SIVAMANI THANGAVEL
DEVELOPER TOOLS FOR NETWORKED WEB APPLICATIONS

Master of Science thesis

Examiner: Prof. Kari Systä
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 7th December 2016
ABSTRACT

SIVAMANI THANGAVEL: Developer tools for Networked Web Applications
Tampere University of Technology
Master of Science thesis, 51 pages, 6 Appendix pages
24 May 2017
Master’s Degree Programme in Information Technology
Major: Communication Systems and Networks
Examiner: Prof. Kari Systä
Keywords: Development frameworks for Internet of Things, liquid IoT, liquid web IDE, networked programming, liquid software, user identity migration

Networked web applications consist of applications running in multiple devices, communicating with each other. The status of the system and data are exposed through Web, thus providing a platform independent user interface which can be accessed without installing any external software. Examples are cloud based services, IoT and liquid software.

The thesis focuses on low power, low cost programmable devices, essentially IoT networks with a Web based liquid user interface that flows between personal computing devices. Features common to IoT systems such as communication, deployment, security, data collection and management can be implemented as a programming framework. Such a framework will help the application developers focus on implementing the domain specific purpose of the network.

An average consumer (in most parts of the world) owns at least two personal computing devices, a laptop and a smart phone. Providing a liquid application for programming and keeping track of the IoT system’s status allows user to access the application from any personal device and switch to a different device with minimal or no effort.

This thesis evaluates suitable frameworks/solutions for programming IoT devices and creating a liquid Web UI. LiquidIoT of Tampere University of Technology, is an open source programming framework for networked devices. Memory usage of this framework is optimized and user interface is redesigned to make it more user friendly as part of the thesis work. Two liquid software frameworks, liquid.js and XD-MVC are analyzed for the suitability of moving application state. A liquid software also needs to move the user identity when moving between devices. Different approaches to moving user identity are considered and evaluated with a prototype.
PREFACE

Thanks to Kari Systä, for guiding me through the research process and documenting it. Thanks to all the developers and researchers, who have contributed to the IoT and liquid software. Special thanks to all of those who have worked on or currently working on the LiquidIoT, Liquid.js, XD-MVC development frameworks and concepts. Thanks to the companies like Nokia, Almamedia who are interested in and investing in the research projects in IoT and liquid software.

Tampere, 24 May 2017
## CONTENTS

1. Introduction ................................................................. 1

2. Networked things and liquid applications ................................. 3
   2.1 IoT ................................................................. 3
   2.2 Liquid Software ................................................... 6

3. IoT: Programming frameworks .............................................. 8
   3.1 Requirements ...................................................... 9
   3.2 Available platforms and frameworks ............................... 13

4. LiquidIoT framework ....................................................... 17
   4.1 Architecture ...................................................... 17
   4.2 Usage ............................................................. 17
     4.2.1 Application development .................................... 20
     4.2.2 Device initialization ....................................... 22
     4.2.3 Deployment .................................................. 22
     4.2.4 Multi-device programming ................................. 24
     4.2.5 User Interface ............................................... 24

5. LiquidIoT improvements ................................................... 25
   5.1 Evaluation ......................................................... 25
   5.2 Memory optimization ............................................. 26
   5.3 User interface redesign .......................................... 29

6. Liquid software: programming frameworks ............................... 35
   6.1 Requirements ..................................................... 35
   6.2 Available tools and frameworks .................................. 37

7. Liquid application with user identity ................................... 40
   7.1 Development using Liquid.js and XD-MVC ........................ 40
   7.2 Approaches for User Identity transfer .......................... 46

8. Developing Networked Web Applications ................................ 49
9. Conclusion and future work ........................................ 51

Bibliography ........................................................................ 52

A. APPENDIX: Screen captures of LiquidIoT new UI .............. 55

B. APPENDIX: Prototypes used for liquid framework comparison .... 59
LIST OF FIGURES

3.1 Simple IoT network ........................................ 9
4.1 LiquidIoT components ..................................... 18
4.2 LiquidIoT programming and deployment .................. 19
4.3 LiquidIoT application template ............................. 21
4.4 LiquidIoT deployment sequence ............................ 23
5.1 LiquidIoT use cases ......................................... 30
5.2 LiquidIoT new home page .................................. 34
A.1 LiquidIoT new projects page ................................. 56
A.2 LiquidIoT new page to edit project ......................... 56
A.3 LiquidIoT new add API page ................................. 57
A.4 LiquidIoT new deploy page ................................. 57
A.5 LiquidIoT new API page .................................... 58
B.1 To-do application with Liquid.js ............................ 59
B.2 To-do application with XD-MVC ............................ 60
# LIST OF TABLES

5.1 Memory usage of Node.js modules .......................... 27

7.1 Liquid.js XD-MVC comparison ................................. 46
**LIST OF ABBREVIATIONS AND SYMBOLS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth Low Energy</td>
</tr>
<tr>
<td>CC license</td>
<td>Creative Commons license</td>
</tr>
<tr>
<td>CoAP</td>
<td>Constrained Application Protocol</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICE</td>
<td>Interactive Connectivity Establishment</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message Queue Telemetry Transport</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NFC</td>
<td>Near Field Communication</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency IDentification</td>
</tr>
<tr>
<td>STUN</td>
<td>Session Traversal Utilities for NAT</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>TURN</td>
<td>Traversal Using Relay NAT</td>
</tr>
<tr>
<td>TUT</td>
<td>Tampere University of Technology</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WebRTC</td>
<td>Web Real Time Communication</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

Internet revolutionized the way we share information. It made it possible for people to share information with each other regardless of the geographical distances. The Internet carries information produced by humans for other humans. With Internet of Things, data is produced and shared by the devices. It can also be referred to as 'for the devices’ when data processing and decision making based on data is automated.

There are already everyday household objects embedded with computing devices accessible remotely (smart fridges and washing machines ¹). Most of the new cars have options for remote accessibility. The number of connected devices per person is increasing rapidly [11]. It is convenient for the consumer, when all devices could be monitored and interacted with remotely.

Programming allows us to automate tasks such as mathematical computations, text processing etc, using a sequence of instructions/commands to a computing machine. Programming the IoT devices lets us to automate data collection and analysis. While a simple sensor can collect data, having a programmable device lets us process the data at the point of collection. As IoT finds its use in every field imaginable [13], having programmable devices lets us to use the same small computers in different fields, with domain specific programs.

With the availability and afford-ability of laptops, smart phones, tablets and also e-readers, more and more of them are getting activated each day [5]. Ideally these devices could work together as a single system with multiple displays. It would be an efficient usage of the resources and allow the user to switch between devices in the middle of an activity. In reality these devices are usually from different manufacturers and do not work together. Mac and iOS from Apple and Android from Google offer some level of liquid experience, but they are limited to vendor specific ecosystems.

agreement can be provided as a liquid application, by moving the application state when the user switches devices. When an IoT network uses user authentication and authorization, the user identity should also be transferred between the devices.

The thesis focuses on the tools and frameworks, for developing networked web applications. Software execution environments in both IoT and personal computing devices vary a lot, depending on the vendor and cost. They might have different CPU and memory capabilities, run different operating systems (Linux, Windows, Mac etc). If a different language (C, C++, Java, C# etc) is used for the development for different platforms, the code for each platform will be different. When a feature needs to be added to the program, it has to be added to the code for all platforms. This makes maintaining the program difficult and costly. The wide variety of computing and physical objects getting connected together needs generic programming models and frameworks that can be used across platforms and devices with a wide range of memory availability.

Chapter 2 provides an overview about IoT systems and liquid software. Chapter 3 analyzes the need and requirements for IoT programming framework and discusses some of the tools and frameworks currently available. Chapter 4 provides an in-depth view of the LiquidIoT framework. Chapter 5 discusses the changes made to the framework as part of this thesis.

Chapter 6 analyzes the need and requirements of liquid programming frameworks and discusses the relevant tools and frameworks. Chapter 7 explains the possible approaches for transferring application state and user identity of a liquid application.

Chapter 8 unifies the concepts of liquid software and IoT to create networked applications with Web user interface. Chapter 9 concludes the thesis, with evaluation of the work done and future work needed.
2. NETWORKED THINGS AND LIQUID APPLICATIONS

The term IoT (Internet of Things) came into existence in a presentation made by Kevin Ashton at Procter & Gamble (P&G) in 1999. The term represented the idea of the computers collecting and processing accurate data about the physical things, through sensors and RFIDs (Radio-frequency identification). This data would help to determine when a thing would need replacing, updating or remodeling, thus reducing the safety risks and also the costs involved [3]. Recommendation ITU-T Y.2060 (06/2012) provides the definition of IoT as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving inter-operable information and communication technologies [8].

The term 'liquid software' refers to a software or an application, which can migrate from one device to another fluidly. This is not a new technology, but a way of programming the devices to provide a seamless experience across multiple devices to the user. The software could be moving between devices owned by a single user or between devices of different users, interacting with each other [15].

2.1 IoT

IoT consists of a wide range of technologies, devices and applications. The main components are the devices embedded or attached to things, network connectivity and a centralized data cloud. In general, it can be defined as a group of physical objects (things) embedded with network connectivity, sensors, actuators etc., which enable them to collect and share data. The devices in their simplest form could be for measuring/controlling environmental conditions like temperature, light etc. It could also be something that provides more advanced service. A fridge aware of its inventory, can update the shopping list when a specific food item is running out [1].

The communication between devices can happen directly or through a centralized system which keeps track of all the devices and applications in the system. An
application can run in its entirety in a single device for the whole of its life cycle or move between devices during its life time. Such liquid application can be server-based if the application state is maintained in a common server and the devices synchronize the state with the server. The application could also be client-based if the devices directly share the state of the application with each other.

