Hazard Analysis
Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Name(s)	Description of changes
Date2	Name(s)	Description of changes
...	...	...

Contents

[You are free to modify this template. —SS]

1 Introduction

Test Text

[You can include your definition of what a hazard is here. —SS]

2 Scope and Purpose of Hazard Analysis

[You should say what **loss** could be incurred because of the hazards. —SS]

3 System Boundaries and Components

3.1 Front End (FE)

Potential Hazards:

- **User Input Errors:** The frontend may not properly validate inputs (e.g., incorrectly labeled data or unauthorized access attempts).
- **UI Design Flaws:** Poor usability could cause users to make mistakes, such as submitting incorrect labels or missing important instructions.
- **Cross-Site Scripting (XSS):** JavaScript code that improperly checks user inputs could allow malicious scripts to be executed in the user's browser.
- **Session Hijacking:** Improper management of session tokens and cookies could allow attackers to impersonate users.

Mitigations:

- Implement thorough input validation and provide clear feedback to users.
- Ensure that critical actions like labeling are supported by warnings or confirmations to reduce errors.
- Use Content Security Policy (CSP) headers and HTML escaping techniques to prevent XSS attacks.
- Secure cookies with `HttpOnly` and `Secure` flags and employ strong session management practices.

3.2 Back End (BE)

3.2.1 Login Service

Potential Hazards:

- **Authentication Failure:** Unauthorized access could allow malicious actors to label data incorrectly or access sensitive customer information.
- **Session Hijacking:** If session management is not handled correctly, attackers could hijack valid user sessions.

Mitigations:

- Implement strong authentication mechanisms (e.g., two-factor authentication) and monitor for suspicious login activity.
- Secure cookies and session tokens, implementing timeout features.

3.2.2 Money Service

Potential Hazards:

- **Payment Failures:** Incorrect payments could cause disputes between the labelers and the platform, leading to operational delays.
- **Financial Data Breaches:** Weak encryption or improper payment processor integration could expose sensitive financial data.

Mitigations:

- Secure payment processing with encryption and multi-step verification for payout transactions.
- Implement industry-standard encryption for payment details and integrate with PCI-DSS-compliant payment gateways.

3.2.3 Task Delegation Service

Potential Hazards:

- **Task Misallocation:** If tasks are assigned to unqualified labelers, the quality of data labeling could decrease, leading to flawed training data.

Mitigations:

- Employ task delegation algorithms that consider labelers' accuracy and expertise.

3.3 Database (DB)

Potential Hazards:

- **Data Corruption or Loss:** Corrupted or missing data, especially labeled data, can introduce significant inaccuracies in model training and analysis.
- **SQL Injection:** If queries are not parameterized, attackers may manipulate query inputs, resulting in unauthorized access or data manipulation.

Mitigations:

- Implement backups and data redundancy mechanisms to prevent loss or corruption.
- Use parameterized queries or prepared statements to prevent SQL injection.
- Ensure proper data validation before storing data.

3.4 Machine Learning Task Allocation Model

Potential Hazards:

- **Model Misconfiguration:** The allocation model may incorrectly assign tasks if it misinterprets labeler performance data, leading to inefficiency and low-quality labeled datasets.
- **Bias in Task Allocation:** If the model favors certain users, this could introduce biases in the labeling process, affecting data accuracy.
- **Adversarial Attacks:** Machine learning models might be vulnerable to adversarial attacks that can skew the task allocations.

Mitigations:

- Continuously monitor and test the task allocation model's performance and adjust for fairness and accuracy.
- Implement adversarial training or model-hardening techniques to defend against attacks.

3.5 Libraries (e.g., BoTorch, TensorFlow, PyTorch)

Potential Hazards:

- **Library Bugs or Vulnerabilities:** External libraries may have bugs or security vulnerabilities that can affect system performance or introduce security risks.
- **Version Conflicts:** Dependency issues may arise when integrating BoTorch with other machine learning libraries.

- **Optimization Failures:** Incorrect hyperparameter optimization might result in poor allocation of labeling tasks.
- **Adversarial Model Attacks:** TensorFlow or PyTorch models may be susceptible to adversarial attacks, leading to incorrect classifications.

Mitigations:

- Regularly update libraries and conduct security audits to ensure all dependencies are secure.
- Use virtual environments or Docker to manage dependencies and avoid conflicts.
- Rigorously test optimization outcomes using test datasets.
- Implement adversarial defenses, such as adversarial training and data augmentation.

3.6 Python (General Back-End Processing)

Potential Hazards:

- **Performance Bottlenecks:** Python can introduce performance issues, especially in CPU-bound processes.
- **Memory Leaks:** Improper memory management can cause resource exhaustion, leading to crashes.
- **Security Vulnerabilities:** Since Python is dynamically typed, unexpected inputs could lead to runtime errors, potentially compromising data integrity.

Mitigations:

- Use Cython or external compiled languages for performance-critical tasks.
- Regularly run memory profiling tools to ensure no memory leaks.
- Implement input validation and use static analysis tools like `bandit` to enforce security best practices.

3.7 Docker (Containerization)

Potential Hazards:

- **Container Breakout:** Misconfigurations or unpatched vulnerabilities in Docker containers could allow attackers to escape the container and gain access to the host system.
- **Resource Exhaustion:** Poor resource management inside containers could lead to resource exhaustion, affecting system performance.

