DESIGNING A PLATFORM FOR NETWORK-RELATED EDUCATION

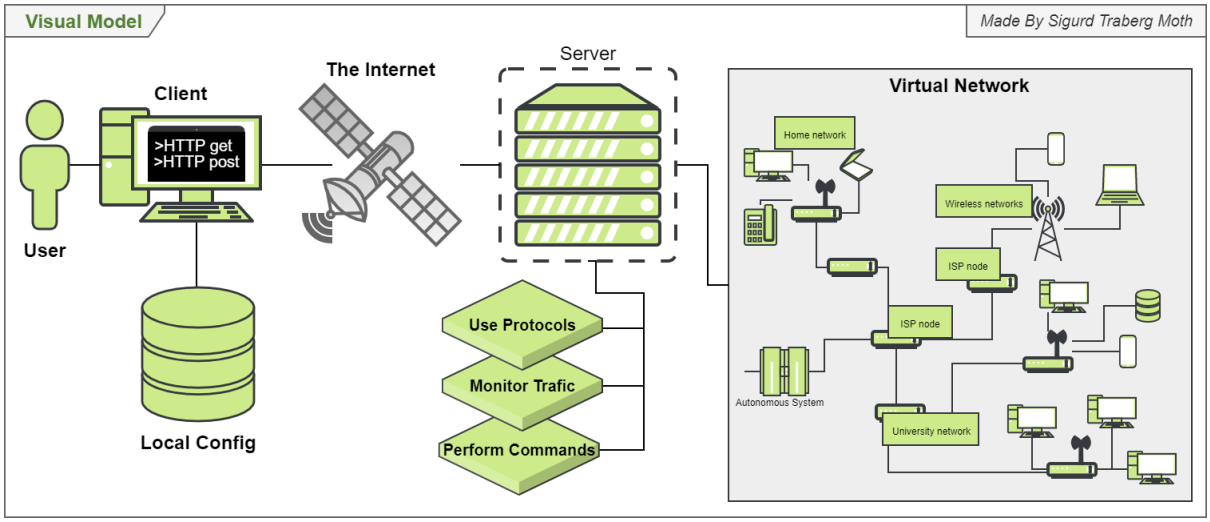
Software Requirement Specification

Written by Sigurd Traberg Moth

Last updated: 3 January 01/2021

Version number: 1.00

Color: #538135



CONTRIBUTORS:

<Contribution> <Name> <Mail[Optional]>

Contents

A. Introduction

A.1 Purpose

This document covers the specified software requirements that have been established by the stakeholders and system users. The document includes use cases, system features, functional and non-functional requirements.

A.2 Document Conventions

This document has been created when the creation of the product was in mind, but before detailed requirements were made. It is required to update this document and its version number every time there are developments made in the requirements that impact the product.

A.3 Intended Audience

This document is intended primarily for developers that wish to further develop the system. That makes updating the document very important as it is expected that students will be the ones to develop on the system. Students will usually only have 1 semester to do work on the project and for that reason, the SRS must be easy to understand, in-depth, and specific to make the transition between developers as smooth as possible.

Finally, it also serves as a lookup document for developers as many specific designs need to be followed to the point.

A.4 Project Scope

The **Virtual Internet Platform**, from now on known as the **VIP**, is a software made to mimic a copy of the Internet. The advantages are that everything is controllable including the amount of traffic, hosts, routers, connectivity, and congestion. Through the VIP users can learn from, monitor, research, and create experiments in a secure and cut off environment while still being able to translate results to the Internet.

The software is written in Java utilizing Spring, Docker, Microservices, Microsoft Azure. Communication between Docker containers is done via JSON.

A.5 References

The VIP is currently available at GitLab under the SDU domain. A link can be found below, though an SDU mail is required.

GitLab link:

GitLab contains a repository with the source code, project documents such as the SRS, report, and SAD.
https://gitlab.sdu.dk/HDCL/virtual-internet-platform-vip

B. Overall Description

B.1 Product Perspective

The goal of the VIP is to engage students in how the Internet and networking in general works. The VIP will allow students to conduct experiments and do assignments that, to some extent, would otherwise not be possible through the Internet.

B.2 Product Features

The goal is that all features below will be implemented at some point in the lifetime of the VIP.

Main features:

- An API that can be accessed through a terminal - in the future this could be a *React user interface* granting access to the server where the virtual network resides and manipulation with the nodes is possible.
- Send and receive messages across a virtual network through hosts, routers, and other nodes.
- Addition, removal, and managing of nodes on the network - preferably also at runtime.
- Monitoring user, network stability, and general network traffic.

B.3 Operating Environment

The VIP will be run in a Unix-driven Docker (WIP) container and is, therefore, platform-independent (aside from ARM machines). Furthermore, the components themselves are made in Java and are therefore platform-independent as well, as long as Java Virtual Machine (JVM) is installed.

B.4 Design and Implementation Constraints

- Other developers can implement features if they **do not negatively affect usability and do not affect performance** severely. Contributions will have to be **tested and verified** in advance of acceptance by the product owner.
- Developers are **not locked to any specific programming language** but must use **JSON** for sending data between features. The programming language and API should be covered in the Software Architecture Document (SAD).

B.5 User Documentation

Documentation concerning different areas will be provided in the following documents:

- Inception document
- Project report
- Software Requirements Specification (SRS)
- Software Architecture Document (SAD)

B.6 Assumptions and Dependencies

Components should be run as jar files in Docker containers when using the system. Separate components can be tested without Docker. In the case of implementation of new components in other languages, these should be documented here and in the SAD.

C. System Features

System features cover implemented functionality based on use cases.

HTTP

1. *HTTP get request*

Users can write an HTTP get request to a node in the network and pull data. Requires setting destination node and network to a target host in the network.

B.1 Receive data, Use HTTP.

2. *HTTP post request*

Users can write a post request to a node in the network and post data. Requires setting destination node and network to a target host in the network.

A.1 Send data, A.3.1 Use HTTP.

Client properties

1. *Set Source Node, Set Source Network*

Users can assign their position in the network to an existing node.

A.1 Send data, B.1 Receive data. A.3.1 Use HTTP.

2. *Set Destination Node, Set Destination Network*

Users can assign a target in the network to write to.