One example of an IoT system is a smart house, where the temperature and lighting are measured and controlled using a network of connected things equipped with sensors and controllers. The system could provide a pleasant temperature and lighting when there are people present in the house and switch to a more energy efficient setting when everyone leaves the house. An example of bigger scale is a smart university building, which is equipped with sensors and controllers to measure temperature, light and noise levels - which could be used to provide a more pleasant environment. A huge network of devices can monitor the noise levels, air quality etc for a whole city, which can be used to regulate traffic, decide the location of industries etc. Analyzing the data collected for a long period of time could help design the buildings and cities better.

When the things in an IoT network are programmable, remote interaction with the physical world is possible using the existing network infrastructure. Instructions for measuring and controlling the environment could be programmed into small computers directly connected to the sensors and controllers. When there is a change in requirement, the small computers can be reprogrammed remotely. The programmable devices in the network usually are small single-board computers (examples are Raspberry pi, Intel Edison etc). Rest of this document refers these small computers as *(IoT)* device.

If there is only one device measuring and controlling the temperature of a small room, there is no need to connect it to network. The device can be programmed to read the temperature every few seconds, switch the heating on if the temperature is below a certain value and switch the heating off and turn cooling on, if the temperature is above a certain value. Programming can be done by simply connecting to it using a cable (most of the small computers have an Ethernet port) and using a software tool (like SSH) which allows remote connection.

The need to connect the things to a network arises, when there is a need to monitor or program them remotely. Even if there are only 2-3 programmable devices located in a small area, it becomes time consuming to connect to each device physically, every time a change has to be made to the program. A program could change due to change in the number of sensors connected, a change in the limits used to trigger
the control or raise an alarm. When these devices are connected to the network and become remotely accessible, this can be categorized as a IoT network.

**Evolution of IoT**

The first appliance connected to the network was a coke dispensing machine at Carnegie Mellon University. It reported the amount of inventory and whether there are cold cokes available for purchase [19]. By connecting a physical device to the network, the coke machine became a remotely accessible computer. This provided the proof of concept of the usefulness of connecting physical things to Internet.

There have been many technical developments since then. Low power communication protocols and RESTful APIs make programming and connecting power constrained devices easier. The devices with no power constraints could use the existing technologies like WiFi or LTE. Low power options such as LoRaWAN, IEEE 802.11ah (an amendment of the wireless networking standard for low power consumption) are some of the technologies suitable for power constrained IoT devices. IPv6 allows addressing a huge number of devices compared to IPv4, enabling us to address huge volume of IoT devices. HTTP over TCP could be used for the communication between the devices. There are also options like the MQTT protocol designed specifically to be light weight, for contained environments like Machine to Machine (M2M) and IoT. MQTT uses a publish/subscribe model for distributing the messages. MQTT’s protocol exchanges and the overhead are kept at a minimum, to reduce network traffic.

IoT could be used in homes, on people to measure the health of each individual, on pets to track their location and to monitor their health, in the soil to measure the temperature and moisture level to provide optimal growth conditions for plants etc [13]. As IoT finds its use in a wide range of fields, this could soon lead to explosion of devices. These devices can cause environmental issues by consuming energy when they are being used and by degrading the environment when they are disposed. In the future, IoT devices can be replaced with much simpler ones that doesn’t come with the need for a power source or battery. They can be printed on environmental friendly material and use energy from naturally available light, energy produced from movements or vibrations etc. They can be powerful enough to collect, process and transmit data from sensors connected to them [10].
2.2 Liquid Software

The multiple computing devices owned by an average consumer are generally separate entities and have no interaction with each other. If software can make them work together as a single entity, it would be easier for the user to switch between the devices in the middle of an activity (like writing an email, reading a book or a blog post, managing an IoT system etc).

"Liquid software manifesto" [17] describes liquid software as follows:

1. In a truly liquid multi-device computing environment, the users shall be able to effortlessly roam between all the computing devices that they have.

2. Roaming between multiple devices shall be as casual, fluid and hassle-free as possible; all the aspects related to device maintenance and device management shall be minimized or hidden from the users.

3. The user’s applications and data shall be synchronized transparently between all the computing devices that the user has, insofar as the application and data make sense for each device.

4. Whenever applicable, roaming between multiple devices shall include the transportation / synchronization of the full state of each application, so that the users can seamlessly continue their previous activities on any device.

5. Roaming between multiple devices shall not be limited to devices from a single vendor ecosystem only; ideally, any device from any vendor should be able to run liquid software, assuming the device has a large enough screen, suitable input mechanisms, and adequate computing power, connectivity mechanisms and storage capacity.

6. The user shall remain in full control regarding the liquidity of applications and data. If the user wishes certain functionality or data to be accessible only on a single device, the user shall be able to define this in a simple, intuitive fashion.

To provide an effortless multi-device switching experience, liquid applications will need to support the following scenarios.

- Sequential screening: User starts using the application in one device and then continues using it in another device. In this case the state of the application will have to move, when the user switches the device being used to access the application.
2.2. Liquid Software

- Simultaneous screening: A single user accesses the application at the same time in different devices. All devices could be set to show the same view or each of them could be set to play a role. This needs the application to broadcast the state changes as it happens, to all the devices involved.

- Collaborative screening: Multiple users using the application at the same time using different devices. The purpose might be to view the same content simultaneously (viewing a photo album together, while located remotely).

Irrespective of the scenario, the user interface of the application must adapt itself to the device being used.

One of the well known software which can be categorized as liquid software is Google documents, which allows user to edit a document from multiple devices and also allow multiple users to edit the same document simultaneously. This is made possible by having the storage in the cloud. Synchronization is done by tracking the changes being made to the document by all browser sessions accessing it and merging the changes in a non-conflicting way. When a user uses one device to edit the document and later switches to a different device, the document is not presented automatically to the user to continue the editing activity. The user has to navigate to the relevant document.

Apple’s Handoff is another example as a liquid software which allows the user to continue an activity from one device to another. For this to work, the devices have to be registered and logged in using the same iCloud identity. The applications that want to support this feature must use the Handoff APIs provided by Apple. This lets the application developers create a seamless switch from one device to another. Being a proprietary feature and requiring Apple iCloud account, this can be used only if the devices involved are all Apple products.
3. IOT: PROGRAMMING FRAMEWORKS

IoT systems share common features, regardless of the domain. These features can be converted into requirements for a programming framework for IoT. The need for a programming framework is analyzed by considering progressively complex IoT networks.

If the network is a simple one with 5 devices or less, we can use the most suitable language for each device depending on the platform, memory and CPU availability of the particular device. This however means that the developer should know multiple languages. It would be easier if the devices have similar specifications as it will allow us to use one language to program all the devices. If the devices are all meant to perform the same action (say, to measure the temperature every 5 minutes and send the data to a central server), it is sufficient to write a single program and deploy the same to all of the devices. A simple example is shown in Figure 3.1.

Even with a network of 5 devices, if we deploy 5 applications to each, we will end up with 5 code bases and 25 application instances to keep track of. Tracking the application instances is necessary to know that they are working as expected. One way is check the status manually, by connecting to each of the devices and checking the application logs. If all the application instances are required to send data to a central server at regular intervals, the server can notify user if some application does not send data anymore.

All of the IoT networks, regardless of their business purpose have the following common features.

- Multiple low power devices connected together. The devices may be running different platforms.
- Devices are running applications for measuring/controlling things.
- The applications send data to a central server or exchange data between each other.
- Data collected from the devices have to be analyzed.
3.1 Requirements

![Diagram of a simple IoT network](image)

**Figure 3.1 A simple IoT network**

Having a framework that supports these features will help the application developers to focus on implementing the purpose of the network. It saves time, effort and provides a standard way of implementing these common functionalities for all IoT networks.

### 3.1 Requirements

The following are the major features an IoT programming framework needs to support.

- **Device management:** The framework should maintain a list of all the devices in the network, provide communication status of each device and notify user about any error condition that needs attention. It should also maintain a list of the applications installed in the devices and the status of the applications. The devices in the network should be reachable through the framework. This is needed to install new applications and update existing ones. There might also be a need to support updating the firmware of the devices.

- **Communication mechanism:** There are communication protocols and architectural principles that make programming IoT devices feasible. Only Application layer communication protocols are considered as part of this thesis.
  - **HTTP:** With the usage of RESTful(REpresentational State Transfer) architecture, HTTP works well as a communication protocol between IoT devices and framework. Framework exposes RESTful services, to be used
by devices for sending data. Devices expose RESTful services that can be used for programming them. Devices can also talk to each other using the same technology. Each device runs a Node.js server which hosts the REST APIs. Framework can also be hosted with the Node.js environment, though for a large network, there might be need for load balancing using tools like Nginx.

- **MQTT**: Message Queue Telemetry Transport is based on a publisher-subscriber model. This protocol is created to support resource constrained networks and has overhead as less as 2 bytes. The publisher and subscriber are linked indirectly through the broker. The publisher publishes the topics and the data to the broker. The subscriber registers the topics it is interested in hearing about. This decouples the publishers and subscribers as it eliminates the need for them to know how to reach each other. An IoT device with temperature sensors can publish the temperature data. Another device wired with LEDs to indicate the temperature can subscribe to the topic and process the data. The protocol handles the case of a subscriber application registering after the publisher has sent the message, by allowing the publisher to inform the broker to retain the message(s) for the subscribers arriving late.

The protocol supports different QoS (Quality of Service) options: QoS 0 is the quickest ’fire and forget’ option which provides no guarantee that the message is published and received by the subscribers, QoS 1 ensures that the message is received, but doesn’t guarantee that the message will arrive just once, QoS 2 is the slowest and guarantees that a message arrives only once. QoS between sender and MQTT broker is set by the sender. Receiver set QoS is used between the broker and receiver.

