British Columbia Institute of Technology

Network Security Applications Development

Cthulhu Project Proposal

COMP 8800 Major Project
Ashkan Jangodaz

Edgar J Coime
A01003601
ecoime1@my.bcit.ca

2025-11-21

Document Control

Document Information

	Information
Document Owner	Edgar J Coime
Document Owner ID	A01003601
Issue Date	2025-10-12
Last Saved Date	2025-11-21
File Name	Cthulhu Project Proposal

Document History

Version	**Issue Date**	**Changes**
1.0	2025-09-19	Initial Proposal submitted.
1.1	2025-10-12	Proposal draft 1.1 submitted reflecting some feedback.
1.2	2025-10-15	Added Methodology and Design portion highlighting agile approach.
1.3	2025-10-18	Created Test Plan section.
1.4	2025-10-22	Created Scope and Depth section detailing application's microservice architecture.
1.6	2025-10-24	Added a comprehensive Development Schedule and milestones. Detailing all project deliverables and divided the milestone timeline into even parts for a the 12-week development period.

Version	Issue Date	Changes
1.7	2025-10-26	Created diagram for System Architecture, File upload flow, and file download flow.
2.0	2025-10-26	Added further details in Scope and Depth. Outlined the complexity of Microservice arch. and the depth that comes learning about and mastering RabbitMQ.
2.1	2025-11-06	Added Milestone 5: Online Functionality, E2E user tests, and CI/CD pipeline.
2.2	2025-11-06	Removed online functionality requirement in milestone 2 and moved it to milestone 5
2.3	2025-11-08	Added Milestone 2 details about incorporating cloud storage, how Authentication service should be source of truth, and plan to cache tokens/sessions in gateway.
2.4	2025-11-09	Expand Milestone 3 to 4 weeks and add details about designing a file life cycle service database to store persistent data.
2.5	2025-11-14	Expand Milestone 4 to 6 weeks. Add details about service performance requirements virus scanning will be computationally expensive. Add SAGA transaction requirements for if file is malicious.
2.6	2025-11-17	Added more details for Milestone 4 about virus scanning properties, detection requirements, and test acceptance thresholds.
2.7	2025-11-21	Added Milestone 5 details about online functionality and HTTPS thorough valid SSL certs. Add requirement for E2E

Version	Issue Date	Changes
		user testing to ensure Cthulhu platform working properly. Add details about CI/CD pipeline, rollbacks, beta environments, and Docker image registry integration.
3.0	2025-11-21	Submission for Proposal draft 3

Contents

Student Background

My name is Edgar Coime, and I am in my 3rd term of the Network Security Applications option of the Applied Computer Science program in BCIT. As per my option, my area of expertise is mainly on Network Security but throughout my tenure at BCIT I have had plenty of experience creating Full Stack Web Applications. One of the projects that I am most proud of is creating an AI learning assistant in 2022 for the Stormhacks Hackathon where two of my schoolmates and I won first place for our inventiveness and overall usability and appeal towards majority of the audience and judges.

I have also had the pleasure of working in the industry for my COOP term in the Computer Systems Technology program in BCIT. During that work term I served the role as project lead helping guide the project through its desired deliverables and communicating directly with the client translating his requirements into user stories and helping his vision come to life. I was also responsible for onboarding future teams into the project so that they can fully understand the codebase and structure of the live application.

Project Summary

CTHULHU is an Anonymous-first file sharing platform designed to address privacy and usability shortcomings in modern file sharing systems. The application will enable users to upload and share files/folders of up to 1GB with automatic deletion after 48 hours for anonymous users. Authenticated users will have more fine grain control to extend file lifetimes up to 14 days. Built using modern microservice architecture, CTHULHU will use React NEXT for the frontend and Golang for backend services, it will use HTTP as the communication layer between the frontend web application and API gateway, while taking advantage of RabbitMQ message queuing for inter-service communication. This architecture ensures scalability, maintainability, and robust security while providing a seamless user experience without traditional account barriers.

Problem Statement and Background

In today's modern age, file sharing is a continuous part of everyday life and has evolved from just a convenience to an essential collaboration tool across industries and education. However, current mainstream file sharing platforms like Microsoft OneDrive and Google Drive impose cumbersome steps that increase overall friction in their platform through mandatory account creation, multi-step authentication, and complex file sharing workflows. These barriers prioritize user profiling rather than accessibility and ease of use.

