A Gather.town Clone in Links

Caitlin McDougall

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2023

Abstract

This skeleton demonstrates how to use the `infthesis` style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and should be used as a starting point for your thesis. Replace this abstract text with a concise summary of your report.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Caitlin McDougall*)

Acknowledgements

Any acknowledgements go here.

Table of Contents

1 Introduction **1**

2 Background **2**

2.1 Web Technologies . . . 2

2.1.1 Classic Approaches . . . 2

2.1.2 Links . . . 2

2.2 WebRTC . . . 3

2.2.1 Client-Server vs P2P . . . 3

2.2.2 ICE . . . 4

2.2.3 Signalling . . . 4

2.3 Video Conferencing Systems . . . 4

2.3.1 Gather.town . . . 5

2.3.2 FluidMeet . . . 6

2.3.3 Accessibility through Multiple Interfaces . . . 6

2.4 Model-Driven Architecture . . . 7

3 Design **8**

3.1 System Overview . . . 8

3.2 Javascript Foreign Functions . . . 9

3.3 Server . . . 9

3.3.1 Grid Creation . . . 10

3.3.2 Broadcast Client Movements . . . 10

3.3.3 Trigger Client Calls . . . 10

3.3.4 Forward WebRTC Messages . . . 10

3.4 Client . . . 11

3.4.1 Client Mailbox . . . 11

3.4.2 Model-View-Update . . . 11

3.5 User Interface . . . 12

3.5.1 Landing Page . . . 12

3.5.2 Spatial View . . . 12

3.5.3 Static View . . . 13

4 Conclusions **14**

Bibliography **15**

A First appendix **17**

A.1 First section 17

B Participants' information sheet **18**

C Participants' consent form **19**

Chapter 1

Introduction

Chapter 2

Background

2.1 Web Technologies

Since its inception, the internet has become integral to the way we live our lives as it allows us to find information instantly, share our interests with peers, buy groceries, and much more. In order to enable the internet to function as smoothly as it does, a variety of web technologies are utilised.

2.1.1 Classic Approaches

Commonly, the client-side consists of markup languages such as HTML and XML to specify the basic structure and content of the web-page, a styling language such as CSS to provide additional styling, and a language such a JavaScript to provide dynamic functionality. On the server-side, languages such as Python, Java and PHP can be used for performing more intensive computations and functionality. Finally, languages such as SQL and XQuery allow connections to databases which store the huge quantities of data which may be requested on a website. These different layers, or tiers, allow modulation of websites and allows for languages which are tailored to specific purposes.

Unfortunately, having this variety of programming languages to choose from at each level creates friction known as the impedance mismatch problem. Impedance mismatch problems are issues which arise as a result of combining technologies which use different archetypes [8].

2.1.2 Links

For this reason, the Links programming language has been created, offering a new approach to web technologies which replaces this three-tiered web model with a single language. Links is a strict, statically-typed, functional language which aims to replace the three-tiered web system with a single-source language [1]. It does so by providing a translator from the Links code to JavaScript and SQL, with the functional aspect of the language providing additional benefits such as database query optimisation, continuations for web interaction, and concurrency with message passing.

The pattern classically used when programming in Links to achieve concurrency is the Actor Model. In this model, an Actor is responsible for managing its own state and performing computations. In this way, there is no global state shared across actors or processes which allows for concurrency since computations are safe from attempting to modify the same memory location. Messages are sent between different actors and when an actor receives a message, it can perform any of 3 concurrent actions: send messages to other actors, create additional actors, or prepare for how subsequent messages will be handled [4]. The type and order of these messages are ensured by defining session types. However, the linear nature of session typing is not well-suited to the implementation of graphical user interfaces (GUI) which makes it more challenging to create webpages in Links.

For this reason, an adaptation of the Model-View-Update (MVU) architecture introduced by Elm has been added to Links [2]. Elm, like Links, is a functional language and, as such, the architecture it presents is particularly well-suited to functional programming. In this architecture, a model contains the state of the application, a view function renders the model, and an update function handles messages produced by these models and produces new models. The extension to the MVU architecture allows for session typing by adding commands (which enable side effects as seen in traditional web technologies), linearity, and model transitions.