Paho\(^1\) is one of the open source libraries that provide implementation for client side of the MQTT protocol. Mosquitto\(^2\) is one of the open source implementations of the MQTT broker. IBM Watson implements the MQTT broker with limited support for retaining messages.

- Platform independence: Programming should be possible using one language. Either all the devices should have the same software platform installed or we have to chose a language that can run in all the devices, if we want to maintain only one version of a program for all the devices in the network. Writing the program in a low level language like C might work, but the program needs to be compiled and tested for each of the platforms separately. High level

---

\(^1\)https://www.eclipse.org/paho/
\(^2\)http://mosquitto.org
languages like Java and nowadays C# can be used as there is runtime support available for a wide range of platforms. But these require a lot of resources in the system, which the IoT devices are usually short of. Of the current languages available, JavaScript with Node.js as runtime is a good option as Node.js supports a wide range of platforms.

Platform independence of the language, doesn’t necessarily guarantee that one version of each program is enough. When it comes to interacting with the sensors and actuators, each hardware vendor may provide their own library that exposes the APIs to interact with them. One solution is for the framework to provide a library that can detect the platform and connected hardware and use the relevant firmware.

- Multi-device deployment: When the number of devices in the network increases, it becomes a tedious job to deploy the programs to each of the devices by connecting to each of them individually. Even with 5 devices and 5 programs, there is 25 application instances to deploy. The framework can simplify the step by providing an option to deploy an application to multiple devices simultaneously. The framework connects to each of the devices and installs the applications in the background. It should report the status of the installation back to the user.

There should also be options to choose only a subset of devices to install an application. If a program is written to measure and report light levels in a room, it makes sense to choose only the devices which have a light sensor attached, for deploying the application to. Another criteria might be based on location of the device. This needs capability in the device to provide location information. A simpler, but time consuming way is to manually configure the location of the devices in the framework when adding it to the network.

Once the applications are all installed successfully, the framework should check the status of the application instances and report it back to the user. Framework can extract the status from the logs of an application or each application can expose an API that can be queried to get the status. The framework can provide a real time status update or provide an option for the user to check the status.

Most programs undergo change, either due to bugs identified in the code or due to changes in requirements of the program. In the initial stage of establishing an IoT network, there will be many new programs to deploy. As time progresses and the network grows, we will have new devices to deploy programs to and existing devices to which we need to send updates of the programs deployed in them. If we assume that program code changes every
3.1. Requirements

week, in a network with 5 devices and 5 programs on each of them, we will have to update 25 application instances. The updates can be done manually with the following steps:

- Connect to the device.
- Stop the instance of the application to be changed.
- Uninstall the existing version of the application and install the new version (this could also be simply replacing the old files with new ones).
- Start the updated application and check that it runs properly.

This becomes a time consuming process, as the number of devices and applications increase in the network. It doesn’t add any value to the network itself. The time could be used for more productive purposes, if this step could be automated. If we have a framework that supports deploying to multiple devices, it should also to support updating and removing applications from multiple devices in one step.

- Security and integrity: With lots of devices being networked and programmed, making the connections secure is very important. Platform should make sure that no rogue devices can be attached to the network, no device can impersonate another device, unauthorized deployment is not allowed etc. Some platforms address this by providing a unique identifier for each device and the device needs to use that identifier for any further communication. This provides some level of security, but does not prevent every possible attack on the system. Since this is a complex topic that would require a thesis on its own, it is not addressed in this thesis.

- Data collection and analysis: The reason behind connecting all things to programmable devices is to collect data and act upon it. IoT framework should provide APIs to the things for sending data to a central point and APIs to the users for analyzing the data collected. It should be possible to access both live data and historical data.

- Support low memory devices: Popular IoT devices currently used like Raspberry Pi, Intel Edison etc have 1 GB of memory. There can be devices with lower memory however and it is important that the framework can support programming devices with memory as low as 64MB.

Similar requirements are defined in this paper [4], with emphasis on the framework being open source.
3.2 Available platforms and frameworks

There are both open source and commercial frameworks available for programming and collecting data from IoT devices. The following sections describe some of the tools and frameworks available currently. Most frameworks provide data collection, analysis and visualization. Only a few have options for programming the IoT devices. The following sections list the major features and the frameworks that implement the specified feature.

Device management and data collection

IBM Watson Internet of Things platform\(^3\) is a data collection service which allows IoT devices to send data to it and exposes APIs to users to analyze the data that has been collected. It provides MQTT endpoints for the device to communicate with the platform and RESTful end points for the applications that want to analyze the data. A sample data collection IoT program provided in the documentation, written using JavaScript, sends movement and acceleration information when run on a mobile device. To be able to send data, the device needs to be registered with the framework. The framework generates a URL for the device. On opening the URL in the IoT device (in this case a mobile device), it loads a script which listens to the movement and orientation change events of the device and sends it to IBM Watson using MQTT. The sample uses the MQTT implementation of the Paho library. The IoT dashboard provides a real time visualization of the data as raw numbers and in a graph. It also provides the device management functionality. It allows users to add and manage the devices. It has a device model which includes a list of attributes like device type, device id, type id, serial number, firmware version etc. It is possible to use both HTTP and MQTT for communication with the platform. TLS(Transport Layer Security) can be used both with HTTP and MQTT, if the certificate used by the device is registered with the platform.

Thingsboard\(^4\) is an open source IoT platform that provides device management, data collection, analysis and visualization. Adding a device requires registering the device with a unique name. The platform provides a unique access token for the device, which must be used when the device connects to the platform. The platform is available for use on-line, but can also be hosted in-house. This also has a set of attributes for the device, which can be configured using the dashboard. The functionalities are available through a Web based UI and also as REST APIs. HTTP,

\(^{4}\)https://thingsboard.io
3.2. Available platforms and frameworks

MQTT and CoAP (Constrained Application Protocol) protocols are supported for communication between the device and the platform. Secure versions of the protocols are also supported. The dashboard has visualization options for the data collected from the devices.

Device discovery

AllJoyn framework\(^5\) is an open source framework that can be used to expose and discover services in a network. AllJoyn comprises of Apps and Routers. Apps can talk to each other through routers. App(s) and Router(s) can reside in the same device or in different devices. In case of low power IoT devices it is recommended to run Routers in a separate device. AllJoyn uses an introspection XML which has the list of properties and methods exposed by a producer in the network. The properties may also expose an event that notifies of changes in their value. This XML works as the interface or contract between the producer and the consumer. This is a generic framework for discovering and advertising services, designed with consideration to low power devices.

Programming

Eclipse IoT is an open source framework that uses Kura \(^6\) for device and application management. Kura is based on the OSGi platform for Java. It provides a dynamic component model by letting the components (termed as bundles in OSGi and written by IoT application developers) expose their interfaces by registering them as discoverable services. Availability of a service, models the corresponding IoT device’s availability.

Communication is accomplished by using either MQTT or CoAP. Both of them are designed for situations where network bandwidth is limited. MQTT runs over TCP/IP using a publish/subscribe model. CoAP, a RESTful protocol, runs over UDP and provides the same resource model as HTTP. Eclipse’s Paho library provides implementation of MQTT and Californium library\(^7\) provides CoAP implementation.

Node-RED provides a flow based programming model for the Internet of Things. It provides functionalities in the form of configurable nodes, which can be arranged in a flow. The flow is stored in JSON (JavaScript Object Notation) format with the

\(^{5}\)https://allseenalliance.org/framework
\(^{6}\)https://www.eclipse.org/kura/
\(^{7}\)https://www.eclipse.org/californium/
nodes, properties of nodes and the connection between the nodes. Output from one node can be injected into the input of another node by dragging between the output port of the previous node to input port of the next node in the UI. Some of the in-built nodes are ‘HTTP in’ to accept HTTP requests, template node to include static content and ‘HTTP out’, the response node to reply to the request. Using a ‘Function’ node allows injecting JavaScript function into the flow. The Web UI for programming is hosted in a Node.js server. Node-RED is also available as npm module\(^8\) and can be embedded into another Node.js application. The functionalities exposed through the Web UI are also accessible via APIs provided by the Node-RED.

The guidance to create new nodes to be added to the palette are provided in the Node-RED documentation \(^9\). Once the Node.js server is installed to each IoT device, flows can be deployed to the device using the Web UI. Flows can also be imported and exported, which allows creating templates which can be deployed later by importing to the device. This allows maintaining a repository of flows in one place and programming a device remotely. Node-RED can also be used to simulate a device as it can be run in laptops and personal computers. IBM-BlueMix has support for simulating a device using Node-RED.

By default, Node-RED is not secure and anyone who can access the Node.js server can deploy and remove flows from the device. All functionalities available in the Web UI are implemented as RESTful APIs in the server. There is in-built support for adding security through configuration or custom implementation. Authentication can be added by configuring the ‘adminAuth’ settings. Authentication can be of type ‘credentials’, in which case user name, encrypted password and corresponding privileges are configured in the settings of the Node-RED configuration. It can be a custom implementation which is configured by setting the ‘adminAuth’ setting to require the module with custom implementation. Once authenticated, Node-RED provides an access token, that has to be added to the ‘Authorization’ header of subsequent HTTP requests.

There is also option for adding authentication to the routes exposed through the ‘HTTP in’ nodes. This is done by configuring the user name and password in the setting ‘httpNodeAuth’. Access to static content can be secured by configuring the user name and password in the setting ‘httpStaticAuth’\(^10\).

LiquidIoT framework developed in the pervasive computing department in TUT is meant for programming devices remotely. It supports programming wide variety of

\(^{8}\)https://www.npmjs.com/package/node-red
\(^{9}\)https://nodered.org/docs/creating-nodes/
\(^{10}\)http://nodered.org/docs/security
3.2. Available platforms and frameworks

devices with different memory availability. One of the major challenges in dealing with programming IoT devices is that they are often from different vendors and run different platforms. Though most IoT devices run a flavor of Linux, there are devices with proprietary Operating System software.