A.1 Send data, B.1 Receive data. A.3.1 Use HTTP.

D. External Interface Requirements

Empty for now

F. Revision History

Empty for now

G. Requirements

The requirements analysis tries to uncover what the system specifically must be able to do to address all actors' use cases. Functional requirements are found through use cases and are connected through a use case model while non-functional requirements are found through quality attributes and tactics.

Use Case Model

Staying true to The Unified Process, the use case model has been used to find the functional requirements and contains the following elements:

- An actor list
- A collection of use cases
- A use case diagram

Actor list

A. **Teachers** are going to be using, monitoring, manipulating, and setting up the VIP for everyone else to use. They want full access to the VIP and the ability to allow others to further develop the system.
B. **Students** are going to be using, monitoring, manipulating, and setting up the VIP as well as part of their education. This will be done through exercises where a path has been laid down with hints for students.
C. **Projects** are usually groups of students that are developing a product. They might use the VIP to test or prove a hypothesis in a more controlled environment. They want to be able to create their own environment wherein they can test their ideas in isolation.
D. **Developers** are those interested in developing features for the VIP. Developers can be and do all of the above.
E. **Virtual Hosts** are bots imitating real users sending and receiving data on the network. As such they are not real users with many use cases, but rather shallow imitations.
F. **Common Users** are a generalization of teachers, students, and project users. It contains many of the use cases such as the ability to send and receive data as this is a feature all users share.
G. **Time** is an actor as several things are going to be happening on a scheduled basis.

Use cases

In this document use cases are separated by letters referring to actors (A, B, …). For each actor, high-level use cases are separated by numbers (A.1, A.2, …). They are then further split up into sub-use cases that more specifically define all actions of a high-level use case (A.1.1, A.1.2, …).

A. Common Users

ID	Name	Description
A.1	**Send data**	Common users want to send data across the VIP.
A.1.1	Send big files	Common users want to send big files across the VIP.
A.1.2	Send several files	Common users want to send several files across the VIP.
A.2	**Receive data**	Common users want to receive data across the VIP.
A.2.2	Receive big files	Common users want to receive big files across the VIP.
A.2.3	Receive several files	Common users want to receive several files across the VIP.
A.3	**Use protocols**	Common users want to use different protocols for different applications.
A.3.1	Use HTTP	Common users want to use HTTP to send and receive messages across the VIP.
A.4	**Monitor system**	Common users want to monitor the VIP for various information.
A.4.1	Monitor throughput	Common users want to monitor throughput or traffic across a given device.
A.4.2	Monitor bandwidth	Common users want to monitor the level of bandwidth is across a given device.
A.4.3	Monitor packet loss	Common users want to monitor packet loss.
A.4.4	Monitor device uptime	Common users want to monitor the uptime of running (and shut down) devices.
A.4.5	Detecting defects	Common users want to detect defects (error handling) when devices are not functioning as expected.
A.4.6	Detecting heartbeat	Common users want to know if devices are continuously running by detecting heartbeat on devices.
A.5	**Display Information**	Common users want to select what data is being logged and displayed
A.5.1	Display protocols	Common users want to select which protocol information is being logged and displayed.
A.5.2	Display packet loss	Common users want to select displaying the packet loss of a device.
A.5.3	Filter layer information	Common users want to select to filter by layers of the internet model.

A.5.4	Display routing network	Common users want to display the routing network so they can identify the setup and possibly join a subnet.
A.6	**Logging information**	Common users want logged information for traversing history.
A.6.1	Logging data	Common users want to log data so they can traverse previous messages.
A.6.2	Logging timestamps	Common users want timestamps to be paired to data.
A.7	**Getting help**	Common users want to access help where they can learn the ropes of the system.
A.7.1	Explaining functionality	Common users want to access help and get an understanding of the system through explanations of functionality.
A.7.2	Explaining keywords	Common users want to access help, where they can get an explanation of keywords and their actions.
A.8	**Configuring virtual hosts**	Common users want to configure virtual hosts.
A.8.1	Selecting if data is being sent	Common users want to select if data is being sent from the virtual host.
A.8.2	Changing data amount sent	Common users want to change how much data is being sent by the virtual host.
A.8.3	Selecting who receives data	Common users want to change who is receiving the data being sent from the virtual host.
A.9	**Joining network**	Common users want to be able to join and interact with the networks on the VIP.
A.9.1	Joining network externally	Common users want to be able to interact with the network as their own external network.
A.9.1	Joining subnet	Common users want to join specific networks and to live as a part of the internal network on a subnet.

B. Teachers

ID	Name	Description
B.1	**Create standard setups**	Teachers want to **create standard setups** for exercises and save them through configuration files.
B.1.1	Add virtual hosts	Teachers want to add virtual hosts to the configuration file. <Details>
B.1.2	Add virtual routers	Teachers want to add virtual routers to the configuration file. <Details>
B.1.3	Add virtual switches	Teachers want to add virtual switches to the configuration file. <Details>
B.1.4	Set routing tables	Teachers want to set up routing tables in the configuration file. <Details>
B.1.5	Enable or disable features	Teachers want to enable or disable features such as general behavior for the VIP. <Details>
B.1.6	Save configuration files.	Teachers want to save configuration files after having configured a setup.
B.1.7	Load configuration files.	Teachers want to load configuration files for each exercise.
B.2	**Change standard setups**	Teachers want to **change standard configuration setups** for exercises.
B.2.1	Add additional components	Teachers want to add additional components (e.g. hosts, routers, switches).
B.2.2	Remove components	Teachers want to remove previously added components from standard configuration setups through configuration files.
B.2.3	Adjust component values	Teachers want to adjust component values such as routing tables and congestion.
B.3.1	**User moderation**	Teachers want to have access to a more **powerful API** than regular users for **moderation** purposes.
B.4.1	**Access externally**	Teachers want to have **access** to the VIP from **outside the lab**.
B.5	**Start network**	

C. Students

ID	Name	Description
C.1	**Reset command line**	Students want to **reset their command line** if they have made errors they cannot undo.
C.2	**Save results**	Students want to **save results and related data** by copying results from the command line.