Beyond just general usability issues, existing platforms lack security first design principles. Many mainstream services focus on indefinite data storage and extensive user tracking and profiling over privacy protection. This flawed approach has proven vulnerable: "80% of companies have been subject to cloud security breaches" (Spacelift, 2025) in the past year. This proves that security and anonymity must be a fundamental design principle rather than an afterthought.

Unfortunately due to the nature of file sharing platforms, file upload is necessary and its mechanisms expose systematic vulnerabilities across the industry. As Uddin & Jabr (2016) note, "file upload from the client end [...] without proper security and sanitization methods [...] presents significant security risks." This is especially the case for unrestricted file upload (UFU) vulnerabilities that arise when platforms do not sufficiently validate file properties such as name, type, contents, or size. Chen et al. (2024) further illustrate the importance of taking these attack vectors seriously as "many of these vulnerabilities are not complex to exploit, often requiring no user interaction or special privileges, yet they profoundly affect the confidentiality, integrity, and availability of systems."

CTHULHU seeks to alleviate these issues approaching file sharing with a security-first design mindset. All uploads will undergo strict server-side validation-verifying filetype, name, size, and magic-header contents-to mitigate UFU vulnerabilities. Second, every file will immediately be processed through a cryptographic hash function (SHA-256), which produces anonymized files and a file fingerprint stored separately from the content. Finally, CTHULHU will be enforcing user-designed life cycles: anonymous users expire by default at 48 hours which can be extended to 14 days for authenticated users. Combined these measures will guarantee CTHULHU's file sharing functionality stays secure and that the platform has a privacy-focused approach with its users.

Inventiveness

CTHULHU's innovation is in its focus on temporal file management and anonymity-fist architecture that eliminate traditional barriers while maintain security integrity. Unlike existing platforms that require persistent user accounts and complex authentication workflows, CTHULHU enables immediate file sharing though its account-optional functionality with a fix 48-hour time bound and its assurance of automatic deletion when that time has elapsed. Unlike more contemporary platforms account creation is not mandatory and only enable the user access to more fine grain control over their files.

The system will also be Integrating advanced microservice architecture utilizing RabbitMQ message queuing for inter-service communication. Microservices is an exciting paradigm and will enable the platform to grow according to its needs. It allows us to add additional features (i.e.. Logging service, notification service, virus scanning service, etc.) without radically changing the codebase, which Monolithic architectures would demand given the same change.

Complexity

Building CTHULHU will involve intensive research on microservice architecture and overcoming the challenges that I have never encountered in any of my previous projects. CTHULHU will demand an understanding of distributed systems and how to effectively manage the asynchronous inter-service communication that comes from RabbitMQ's message queue system. It would require implementing a robust message queue protocol, ensuring data consistency across multiple independent services, and making sure that data remains ACID compliant during transactions that may need to be rolled back if catastrophic failure occurs.Event driven architecture is also something that I have never had experience with and is quite infamous for its difficulty. It would require overcoming complex service orchestration challenges and understanding how to handle transactions through CAP theorem and SAGA patterns.

File upload functionality also introduces critical security vulnerabilities that require mitigation strategies for: preventing XSS attacks through malicious file uploads, implementing file type validation and sanitation, defending against path traversal attacks, and many more. The technical complexity starts to balloon when we account for the sophisticated message broker architecture, distributed system fault tolerance, and malware detection integration. Therefore, the complexity for CTHULHU I feel is more than sufficient for this major project.

Technical Challenges

Most of the applications I have built have been primarily monolithic architectures with the occasional mix of microservice through client and server. CTHULHU gives me the opportunity to fully dive into the inner workings of microservice architecture applications especially when incorporating an asynchronous communication layer through RabbitMQ. Through this project I will have to learn about microservice methodologies and communication patterns such as event stream pub/sub, event sourcing, Command Query Responsibility Segregation (CQRS), and the SAGA pattern for distributed transaction management.

Service-to-Service authentication and Authorization in microservices is foreign to me and is completely different to more synchronous communication patterns like HTTP. In HTTP we can make use of existing user sessions but microservices require machine-to-machine communication usually through JWT tokens or mutual TLS. But we also need strict access control policies to mitigate risks from compromised services or malicious lateral movement in the network.