LiquidIoT provides platform independent programming by using JavaScript as the programming language and Node.js as the execution engine, which is supported by almost all of the Operating Systems. The framework provides options to develop applications using the Web UI, deploy applications to one or more devices, remove installed applications from the devices, update and monitor the applications. This framework’s features are discussed in more detail in the next chapter.
4. LIQUIDIOT FRAMEWORK

This chapter provides an in-depth analysis of the LiquidIoT framework’s architecture, usage and guidance for the creation of new projects and adding new devices to the framework.

4.1 Architecture

The main components of the LiquidIoT as shown in Figure 4.1 are the following:

- Framework Web UI (IDE): provides the user interface for programming the devices. It also has options to deploy, update, monitor and remove applications.

- LiquidIoT server: Back end of the IDE. Manages developer written applications and communicates with Resource Registry and devices.

- Resource registry: maintains the list of devices in the network, along with the URLs to reach each of the device, physical location, device capabilities, installed applications, implemented REST APIs etc

- LiquidIoT application server in device: This is installed to a device, when the device is added to the network. This is a Node.js hosted component, that turns the device into an IoT application host, which can receive applications to be deployed and run them.

- Applications: These are the programs written by the developers who use the framework and deployed to the IoT devices. The purpose of the application depends on the domain. In general, applications send data to some central data collection unit. They could also record and process the data themselves and raise alarms if the data exceeds certain limits.

4.2 Usage

For accessing the devices and programming them, the framework provides a Web based UI. Devices and applications can also be monitored and managed through
the UI. The following sections explain the steps of adding a device to the network, creating a program and deploying the program to the device. The sequence of these steps is represented in Figure 4.2.

- **Add the device to the network**: The process of adding the device is not automated and have to be done manually as explained in the steps below:

  - This step might differ for each of the device. Different vendors provide different options. Some of the devices need configuring a WiFi network with the credentials supplied by the vendor. The device connects to this network. The user can then take remote control of the device (using tools such as SSH) and change the WiFi settings to point to the network, in which the framework server is running. Some devices have an Ethernet port and this makes it easier to directly connect to the relevant network or configure the WiFi settings.
4.2. Usage

![Diagram](image)

**Figure 4.2 Programming with LiquidIoT**

- Install the LiquidIoT application server to the device. This component is hosted in a Node.js application and it handles the communication between the framework and the device.

- The next step to adding the device to the LiquidIoT network is to configure the device capabilities. The list of capabilities is dynamic. The developer can define a capability ‘measureTemperature’ to all devices connected with a temperature sensor. The only requirement is that the developer should use the name consistently. Capabilities are used by the framework to check the compatibility of a program to a device.

- Start the LiquidIoT application server in the device. When the application is run for the first time, it registers the device to the resource registry and to the framework server (if it is not already registered). While registering it notifies the framework of the URL to reach itself, device capabilities, location etc.

- The device should now be present in the list of devices in the UI of the framework. Check the IDE to make sure it is listed.

- Write the program: The Web UI provides interfaces for creating a project.
4.2. Usage

When a new project is created, the framework presents the user with a template which has a sample implementation of the interface. The developer can change the implementation of the interface, based on the purpose of the program. Each program can be configured to require certain capabilities to be able to run. The project can also expose REST APIs. The APIs are defined using Swagger editor. The REST API definition can also require specific device capabilities that need to be supported. A program with the requirement ‘measureTemperature’ can only be deployed to devices, which has the capability ‘measureTemperature’.

- Chose the device and deploy the corresponding program to the device. The framework sends the application to the device, by using the framework’s device component. It installs the application and reports the status to the framework server, which updates the UI to report the status to the developer.

The framework also provides the following functionalities.

- Multi-device deployment: The framework provides options for deploying an application to all of the devices or a subset of the devices (based on the device’s name, location, capabilities, already deployed applications etc).

- Updates: The framework provides options for choosing the set of devices, that has a specific application deployed and then update this application.

- Monitoring: There are interfaces in the framework for getting the status of a deployed application and for checking the logs of the application.

- Multi-device programming: It also provides the features for adding a REST API to a project. This REST API in one application can be consumed by another application, thus providing a basic infrastructure for multi-device programming.

The following sections provide detailed information about the interfaces expected to be implemented by the projects and how the device Node.js component handles registration of the device and deployment of the applications.

4.2.1 Application development

The Framework Web UI provides a code editor, which is capable of verifying the JavaScript syntax. When a new project is created, framework adds the files which
have the necessary infrastructure of LiquidIoT application. There are 3 methods that need to be implemented by the developer of the IoT application. Each of these get a callback function of the infrastructure component as a parameter. A sample is shown in the Figure 4.3.

- Initialize - Any steps that is needed to initialize the resources used by the application should go here. It needs to invoke the callback supplied as parameter to indicate to the framework that the initialization is complete.

- task - The main purpose of the application is implemented here. In case of sensors, it is usually measuring variable(s) and sending the measured value to a central computer. It can also keep track of the measurements and report only if the value exceeds certain limits.

- terminate - This interface is called when a user shuts down the application. This can also be called by the framework, when a user wants to upgrade/-downgrade the application to a different version.
4.2.2 Device initialization

When the IoT device component in the device runs for the first time, it registers the device by calling the Resource Registry. It provides the following information to the registry.

- Address which the framework can use to reach the device for further communication. This includes the IP address where it can be reached and the port number the device component is listening to. All the communications between the device and framework happens through the device component.

- It also provides the location information (location is typically preset and is configured by the developer when adding the device to the network).

- Capabilities: This is based on the sensors and actuators a device is connected to. A device with a temperature sensor and a speaker attached to it, will have the capabilities canMeasureTemperature and canPlaySound (configured by the developer currently). This can be automated if the device component can detect the sensors and actuators connected directly to the device.

When writing an application, the developer specifies the device capabilities required to run the application. If the application is for measuring the temperature, the device should have a temperature sensor attached to it and should have reported it to the device registry. This check is done by the framework when deploying an application. The framework imposes no restrictions on the capabilities that can be used in the system. It only checks that the application’s required capabilities are part of the device’s capabilities according to the Resource Registry entry. Some applications may not require any specific capabilities (an example is one that implements a heartbeat which helps the framework detect whether a device is online). There is no restriction on what the application measures and how it processes the measured value. This allows flexibility to connect any device that can run Node.js framework. Only constraint is that the device should have enough memory to run the framework device component and the deployed applications.

4.2.3 Deployment

It is possible to deploy an application to a set of devices. The framework allows choosing the devices based on one or more of the following criteria:
4.2. Usage

**Figure 4.4 Deploying an application**

- Name: Either with exact match or devices with similar names. This will be most useful in a system which follows a standard naming convention.

- Capabilities: This criteria can be used when there is a need to install applications to devices with similar capabilities.

- Location: This is useful when the network is spread across well defined locations. An example use case is to change the temperature limits for raising an alarm in a particular part of a building.

- Deployed applications: This is useful in cases where a new application has to be installed to devices which are running a specific application. An example use case is to add a light sensor to all devices with temperature sensors and then deploying the program corresponding to the light sensor.

The sequence corresponding to deploying application to an existing device is shown in Figure 4.4.
4.2.4 Multi-device programming

REST API can be added to an application, by first defining the API in the customized Swagger Editor \(^1\), adding it to the project and implementing it. Another application running in the same device or a different one can then use the API to interact with this application. This allows the devices to communicate with each other directly, without involving the framework.

Previous implementation required the developer to hard code the address of the REST API in the calling application. Recently, this has been automated by one of the developers of LiquidIoT. When an application is registered to the Resource Registry, the APIs it implements are also sent as part of the application information. Consumer of an API requests a framework component deployed as part of the application, to call the API by passing the name, HTTP method etc. Consumer can choose whether to call only one implementation of the API or call all of the implementations present in the system. Framework component consolidates the results of the call and sends it to the application.

4.2.5 User Interface

The IDE is the part of the framework the developer interacts with and is provided as a web application. The UI is responsive/adaptive to desktop, laptop, tablet and mobile devices. A developer using the UI gets the current state of the system. Since the IDE is provided as a Web UI, there is no need to worry about installing/updating it in multiple personal devices\(^6\). When a developer edits a project, the changes are instantly saved to the LiquidIoT server and this lets a developer who uses multiple devices for development (example desktop and laptop) switch easily between different devices and continue with the task. This supports the scenario of sequential screening of liquid software.

\(^1\)https://github.com/ahn/swagger-editor
5. LIQUIDIOT IMPROVEMENTS

The fitness of LiquidIoT against the major requirements for an IoT programming framework was evaluated. Optimizing the memory usage and redesigning the UI to make it more user friendly are the major improvements identified and implemented as part of the thesis.

5.1 Evaluation

- Device management: The IDE has both device and application management.
- Platform independence: Programming is done using JavaScript and Node.js is used as the runtime environment. As Node.js is supported for almost all the platforms, this requirement is satisfied.
- Multi-device deployment: IDE has options for choosing multiple devices and deploy an application to all of them at once. It also supports monitoring and updating the deployed applications.
- Easy to use UI: This requirement needed changes to make the sequence of actions clearer as described in section 5.3
- Support low memory devices: This requirement needed changes to the framework and is described in detail in section 5.2.
- Data collection and analysis: The framework’s primary purpose is to program IoT devices remotely. Data collection can be implemented with one of the frameworks listed in section 3.2.
- Security and integrity: This is not handled by the framework. Having the resource registry generate a unique id before adding the device to the network and mandating this id to be used when the device registers itself, will prevent rogue devices getting added to the network.
5.2 Memory optimization

Most of the devices in an IoT network has low memory and power, compared to the laptop and desktop computers we are used to programming for. Though the devices like Raspberry Pi and Intel Edison has usually 1GB or even more memory, the possibility of having other devices with memory as low as 64MB in the network cannot be ignored. An analysis of the memory usage of the LiquidIoT application server was done to find out the memory requirements for running LiquidIoT applications in a device.