D. Projects

ID	Name	Description
D.1	**Separate environment**	Project groups want to separate their own VIP environment from others in case of NDA or for a more stable environment.
D.1.1	Save configuration files	Project groups want to save configuration files when doing projects.
D.1.2	Load configuration files	Project groups want to load configuration files when doing projects.
D.2	**Track statistics**	Project groups want to track statistics for analyzing workloads, etc.
D.2.1	Track packet loss	Project groups want to track packet loss when doing projects.
D.2.2	Track delay	Project groups want to track delays between packets when doing projects.
D.2.3	Track uptime	Project groups want to track the uptime of devices when doing projects.

E. Developers

ID	Name	Description
E.1	**Hook up to API**	Developers want an API to hook up new components to.
E.2	**Build from templates**	Developers want templates to build components from. (beyond virtual hosts, routers, and switches).
E.2.1	Template protocols	Developers want the option to build protocols from a protocol template.
E.2.2	Template routers	Developers want the option to build routers from a router template.
E.2.3	Template switches	Developers want the option to build switches from a switch template.
E.2.4	Template hosts	Developers want the option to build virtual hosts from a host template.

F. Virtual Host

- No use cases

ID	Name	Description
F		

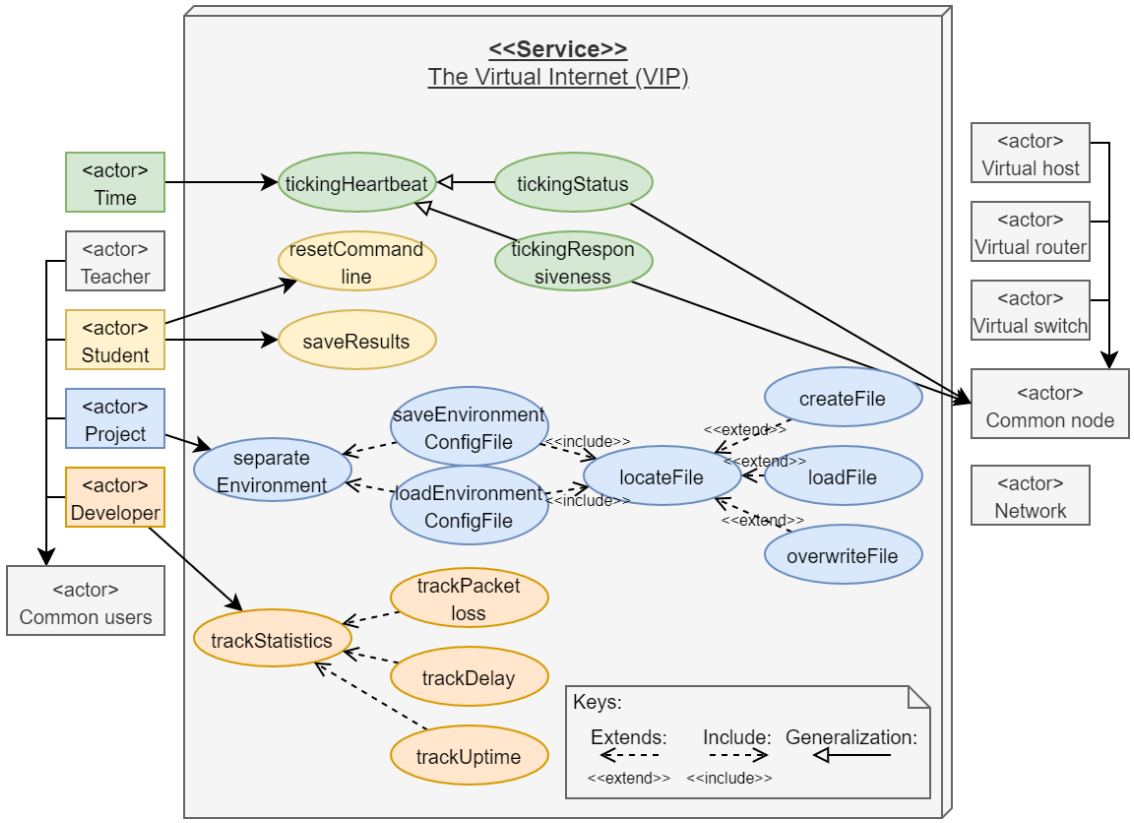
G. Time

ID	Name	Description
G.1	**Ticking heartbeat**	With a certain frequency, components of the VIP should be checked on.
G.1.1	Ticking status	With a certain frequency, the status of components should be checked (up/down/unresponsive.
G.1.2	Ticking responsiveness	With a certain frequency, the responsiveness (speed of reply) should be checked.