Methodology and Design

Methodology

CTHULHU will follow an agile approach to software development to ensure modular software design that will emphasize flexibility, iteration and continuous feedback. The project will be organized into short development sprints tailored towards the milestones that are outlined later in the document.

Since the system is a microservice architecture platform, development cycles will focus on integrating individual services such as the frontend, API gateway, file manager, authentication, lifecycle, and auth services. This approach will allow for rapid development of new features and allow the platform to explore additional technologies in the future, such as incorporating other programming languages or frameworks as new functionality requires.

Design

Microservices enable CTHULHU to maintain a highly modular architecture, facilitating seamless expansion of functionality as the platform evolves. This design approach also allows the system to leverage distributed computing principles, supporting independent deployment and horizontal scaling. As a result, services experiencing higher demand can run multiple instances dynamically, ensuring improved performance, fault tolerance, and balanced system load.

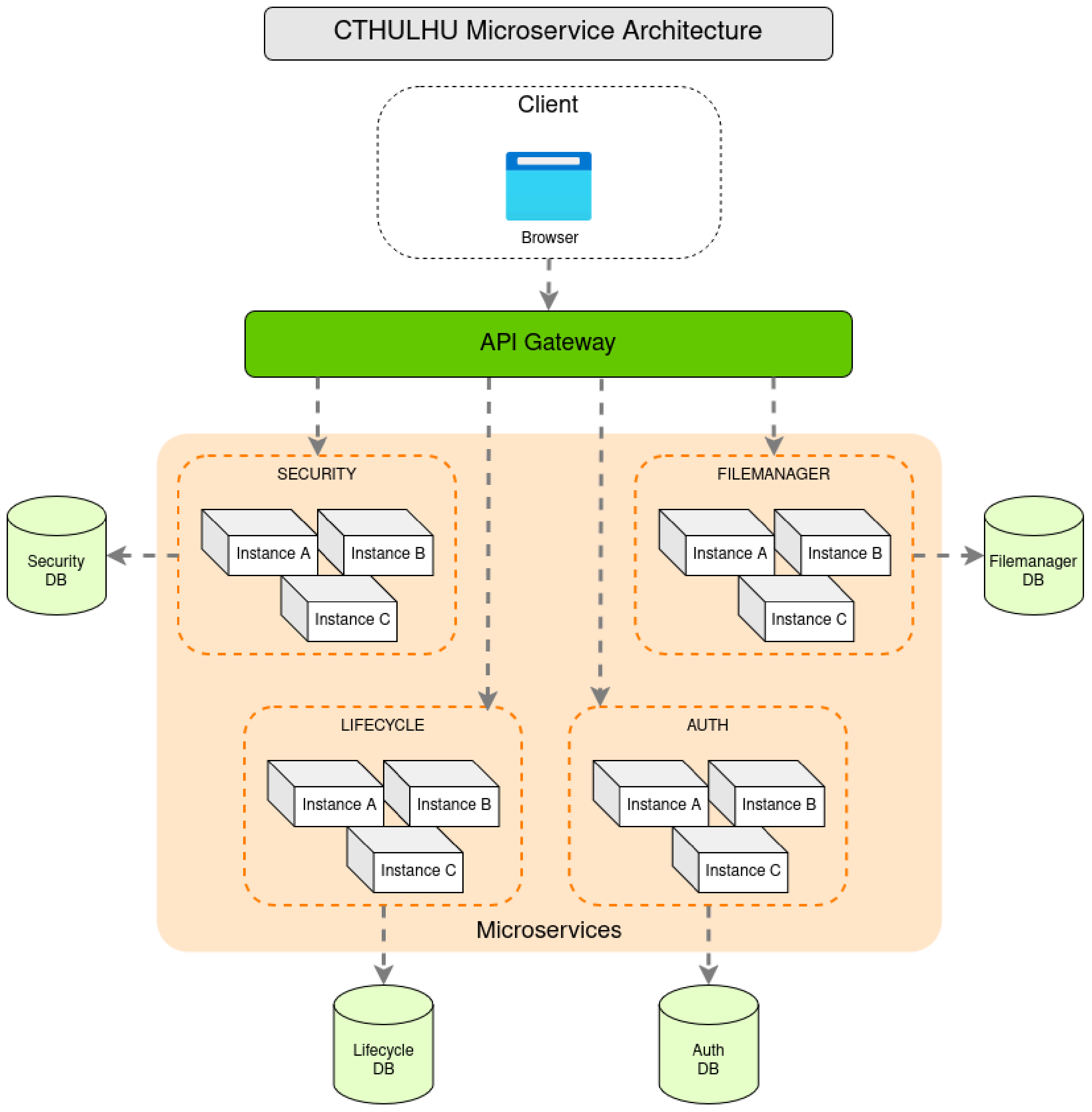

Figure 1: Microservice System Architecture Design of CTHULHU

All microservices within CTHULHU will seamlessly interact and coordinate as if operating within a monolithic architecture. By leveraging message queues and implementing the SAGA pattern, the system will maintain reliable communication and coherent workflows across distributed services. This approach enables the platform to support ACID-like transactional consistency within an inherently asynchronous and decoupled environment, ensuring data integrity while preserving the benefits of scalability and fault isolation.

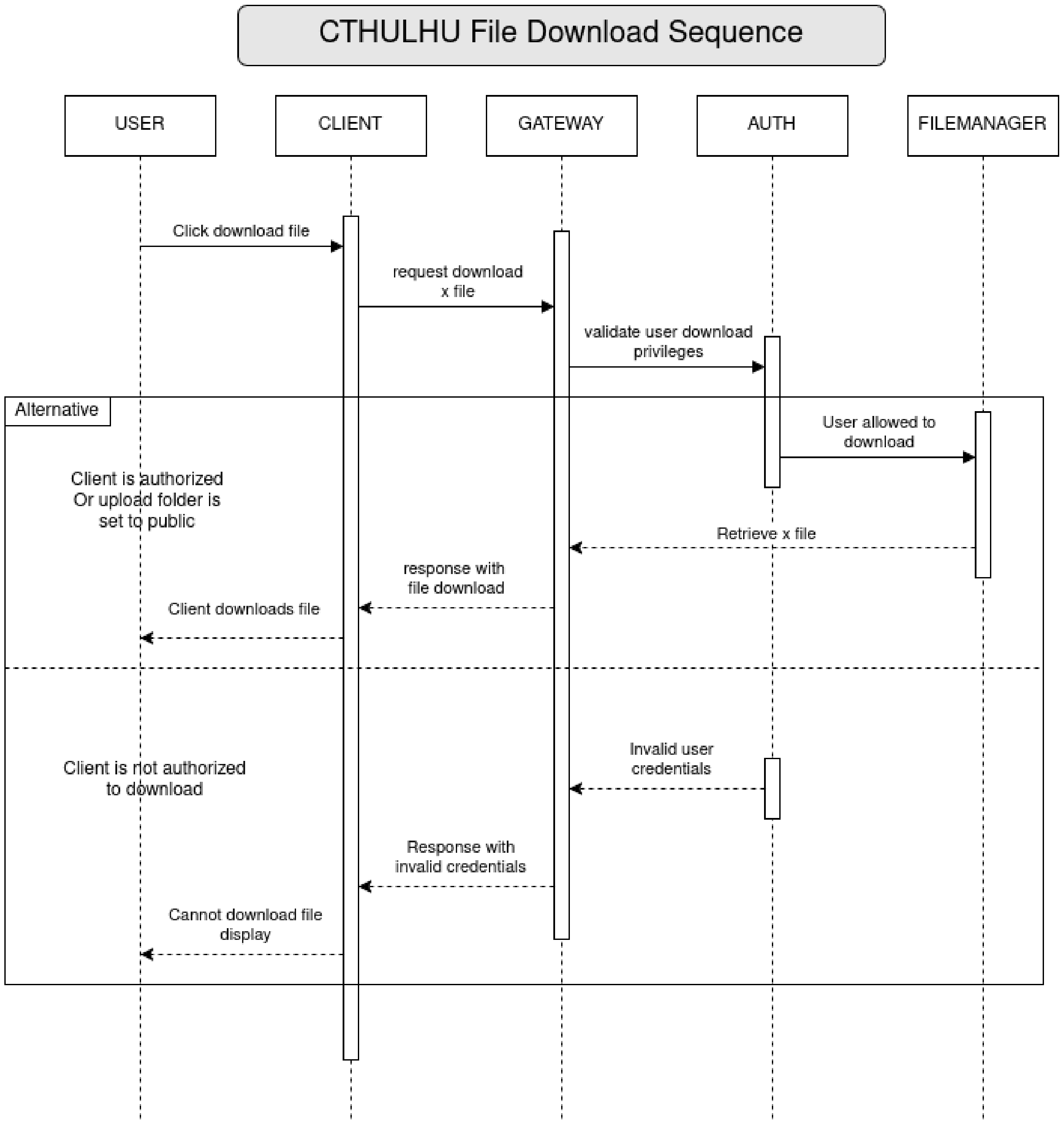
CTHULHU File Download Sequence
USER
CLIENT
GATEWAY
AUTH
FILEMANAGER
Click download file
request download x file
validate user download privileges
Alternative
User allowed to download
Client is authorized Or upload folder is set to public
Retrieve x file
response with file download
Client downloads file
Invalid user credentials
Client is not authorized to download
Response with invalid credentials
Cannot download file display