Every device added to the LiquidIoT framework’s network is installed with the Node.js LiquidIoT application server. This application exposes APIs for deploying, running, pausing and deleting the programs. It also relays any calls made to the applications. Each of the deployed applications was being run as a child process of the device server.

Analysis

Both the LiquidIoT application server and the user deployed LiquidIoT applications were found to be using 19-50 MB of memory. This is considerably high, if the framework needs to support devices with 64MB memory. Even with 128MB memory devices, it would be a limiting factor on how many IoT applications can be deployed. This led to the analysis of the memory usage and ways to optimize the usage. It was found that there isn’t much of a difference between the child process started using the ‘Spawn’ and ‘Fork’ options, in terms of memory usage.

The memory usage was analyzed using two methods. One using the Linux tools which provide memory and CPU usage information about a process. And the other using the memoryUsage method of ‘process’ Node.js module. The method returns memory usage as 3 parameters: Resident Set Size, total heap space allocated and total heap being used. Resident Set Size refers to the amount of memory a process currently occupies in the main memory.

The LiquidIoT application server running in the device uses about 13 Node.js modules, including the core http, express etc. Approximate memory usage of each of the module being used, was measured by loading the modules one by one and measuring the increase in memory usage. Requiring a module in Node.js loads the module and also its dependencies. Table 5.1 shows the memory usage of each of the modules being used. These values are not 100% accurate for each individual module, as loading a module also loads its dependencies. If a module having same dependencies
### 5.2. Memory optimization

<table>
<thead>
<tr>
<th>Module</th>
<th>Memory usage in MB</th>
<th>Usage in framework</th>
<th>Possible optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>19</td>
<td>Used for uploading the project files when deploying a user written program</td>
<td>Can be replaced as we upload a single compressed file that has all the project files. Express-fileupload is a possible alternative.</td>
</tr>
<tr>
<td>multer</td>
<td>6.7</td>
<td>For making calls to the Resource Registry and to the framework backend server</td>
<td>Write our own method to do what request does.</td>
</tr>
<tr>
<td>request</td>
<td>6.7</td>
<td>Used for routing</td>
<td>Koa might be easier and cleaner to use than express.</td>
</tr>
<tr>
<td>express</td>
<td></td>
<td>For reading and writing to config files. Used also when deploying applications</td>
<td>Fs' is a possible alternative. This module has rimraf below, as dependency.</td>
</tr>
<tr>
<td>http</td>
<td>0.7</td>
<td>Used for listening to http requests</td>
<td>This is not needed as framework is changed to run applications under one process.</td>
</tr>
<tr>
<td>tar</td>
<td>0.2</td>
<td>Used to extract the .tar files</td>
<td>Linux command 'cp -r' is an alternative. Can use child_process module to execute the linux command</td>
</tr>
<tr>
<td>tree-kill</td>
<td>0.2</td>
<td>Used to kill the child processes</td>
<td></td>
</tr>
<tr>
<td>ncp</td>
<td>0</td>
<td>Used for recursive copying</td>
<td>If the apps can call httpServer.listen(), it assigns a random free port, which can be obtained with httpServer.address().port, once it starts to listen.</td>
</tr>
<tr>
<td>path</td>
<td>0</td>
<td>Used for handling file and folder paths</td>
<td></td>
</tr>
<tr>
<td>portscanner</td>
<td>0</td>
<td>Used to find free ports</td>
<td></td>
</tr>
<tr>
<td>rimraf</td>
<td>0</td>
<td>Used to remove the folders – when deleting an application</td>
<td>This handles deleting folders which may have many levels of sub-folders and also symbolic links. One alternative is to execute 'rm -rf' using the child_process module.</td>
</tr>
<tr>
<td>slick</td>
<td>0</td>
<td>Used to parse css selector into JSON.</td>
<td>No need to replace as this is quite small</td>
</tr>
<tr>
<td>zlib</td>
<td>0</td>
<td>Used to decompress a buffer with Gunzip.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.1 Memory usage of Node.js modules**
is loaded earlier, the current module will seem to take less memory, compared to loading it as the first module. Hence, changing the order of loading the modules gives different results, depending on whether the module’s dependencies are already loaded. The table provides information about what the module is used for and what are the possible alternative modules/methods that could be used to reduce the memory usage. Some of the modules with '0 MB’ memory usage are quite small and some get loaded as part of the core or as dependency of other modules.

**Optimization**

Since a basic Node.js process consumes about 19MB of memory even with no additional modules loaded, it is considerably better to run the user written programs inside the LiquidIoT application server process instead of creating separate process for each of the deployed LiquidIoT applications.

The user applications still listen to their own ports. Although running user written code inside the application server is not a secure option, it helps reducing the memory consumption which is the primary concern. The user applications are generally smaller and do not use a lot of external modules. The external modules used are generally for communicating with I/O devices such as sensors. This already makes running the framework on low memory devices more feasible.

One more solution is to load and unload modules as needed. For example: some the modules are needed only for specific tasks, like the tar, zlib packages and can be loaded inside the method in which it is used and unloaded at the end of the method.

```
1   // Loads the module
2   require('zlib')
3   ..... 
4   // Code that uses the module
5   ..... 
6   // Unloads the module
7   delete require.cache[require.resolve('zlib')]
```

*Program 5.1 Loading and unloading module*

These modules are used only when deploying or updating an application. Since deploying an application would happen may be a few times a day to the maximum, the performance effect of loading and unloading shouldn’t be an issue.

Running Node.js with the command line option 'optimize_for_size' shows a reduction of 10MB in memory usage. The only issue is that, this is V8 engine’s \(^1\) command

---

\(^1\)V8 is Google’s open source JavaScript execution engine, used by Chromium, Node.js etc
line option and hence can be considered as internal option, which might get changed or removed in the future \(^2\) \(^3\).

Another alternative considered is duktape\(^4\) which has a much less memory footage compared to Nodejs. This is not explored as part of this thesis as it needs architecture change and also duktape doesn’t have any inbuilt http server.

With the changes implemented, it is feasible to run the LiquidIoT application server and applications in a device that has at least 128 MB of memory. It is possible to run it in a 64MB device, if we can further reduce the memory usage. This has not been tested with a real device.

### 5.3 User interface redesign

The major challenge for designing UI for the framework was separating programming and deployment flows. These flows still need to be easy to move between and at the same time clearly separated. The first step in redesigning the UI was to identify the use cases. This was done by exploring the features available in the existing UI and by checking the components that needs to be exposed through the user interface. Some components are only partly exposed to the UI.

The existing UI treated the API templates as a major functionality, which reduced the clarity. Treating it as a sub-functionality of editing a project makes it easier to understand the usage of it, although the REST APIs are needed for the important functionality of supporting the orchestration of the devices. Orchestration refers to the case in which one device could call another device directly. An example would be an alarm device that calls the temperature devices in a specific location, calculates the average of the returned values and raises an alarm if the temperature is above or below limits.

In the background, the framework uses a Resource Registry, which is another Node.js server. This keeps a record of the devices and their properties, sent by the device while registering to the framework. This information is accessed by the framework when deploying applications, controlling their status and monitoring them. As this is only in the background, there is no need to expose this in the UI, at least until adding the device to the framework is automated. The decision to change the UI to be more user friendly was influenced because of the difficulty a new user had with

\(^2\)https://blog.heroku.com/node-habits-2016
\(^3\)https://github.com/nodejs/v8/blob/master/test/optimize_for_size.gyp
\(^4\)http://duktape.org/index.html
Figure 5.1 Use cases

using the framework and also the difficulty in using the UI to present the framework to potential customers and in conferences.

Use cases

The major use cases as shown in Figure 5.1 are the following.

- When the framework is installed and used for the first time:
  - Check what the framework does: This needs a page that describes the purpose and usage of the framework.
5.3. User interface redesign

- Add a new device: As this isn’t automated yet and it in itself is a major use case which also needs to authenticate the devices, for now this will only be an instruction page that describes the steps needed.

- Configure device’s capabilities: This is currently done by configuring the framework component installed in the device to add it to the network. The configurations are all in JSON format to make it easier to edit.

- Write an application: This step needs UI that allows user to create a project and add his/her own implementation to the project. It also needs to have the options to upload code files and resource files which could be audio or video files (the framework does not restrict the type of files that can be added as resources to the project). It should also provide interface to add an API template as a method to the project. There has to be an option to edit the project after it is created. In the current implementation, editing a project is equivalent to creating a new version.

- Deploy the application: This step has the following sub-steps:
  * Choosing the device(s): Currently the implementation uses CSS selectors and supports choosing devices based on the name, location of the devices (which is configured when adding the device to the network), currently deployed applications, device capabilities etc.
  * Deploying the applications: Deployment is done by first contacting the RR (Resource Registry) which provides the framework with the URLs of the chosen devices. This URL corresponds to the Node.js component of the framework which exposes APIs to install, uninstall and monitor the status of the applications in the device.
  * Updating the applications to a newer version/Revert to an old version: The framework currently supports updating the application to another version as a blue-green deployment scenario. Using this concept lets only two versions of the application to reside in the device at any point of time. But it gives the flexibility of switching between these versions easily.
  * Monitoring application status: The framework’s component in the device both controls and maintains the status of the applications. The status is currently restricted to ‘Running, Paused, Error’. Any internal errors that might prevent the application from functioning properly doesn’t bubble up to the framework component. The application could write to the console, which is accessible for the developers as the application log.
* Check deployment result: The framework reports the result of the deployment process, which needs to be displayed to the user. This gets tricky when the user selects a large number of deployments and only a few succeed. Currently this is reported in the form of statistics (ex: 5 out of 7 deployments succeeded). Once the deployment succeeds, the user needs to be presented with a UI, through which he/she can control the applications installed (Running it, pausing it, checking the logs etc).

