University of Essex

Proposal document

Team Strategy

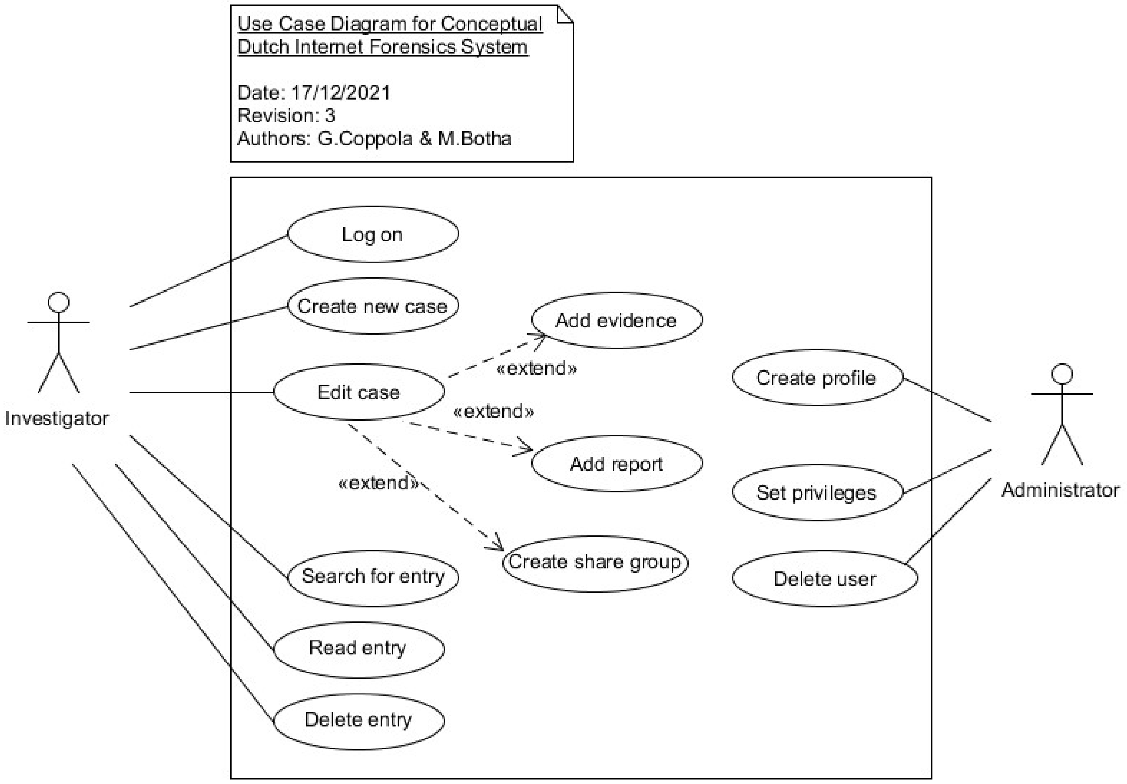
Revisions

Version	**Author**	**Date**
0.1	Michael Botha	5/12/2021
0.2	Gennaro Coppola	8/12/2021
0.3	Michael Botha	12/12/2021
0.4	Gennaro Coppola	12/12/2021
0.5	Michael Botha	13/12/2021
0.6	Michael Botha	14/12/2021
0.7	Gennaro Coppola	14/12/2021
0.8	Michael Botha	15/12/2021
0.9	Michael Botha	16/12/2021
1.0	Michael Botha	18/12/2021
1.1	Michael Botha	19/12/2021
1.2	Gennaro Coppola	19/12/2021
1.3	Michael Botha	20/11/2021

Contents

System Requirements and Assumptions

The Dutch Police domain requires a mock system to be developed to facilitate the creation and alteration of forensic investigation cases, related to Internet security within the Netherlands region (Borges, 2021; Government of the Netherlands, N.D; Microsoft, 2008). Evidence and other related information will thereby be captured on a database (Borges, 2021; Government of the Netherlands, N.D; Microsoft, 2008). Once captured, investigative experts can process the data to produce findings (Borges, 2021; Government of the Netherlands, N.D; Microsoft, 2008). Therefore, allowing more Internet security awareness and better preparedness with regards to cybercrime (Borges, 2021; Government of the Netherlands, N.D; Microsoft, 2008). Please see Figure 1 below for use case analyses of the envisaged system.


Use Case Diagram for Conceptual Dutch Internet Forensics System

Date: 17/12/2021
Revision: 3
Authors: G.Coppola & M.Botha

Figure 1 – Use Case Diagram

It is assumed that in future the system would also extend to cover cases for the entire Internet to facilitate a global footprint (Government of the Netherlands, N.D; Microsoft, 2008). However, this phase of development will be implemented in one selected police station, and produce a prototype that provides Local Area Network (LAN) access to the relevant database using a Command-Line Interface (CLI) (Sommerville, 2016). If time permits, a web server will be created to facilitate a browser-based front-end. Ultimately, a network architecture as seen in figure 2 is envisaged should the project continue development phases (Fortinet, N.D).

During this initial project phase, program usage patterns and profiling can be produced to better appropriate hardware requirements relevant to application facilitation (Goh, 2021; Microsoft, 2014). Until then, the selected hardware is as per Table 1.

