extra, that's fine
- Data storage will look to leverage NoSQL, SQLite, or MongoDB for reports.
- Flask

Other:

- If time permits, we will add and create new additions.
- Diagramming will focus on the bare bones, and other team members can focus on the security viewpoint.
- Shall we maintain change logs?
    - Architectural changes will require change log, not everything only as much as is important.
- We shall provide comments on class diagram.
- Shall we capture why we use a specific technology?
- Consider scalability and look at other
- Utilise reference material from Cathryn Peoples
- Academic references will be added to assumptions, where possible.