Figure 2: User download request workflow

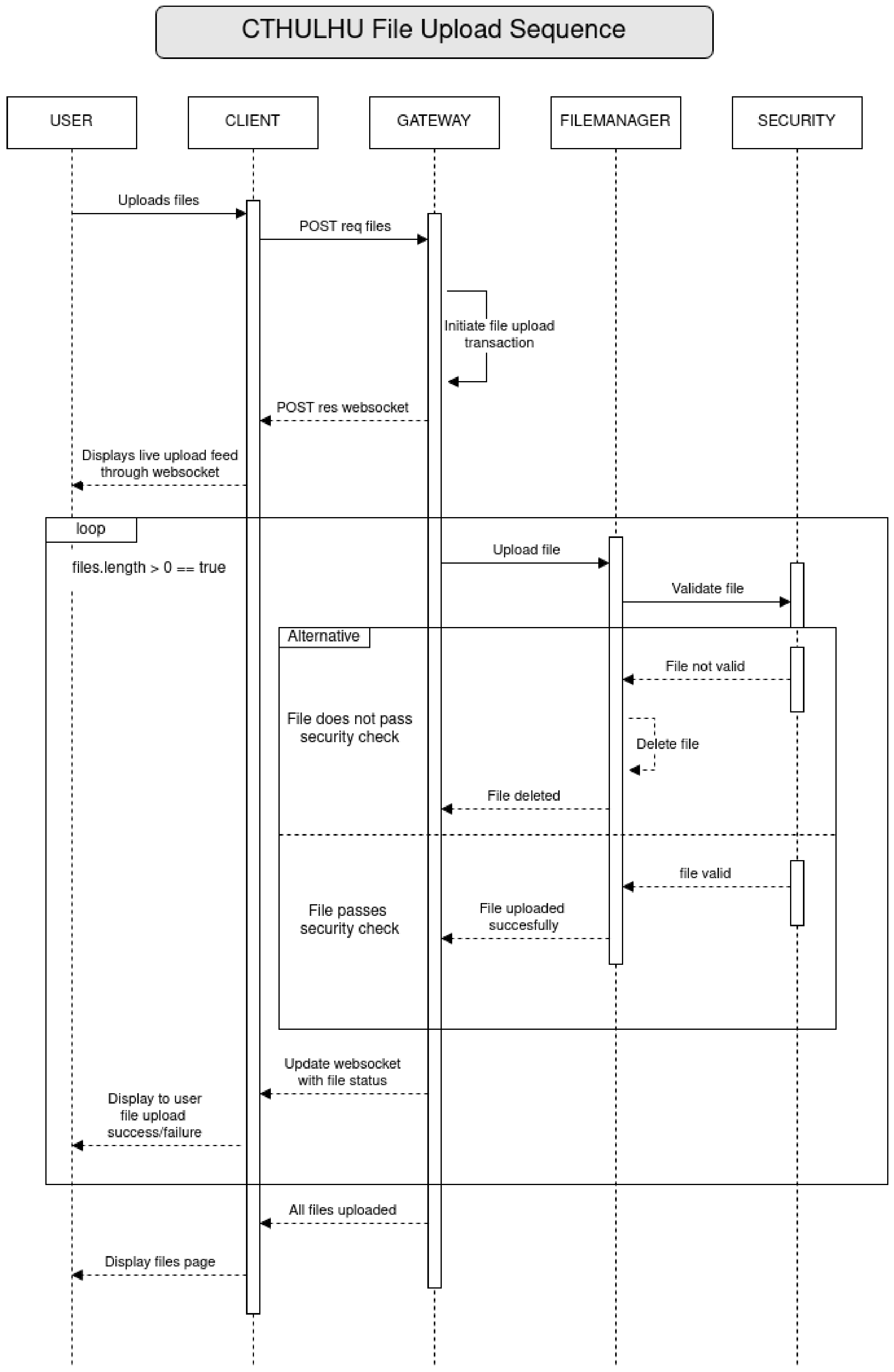
CTHULHU File Upload Sequence
USER
CLIENT
GATEWAY
FILEMANAGER
SECURITY
Uploads files
POST req files
Initiate file upload transaction
POST res websocket
Displays live upload feed through websocket
loop
files.length > 0 == true
Upload file
Validate file
Alternative
File does not pass security check
File not valid
Delete file
File deleted
file valid
File passes security check
File uploaded succesfully
Update websocket with file status
Display to user file upload success/failure
All files uploaded
Display files page