Processor	**Memory**	**Mass Storage**	**Networking**
- Quad core - 2 threads/core - 3 GHz	- DDR4 - 2400 MHz - 16 GB	- SSD - 1 TB	- 2 FastEthernet ports - 100 Mbps

Table 1 – Server Machine Specifications

An Information System is comprised of Hardware, Software, People, Processes, and Data (Bourgeois, 2014). Therefore, various factors other than the application architecture and design affect its wholistic performance and security (Nieles et al., 2017). It is assumed that such aspects are well catered for by the various domain-specific teams, outside of the application development team. Examples of such reside in Table 2 below.

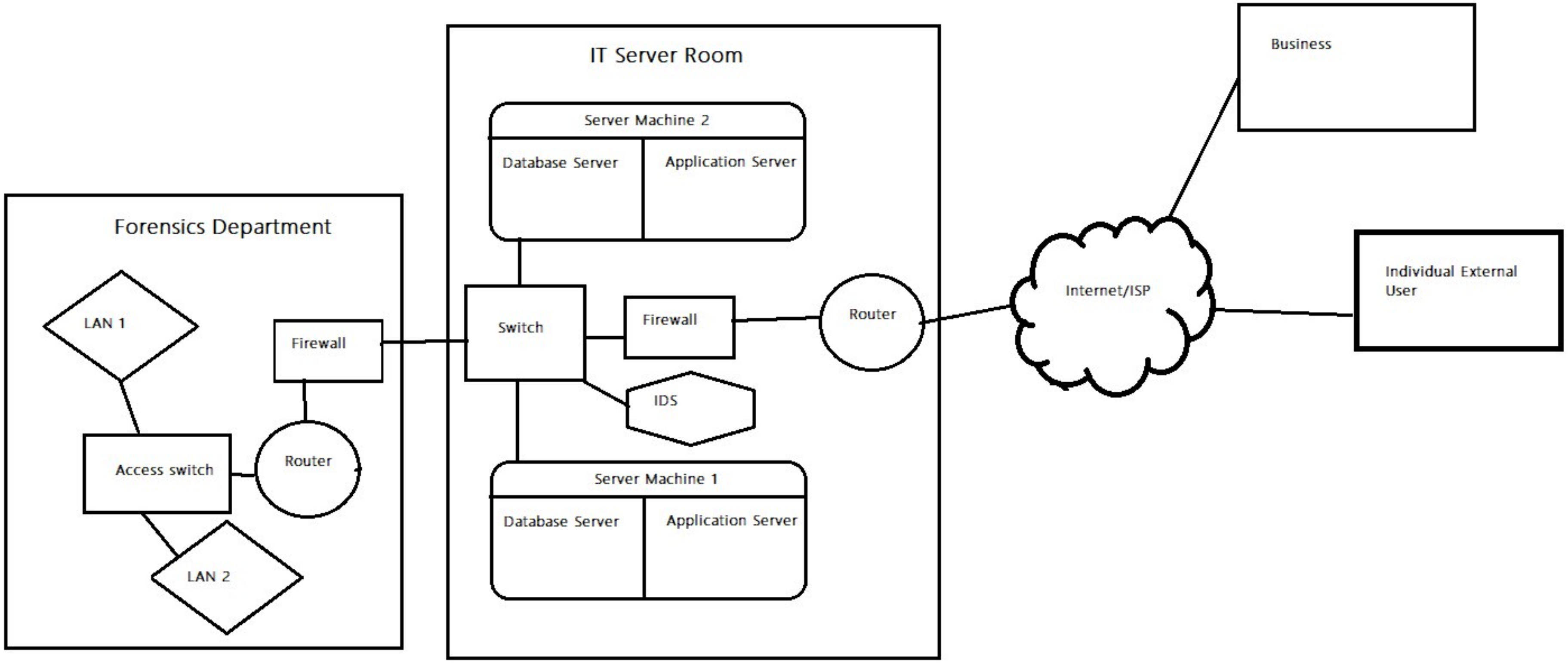


Figure 2 – Envisaged Future Network Architecture

Policies & Procedures	**Physical Security**	**Infrastructure**	**Networking**	**Software**
1. Fair-use policy 2. Employee background checks 3. Incident management procedure 4. Disaster recovery plan 5. Training	1. Building access control 2. Security guards 3. Perimeter fencing	1. Airconditioning systems 2. Cabling regulations 3. Redundant power sources and supplies 4. Server redundancy and the use of virtual machines 5. Cryptographic accelerators 6. Fire suppression systems	1. Network throughputs 2. Virtual network partitions 3. Firewall port allocations 4. A demilitarized zone 5. Intrusion Prevention or detection system 6. Redundant network links 7. Load balancer	1. File permissions 2. Host-based firewall 3. Up-to-date antivirus

Table 2 – Assumed Conditions to Ensure Security In-Depth

Application Security

Performing a brief risk assessment and impact evaluation pertaining to the software system prior to any application hardening, and only considering the more probable and high impact risks yielded the below Table 3 (Nieles et al., 2017).

Risk	Probability	Impact	Control
Overall system failure	Medium	High	Monitoring and logging
Packet Sniffing	High	High	Data in-transit encryption
Unauthorised access	High	High	Authorisation
Data breech	High	High	Authentication & data at-rest encryption

Table 3 – Risk Assessment and Evaluation

Further to the above, the Open Web Application Security Project's (OWASP's) top 10 vulnerabilities were used as a reference risk profile to produce Table 4 (OWASP, N.Da). Additionally, the specific vulnerabilities inherent in the Python interpreter and language were considered (Pillai, 2017). Although these vulnerabilities are from a generic set they will sufficiently cover the domain-specific risks too. OWASP's control measures were referenced for a risk mitigation strategy (OWASP, N.Db).