- Orchestrate: This can be added as an instruction page in the future. Currently this is considered as a separate project in which the orchestration can be configured (details include who calls who, how to reach each other and what to do with the results of the API calls)
  - Add API: Although the orchestration framework is not part of the UI, the API templates still are, as they can be used manually by the developer by writing code in one device to call another.
  - Implement API: Implementation is done through the code editor and doesn’t need a specific UI.

- Network already has devices, applications, APIs
  - Check device status: This needs a consolidated view of the network with all of the devices and the applications installed in the devices including the status of the applications. When the number of devices increases over a few tens, say 50, this UI needs to consolidate the devices into groups (could be based on location of the devices, capabilities). The criteria for grouping could be a made into a configurable one.
  - Check application status: This needs UI to allow the user to both view and control the status of the applications. This may also need two views, one in which the grouping is done based on device information and one in which the related applications are grouped together.

**Other considerations**

Apart from the use cases, there are also other considerations in designing the UI.

- Brand the UI as something developed in TUT (currently only the color scheme is designed to be similar to the TUT’s website).
5.3. User interface redesign

- If the orchestration needs to be included in the UI at some point in the future, we need to keep a separate main menu option for accessing the API templates and creating new ones.

- The framework must implement an authentication logic for accessing the IDE. The menu options for signing up and logging in are added, with no functional implementation.

- There will also be need for administrator user accounts which will have the privileges of deleting projects and removing devices from the Resource Registry. An administrator user should be provided with these additional functionalities in the UI.

New UI

The design was done using different methods to come up with a few ideas. One of them was to create small sketches (drawing in a smaller scale forces us to focus on the major idea and ignore the minor details), the other was to draw at least 4 designs, each with a different flow. Although drawing on paper is better for the UI designer to come up with the ideas, it proved to be much easier to present the ideas in the form of a mock up. After the ideas presented using mock up were discussed and improved, the implementation was done using the same tools and technologies used previously.

- Grunt as the automated development environment which is configured to monitor for the changes and refresh the page automatically which made it easier to test the changes.

- UI is written using the template engine Pug (previously named Jade), which provides a clean white-space sensitive syntax.

- visjs for the network visualization (D3js was considered as an alternative, but wasn’t chosen as it follows a different programming paradigm which might be difficult and time consuming to learn).

- Sass is used for writing the styles for the HTML elements.

- Yeoman is used as the scaffolding tool for generating new pages and to add them to the Grunt scripts.

- Angular is used for writing the scripts needed for the pages.
Figure 5.2 New home page

- Node.js is the execution environment, npm is the server side package manager and browserify is the package manager for front end.

- MatchHeight is added newly to help make the blocks of the device/project details to be of the same height in a row. This makes the UI looks more orderly than the default size aligned by bootstrap grid.

The screen-shot in Figure 5.2 shows the home page from the new UI. A new feature is added to the UI, which describes the flow of different steps in the IDE. The current step is highlighted and changes according to the page the user navigates to. Rest of the pages are available in Appendix A.
6. LIQUID SOFTWARE: PROGRAMMING FRAMEWORKS

Liquid software requires the state of the application to move seamlessly between different devices. It could be a native application or an application served through the Web. In an ideal liquid environment, all the applications should be able to move between devices with minimal or no explicit action by the user. This liquid experience can be achieved at the operation system level, where the whole of a device state is moved from one device to another, or at the application level where individual application state moves between devices. This thesis focuses only on the liquid applications delivered through Web.

The following are common functionalities of a liquid application to support sequential, simultaneous or collaborative scenarios.

- Discovering the devices between which the application has to flow. The devices might be physically close by or far apart.
- Communication between the devices.
- Transferring/sharing the state.
- Application should be platform agnostic.

Implementing these common features as a framework allows us to create a standardized and reusable implementation. Application developers will be able to focus on the domain specific or business purpose of the applications and add liquid feature with minimal effort by using the framework.

6.1 Requirements

The requirements for a liquid framework are as follows:
6.1. Requirements

- Device discovery: Framework should be able to provide a list of devices that the application can be moved to. In a server based system, the server can keep track of the devices connected to it. In case of a peer to peer system, the framework can use technologies like Bluetooth or NFC (Near-field communication) to find the devices near by.

- Communication mechanism: Communication can be achieved by using IP(Internet Protocol) networks or using other technologies such as Bluetooth, Bluetooth low energy, NFC etc. WebRTC and WebSocket are two of the options when using IP networks for communication.

In a Web application, communication is always initiated by the client with a HTTP request and the server sends data in the HTTP response. Two-way communication is needed for simultaneous and collaborative scenarios, as the application instances running in different devices need to share data with each other. Long-polling connection is a mechanism used to achieve two way communication. It is a pull request and the client expects that the server will not reply immediately. The server keeps the request open and sends a response, when it has the actual data. WebRTC and WebSocket can also be used to implement the two-way communication needed.

WebSocket supports two way communication and allows the server to send messages to the client. socket.io is a Node.js library that provides easy to use interfaces for connecting and sending data as JSON\(^1\).

WebRTC uses a peer to peer connection. It needs a server for the devices to find each other and establish the connection. Once the connection is established, data is exchanged directly between the peers. Discovery mechanism is not defined as part of the WebRTC specification. If the devices are behind NAT(Network Address Translation), ICE (Interactive Connectivity Establishment) protocol implementation can be used to find the best possible connection\(^2\). ICE makes use of the STUN (Session Traversal Utilities for NAT) protocol and its extension TURN (Traversal Using Relay NAT). STUN server provides an external IP address and TURN server relays traffic if peer to peer connection fails.

- Connection establishment: Framework should provide interfaces for choosing a peer device to synchronize to.

- Real time state transfer: Simultaneous and collaborative screening requires support for notifying the changes done in any device to all interested devices

\(^1\)https://socket.io/get-started/chat/
\(^2\)https://tools.ietf.org/html/rfc5245
in the system. This can be supported using publish-subscribe model.

- Support heterogeneous platforms: Writing the application as a native application provides the advantage of creating a rich UI and being able to use native APIs. This is a good option if the liquid application targets one specific platform. Apple Handoff is an example of a liquid software feature for proprietary platform. When the devices involved are running different platforms, it is necessary to use a common programming language and runtime. If platform specific APIs are used for creating liquid applications, the application has to be developed separately for each of the platforms. A generic application can be developed by providing a Web interface and using JavaScript.

- Responsive/Adaptive UI: User interface should adapt itself to the device it is currently being viewed, as the devices might have different screen sizes and resolutions. There are open source responsive web frameworks that can be used to implement an adaptive UI.

In case the application is served through web and it serves only static content, user can switch to a different device and continue reading the content simply by copying the hyperlink. A more convenient way than typing the URL, is to convert the URL to QR code and scan it with the destination device. More information about the content, like the length of the video that has been played thus far could be added to the hyperlink to allow the user to continue from where he/she left it in the previous device. If the application uses user authentication, moving the user session from one device to another requires the session identifier(which is providing the authorization) and the corresponding session data to be moved together.

### 6.2 Available tools and frameworks

There are a few open source frameworks that can be used for developing liquid application. Most of them are academic research projects. Apple Handoff is an example of liquid API that allows applications to move between devices. Google Sync provides liquid functionality for its own products by saving the data in cloud\(^3\). The following sections list the major functionalities and the relevant frameworks.

#### Device discovery

Discovering devices needs using native applications such as Bluetooth. It is not accessible through JavaScript running in browser. Experimental technologies like

---

\(^3\)https://www.google.com/sync/index.html
Web Bluetooth\(^4\) are not supported by all browsers yet.

**Connichiwa** is an open source framework that uses a native application to use BLE(Bluetooth Low Energy) to detect nearby devices. Once connection is established, the framework calculates the distance between the devices using the strength of Bluetooth signal and provides support to create a synchronized visualization of media, distributed across the connected devices. The framework is under development and supports iOS and Mac OS platforms natively. It also allows other clients to connect to the framework using Web UI [16].

**Liquid.js** framework is an open source framework\(^5\) that uses WebSocket for communication and keeps track of the clients connected to the server. It provides interfaces for getting the list of other connected clients. A client can choose a peer to synchronize the state [20].

**XD-MVC[7]** is a framework that brings Cross-Device capabilities for Model View Controller frameworks\(^6\). It provides WebSocket and WebRTC as communication options and keeps track of the clients that connect to the server. A client can choose another client to establish a synchronizing connection.

### Application state transfer

The state of the application can be transferred as a whole or partly, depending on the application requirements and framework support.

If the application is a static web page, it can be transferred easily by copying the URL to destination device. In cases where the application is dynamic, changes made by the user need to be copied to the destination device. Depending on the amount of data, this can be added to the URL as parameter or saved to the server and sent to the destination device. In case of simultaneous and collaborative screening, the changes made by the user(s) have to be sent to the participating devices in real time.

Liquid.js framework provides this feature by creating a virtual DOM(Document Object Model). For synchronizing the applications, the framework computes the difference between the initial and the current DOM. This reduces the amount of information needs to be shared. This is well suited for single page applications. XD-MVC provides interfaces for establishing a connection and sending data to be

\(^4\)https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API  
\(^5\)https://github.com/Zharktas/liquid.js  
\(^6\)https://github.com/mhusm/XD-MVC
synchronized as JSON. Liquid.js and XD-MVC are discussed in more detail in section 7.1.