Figure 3: User upload request workflow

Test Plan

CTHULHU’s test plan will use a layered approach do validate reliability, security, and the overall functionality of not only the individual services but how it interacts with the entire system. Each Go-based backend service will undergo unit testing to ensure internal logic correctness, followed by integration testing to verify proper message exchange through RabbitMQ and the API Gateway.

Tools such as Go’s build-in testing framework and postman. Validation success will be measured by zero critical errors, stable throughput, and fully working and expected API responses and file handling limits.

Scope and Depth

Scope

CTHULHU’s scope includes the design, development, and deployment of a microservice based anonymous file-sharing platform that prioritizes user privacy while maintaining and upholding proper security measures. The project seeks to deliver a production ready system enabling anonymous and authenticated file sharing with comprehensive management.

- **Frontend Application:** A modern and responsive client built with next.js that provides an intuitive user interface for file uploads, downloads, and management.
- **API Gateway Service:** Centralized entry point managing all HTTP traffic routing, request validation, and initial authentication/security checks.
- **File manager Microservice:** Responsible for core file operations including upload processing, temporary storage management, download delivery, and metadata tracking.
- **Authentication Microservice:** Implements proper authentication procedures that keep user data safe and secure. Manages user sessions, access tokens, user file inventories, and enforces authorization for protected operations.

- **Security Microservice:** An autonomous service dedicated to scanning uploaded files for malicious content and industry-standard antivirus engines.
- **File life cycle Microservice:** Manages the temporal features of file availability by tracking upload timestamps, enforcing retention policies (48 hours for anonymous users), and scheduling automated deletion.

Out of Scope

- **Advanced Encryption features:** Client-side encryption where users can manage their own encryption keys.
- **Mobile Native application:** No iOS or Android native applications.
- **Advanced analytics and metrics:** User behavior, detailed usage statistics, and fully detailed dashboard.

Depth

This project's technical depth comes from its sophisticated microservice orchestration and event driven architecture combined with its high privacy requirements and security-first focus. CTHULHU will do more than just enable users basic file upload functionality; it also explores the challenges that come with distributed system design ad asynchronous processing.

- **Microservice architecture Complexity:** Implementing true microservice patterns is a true challenge and requires advanced design concepts and principles in data and system design. It also demands sophisticated error handling across services, implementing circuit breakers for resilience, and designing compensating transactions for distributed failure scenarios.
- **Asynchronous Communication Mastery:** Leveraging RabbitMQ for inter-service communication introduces significant technical sophistication beyond synchronous HTTP APIs. This communication layer requires understanding of message routing

patterns, exchange types (topic, direct, fanout), queue durability, and acknowledgment strategies.

- **Authentication and authorization architecture:** The authentication service generates and validates JWT tokens, implements token refresh mechanisms, manages session expirations, and coordinates authorization decisions with other services.

Development Schedule and Milestones

Development Schedule

The development of CTHULHU will follow an Agile methodology with sprint-based iterations lasting 3-6 weeks depending on the scale and depth of the Service being developed. This approach aligns naturally with the microservices architecture, allowing each service to be developed, tested, and deployed independently while maintaining integration with the system as a whole.