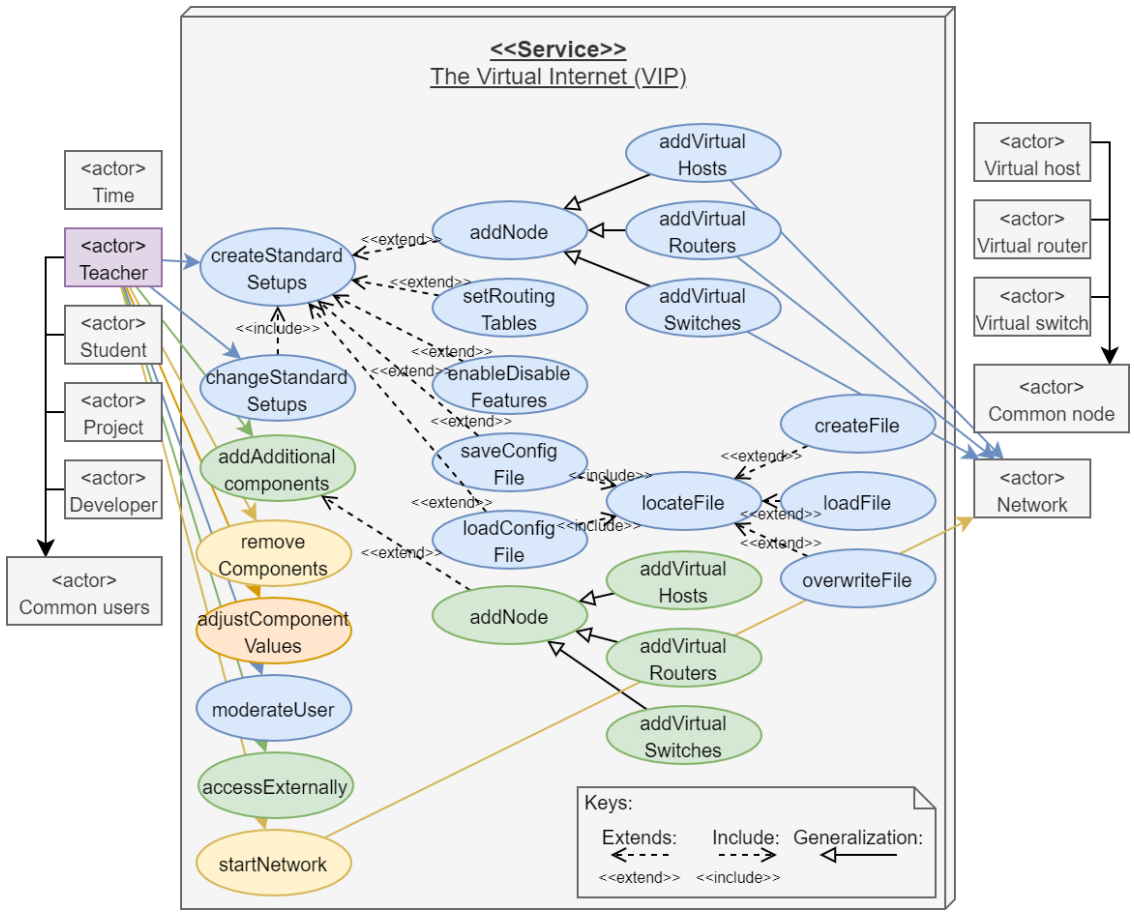
Use case diagrams

The use case diagrams are based on the use cases and actor lists and give a visual overview of which actors are dependent on which use cases.

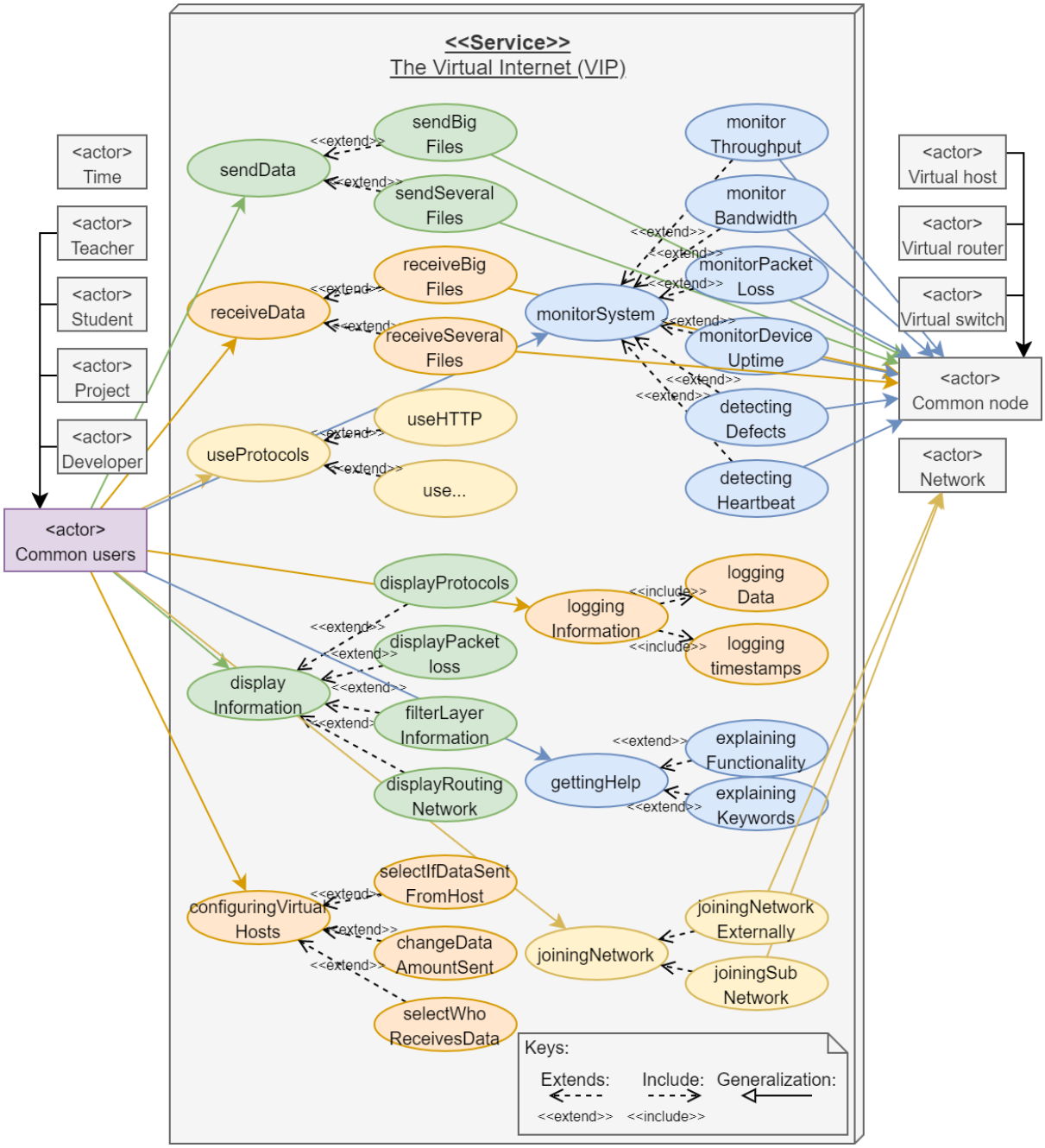
Time, Student, Project, Developer Combo diagram



Teacher diagram



Common user diagram



Finally, it should be clear that the Common users use case diagram is the biggest. This is in part because this is where the focus was, and the rest of the diagrams lack development. All diagrams are unfinished versions in various degrees.

Prioritization of use cases

A Basic MoSCoW analysis (Replaceable with AHP[1]).