**User identity transfer**

Most of the applications available through Internet, has registered or subscribed users. These applications provide personalized content and in case of paid services, privileged content and services. When a user moves from one device to another, the user identity also needs to be transferred in order to continue providing personalized content. This needs special considerations and is described in more detail in section 7.2.
7. LIQUID APPLICATION WITH USER IDENTITY

Liquid application that uses user authentication needs to transfer application state and user identity, when the user switches between devices to access the application. Option to switch from one device to another can be provided in the UI and when the user initiates the transfer, session data can be moved to the destination device.

Liquid.js and XD-MVC frameworks are considered for transferring application state. Approaches to transferring user identity are discussed in section 7.2.

7.1 Development using Liquid.js and XD-MVC

We compared the XD-MVC[7] and Liquid.js[20] frameworks, from the point of view of the Application Developer. A prototype which provides a to-do list through Web was created with both frameworks to evaluate the ease of use and functionalities. More details of the prototype are provided in Appendix B.

In both cases, there is no need for any software to be installed on the client side, as they are both web based frameworks. Both use a Web server to which the clients connect to. Every client connecting to the server is tracked and is discoverable by other clients. The security aspect of the liquid capability, ownership and authorization, location discovery are not handled by neither of the frameworks at the moment. There is no authentication mechanism in place. All requests to connect are accepted by default.

Liquid.js can synchronize instances of the same application running in different devices (the UI and code needs to be the same across devices). XD-MVC can synchronize clients having different UI and code, as it leaves the handling UI synchronization to the application developer. With XD-MVC, if we want to use the WebRTC mode, then the application have to be a single-page one. We want the clients to work even after losing connection to the server. They will need server connection to get new pages if the application contains multiple pages.
7.1. Development using Liquid.js and XD-MVC

LiquidJS

This framework uses WebSocket for communication and Virtual-DOM to transfer data. Virtual-DOM is a object based representation of the DOM, in which each of the DOM node is represented by an equivalent object. This makes it easy to compare and compute changes between two DOMs. Only the changes (computed against the initial DOM state of the application) are serialized and transferred, reducing the amount of information to transfer.

This framework is well suited for single page web applications. The application developer only needs to initialize the framework to make the application to work fluidly between different devices. The framework adds a partial UI to the page using it. The framework UI has options for viewing the list of all other clients and choosing one or all of other client(s) to synchronize with. Framework removes this part of the UI before computing the differences between DOMs. The framework then compares the rest of the DOM to find the changes from initial state. This is done as the list of peers is different for every device and this information should not be synchronized. This procedure can be improved by letting the application developer choose only part of the DOM to be synchronized.

Server implementation

The framework doesn’t provide any server side component. Since the client side component of the framework uses socket.io.client\(^1\) package, the application will have to use socket.io on the server side. This is necessary as the socket.io’s client and server side components communicate through specific named events.

Application developer needs to implement the following on the server side (as shown in the sample code 7.1).

- Attach a socket.io server to http. Socket.io package uses xhr-polling which is also referred to as long polling to reliably initiate a connection and switches to WebSocket for communication if supported. Socket.io assigns unique id to the connecting clients and also maintains a list of clients connected to the server and provides APIs for subscribing and publishing events.
- Subscribe to the socket.io events ’sync’, ’getList’ and ’syncToClient’.
- On receiving ’sync’, broadcast the same event to all of the connected clients.

\(^1\)https://socket.io/
• On 'getList', return a list of all of the connected clients, excluding the request-
ing client.

• On 'syncToClient', publish 'sync' event to the client id specified in the request.

```
var io = require('socket.io')(http);

// Called when a new client connects
io.on('connection', function(socket){

  // Handler for the socket.io sync event
  socket.on('sync', function(SyncObj){
    // Broadcast the information to all clients
    socket.broadcast.emit("sync", SyncObj);
  });

  // Handler for the socket.io getList event
  socket.on('getList', function(){
    // Get the list of all connected clients
    // and remove the calling client from the list
    var clientList = Object.keys(io.engine.clients);
    var index = clientList.indexOf(socket.id);
    clientList.splice(index, 1);
    io.to(socket.id).emit("list", clientList);
  });

  // Handler for the socket.io sync to specific client event
  socket.on('syncToClient', function(clientObj){
    // Synchronize data to the specific client
    socket.to(clientObj.client).emit("sync",
    clientObj.sync);
  });
```

Program 7.1 Liquid.js server implementation

The framework supports sequential screening as the user can send the latest changes
from one device to another. It supports simultaneous screening scenario in which
same content is displayed in all devices. It does not support simultaneous screening
scenarios where the content needs to be distributed across the devices (viewing a
different part of the content in each display).
7.1. Development using Liquid.js and XD-MVC

Client implementation

The framework provides a UI component which lists the clients and provides the options to sync to a specific or all of the clients. It maintains a virtual version of the initial DOM. When the user requests a sync, it compares the current DOM with the initial one, computes the changes and sends the changes to the server, which passes it on to other client(s). On receiving the changes, a client restores its DOM to the initial state and then applies the changes.

Application developer implements the following in the client side.

- Create the application specific UI and initializes the Liquid framework.
- If there are event handlers added dynamically (example: A newly added to-do item can add a checkbox to the UI, and attach a click event that strikes through the to-do item or removes it), then the application have to register these handlers to the liquid framework, to get them also to be part of the virtual DOM to be sent.

Pros and Cons

The following are the positives of the framework.

- Easy for the Application developer to use.
- Application developer doesn’t have to take care of the data synchronization.
- Provides a generic model for creating a fluid application with Web UI.
- It supports synchronizing both data and dynamically added event handlers (code mobility).

Following are the limits and issues of the framework.

- A client joining an existing group, doesn’t get synchronized, until some other client decides to push the updated DOM to it.
- It doesn’t handle the scenario, if the application has multiple pages and the users are on different pages.
- There is no option to connect to and be in sync with specific clients.
• Doesn’t deal with responsive UIs (UIs which adapt to the device screen based on whether it is mobile or desktop or tablet)

• It is up to the client application to maintain the list of clients, it wants to send the updates to (socket.io package does this and the application developer can use it’s API). Only the sender has the control on who gets the updates.

**XD-MVC**

This framework monitors an object or array for changes and synchronizes data chosen by application developer. It uses WebRTC if supported by the browser, otherwise uses socket.io. It supports defining and configuring roles. The server component maintains the list of roles, defined and assumed by each of the clients and exposes an API to query roles of a peer. Client side component maintains the roles of that client. The framework itself doesn’t define any roles. Data to be synchronized is transferred using JSON format. The application can pass a callback to the API xdmvc.synchronize. The callback is called by the framework, when some other device changes the data. The framework also provides a Polymer based API.

It supports both sequential and simultaneous screening. Any change in the data the application is interested in, is immediately distributed to the connected devices.

**Server components**

The provided server component starts socket.io(Websocket), PeerServer(WebRTC) \(^2\) and Ajax servers listening on different ports. When a client connects to it, this component assigns a unique ID to the client and notifies other clients. Client can choose a set of peers to be in sync with. The component exposes APIs to request a specific ID to be assigned, to get the list of peers and sync with a peer. It also provides information about whether or not a client supports WebRTC.

Application developer needs to write the code to serve application specific pages and start the xdmvcserver.

**Client components**

Framework doesn’t provide any UI components. Framework’s client component checks if the browser type and version supports WebRTC, connects to the server

\(^2\)http://www.peer-server.com
and gets an Id. It also maintains list of roles defined and the current role. It listens
to and passes on the following events from server to the application code: When
a peer connects or disconnects; Some client sends data to sync (passed on only if
the current role is configured to sync). When a connection is established, the client
sends its roles and device information to the peer.

Application developer needs to implement the following.

- Implement the application’s own UI. And UI for listing the peers and options
to choose a peer to connect to.
- Initialize xdmvc, define roles, register objects to be monitored (and provide a
callback to receive changes to that object by other clients).
- Initiate sync when user changes something.

Pros and Cons

The following are the positives of the framework.

- Easy for the Application developer to use.
- Application developer chooses the data that needs to be synchronized.
- The data is not tied to the visualization (unlike Liquid.js which uses DOM).
- Manages the connections between clients. It creates a group for the connected
  clients. Everyone in the group receive updates.
- Connection details are persisted and the framework attempts to reestablish
  the connections when relaunched (optional behavior which can be disabled).
- Provides an API integrated with polymer, if the client wants to use polymer
  for UI.
- A new instance of the client application joining to an existing group of peers,
can get the latest state and update its UI. (The peer, the new client is con-
necting to, detects that it is joining newly to the group and sends the latest
state information to it).

Following are the limits and issues:
7.2 Approaches for User Identity transfer

- It allows synchronizing only the data (in the form of Arrays or Objects). The application developer has to update the UI based on the data received.
- Even though it supports WebRTC, the server needs to be running, for the new peers to discover each other.
- There is no support for code mobility.

Summary

Table 7.1 Comparison of Liquid.js and XD-MVC

<table>
<thead>
<tr>
<th>Liquid.js</th>
<th>XD-MVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication support: Websocket.</td>
<td>Supports Websocket and WebRTC (when available). WebRTC allows communication between clients, even when the server connection is lost.</td>
</tr>
<tr>
<td>Synchronization is done using Virtual DOM (specific to HTML based UI).</td>
<td>Synchronization is done by sending data in JSON format. Application developer is responsible for updating the DOM.</td>
</tr>
<tr>
<td>No server side component is provided. Application developer must add Websocket support in the server.</td>
<td>Provides server side component.</td>
</tr>
<tr>
<td>No support for roles.</td>
<td>Supports defining and configuring roles. Synchronization can be customized based on the role of the client.</td>
</tr>
<tr>
<td>Does not support multi page applications.</td>
<td>Supports multi page applications.</td>
</tr>
</tbody>
</table>

From the analysis of the frameworks and the experience of creating a prototype with them, Liquid.js is better if the developer wants to have a single page application to have liquid capability. It is easier to adapt with few lines of code. While both of the frameworks have their advantages and limitations, XD-MVC comes off as a better choice, if the developer wants to have control over the data to be synchronized between the devices. It also maintains and synchronizes the devices automatically once the devices are paired by the user/application. Table 7.1 provides a summary of the similarities and differences between the frameworks.