Milestones:

Milestone 1: Client, Gateway, and File manager functionality

- **Duration:** 3 weeks
- **Effort:** *High* - Establishing the foundational infrastructure requires careful architecture planning and testing service communication patterns
- **Details:**
    - Working next.js react application for the user to upload and download files.
    - API gateway service with basic routing capabilities.
    - Docker containerization and for easier deployment and docker-compose configuration.
    - File manager service basic upload and download functionality.
    - API Gateway service database to persist data such as user sessions.

    - File manager service database to persist file and bucket metadata.

Milestone 2: File manager expansion and Authorization

- **Duration:** 4 weeks
- **Effort:** *high* - Authentication implementation will require thorough security testing, token life cycle management, and close integration with the API gateway.
- **Details:**
    - Design and implement an efficient Authentication Service database to store User metadata and keep track of user sessions.
        - Design how Authentication service can be source of truth but also limit redundant calls from the gateway by using caching in the gateway.
    - Expand File manager service functionality to incorporate cloud storage.
    - Authorization middleware for API gateway integration.

Milestone 3: File life cycle manager service

- **Duration:** 4 weeks
- **Effort:** *Moderate* — Requires implementing reliable background job processing and message queue consumers all within 2 weeks or a more intense schedule.
- **Details:**
    - Schedule the service to automatically manage files throughout the Cthulhu platform such as file expiration (48 hours for anonymous and up to 14 days for authorized users)
    - RabbitMQ-based event system for file lifecycle notifications.
    - Design and implement a File lifecycle service database that persists and stores required data to update, create, and delete files in Cthulhu as well as keep track of notifications to send to the user.

Milestone 4: Security Service

- **Duration:** 6 weeks
- **Effort:** *High* — File scanning requires deep integration with ClamAV antivirus engine, complex event-driven architecture using RabbitMQ, and careful orchestration of asynchronous workflows across multiple services.
- **Details:**
    - Integration with ClamAV open-source antivirus engine for malware detection.
    - Signature based detection for known malware patterns (trojans, viruses, worms, ransomware, etc.)
    - Deep content inspection to verify true filetype against declared file extension.
    - Will have to find a suitable VPS to power and allow for concurrent scans of files to ensure no bottleneck in this service.
    - Design a SAGA sequence for scanning files and what to do when a file is malicious (Request deletion from Filemanager and emit a notification back to the user client).
    - Establish a thourough testing suite to validate the Security Service's accuracy and reliability across both malicious and benign files.
        - Malicious file detection testing (trojans, viruses, worms, ransomware, etc).
        - Safe file false positive testing validating that benign files are correctly classified as safe.
        - Target a true positive rate of more than 95% and false positives below 2% in accordance to AV-TEST industry standards.

Milestone 5: Online Functionality, E2E user tests, and CI/CD pipeline

- **Duration:** 5 weeks
- **Effort:** *High* — Deployment is the most important part of the platform for end users. The Cthulhu platform and its required services need to be fully deployed, end to end

user tested, and automatic CI/CD pipelines established to ensure smooth future updates and maintenance.

- **Details:**
    - Cthulhu needs to be viewable online on a dedicated domain and be protected with HTTPS using a valid SSL certificate.
    - End to end user testing simulating real world usage scenarios to validate the entire platform's usability, performance, and reliability.
    - CI/CD pipeline should be integrated using jenkins or other pipelines to ensure that testing and artifacts are created handled properly.
    - If tests fail the development pipeline needs to rollback and abort the deployment to avoid breaking changes.
    - If tests pass and then a seperate "beta" environment should be viewable online for final approval before transitioning to production servers.
        - Beta environment should run independent to the production servers to avoid any conflicts.
    - CI/CD pipeline should push Docker container images to a container registry such as Docker Hub or AWS ECR for easy deployment and versioning.

Bibliography

Chen, Y., Akhawe, D., Hanna, S., Mao, F., McCamant, S., & Song, D. (2024). URadar: Discovering Unrestricted File Upload Vulnerabilities via Adaptive Dynamic Testing. *IEEE Transactions on Information Forensics and Security.*

Spacelift. (2025,). *100+ Cloud Security Statistics for 2025.*

Uddin, N., & Jabr, M. (2016,). File Upload Security and Validation in Context of Software as a Service Cloud Model. *2016 6th International Conference on Information Technology Convergence and Security (ICTS).*