Vulnerability	Mitigation technique
Broken access control	- Log authorisation failures - Implement least privilege - Assign authors – record ownership
Cryptographic failures	- Encryption of sensitive data information - Use up-to-date encryption and hashing algorithms
Injection	- Use Input sanitisers for any data received by the user
Insecure design	- Iteratively Implement security at the design level - Use a secure scrum approach - Use trusted third-party libraries
Identification and Authentication Failures	- Monitor all login attempts - Implement strong password policy including salting before encryption - Implement a login mechanism with limited tries.
Security Logging and Monitoring Failures	- Implement a contextualised log format to better understand and manage logs - Encrypt logged data - log application exceptions raised

Table 4 – System Vulnerabilities and Controls

Whilst the application resides on the user's computer during the CLI phase, the normal user will only have execution rights. Thus, preventing any read or write operations with regards to the associated Python application's source code, as it is interpreted and therefore won't reside as human-unreadable machine executable code, such as with a compiled C program (University of Essex Online, 2021).

The proposed solution will be GDPR compliant according to the requirements laid out thereby (Information Commissioner's Office, N.D). Please see Table 5 below for related information.

Concern	Action
Lawful basis and transparency	- The system will store personal data regarding police members accessing it. This data will only be stored for the duration he/she is supposed to have access to the system. - Information pertaining to suspects will remain secret, in the best interests of the public.
Data security	- Data security will be enforced through encryption, and the implementation of security policies for all people involved in the use of the system. - Security in-depth principles will be followed as previously mentioned
Accountability and governance	- Because the system pertains to a public European institution, a committee will be established to ensure continual GDPR compliance.
Privacy rights	- Users will have the right through a change control process, and the administrator to request the correction of their data, should errors exist.

Table 5 – GDPR Considerations

Architectural and Design Philosophies

It is important to create software which aligns with the architectural quality attributes reflected in Table 6 below (Pillai, 2017).

Quality Attribute	Method to Achieve Attribute	Implementation Method
Modifiability	- Readability - Simplicity - Code cohesion	- Follow Python style guide - Docstrings and comments in source code - Linters - Single-purpose classes and functions
Scalability	- Microservices - OOP	- Split application into separate core and authentication microservice programs
Testability	- Code cohesion - Separation of duties	- Single-purpose classes and functions - Limit inheritance hierarchy to three levels - Don't use deeply nested conditional and loop structures
Performance	- Efficient algorithms - Multi-threading	- Where possible keep core algorithms to less than O(n) complexity - Create multiple threads to run concurrently when parallel processing is possible
Availability	- Exception management - Redundancy	- Use Python exception classes to cover the raising of various errors - Microservices - Concurrent multi-threading
Security	- See security section	- See security section
Deployability	- Limit dependencies - Highly available 3rd-party libraries	- Use libraries from PyPi - Implement a Python virtual environment - Where possible, used libraries should come with their dependant libraries

Table 6 – Architectural Quality Attributes

A Microservices Architecture will be used to support separation of concern and enhance security, as well as provide scalability options (Pillai, 2017). HTTPS is the application layer protocol to be used for the transfer of JSON document objects between microservices (Pillai, 2017). The relevant applications are represented in Table 7 below.

Microservice	Functions
Core program	- Provide database CRUD features - Monitoring and logging - Encryption - Item sharing - Searching
Authentication	- Create and edit user profile and privileges - Encryption - Monitoring and logging - Authenticate/log-in user - Deliver user's authorisation level

Table 7 – Microservices

Design patterns will be used as per Table 8 below in an amalgamation with individual stylistic patterns inherent to each developer (Pillai, 2017).

Creational	Structural	Behavioural
Factory pattern for case creation	- Association: to avail the use of services like authorisation and distributed application access - Aggregation: to tie evidence to a case - Composition for the creation of cases within a repository object - Inheritance to specialise evidence whilst having a generalised type of meta data for evidence details - Proxy pattern for case access	A pattern similar to the Observer pattern to allow concurrent multithreading

Table 8 – Design Patterns

Finally, a relational database will need to be developed to simulate the database requirements. This database will use a standard Relational Database Management System (RDBMS) and not have any conversion to Microsoft Word or Excel formats (Connolly & Beg 2015). A database technology, typically of the NoSQL type would be essential for the facilitation of multimedia artefacts in future project phases (mongoDB, N.D). The relational database will be normalised to the third normal form to ensure no redundancy in the system (Connolly & Beg 2015).

Design

Extending the Use Case diagram of Figure 3 to create the core procedural flows for an investigator, produced the Activity diagram in Figure 2 below (Ambler, 2003). Please note that the actual program will provide more return functionality. For instance, to place a user at a previous menu. Furthermore, by observing Figure 2 a programmer should have sufficient insight to code the other user roles into the source code (Ambler, 2003).