MoSCoW analysis

M/S/C/W	ID	Use case	
M	**A.1**	**Send data**	Common users want to send data across the VIP.

[1] Analytic hierarchy process https://en.wikipedia.org/wiki/Analytic_hierarchy_process

	A.1.1	Send big files	Common users want to send big files across the VIP.
	A.1.2	Send several files	Common users want to send several files across the VIP.
M	**A.2**	**Receive data**	Common users want to receive data across the VIP.
	A.2.1	Receive big files	Common users want to receive big files across the VIP.
	A.2.2	Receive several files	Common users want to receive several files across the VIP.
M	**A.9**	**Joining network**	Common users want to be able to join and interact with the networks on the VIP.
	A.9.1	Joining network externally	Common users want to be able to interact with the network as their own external network.
	A.9.2	Joining subnet	Common users want to join specific networks and to live as a part of the internal network on a subnet.
M	**A.3**	**Use protocols**	Common users want to use different protocols for different applications. (Contains many sub-use cases)
	A.3.1	Use HTTP	Common users want to use HTTP to send and receive messages across the VIP.
S	**A.5**	**Displaying data**	Common users want to select what data is being logged and displayed
	A.5.1	Display protocols	Common users want to select which protocol information is being logged and displayed.
	A.5.2	Display packet loss	Common users want to select displaying the packet loss of a device.
	A.5.3	Filter layer information	Common users want to select to filter by layers of the internet model.
	A.5.4	Display routing network	Common users want to display the routing network so they can identify the setup and possibly join a subnet.
S	**A.6**	**Logging information**	Common users want logged information for traversing history.
	A.6.1	Logging data	Common users want to log data so they can traverse previous messages.
	A.6.2	Logging timestamps	Common users want timestamps to be paired to data.
S	**A.7**	**Getting help**	Common users want to access help where they can learn the ropes of the system.
	A.7.1	Explaining functionality	Common users want to access help and get an understanding of the system through explanations of functionality.
	A.7.2	Explaining keywords	Common users want to access help, where they can get an explanation of keywords (parameters -x) and their actions.
W	**A.4**	**Monitor system**	
	A.4.1	Monitor throughput	Common users want to monitor throughput or traffic across a given device.
	A.4.2	Monitor bandwidth	Common users want to monitor the level of bandwidth is across a given device.
	A.4.3	Monitor packet loss	Common users want to monitor packet loss.
	A.4.4	Monitor device uptime	Common users want to monitor the uptime of running (and shut down) devices.
	A.4.5	Detecting defects	Common users want to detect defects (error handling) when devices are not functioning as expected.
	A.4.6	Detecting heartbeat	Common users want to know if devices are continuously running by detecting heartbeat on devices.
W	**B.1**	**Create standard setups**	
W	**B.2**	**Change standard setups**	
W	**E.1**	**Hook up to API**	

Non-functional requirements

To determine the best-suited architecture, non-functional requirements are found. They are written down through utility trees and further defined with belonging tactics (Len Bass, 2013). This ensures that requirements are made explicit, and that the architecture is well thought out when it is designed and realized and so the architecture does not cause havoc down the line.

Business goals

Developer objectives

- (**Important**) Developers will in general have more experience with Java than other programming languages due to SDU's preference for Java in the curriculum. That does not mean students will not have their own favorite programming languages.
- (**Extra important**) It should be easy to introduce users and developers to the system. This means creating a clearly defined architecture that is easy to **modify**, **extend,** and **maintain** with an ever-switching group of developers.
- (**Important**) Developers want to have test setups of the Internal Network to easily test configurations of the VIP.
- Developers are going to need some kind of monitoring service that can track uptime, performance, and reliability of services.
- Developers are going to need a ping system that makes it possible to interact directly with services.

User objectives

- The Internal network of the VIP should be so fast that a class can be using the VIP in a networking course. Courses could contain assignments such as sending lots of data between hosts, creating network congestion, etc. These concepts should act accordingly to the Internet.
- Users want to have configurations of the Internal Network available for courses. In this way, they will be able to save, restore to a previous state, or quickly start a new assignment.
- (**Important**) Users want to be able to use their own pcs to act as clients for the whole system.
- Users need to able to check if nodes they create are available in the Internal Network.

Product quality

- (**Important**) The VIP should be reliable enough to resemble the Internet which is very reliable. The Internal Network is going to have other requirements for reliability than other services in the VIP system as the Internal Network must have the same error handling as the Internet.
- When many users are using the VIP, the system should not bug down due to repeated errors, stalling service, dying services, etc.

Utility Tree: Important

ID	Attribute refinement	Architectural Specific Requirements (ASR) and Tactics
1	**Java programming language**	The core microservices should be developed in Java as Java is the primary programming language taught at SDU, at least currently. This will allow for a common ground for most students possible. Java also supports many useful frameworks. Quality attributes: **Modifiability**, **Usability, Compatibility**
2	**Springboot framework**	The core microservices will be using a combination of Springboot and Maven through the build tool Gradle to realize the microservice architecture. This is a choice by developers early on as the framework should reduce complexity with

		interfaces within each microservice. Additionally, Springboot provides great implementations of RESTful services. Quality attributes: **Modifiability**, **Extendibility, Maintainability**
3	**Programming language independence**	JSON will be used to communicate between microservices. This is done to keep every microservice independent of a programming language and every input/output independent of the service itself. This means that any developer can pick up the project with their own favorite programming language and add additional microservices. Primarily protocol microservices. Quality attributes: **Extendibility**
4	**Layered Architecture**	Using a standardized way of organizing projects can improve uniformity and reduce complexity when the standard is understood by the developer. A 3-layer architecture is a common way to do this where each service is built out of a presentation, domain, and persistence layer. Presentation will handle input from other services, domain will transform data, and persistence will handle output as well as configuration data (databases or locally stored files. Quality attributes: **Modifiability**, **Extendibility**, **Maintainability**
5	**Microservice responsibility**	Every microservice will only have a single responsibility. That is, a service will be taking input, a service will be translating input, a service will be handling nodes, etc. This is done to reduce complexity and increase uniformity. Quality attributes: **Modifiability, Extendibility, Maintainability**
6	**REST / CRUD**	Microservices will benefit from using the REST architecture through the Springboot framework and JSON standard. REST will uniformize microservices and reduce complexity. Quality attributes: **Modifiability, Simplicity**, **Operability**
7	**Configuration file responsibility**	It is important that configurations can be saved so clients can maintain correct settings across multiple sessions (sittings). That does mean that these configuration files should also either be encrypted or (preferably) the code should have limits to configuration values so that users do not manually set them too high or low and crash the system. Quality attributes:
8	**Device-independent**	The VIP should have the potential to be set up on different devices concurrently. This means that the workload can be spread between devices. As the VIP will be based on Springboot and REST this requirement will be easier to fulfill. Quality attributes:

A. Utility Tree: Availability

Since the VIP is a virtual system with no critical or life-threatening elements, it means that system uptime is less of a priority. Yet, there are still several elements that are important and make for a reliable and useful product experience when using the VIP.