Since the Links language is still in active development, it still lacks some functionality which is provided by JavaScript. Fortunately, Links offers a Foreign Functions Interface (FFI) which allows custom JavaScript functions to be written and called from a Links application such that any missing JavaScript functionality can still be included by the programmer. In particular, this paper will make use of FFI to make use of the WebRTC framework to achieve real-time communication between clients.

2.2 WebRTC

WebRTC is a set of standards making use of peer-to-peer connections to enable real-time applications such as text-based chat, audio sharing, and live video-conferencing [10]. This framework allows peers to send and receive information directly, without first sending the data through a server which can create delay.

2.2.1 Client-Server vs P2P

In the traditional Client-Server network architecture, devices in a network act as either a client or a server. Clients make requests for services and content which are received and actioned by the server, a higher-performance entity which is usually connected to a large number of clients [12]. In this way, clients are not connected directly to one another and in order to share content must instead send content first to the server which can in turn forward this to the destination client.

On the other hand, in the peer-to-peer (P2P) architecture, devices in the network can act as either a client or service provider at any point. In this way, devices in a P2P network share their combined resources in order to send content directly to one another without passing through a central entity [12]. WebRTC chooses to make use of the P2P network architecture due to the fact that these direct connections between clients give less delay for real-time applications while also giving more privacy to those communicating [10].

Although WebRTC uses P2P connections for file transfers during a session, it does make use of servers to properly manage each P2P connection. For example, when two clients access a video-conferencing website and want to communicate with one another, a signalling server will be set up to allow initial communication between the two clients.

2.2.2 ICE

In order for two clients to communicate with one another directly, they first need to know where within the network the other client is located, or in other words, their IP address. However, it is likely that both clients are connected to a router which performs Network Address Translation (NAT). This means that the local IP address of the client is hidden from the public network and replaced with a public IP address which is the address external hosts should use to communicate with the client. Since this translation happens at the router, the client does not know its own public IP address and so can't automatically tell other clients which address to use to connect to it.

Fortunately, Session Traversal Utilities for NAT (STUN) servers give clients the ability to ask for their own public IP address such that they can distribute this information to clients or servers they want to be able to connect to them. If, however, the router uses symmetric NAT, which means that the NAT mappings are dependent on both the source and the destination IP addresses, this no longer works as other servers would still be unable to connect since their IP address is not trusted. In this situation, Traversal Using Relays for NAT (TURN) servers are instead utilised to route messages to and from the clients, therefore creating a client-server connection rather than a P2P connection.

Interactive Connectivity Establishment (ICE) is used to describe the framework of collecting the different candidates which may be used to connect to the client such as their local IP address, public IP addresses given by STUN servers, or TURN server IP addresses.

2.2.3 Signalling

The ICE candidates are bundled up along with information about available media types and formats according to the Session Description Protocol (SDP). Clients can then exchange these SDP formatted messages as an offer to the other client and the other client responds with an answer until an agreement on how to communicate is achieved. This process is called Signalling and is facilitated by the Signalling server. Once the direct connection is agreed, the clients can then exchange media using a P2P connection.

2.3 Video Conferencing Systems

Technology has long been used to communicate with one another from afar, but mostly in the form of telephone calls or text-based messaging. However, the ability to send and receive video streams has revolutionised the way businesses, schools, and social circles function. The ability to share video with multiple users at the same time has become particularly relevant in the wake of the Covid-19 pandemic which saw the use of video-conferencing systems (VCS) boom as this became the closest to face-to-face meetings we could get.

A variety of VCS are available to use, with popular systems including Zoom, Microsoft Teams, and Cisco Webex all sharing a similar interface. The interface consists of live video feeds for a subset of participants, a larger area displaying the current speaker or currently shared screen, a small area showing the user's own live feed, and an area for text-based conversation. These systems also come with an increasing number of additional features to mimic the way we interact in real meetings such as raising hands, white-boarding and breakout rooms.

However, even with the inclusion of breakout rooms, these systems fail to adequately represent the spatiality of real meetings and social interactions. For example, a teacher may walk around the edge of a classroom to get a sense of who is struggling, while in these virtual environments they need to enter every breakout room, potentially interrupting the conversation. This limitation of traditional VCS has motivated new approaches to video conferencing which take into account this spatial awareness and enable more natural virtual interactions.