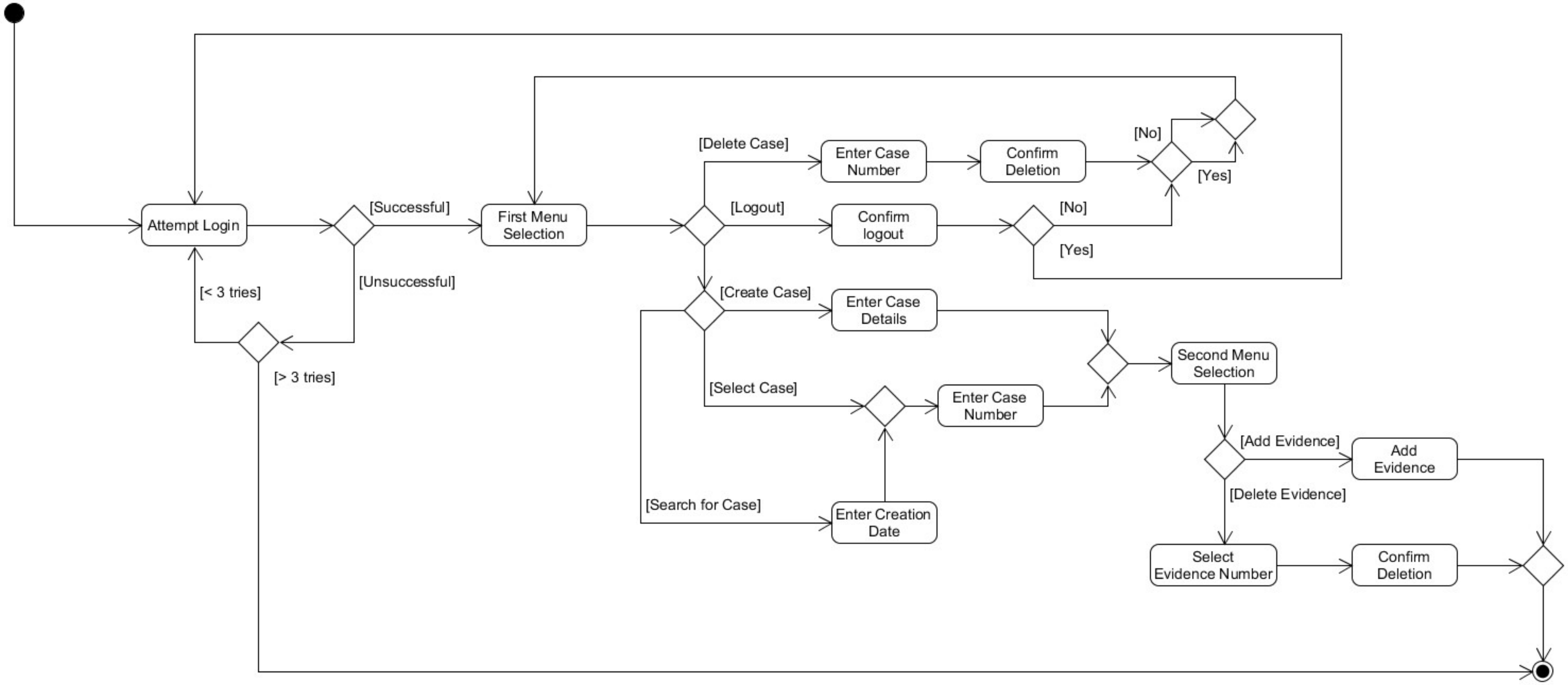


Figure 3 – Activity Diagram for Investigator Interacting with the Program

The Class diagrams Figure 4 and Figure 5 capture the Object-Oriented design of the two microservices. Table 9 and Table 10 explain the objects within the Class diagrams.