ID	Attribute refinement	Architectural Specific Requirements (ASR) and Tactics
A.1	**Ping/Echo**	As a user, it is necessary to be able to detect components to know that they are alive. A way to do this is to make components pingable to determine that they are functioning as intended. Pings are sent by clients to microservices and nodes in the Internal Network.
A.2	**Monitor**	A monitor service will allow for monitoring the “state of health” of individual components and the system. It should also be able to detect congestion on the network as well as errors and inform administrators of the server-side that something is behaving incorrectly.
A.3	**Heartbeat**	In general, every component that is part of the VIP should send out a heartbeat to let administrators know that the components are alive and responding. If the heartbeat expires this should indicate a fault from a component. Heartbeats are sent from the components to the server (opposite of Ping) and in that sense only detect, handle,

		and inform server-side (administrators). **A.3 Heartbeat** is somewhat part of the **A.2 Monitor** pattern.
A.4	**Timestamps**	Timestamps allow for the detection of incorrect sequencing of events, serves as general information as well as when faults have occurred. These are all attributes that are very much tied to the interests of the VIP.
A.5	**Exception handling**	Exception handling allows the VIP to handle issues on its own without potentially requiring a restart. Exception handling becomes increasingly necessary as more and more users rely on the VIP. It is especially important that a client cannot cause errors on the server-side.
A.5.1	Error recovery	A way to handle error recovery is in general to cancel the requests that are outside the spectrum of acceptable values and then return an informative message to the service or client that sends the message. Besides errors in the Internal Network of which some can be considered "realistic features", services should have the ability to survive connection timeouts, incorrect input (at all services), and services should inform why errors happen.
A.5.2	Relaunching services after error	Like in A.5.1, all errors not considered a part of the Internal Network, should force a clean-up and relaunch of the service unless an argument can be made. There is an expected data loss in such situations, and it is important that they are handled accordingly and that the administrator and user are informed.
A.5.3	Removal from service	If a component fails and the error is not human-related, it is possibly better to log the error server-side and remove the component from service. When the system detects the removal, components should be reassigned. Components can be both nodes in the Internal Network or microservices.
A.5.4	Exception prevention	For the most part, the VIP should be so stable (just like the internet) that it can handle queues, live addition/removal of nodes, overload, and packet loss within the Internal Network without crashing. The VIP should also be able to handle disappearing clients and virtual host overtaking / dropout.

B. Utility Tree: Interoperability

As the VIP is two systems of systems (SOS), it is necessary to consider interoperability. That is, how the VIP will discover services, nodes and manage these components, and their interaction between them. Both on a service level, but also inside the Internal Network.

ID	Attribute refinement	Architectural Specific Requirement (ASR) and Tactics
B.1	**Discover Internal Network node**	When a user joins the Internal Network, a virtual host is created or overtaken with an (IP) address. When the user tries to interact with another user (through an (IP) address), the other user is discovered through the Internal Network.
B.2	**Discover client**	When a user joins the VIP by the client service, the client must be registered until no longer active. Logging information, if requested, should be able to automatically flow to the client.
B.3	**Orchestrate**	The VIP is managing a complex network of routers, switches, and hosts with both physical and virtual limitations to how much they can transmit at one given time. Therefore, it is necessary to manage these limitations and in accordance with the Internet where queue sizes and packet dropping plays a vital part.

C. Utility Tree: Performance

There are several important aspects of performance. The most important aspect is that the VIP must be so efficient (fast), that it will not hinder the realistic behavior of the VIP in comparison to the Internet. In other words, the system must be able to process network data. This creates a couple of issues such as how scalable the system can be, how "amateurish" features can be designed, and if potential latency

from processing (because of being slow) could create packet loss due to queues overflowing. A final note is that a lot of the optimization should be done according to the Internet's structure in the Internal Network.

ID	Attribute refinement	Architectural Specific Requirement (ASR) and Tactics
	Node speed in networks	Interactions between nodes must not degenerate to the point where it cannot be used to test systems.
C.1	**FIFO queues**	The VIP will utilize several FIFO queues as each node in the network will have arriving and departing packets. FIFO queues can have issues if a node is sent a large number of packets as the queue will potentially block up. This is not an issue to the Internal Network as it is a functionality that queues can fill up. Other queue types such as Round Robin will become relevant as networking features are developed.
C.2	**Priority queues**	There will be important packets that are prioritized over general network traffic. These packets are generally network health-related packets.
C.3	**Manage event rate**	The VIP is going to be highly event-driven, but to improve performance the interval for repeating scheduled events should be so low that a significant number of users (about a class, 25) will not impact the reliability or speed of the system significantly.
C.4	**Keeping resource cost low**	The system should use as few resources as possible.
C.4.1	Cheap containers	The VIP should only utilize the bare minimum of containers necessary and they should run cheap to run OS's.
C.4.2	Cheap algorithms	The VIP should not use expensive algorithms, and only if necessary, they should be 1:1 compared to the algorithms they are made to replace on the Internet.
C.4.3	Cheap protocols and packets	The Internal Network should not have to pack all the extra data sent from the client, rather, it should be used by a monitoring system to access the nodes and track data directly instead.
C.5	**Limited retries**	Services and nodes in the Internal Network should limit their retries on less important actions such that they do not slow down the service. By limiting retries typically to within a handful, harmful loops will not occur.