2.3.1 Gather.town

One such VCS offering proximity-based interactions is Gather.town [6], an online platform which allows companies or individuals to create a gamified virtual meeting space in which customised avatars can move around and interact with other avatars. When two avatars are within close proximity they will be connected and begin sharing video, whereas when they are far apart the connection stops and they are unable to see or hear one another. In addition to this, spaces can be set up for broadcasting to a large room of people as would be possible during a presentation and tables are available in which everyone can communicate with one another similarly to the traditional Zoom and Teams approaches.

Additionally, Gather.town is increasingly incorporating many of the features offered by traditional VCS such as white-boarding, screen-sharing and file-sharing. This ensures that users do not miss out on any functionality through switching to Gather.town. Gather.town also provides features not offered by static VCS as it has the unique ability to provide interactive games which users can play. This allows for a much more natural experience in environments such as classrooms or icebreaker sessions. Games offered include Chess, Codenames, Tetris, and more [7].

Some limitations of Gather.town which have been highlighted are its limit of 25 participants in the free version, and the performance of the software when multiple users are connected on a poor internet connection [13]. It has also been suggested that this visual interface could be inaccessible for those who find too much visual stimulus overwhelming or those with visual impairments who may struggle to interact easily with the virtual environment.

Overall, Gather.town provides a much more interactive experience for users and has been shown to reduce fatigue associated with virtual meetings. In addition, a study by [11] determined that both educators and students preferred using Gather.Town in comparison with using other static VCS such as Zoom and Teams. However, accessibility of Gather.Town may be an issue for those with visual impairments or sensitivity to visual stimuli. Therefore, there is a need for a Video Conferencing system which provides the benefits of Gather.Town's spatial and gamified interactions while maintaining an appropriate level of accessibility for those with visual impairments.

2.3.2 FluidMeet

Another alternative approach to live Video-Conferencing is proposed by FluidMeet [5]. In this system, the authors have focused on creating flexible boundaries between conversations (unlike the rigid break-out room boundaries offered by alternatives such as Teams). In doing so, they aim to offer more natural transitions between conversations and provide a more visual representation of different conversations.

FluidMeet achieves more natural transitions by offering the option for breakout rooms to be fully or partially open to others [5]. When fully open, users who are not in the group are able to hear and see what is going on in that group before joining. When partially open, non-group members instead have access to keywords being used in the conversation in the form of a word cloud and they can view audio visualisations in order to gain a sense of the atmosphere of a group.

Having access to all this information could be overwhelming for users, but FluidMeet offers the ability to choose which of these features are displayed, reducing the visual overload which may occur. This is an important feature to note as allowing users to customise how much information they receive enhances the accessibility of FluidMeet to those with visual impairments or those who just prefer a simpler interface.

2.3.3 Accessibility through Multiple Interfaces

A study by Gappa and Nordbrock (2004) explored ease of use in internet portals (search engines) which included individuals with hearing impairments, visual impairments, learning difficulties, and the elderly [3]. Their findings showed that all participants valued a clear and simple design. This shows that adding many layers of complexity which may be preferable for some, is not fit for purpose in all situations.

In the context of Video Conferencing, it is extremely important that disabled students, employees, and teachers have the same access to these systems and can communicate effectively with peers and colleagues. Therefore, if companies and schools want to use these interactive and spatial VCS, there need to be features included to allow disabled or elderly individuals to participate equally.

The ideal system is one which caters to both the needs of those who prefer simpler interfaces and those who prefer a more interactive experience. Therefore, this paper explores offering a choice between a simplified interface offered by static VCS and a gamified experience as offered by Gather.town. However, this produces a challenge in terms of allowing those using one interface to interact with those using the alternative interface.

2.4 Model-Driven Architecture

A Model-Driven Architecture (MDA) is one approach which allows us to abstract this problem of having multiple views from the underlying technical details. MDA uses an incremental approach to the system design process in which models are kept at the forefront of thinking [9]. These models represent different levels of abstraction of the underlying system. For example, in our VCS, the two visual representations offered to the user are two different models which each represent a high-level abstraction of the underlying system model. Once these models have been designed from the highest to the lowest level, and the interactions between them have been defined, they can then be converted to code.