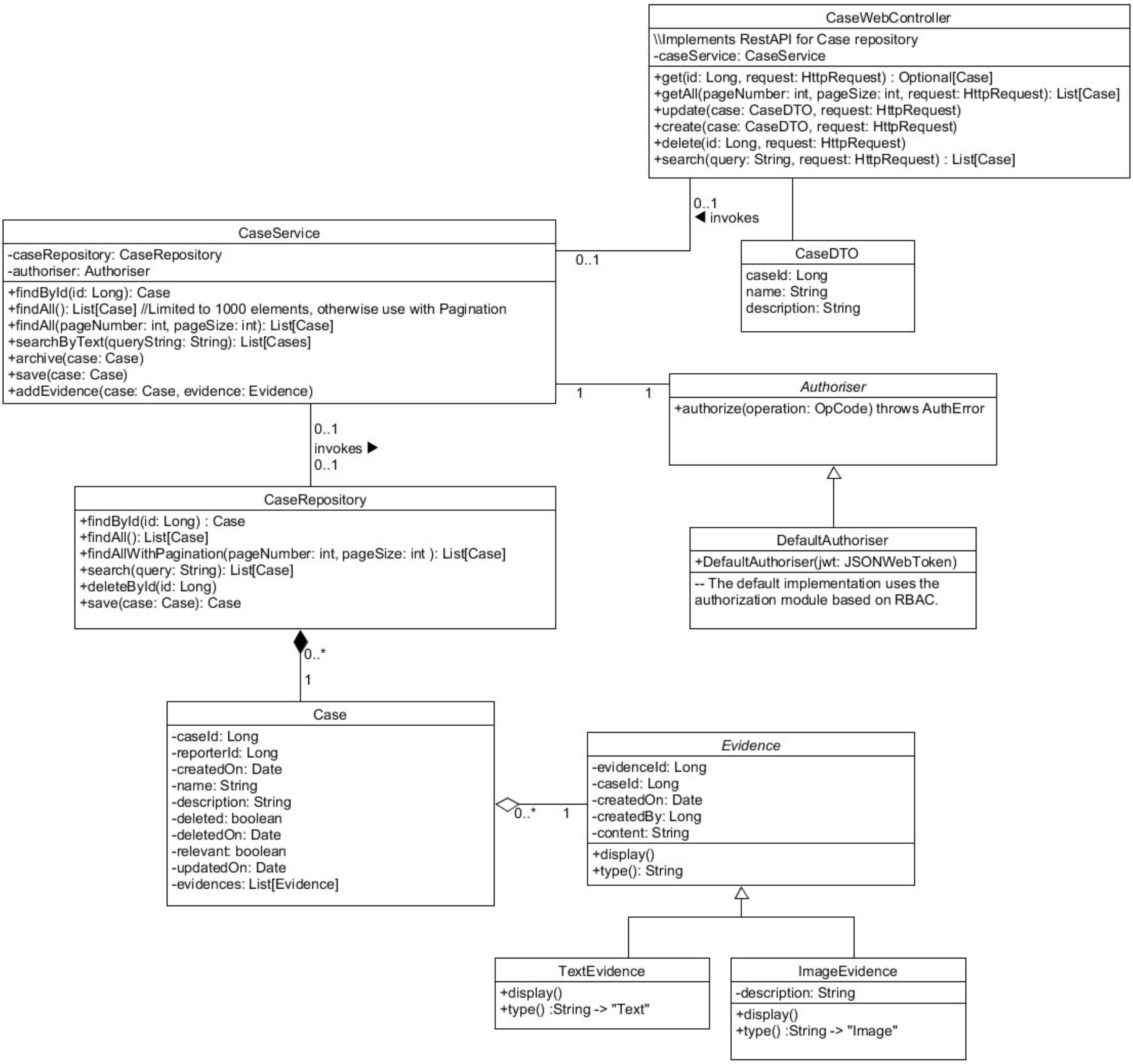

CaseWebController
\\Implements RestAPI for Case repository
-caseService: CaseService
+get(id: Long, request: HttpRequest) : Optional[Case]
+getAll(pageNumber: int, pageSize: int, request: HttpRequest): List[Case]
+update(case: CaseDTO, request: HttpRequest)
+create(case: CaseDTO, request: HttpRequest)
+delete(id: Long, request: HttpRequest)
+search(query: String, request: HttpRequest) : List[Case]
0..1
◀ invokes
0..1
CaseDTO
caseId: Long
name: String
description: String
CaseService
-caseRepository: CaseRepository
-authoriser: Authoriser
+findById(id: Long): Case
+findAll(): List[Case] //Limited to 1000 elements, otherwise use with Pagination
+findAll(pageNumber: int, pageSize: int): List[Case]
+searchByText(queryString: String): List[Cases]
+archive(case: Case)
+save(case: Case)
+addEvidence(case: Case, evidence: Evidence)
1
1
Authoriser
+authorize(operation: OpCode) throws AuthError
0..1
invokes ▶
0..1
CaseRepository
+findById(id: Long) : Case
+findAll(): List[Case]
+findAllWithPagination(pageNumber: int, pageSize: int): List[Case]
+search(query: String): List[Case]
+deleteById(id: Long)
+save(case: Case): Case
DefaultAuthoriser
+DefaultAuthoriser(jwt: JSONWebToken)
-- The default implementation uses the authorization module based on RBAC.
0..*
1
Case
-caseId: Long
-reporterId: Long
-createdOn: Date
-name: String
-description: String
-deleted: boolean
-deletedOn: Date
-relevant: boolean
-updatedOn: Date
-evidences: List[Evidence]
0..*
1
Evidence
-evidenceId: Long
-caseId: Long
-createdOn: Date
-createdBy: Long
-content: String
+display()
+type(): String
TextEvidence
+display()
+type() :String -> "Text"
ImageEvidence
-description: String
+display()
+type() :String -> "Image"

Figure 4 – Class Diagram for Core Microservice

Class	Function
Case	- The main entity - Consists of a list of evidence
Evidence	- Stores evidence details as objects
TextEvidence	- A specialisation of Evidence for text evidence
ImageEvidence	- A specialisation of Evidence for Multimedia evidence.
CaseRepository	- Implements CRUD operations for cases
CaseService	- Provides CRUD operations for cases, with support from Authoriser
Authoriser	- Interface to support authorisation. Used by CaseService, but could be used by new components too
DefaultAuthoriser	- Default implementation for authorization. At the moment of document creation, the idea is to have a module based on Role-Based Access Control (RBAC) to provide authorisation for system operations
CaseWebController	- Exposes Rest API for operations on cases
CaseDTO	- Data Transfer Object (DTO) to be used by the web layer