Mitigations:

- Use security best practices such as running containers with the least privilege, using `Seccomp` profiles, and isolating sensitive workloads.
- Set resource limits on containers to prevent exhaustion of host system resources.

4 Critical Assumptions

[These assumptions that are made about the software or system. You should minimize the number of assumptions that remove potential hazards. For instance, you could assume a part will never fail, but it is generally better to include this potential failure mode. —SS]

5 Failure Mode and Effect Analysis

Design Function	Failure Modes	Effects of Failure	Causes of Failure	Detection	Recommended Action	SR	Ref
Account Creation	User already exists	User can not create an account	Email is duplicated	Compare the email entered with the user database records to see if the email is in use	Notify the user that the email is associated with another account Prompt them to give another email or sign in with the one they entered		H1-1
	Invalid input syntax and length	User can not create an account	1) Email is not valid 2) Password is not strong enough	1) Use regular expressions to detect if the string pattern is valid 2) Use regular expressions to detect if all password requirements are met	1) Notify the user that they must enter a valid email and give an example of a valid email 2) Tell the user what password requirements they have and have not satisfied		H1-2
Log In	Incorrect credentials entered	User can not access application	1) No account with the entered email exists 2) Password does not match records	1) Compare email entered with database records to see if account exists 2) Compare password entered with what is stored in the database for the entered email	1) Tell user account does not exist and prompt them to make one 2) Tell user password is incorrect and prompt password recovery		H2-1
	Excessive permissions given	Users can perform unauthorized actions	Application paths are unprotected	Check user login token each time a new page of the website is accessed	Tell the user to log in Deny access if they try to access a page they should not		H2-2
Labeling Satallite Images	Internet connection is lost	Users can not submit labeled images or navigate the website	Internet connection is weak or power is lost	Device shows no internet connection	Any labeled photos or created projects that have already been submitted have been saved Progress is resumed when connection is re-established		H3-1
	Application is closed	Same as H3-1	Power outage or misclick	Application is no longer running on the users device	Any labeled photos or created projects that have already been submitted have been saved Progress is resumed on log in		H3-2

	Unlabelled data is submitted	Bad data is added to the dataset	Misclick	On submission, application checks that there are as many labels as requested by the job	Reject a submission if no labeling was done		H3-3
	Mass labeling done too quickly	Bad data is added and reward system is abused	Bots have been deployed to make quick labels	User is submitting data at an unreasonable speed	Implement a submission cool down to prevent bot submissions Reward system is based on accuracy		H3-4
Backend Server and API Requests	Server crashes	All services provided by the server are down	Software error on server side	Error found in logs	Monitor errors in logs Notify users that the server is down		H4-1
	API is not responding	All services provided by an API do not work	API service provider is down or overwhelmed	Response from the API has an error code	Retry all API requests after a specific amount of time Monitor errors in logs		H4-2
Data Storage	User account is compromised	User info is exposed and they will be dissatisfied with the application	Lack of encryption and protection of sensitive infomation	User notifies the team of lost reward balance or lost account access	Ensure user passwords are encrpyted when stored Ensure financial transactions are secure Password reset occurs through a trusted source such as email		H5-1
	Duplicate entry occurs	Data inconsistency, unneccesary storage usage, and slower query performance	Lack of constraints/validation	Check the database entries	Ensure the database has unique keys Set up a duplicate key procedure on the database		H5-2
	Database is compromised	Data inconsistency, malicous entries, and data leaks	SQL injection	Check database entries	Use parameterized queries Avoid dynamic SQL strings		H5-3

6 Safety and Security Requirements

[Newly discovered requirements. These should also be added to the SRS. (A rationale design process how and why to fake it.) —SS]

7 Roadmap

[Which safety requirements will be implemented as part of the capstone timeline? Which requirements will be implemented in the future? —SS]

Appendix — Reflection

[Not required for CAS 741 —SS]

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable? sample text
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?
4. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?

8 References

- OWASP. Cross-Site Scripting (XSS). Available at: https://owasp.org/www-community/attacks/xss/.
- The SSL Store. The Ultimate Guide to Session Hijacking (aka Cookie Hijacking). Available at: https://www.thesslstore.com/blog/the-ultimate-guide-to-session-hijacking-aka-cookie-hijacking/.
- Mozilla Developer Network. Content Security Policy (CSP). Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.
- Neontri. PCI DSS Compliance. Available at: https://neontri.com/blog/pci-dss-compliance/.
- Imperva. Session Hijacking. Available at: https://www.imperva.com/learn/application-security/session-hijacking/.

- Label Your Data. Bias in Machine Learning. Available at: https://labelyourdata.com/articles/bias-in-machine-learning.
- Reversing Labs. How to Harden ML Models Against Adversarial Attacks. Available at: https://www.reversinglabs.com/blog/how-to-harden-ml-models-against-adversarial-attacks.
- Hyperopt GitHub Issues. Hyperopt Version Conflicts. Available at: https://github.com/hyperopt/hyperopt/issues/642.
- HiddenLayer. What's in the Box? Understanding Adversarial Attacks on ML Models. Available at: https://hiddenlayer.com/research/whats-in-the-box/.
- Medium. Python Security Best Practices for Writing Secure Code. Available at: https://medium.com/@VAISHAK_CP/python-security-best-practices-for-writing-secure-code-a6a9130e3748.
- Docker Documentation. Seccomp Security Profiles. Available at: https://docs.docker.com/engine/security/seccomp/.