Chapter 3

Design

3.1 System Overview

Links aims to provide a frontend language which provides programmers with the tools to create any application which can be created using traditional web-development technologies. In order to further test that this is possible, we outline the design of a Video-Conferencing web application written in Links. This application not only facilitates live video and audio communication between users but also takes advantage of the Model-View-Update architecture provided by Links to offer multiple interface options to the user.

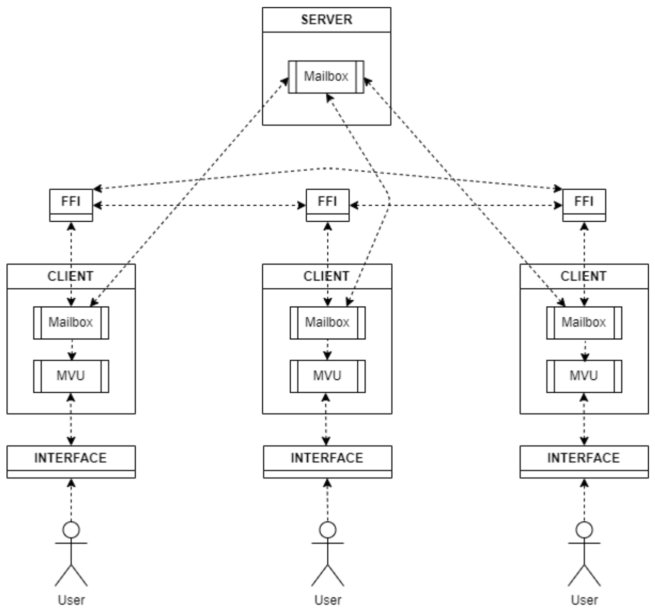

Figure 3.1: System diagram of VCS components and interactions.

A diagram of the components and their relationships is shown in Figure 3.1. The application consists of a Server which maintains the overall state of the application. When a user enters the web address at which the application is hosted, a Client process is generated that hosts a variety of sub-components: a mailbox which allows message exchanges with the server; a JavaScript-based foreign functions interface which handles WebRTC interactions and media capture; a model to maintain the client state; an update function which updates the model in response to actions; and a view function which uses the model to generate HTML to be displayed to the user. The clients are initially only able to exchange messages with the server but once an RTC connection is made between two clients, they are able to exchange messages directly.

As described in subsection 2.3.1, allowing more natural, spatial interactions between users in a video-conferencing system can be beneficial for enhancing engagement with online meetings and reducing fatigue. However, it is also noted in subsection 2.3.3 that it is important to offer a more traditional, simplified interface for those who may be overwhelmed by too much visual information at once. Further, if both of these interfaces are to be offered, it is vital that the individuals utilising different interfaces are not segregated from one another. Therefore, our application will make use of Links' in-built MVU architecture to allow different view functions of the same underlying model state such that users with each interface can interact with one another. From the perspective of the user, when they access the application they will first be prompted to choose their preferred video-conferencing style, static or spatial, and this choice will influence how the underlying model state is displayed to the user by the View function.

A full description of each component of the system is given later in this chapter.

3.2 Javascript Foreign Functions

In order for clients to send and receive live video and audio data, our video-conferencing application first requires the ability to identify and access available media devices of the clients. Additionally, the WebRTC framework API is integral to the live communication of this video and audio between clients. However, these functionalities cannot currently be accessed directly from Links.

Fortunately, as discussed in subsection 2.1.2, we are able to make use of Links' Foreign Functions Interface (FFI) which allows custom Javascript functions to be called directly from the Links code. This allows us to access JavaScript functionality which has not yet been added to the Links library such as media capture and WebRTC API calls.

The primary functionality provided to the Links application by our Javascript library is to maintain and manage RTC connections between peers. In order to achieve this, the library must make several functions available: capturing local streams from client media devices, setting up new RTC connections and adding any local or remote video streams, creating and receiving offer SDP messages, creating and receiving answer SDP messages, checking for new local ICE candidates, adding new remote ICE candidates, and closing existing RTC connections.

3.3 Server

The server is an integral component within the application that maintains the system's overall state. This includes knowledge of which clients are currently accessing the application, the overall grid of squares in which clients may be located on the map in the spatial view, and the location of different rooms on this map. Additionally, although the WebRTC process will primarily depend on direct connections between clients to allow for maximum communication latency, a server is still necessary to act as a middle-man between these clients when they initially set up these direct Peer-to-Peer connections.