Table 9 – Core Microservice Class Explanation

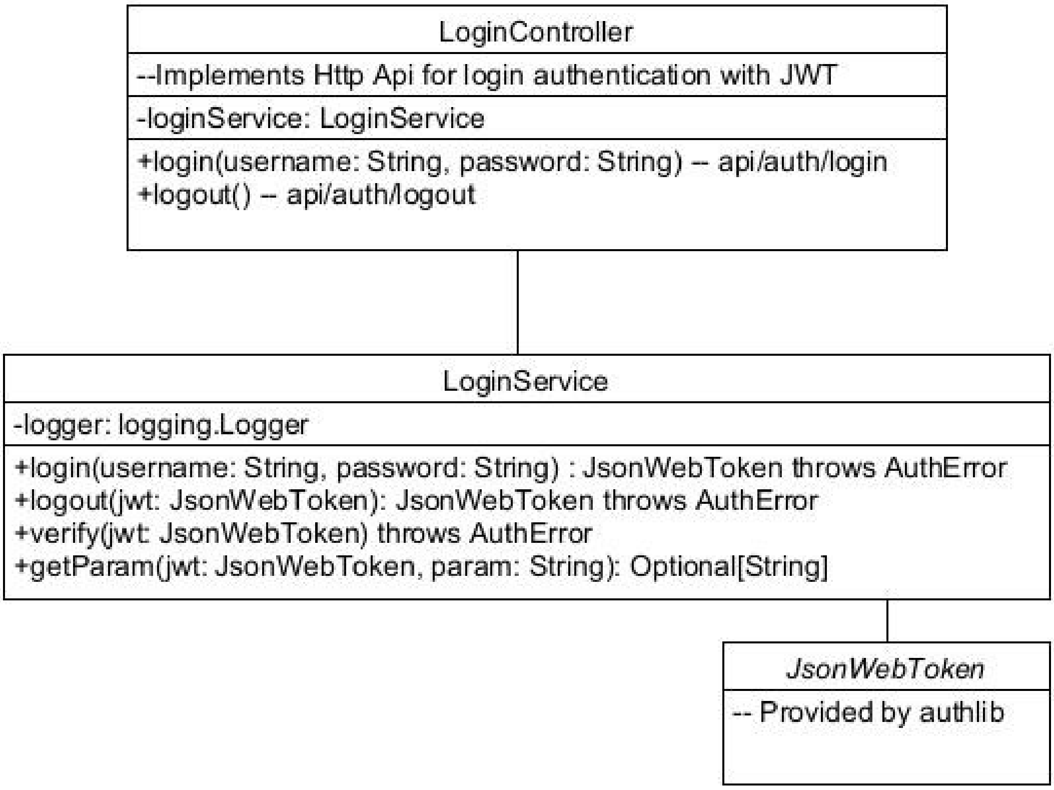

LoginController
--Implements Http Api for login authentication with JWT
-loginService: LoginService
+login(username: String, password: String) – api/auth/login
+logout() – api/auth/logout
LoginService
-logger: logging.Logger
+login(username: String, password: String) : JsonWebToken throws AuthError
+logout(jwt: JsonWebToken): JsonWebToken throws AuthError
+verify(jwt: JsonWebToken) throws AuthError
+getParam(jwt: JsonWebToken, param: String): Optional[String]
JsonWebToken
-- Provided by authlib

Figure 5 – Class Diagram for Authentication Microservice

The user session is stored as a JSON Web Token, which is a standard mechanism that allows third party developers to integrate web applications or other services (Pillai, 2017).

Class	Function
LoginController	Exposes the microservice through a Rest-API
LoginService	Implements authentication logic
JsonWebToken	Contains the login context, username, and user session

Table 10 – Authentication Microservice Class Explanation

The three Sequence diagrams, Figure 6, Figure 7, and Figure 8 represent use case object interactions for insight into object communications.