D. Utility Tree: Modifiability

As it has been said, again and again, the VIP is an ongoing project that developers are going to be working on in sprints the size of 1 semester (assumption). Modifiability is for that reason probably the most important quality attribute that we can gather information about. The end goal is that the VIP will be so modular, that any developer can pick up the project and develop new features in their favorite programming language, given that they know how to interact with the API. As long as documentation is kept up to date, developers will be able to develop new features synchronously/asynchronously as everything is componentized. Modifiability is partly covered in the first section "Utility Tree: Mixed".

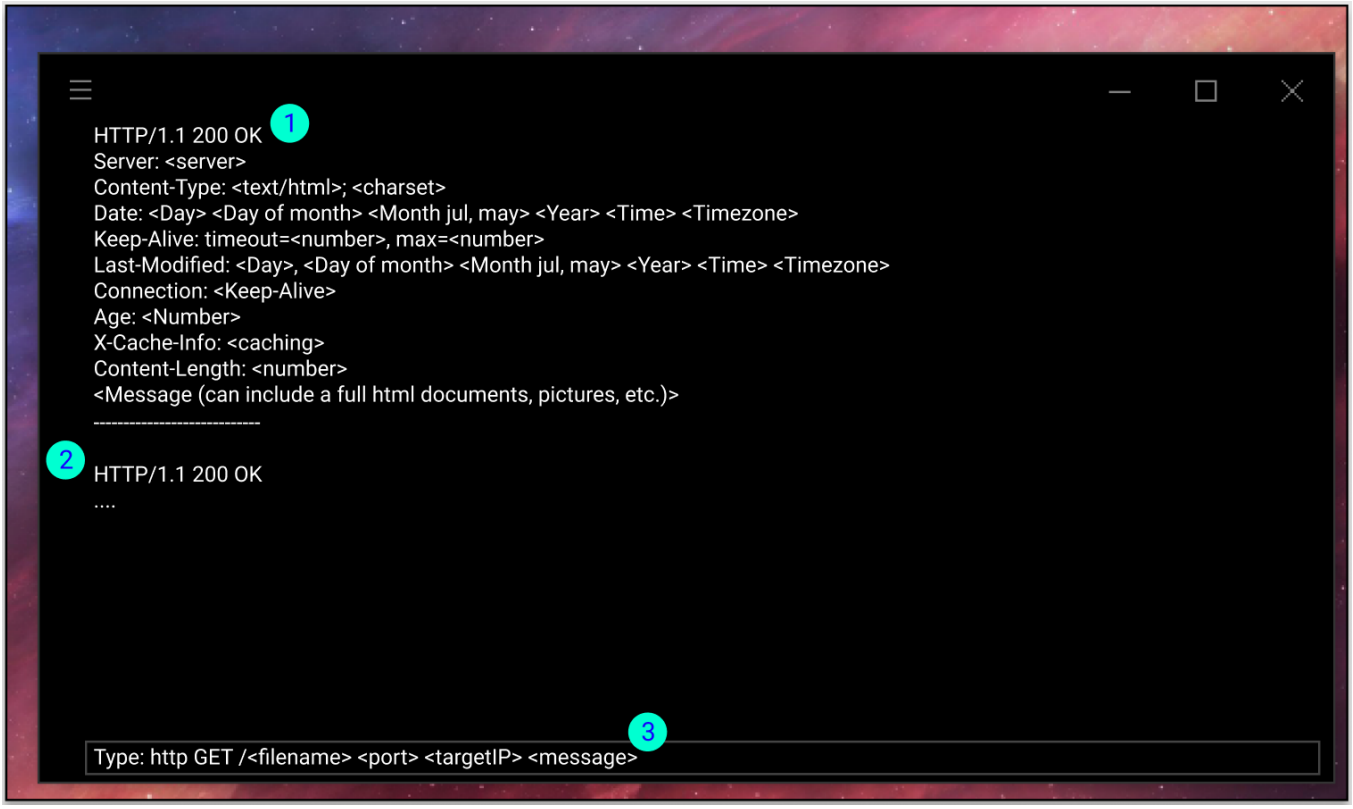
ID	Attribute refinement	Architectural Specific Requirement (ASR) and Tactics
D.1	**Split modules**	Through componentization, modules can be split into separate modules with their own purpose. By splitting modules, they become less costly to change in the future.
D.2	**Increase semantic cohesion**	It is important to ensure that the responsibilities of components are not mixed. If a change to a module that changes core functionality does not change all functionality, the module should probably be split.
D.3	**Encapsulate services and layers**	It is important that at least every microservice uses an API (interface) to communicate through. Preferably every microservice also has interfaces between layers allowing for better testability and modifiability.

D.4	**Use an intermediary**	The VIP will benefit from using a publish-subscribe pattern where hosts will publish themselves to the server as they join the session. Routers will (through routing tables) explore and subscribe to IP addresses. A publish-subscribe pattern will also be used for features such as Protocols.
D.5	**Restrict dependencies**	Visibility of interfaces should be hidden by layers and through componentization. Microservices may only depend on wrappers of other microservices, while internally interfaces should be one way down through the stack.
D.6	**Refactor**	It is important to avoid code duplication as it unnecessarily complicates the code (having to correct it in several places). Code should therefore be refactored depending on the case, into methods, modules, or sub-modules (common services).
D.7	**Manuals**	The project should contain manuals for greater architectural design decisions and must also be defined in the SRS and report with diagrams.

Finally, one issue remains. That is, there is going to be a trade-off between modifiability and performance. The question is if the performance will suffer from all the planned architectural constructs. For the answer to this question, it will have to be developed first and run on a powerful machine, worst case, the system must be simplified and optimized.

Mock-up

As explained in the Inception document. For now, the application will consist fully of a command-line interface where messages can be sent and received through updates and HTTP GET requests.



First iteration

1. An HTTP response from the server.

2. Another response representing that several responses can be viewed at once like a history (with scrolling).
3. An HTTP request that can be sent to a host with a message.