3.3.1 Grid Creation

In order to ensure easy interaction between clients accessing the Spatial view and those accessing the Static view, a grid system is used to represent the map of rooms seen in the Spatial view. This grid consists of entries which each represent a square area of the map with each square holding a pixel location, which room it is a part of, and how many clients are currently on that square. In this way, when a client in the Spatial view moves to a new square in the grid, it is easy to find both the pixel location for displaying the character in the correct position and the room they are in which will be displayed within the Static view. Additionally, when a client in the Static view selects a new room to enter, they can be placed in a square within the correct room and using the number of clients on each square, they can be placed in a square which is unoccupied if such a square exists.

Upon initialisation of the server, it will immediately begin creating this grid such that it is ready when clients begin accessing the application. The server is also responsible for updating the grid when clients move in and out of squares to ensure each square holds the correct number of occupants.

3.3.2 Broadcast Client Movements

Clients using the Spatial view need to be aware of where other clients are on the map to allow them to communicate. Those in the Static view instead need to know which room each client is in. For both of these to work, the server must broadcast any changes in the client's position or room to every other client.

3.3.3 Trigger Client Calls

The server is aware of each room's occupants and is therefore in charge of alerting relevant clients of any changes they should make to their connections when a client enters a new room. In particular, it must signal each client in the new room to begin a call with the entering client as well as signalling every client in the previous room to close their connection with this client.

3.3.4 Forward WebRTC Messages

The server must then forward any SDP messages between the connecting clients which will inform them of which addresses they can use to communicate directly with the other client. Once this P2P connection is made, the clients can communicate live data directly but the server must still handle the forwarding of any changes in the ICE candidates available to a client to ensure they can continue communicating smoothly.

3.4 Client

3.4.1 Client Mailbox

When a user accesses the application webpage, they will be assigned a client process which includes a mailbox for receiving and handling messages sent to it by the server process. All functionality provided by the client process will be executed within the client's browser rather than on a separate server system and thus each client maintains only their own state without direct access to any other client's state. In order to share state information with the server and other clients, the client process must exchange messages with the server which will then determine whether the information should be propagated to other clients.

Messages received by the client's mailbox can invoke a variety of functionalities, many of which invoke the WebRTC JavaScript Foreign Functions as described above. Functionalities offered by the mailbox include opening and closing RTC connections to maintain the state of client calls, initiating calls by generating and sending SDP messages to the server which will propagate this to the desired client, handling SDP messages propagated by the server from other clients, and sending newly available ICE candidates to connected clients through the server.

3.4.2 Model-View-Update

So far, the client follows Links' standard Actor-based model. However, as described in subsection 2.1.2, this model's linear session typing is not ideal for the creation of GUIs and so our client also makes use of the Model-View-Update architecture. This architecture instead allows us to utilise side-effects as exist within JavaScript such that the client's state can be maintained and smooth transitions between models can be made. In our application, the model held by the client tracks the state of both this client and information about other clients received from the server. This information includes the id, name, grid position, pixel position, and room of each client. The model also tracks the current view we should be displaying to this client (i.e, the landing page, the static view, or the spatial view), as well as a list of rooms and their current members for displaying in the static view.

The Update function accepts messages and updates the state according to the message type and parameters. Our update function has multiple purposes: setting the unique ID of the client as assigned by the server, setting the name and icon of the user once selected, updating the view upon selection and alerting the server that the client has entered this view, handling subscribed events such as pressing the arrow keys to move, updating the rooms and positions of clients when the server signals a change, and responding to requests for its own current position which will then be broadcast to other clients.

However, since the Update function is only accessible by the client through user actions and interactions with the user interface, we need a method of changing the model's state when it receives messages from the server instead. In order to achieve this, we use the Dispatch functionality offered by Links for MVU-based applications. In this manner, the server sends messages to the client's Actor-based mailbox as usual, and then the client itself can then dispatch this message to its model's update function to be processed and update the model accordingly.

The View function of the client will then generate and display an HTML representation of the current model state. Since the application allows different choices of view, the layout of the HTML entities depends on the current view state. This allows us to maintain almost exactly the same underlying model and update functionality for each system, with different high-level representations of this state observed by users.