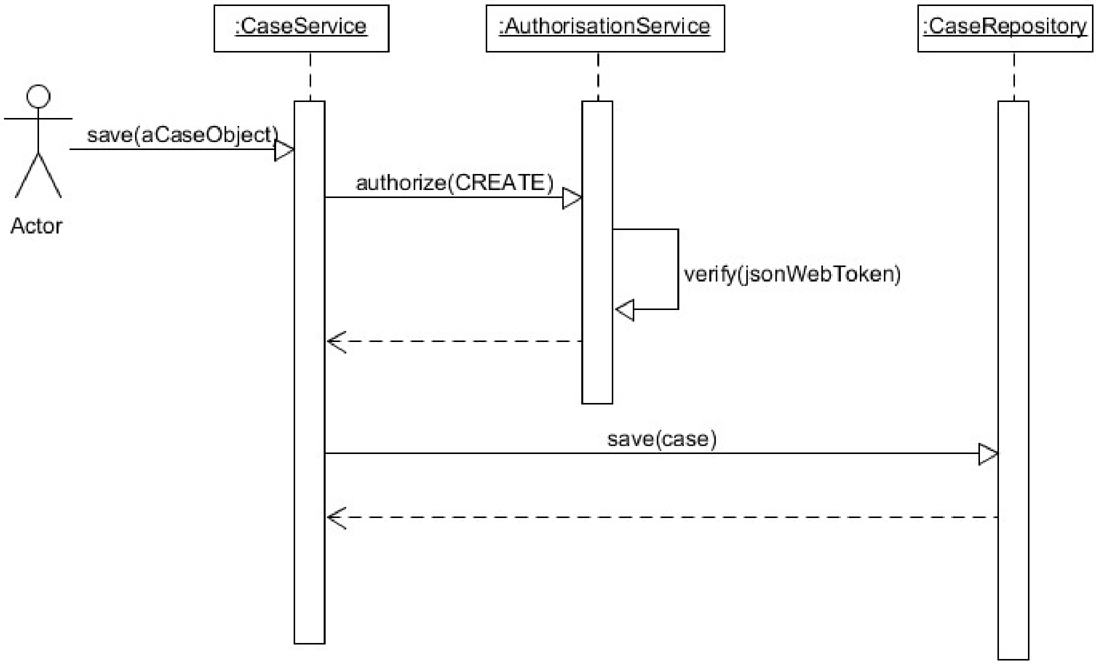


Figure 6 – Sequence Diagram Presenting the Creation of a new Case

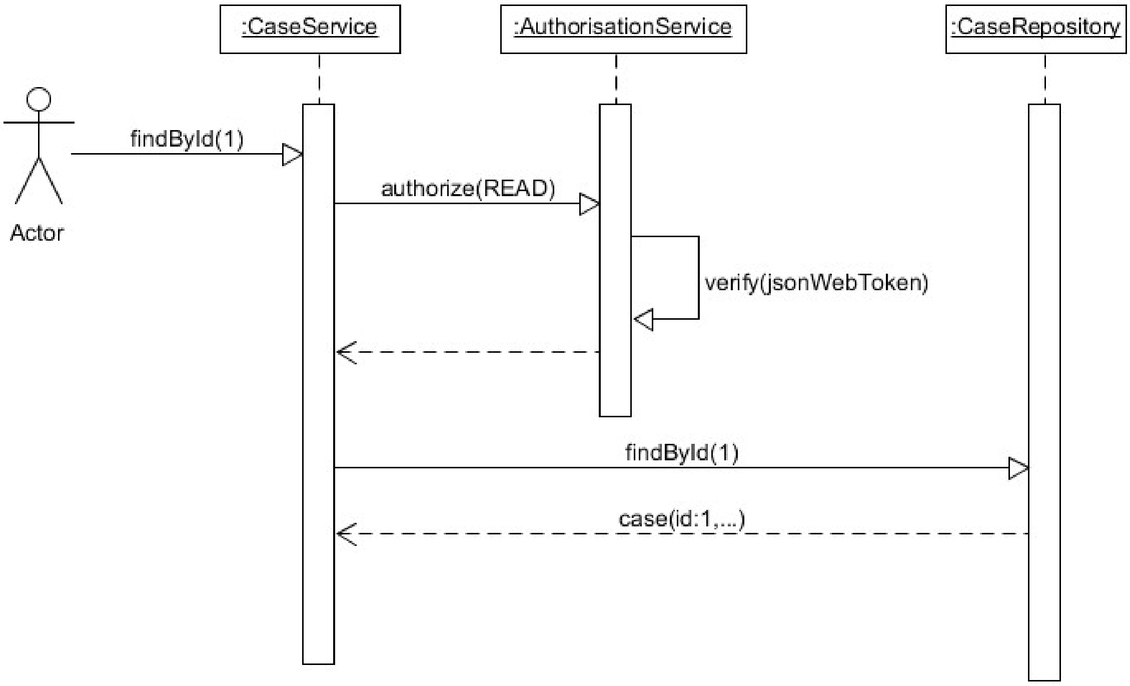


Figure 7 – Sequence Diagram Presenting a User Attempting to Read a Case from the Database