7.2 Approaches for User Identity transfer

Most of the content providers provide an option for the users to register, so that they can get personalized content. This personalized content is in some cases, a paid
7.2. Approaches for User Identity transfer

one. When a user logs in using one device and later wants to move to a different device, the user identity has to be moved along with the current URL, to be able to access personalized content. In case of Web applications like LiquidIoT, when user management is present, then the user identity is needed to provide authorization.

When a user is authenticated in a Web application, a session is created to identify further requests from the same user. This is needed as HTTP is stateless and the user cannot be expected to provide authentication details for every request. Transferring the user identity, hence requires the session to be cloned or copied to the new device.

The copying of the session and data from one device to another

1. Can be done manually by typing the URL in the destination device.

2. By providing a QR code that the user can scan with mobile devices. This is much more quicker and convenient [2].

3. By using NFC to identify the nearby devices and allow the user to choose the device to transfer to. This requires all devices involved to have NFC capability.

4. By using Bluetooth to identify the devices close by and letting the user choose the destination device.

The focus of the thesis is mainly on the techniques to transfer the identity. Techniques to discover devices are not explored in detail. The prototype used for evaluation provided a QR code to move the identity. A similar solution to transfer anonymous sessions is described here [2].

The session can be classified into 2 types, depending on how the session related data is stored.

Server session: Data is stored on the server side and the client is provided only with a session identifier. Identifier is recommended to be a randomly generated string, to prevent session spoofing attacks. If the identifier is generated in a predictable way, hackers can launch a brute-force attack and spoof the sessions of legit users. The server stores the identifier-user association in the database. When it gets a request with session identifier, it looks up the list of associations to identify the user of the request.

Client session: Session data is stored in the client side. The data is encrypted or signed to prevent attackers from copying the session data and spoofing the
users. Since the data is encrypted, the client cannot decrypt it and it simply sends this with all the further requests after authentication [12, 14]. Server decrypts it and can verify that the session belongs to a valid logged in user. If techniques such as JWT (JSON Web Token) [9] are used, session data is signed and cannot be modified by the client or by the attackers. The main issue with client side session is that the server cannot invalidate the active sessions, say, when a user changes password.

In our ongoing research [18], we found that being able to invalidate sessions is an important functionality and hence the focus is only on transferring server side sessions. In case of server side sessions, copying the session identifier when the user moves to a different device is sufficient to move the user identity. This however does not copy the data stored in cookies in the client side (an example of this is last seen item stored by client side script, which is used to return the user to that item when the browser is closed and reopened). This data also needs to be copied when the user moves to a new device. It is less secure to expose the session identifier through QR code and to clone the same session identifier to destination device in case it is a public or borrowed device. Using a short lived code provides a more secure option. Transfer is done using this code and a new session identifier is created for the new device. If there is data stored in cookies that needs to be transferred, client can send them in the request for transfer code. Server stores this information in a database. When the transfer is requested from a new device with this code, server attaches the stored cookies to the response.

This helps the user to transfer the identity, without having to enter the credentials again in the new device. Even in cases where the application provides a 'remember me' option, transferring the identity this way can help the user continue from where he/she left off, instead of having to navigate the page(s) manually. It is also useful for cases, where the user uses a public device or borrows a device from a friend or a family member. The application must demand password for sensitive operations such as changing password, particularly if the session is a transferred one and not the one authenticated with user credentials.
8. DEVELOPING NETWORKED WEB APPLICATIONS

This chapter unifies the concepts of IoT and liquid software to envision a smart space where the physical boundaries of the devices disappear (more like hidden from the end user). IoT systems provide a holistic view and control of multiple physical things and liquid software flows between device boundaries, making multiple computing devices look like a single device with multiple screens. Combining these concepts will take us a little closer the grander vision of ubiquitous computing[21].

LiquidIoT enables us to develop systems, that consist of multiple devices talking to the central server and also with each other. Liquid software enables us to roam between our personal devices seamlessly, independent of their platforms. They are both aiming at making multiple devices work seamlessly as a single system.

In future, LiquidIoT can be extended to enable programming of liquid applications for personal computing devices. LiquidIoT specializes in programming IoT devices, but it does not restrict programming more powerful devices. A Liquid Software (LiquidSW) host can be created and installed to the personal devices. IDE will enable creating liquid applications and deploy them to devices through LiquidSW host. End user can choose the devices he/she wants an application to move seamlessly. LiquidSW host will be responsible for the communication between the devices. Communication can be peer-peer or through the framework server. When an application state changes in one device, it will notify the LiquidSW host in that device. LiquidSW host will then notify all other devices, interested in the state of that application. Implementation of the LiquidSW host can be based on XD-MVC and Liquid.js frameworks.

There is also a need for developing liquid applications for IoT devices. This will be helpful to move the LiquidIoT applications to a new device, when a device needs to be replaced. This feature will also be useful for a network that is distributed across a large area, with network routing devices and firewalls in between. Instead of the framework deploying applications individually to each device, it could send the program to one endpoint behind a firewall and then the application clones itself
to IoT devices behind the firewall.

User interface for the end user of an IoT system can be provided as a clean high level map of information, by hiding the details about sensors and small computers. For example, user interface of a simple smart house will show the temperature and light levels in different rooms and let the user control them. This user interface will be able to flow between personal devices owned by the user and be able adapt to different screen sizes.

IDE of the LiquidIoT can be implemented as a liquid application as well, enabling the developers to switch between different devices during development. If there are multiple users viewing the same page, they should all get the same status of the system. Any changes made to the state of the system by user action or a background incident (a device might lose network connection, some application instance might crash or a new device could get added to the system), should reflect in the user interface immediately. Current implementation of LiquidIoT has the infrastructure ready for the implementation of user management, with option to provide administrative privileges for users. When this is implemented and there are multiple users using the system at the same time, the IDE should be able to transfer the user identity along with application state when a user roams between devices.
9. CONCLUSION AND FUTURE WORK

The thesis work has helped us achieve the results we aimed to. Through evaluation of IoT frameworks and analyzing the approaches for transferring liquid application state and user identity, we now have a list of development tools and frameworks to use for developing networked web applications.

With the changes implemented as part of the thesis to the LiquidIoT framework, it now supports devices as low as 128 MB. It is theoretically possible to run it devices with 64 MB of memory too, though this needs to be tested in a real device. The new user interface provides a clear flow of the primary use case. There are improvements needed in the page providing the option to deploy and update applications. It might be easier to create two separate pages, one which allows users to deploy applications and one which allows updating applications. Liquid experience of the IDE of LiquidIoT can be provided by making use of the XD-MVC framework.

Transferring user identity requires stricter security considerations, compared to transferring merely the application state between devices. Requirements and techniques have been discussed in more detail here [18]. How user identity is transferred depends on whether the session data is maintained in the client side or server side. Using a temporary code to transfer the session is just one solution suitable for web applications. More research is needed for native applications and for making use of technologies such as Bluetooth, NFC for discovering devices and transferring identity between the devices.

Though there are both open source and commercial frameworks available for IoT, developing a holistic framework which supports device discovery, programming, data collection and analysis will allow the system/application developers to focus on the domain specific implementation.

Liquid software frameworks that are platform agnostic needs more work to provide a complete liquid experience. A lot more work needs to be done before the personal computing devices can work together as a single entity.
BIBLIOGRAPHY


A. APPENDIX: SCREEN CAPTURES OF LIQUIDIOT NEW UI

The following are the screens of the new UI developed for LiquidIoT. All of the pages use the same color scheme as the TUT official website \(^1\). In each page, the corresponding step is highlighted in the flow of the main use case.

Projects page shown in Figure A.1 lists the existing projects, provides option to navigate to the page to edit the code of the existing projects and option to create new project. The page to edit the code (Figure A.2) of the project lists the files present in the project and when a file is selected, opens it in the code editor. The editor has syntax checking capabilities for JavaScript. The page provides options to upload new code files and resource files to the project. It also has interface to choose API templates and add them to the project. Navigating to the deploy view from this page helps to choose devices and deploy the current project. The option to add API in the project page, opens a modal dialog (Figure A.3) which lists the current templates. User can choose one or more templates to add to the project. Deploy page shown in Figure A.4 is similar to the old UI and needs more changes to separate the deploy and update use cases. User can choose the devices and applications, deploy new applications and update existing ones in this page. This page shown in Figure A.5 lists the APIs present in the system. User can also create new APIs and define them in the Swagger editor, similar to the old UI.

---

\(^1\)http://www.tut.fi/fi/etusivu
Figure A.1 New projects page

Figure A.2 New ‘edit project’ page
Appendix A. APPENDIX: Screen captures of LiquidIoT new UI

Figure A.3 New add API page

Figure A.4 New deploy page
Figure A.5 New API page
B. APPENDIX: PROTOTYPES USED FOR LIQUID FRAMEWORK COMPARISON

Simple to-do list prototypes were developed using both Liquid.js and XD-MVC frameworks. Information about the prototypes are presented below.

The to-do application shown in Figure B.1 shows the prototype developed with Liquid.js. Bottom of the page has the UI provided by the framework. The page on the left side of the picture would look the same as right side, when the user clicks on the ‘sync to client’ or ‘sync to all’ button.

The to-do application shown in Figure B.2 shows the prototype developed with XD-MVC. Page on the left side shows a client connected to the client in the page on the right side. Synchronization happens in the background with no explicit user action. Both pages show a third client, to which they are not currently connected to.

![Figure B.1 To-do application with Liquid.js](image-url)
Figure B.2 To-do application with XD-MVC