3.5 User Interface

Since the principal aim of this application is to allow a choice of user interfaces to the client, the client's MVU architecture described above gives the perfect solution for providing this. The application can therefore provide 3 view functions depending on which page the user is on: The landing page, the Spatial map-based view of the VCS, and the static view of the VCS.

3.5.1 Landing Page

The landing page of the application is the same for every client when they enter the application. It prompts the user to enter information to be displayed once they enter the room such as their name, and their character icon. Once the user is happy with the information they have entered, they will select which view they would like to be displayed to them: spatial or static.

3.5.2 Spatial View

The spatial view of the application is designed to give the user a gamified experience of interacting with other users which more closely resembles the real-life experience than classic VCS systems such as Zoom and Microsoft Teams. In this view, the rooms available for entry are displayed to the user as the map displayed in **IMAGE** with the passages in between the labelled rooms representing the Lobby which is the default room upon entry. The client and other users of the VCS are displayed as pixel art icons along with their names and will appear in the position/room in which they are currently situated. In order to navigate between rooms, users click the arrow keys to move their character from grid square to grid square. When a client enters a new room, the video and audio feeds of other clients in the new room will appear while those from the previous room will disappear. If other clients are instead using the static view of the application (without a map), they will still be displayed in the correct room on the map view in an unoccupied grid square such that those using the spatial view are still aware of which room these users are in and can join their room if desired.

3.5.3 Static View

This previously described spatial view works well for engaging users and providing a more natural environment for conversations. However, to those with visual impairments or learning disabilities, this visual-heavy interface may prove overwhelming and discourage them from engaging in social gatherings taking place in a VCS environment. For this reason, the static view of the underlying application instead provides the user with the minimum necessary information to allow them to join calls with other users. This view resembles the classic VCS application such as Zoom and Microsoft Teams, displaying the client's own video along with those of connected clients. The application also allows the client to navigate between rooms by displaying each available room as a button along with which users are currently present in that room.

A side-by-side comparison of the spatial and static views of the same state is displayed in **IMAGE + EXPLAIN**.

Chapter 4

Conclusions

Bibliography

[1] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web programming without tiers. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, pages 266–296, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[2] Simon Fowler. Model-view-update-communicate: Session types meet the elm architecture. *CoRR*, abs/1910.11108, 2019.

[3] Henrike Gappa and Gabriele Nordbrock. Applying web accessibility to internet portals. *Universal Access in the Information Society*, 3(1):80–87, Mar 2004.

[4] Carl Hewitt. Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.

[5] Erzhen Hu, Md Aashikur Rahman Azim, and Seongkook Heo. Fluidmeet: Enabling frictionless transitions between in-group, between-group, and private conversations during virtual breakout meetings. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Gather Presence Inc. Gather.town. `https://www.gather.town/`, October 2022.

[7] Gather Presence Inc. Integrated games. `https://support.gather.town/help/integrated-games`, October 2022.

[8] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A classification of object-relational impedance mismatch. In *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*, pages 36–43, 2009.

[9] Amal Khalil and Juergen Dingel. Chapter four - optimizing the symbolic execution of evolving rhapsody statecharts. volume 108 of *Advances in Computers*, pages 145–281. Elsevier, 2018.

[10] Rob Manson. *Getting Started with WebRTC*, pages 24–34. Packt Publishing Ltd, 2013.

[11] Colin McClure and Paul Williams. Gather.town: An opportunity for self-paced learning in a synchronous, distance-learning environment. *Compass: Journal of Learning and Teaching*, 14(2), 2021.

[12] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102, 2001.

[13] Xin Zhao and Colin Derek McClure. Gather.town: A gamification tool to promote engagement and establish online learning communities for language learners. *RELC Journal*, 0(0):00336882221097216, 0.

Appendix A

First appendix

A.1 First section

Any appendices, including any required ethics information, should be included after the references.

Markers do not have to consider appendices. Make sure that your contributions are made clear in the main body of the dissertation (within the page limit).

Appendix B

Participants' information sheet

If you had human participants, include key information that they were given in an appendix, and point to it from the ethics declaration.

Appendix C

Participants' consent form

If you had human participants, include information about how consent was gathered in an appendix, and point to it from the ethics declaration. This information is often a copy of a consent form.