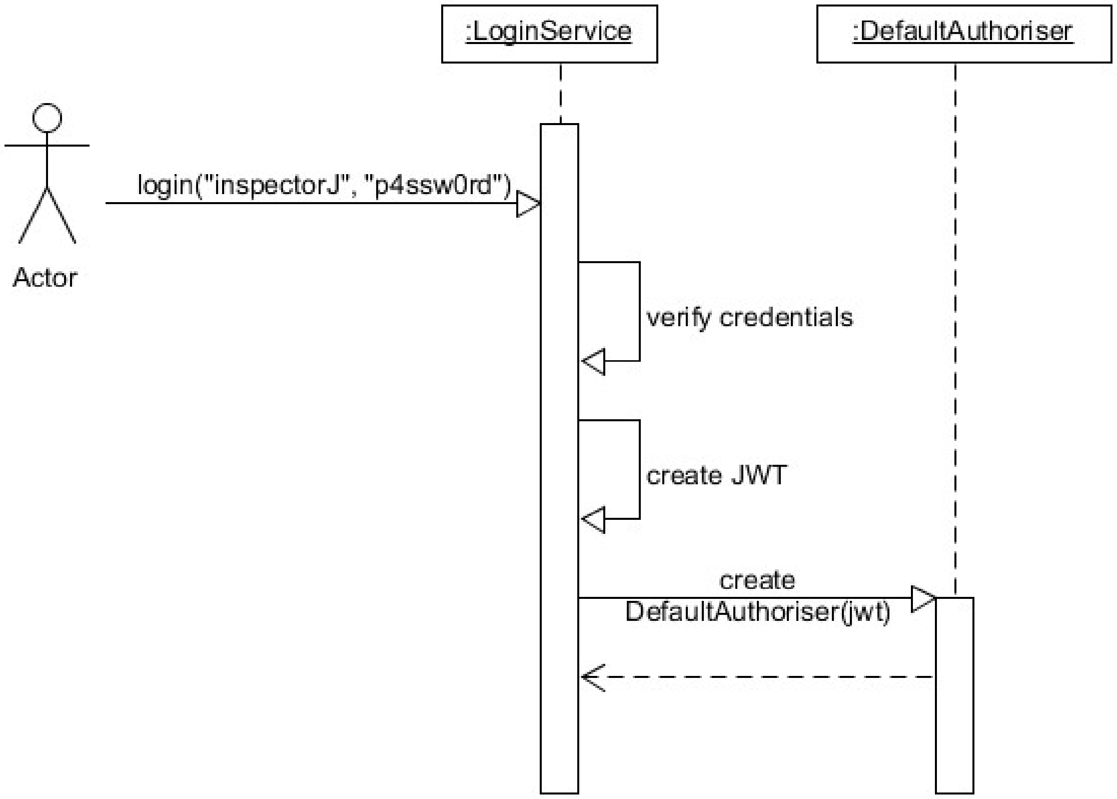


Figure 8 – Sequence Diagram Presenting a User Attempting to Log in to the System

Below is a list of general prescribed approaches for the development team to follow:

- The system is to be realised through the Python and SQL programming languages.
- An Agile Secure Scrum software development approach will be used (Phol & Hof, 2015). Once the initial backlog has been developed the workload will be separated into individual tasks for each member to complete (Phol & Hof, 2015).
- Developers are to work independently on tasks and merge outputs into a common GitHub repository when completed.

- Libraries and tools as described in Table 11 will aid to produce an efficient and secure development process.
- The minimum strength symmetric encryption algorithm allowable is AES-128 (OWASP, N.Dc).
- The minimum asymmetric encryption algorithm allowable is RSA-2048 (OWASP, N.Dc).
- The minimum strength hashing algorithm allowable is SHA-2 (OWASP, N.Dd).
- Salting can be performed to further enhance obfuscation (OWASP, N.Dd).

Functionality	Library or tool's name
Encryption	Pycrypto - https://pypi.org/project/pycrypto/
Authorisation	Py_RBAC - https://pypi.org/project/py-rbac/
Authentication	Authlib - https://pypi.org/project/Authlib/
Monitoring	Logging – Python standard library
Database	Sqlite3 – Python standard library
Style guide	PEP3 style guide from Python

Table 11 – Project Libraries and Tools

References

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

Borges, E. (2021) Cyber Crime Investigation Tools and Techniques Explained. Available from: https://securitytrails.com/blog/cyber-crime-investigation [Accessed 20 November 2021].

Bourgeois, D. (2014) Information Systems for Business and Beyond. Washington: The Saylor Academy.

Connolly, T. & Beg, C. (2015) *Database Systems: A Practical Approach to Design, Implementation, and Management.* Global Edition. Edinburgh: Pearson.

Fortinet. (N.D) DMZ. Available from: https://www.fortinet.com/resources/cyberglossary/what-is-dmz [Accessed 20/12/2021].

Goh, E. (2021) How to Estimate vCPU Core, Memory & Disk Size for a Cloud Server?. Available from: https://medium.com/geekculture/how-to-estimate-vcpu-core-memory-disk-size-for-a-cloud-server-31fa26c883f5 [Accessed 12 December 2021].

Government of the Netherlands. (N.D) Fighting Cybercrime in the Netherlands. Available from: https://www.government.nl/topics/cybercrime/fighting-cybercrime-in-the-netherlands [Accessed 20 November 2021].

Information Commissioner’s Office. (N.D) Guide to the UK General Data Protection Regulation UK GDPR). Available from: https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/ [Accessed 28 November 2021].

Microsoft. (2008) The Open Computer Forensics Architecture (OCFA). Available from: https://docs.microsoft.com/en-us/archive/blogs/ms_schweiz_security_blog/the-open-computer-forensics-architecture-ocfa [Accessed 20 November 2021] .

Microsoft. (2014) Calculate Your Server Size. https://docs.microsoft.com/en-us/previous-versions/tn-archive/bb124226(v=exchg.65)?redirectedfrom=MSDN [Accessed 12 December 2021].

mongoDB. (N.D) What is NoSQL. Available from: https://www.mongodb.com/nosql-explained [Accessed 20 December 2021].

Nieles, M., Dempsey, Kelly., Pillitteri, V. (2017) An Introduction to Information Security. Revision 1. United States of America: National Institute of Standards and Technology. Available from: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf [Accessed 18 September 2021].

OWASP. (N.Da) OWASP Top Ten. Available from: https://owasp.org/www-project-top-ten/# [Accessed 5 December 2021].

OWASP. (N.Db) OWASP Proactive Controls. Available from: https://owasp.org/www-project-proactive-controls/ [Accessed 5 December 2021].

OWASP. (N.Dc) Cryptographic Storage Cheat Sheet. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html [Accessed 20/12/2021].

OWASP. (N.Dd) Password Storage Cheat Sheet. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html [Accessed 20/12/2021].

Pillai, B. (2017) *Software Architecture with Python*. 1st Edition. Birmingham, UK: Packt Publishing

Pohl, C. & Hof, H-J. (2015) *Secure Scrum: Development of Secure Software with Scrum*, in Proc. of the 9th International Conference on Emerging Security Information, Systems and Technologies

Sommerville, I. (2016) Software Engineering. 10th ed. Essex: Pearson Education Limited.

University of Essex Online. (2021) *Programming Languages* [Lecturecast]. SSDCS_PCOM7E November 2021 Secure Software Development November 2021. University of Essex Online