Glossary

ID	Word	Description
	Virtual Internet Platform	The Virtual Internet Platform (VIP) is the product of this project, its function is to mimic the behavior of the Internet but without the outside interference that is real traffic. Synonym: The VIP might also be referenced to as “the platform”.
	Microservices	A Microservice is an isolated project with a business goal that serves the purpose of taking data as input and transforming it based on the business goal. Every microservice is independently maintainable and testable as they are isolated from the other microservices.
	Component	A component is an instance of an object with its own behavior. In general, this can be seen as both nodes in the Internal Network as well as microservices.
	Java	Java is a programming language that is OS independent and can be compiled to run in Java Runtime Environment (JRE).
	Springboot	Springboot is an implementation of the Spring framework and its goal is to provide a framework that makes it simple to develop quality applications with “minimum fuss”. It runs on Java.
	Gradle	Gradle is a build tool which for example can be used to produce pom files (Maven) which manages dependencies, versioning, repositories, and much more. The primary goal of Gradle is to avoid having to type XML. It also contains additional tools for Maven.
	Maven	Maven is a build automation tool just like Gradle, it uses XML to manage dependencies, versioning, repositories, and much more through pom files.
	REST	REST is an architecture pattern used for web development. As almost every application uses the Internet, it has become a standard in the industry. RESTful API’s follow the HTTP protocol and use the methods Get, Post, Put, Delete. REST is stateless and therefore fast and reliable.
	Administrator	An administrator when talking of the VIP is a user from one of the actor categories which in the current situation is managing the VIP server side.
	Virtual Host	The distinction between a host and a virtual host is that the virtual one is a bot within the VIP representing or faking a real user sending and receiving data to make the VIP more “real”.
	TCP/IP model and Layers	In general, layers refer to the 4 layers of the TCP/IP model that the Internet uses. It consists of the Application layer, the transport layer, the network layer, and the link layer.
	Application layer	The application layer handles application to application communication. They handle byte streamed data mostly used in conjunction with HTTP though many other protocols exist on this layer.
	Application layer: Message	A message is a packet of the application layer containing data.
	Protocol	A protocol in the context of this project will always be related to a specific ruleset for network-related code. Homonym: The word “protocol” should not be confused with the regular use of protocol which means a ruleset for a series of actions as in a procedure.
	HTTP	HTTP or Hyper Text Transfer Protocol is an application layer protocol that handles the encoding and decoding of data.
	Transport layer	The transport layer handles correctly delivering data from one device to another ensuring that the end host is the correct receiver. It also ensures

		that data ends up in the correct order for the application layer to process. The transport layer also handles packet loss, that is if the network layer drops packets, they are retransmitted.
	Transport layer: Segments	A segment is a packet of the transport layer containing an application layer message inside. The segment has a segment-specific header.
	TCP	TCP is a transport layer protocol that is iconic for its "handshake" approach to connections. It works by first establishing a connection both ways before transactions of data occur.
	UDP	UDP is a transport layer protocol that is simpler than TCP. It works by a sender sending data to other hosts but without checking for packet drops and confirming that packets arrive. This makes it a lot cheaper but also less stable and as such, it is rarely used for data-sensitive transactions such as emails unless modified.
	Network layer	The network layer handles routing through the Internet and thus is the network itself. Through routing tables, routers send data across the Internet for it to eventually end up at the correct host.
	Network layer: Datagram	A datagram is a packet of the network layer containing a transport layer segment inside. The datagram has a datagram specific header.
	Link layer	The link layer handles data delivery between routers and switches through ethernet, Wi-Fi, etc.
	Link layer: Frame	A frame is a packet of the link layer containing a network layer datagram inside. The frame has a frame specific header.
	Subject	The subject is the system domain that actors are going to interact with. It is thus the one element that developers can modify to accommodate every actor.
	Actor	An actor is any kind of user that must be able to interact with the system.
	Use case	A use case is a specific scenario describing an actor's interactions to arrive at a goal. Synonym: Scenario
	Functional requirement	Functional requirements are derived from use cases and define features a system needs to satisfy actors of a system.
	Non-functional requirement	Non-functional requirements qualities of functional requirements and the overall product architecture. Nonfunctional requirements are derived from business goals and not only use cases, unlike functional requirements.
	Quality attribute	Quality attributes are related to nonfunctional requirements and are labels for general positive traits a system can have such as performance, modifiability, extendibility, usability, etc.
	Architecture Specific Requirement (ASR)	Architecture Specific Requirements or ASRs are more in-depth descriptions of what quality attributes a system needs.
	Tactics	Tactics are specific solutions to ASRs with smart goals that are achievable.
	Utility tree	A table that lists ASRs tactics and quality attributes together as a solution to a nonfunctional requirement.

Exercise examples

This section describes the types of exercises that students could be working on. The exercises are roughly based on the already existing exercises to give an idea of what students might encounter and therefore how the students might interact with the VIP.

Lab 1 - Network introduction

1. Understanding a request response.
    a. Identifying a packet header.
    b. Identifying protocol.
    c. Identifying response message information.

Lab 2 - HTTP

1. Retrieving long documents in several responses.
    a. Document size.
    b. Images.
2. Look at / pause displayed message flow/history.
3. Understanding status codes.

Lab 3 - DNS

1. Identifying IP based on DNS request.
2. Understanding IP Config.
3. Understanding DNS cache.
4. Analyzing what transport layer protocol DNS uses.

Lab 4 - UDP

1. Selecting and analyzing a UDP packet.
2. Identifying source and destination ports.

Lab 5 - TCP

1. Filtering for TCP packets.
2. Identifying SYN/SYNACK messages.
3. Identifying data fields.
4. Identifying RTT (Round Trip Time) and timestamps.
5. Monitor congestion information.

Lab 6 - ICMP

1. Pinging servers and identifying ICMP protocol.

Lab 7 - Ethernet and ARP

2. Identifying Ethernet frames.
3. Identifying ethernet addresses.
4. Identifying ARP messages.

Bibliography

Frank Tsui, O. K. (2018). *Essentials of Software Engineering, Fourth Edition.* Jones & Bartlett Learning.

Jim Arlow, I. N. (2009). *UML 2 and the Unified Process, Second Edition.* Pearson Education Inc.

Len Bass, P. C. (2013). *Software Architecture in Practice.* Pearson Education Inc.

Nikolaidis Pavlos, J. C. (2011). *Software Requirements Specification for JHotDraw.*