REDBACK BIM

Developing 'De-localised' Open-source Architecture-centric Tools

E. Y. LEUNG,
University of New South Wales, Sydney, Australia
emily_y_leung@outlook.com

Abstract. Emerging technologies that use data have contributed to the success of communication all over the world. Social media and gaming industries have already taken advantage of the web to provide synchronous communication and updated information. In contrast, existing methods of communication within the AEC industry require additional platforms, such as emails and file sharing platforms in conjunction with the 3D model, to inform changes made by stakeholders. This has resulted with duplication of files and limited accessibility to the latest version, which has contributed to the myriad of inefficiencies derived from existing practice. As communication is critical to the success of a project and should be enhanced, Redback BIM promises to establish a workflow for a dynamic platform, that can achieve similar results to that of a 3D modelling program hosted on the web. Using existing open-source web development software, multiple users will be able to collaboratively organise and synchronise changes made to the 3D architectural model in real-time. Such applications can bridge the communication gap between multiple stakeholders within the life of a project.

Keywords. De-localised Workspaces: Web-based Software Programs; Data; Open-source; Collaboration.

1. Research Motivations

Architectural practice should and can take greater advantage of emerging technologies that use data, whether it be through data analytics, artificial intelligence, robotics, digital supply chains and advanced construction (Raisbeck, 2016). Critical to current architectural practice is the use of 3D modelling programs that allow for accurate design and documentation of spaces and buildings. Even though the development of these programs has meant streamlined changes, through the introduction of visual scripting platforms, such as Grasshopper and Dynamo, the communication of information between multiple stakeholders remains a key challenge.

Since the release of Web 2.0, and more recently the advent of cloud computing, there has been a significant growth in synchronous collaborative web-based software platforms, such as Google's G Suite (Google Docs, Google Slides, etc), Microsoft's Office 365, Trello, Realtime Board and Slack. These programs, that allow multiple users to access and edit live documents without requiring software licensing or hosting on localised private servers or devices, are referred to here as 'de-localised' platforms. Many of these existing platforms are already implemented within the architecture, engineering and construction (AEC) industry for text editing and visualisation purposes that typically require the input of multiple stakeholders. However, emails, phone calls and file sharing platforms are seen to bridge the communication gap, clarifying information about a 3D building. As effective communication often relies on the right medium for its expression (Norouzi et al, 2015), this research argues that the existing communications practices can be enhanced by creating a de-localised 3D modelling interface.

Within the AEC industry, some of the most widely implemented software are used to create 3D models and drawings. Rhinoceros, Revit, AutoCAD and Sketchup are popularly adopted to achieve a level of visual representation and communication between designer, clients, builders, and engineers. A key constraint of these examples is that this software are localised, in short, they sit on remote desktops, and files are stored on private servers. As the AEC industry typically requires the input of multiple stakeholders, who can be located across vast geographic distances and time zones, communication between all stakeholders significantly influences the success of a project. As a result, other forms of communication are required in conjunction with the 3D building model to clarify changes, and access to the files are not only restricted, but are also continuously outdated within a matter of hours.

Additionally, while there are a myriad of web-based software platforms for the AEC industry, a review of these applications indicates that there are inefficiencies in practice, leading to duplication of files and outdated

information. Existing web-based software that seeks to enhance collaboration and address issues of version control, file storage, downloading, model viewing and commenting include cloud-based applications such as AutoDesk 360 BIM, Dropbox and Aconex. Yet, these are costly alternatives and at times require additional licensing. These software are limited in functionality and do not enable real-time updates, whereas the combination of synchronous collaborative spaces and 3D modelling programs that this research explores, could be made possible when repositioned on a web-based platform.

Current multiplayer online games provide key examples of the potential of synchronous collaboration within a common virtual environment. Minecraft, for example is a popular game that has a powerful multiplayer mode. Specifically, users can work or converse with anyone in the world at the same time within the same "seed" (equivalent to an ID) to a world. The emphasis on synchronous collaboration is shown through all users having the ability to build on top of, or to modify each other's work in real-time. This idea of making changes and everyone seeing these changes instantaneously is something only the web can achieve. Therefore, synchronous collaborative environments show promise in delivering improved design lead times and design quality from unified workspaces (French et al, 2016).

Contemporary architectural practice is increasingly characterised as a collaborative environment that challenges the traditional idea of curatorship from a single keeper and organiser of information, to one where authorship is the result of teamwork oriented practice (Agulrre Leon and Molina 2009). As a result, the tools that individuals use comparative to current methods of practice has shown the web as the common denominator within a collaborative scenario. The management of project information using de-localised platforms combined with 3D modelling capabilities, stands to offer significant benefits in terms of workflow and project collaboration.

2. Research Aims and Objectives

The overarching aim of this project is to develop a workflow for a dynamic platform that can achieve similar results to that of a 3D modelling program hosted on the web. This aims to allow multiple users to collaboratively work on a consolidated 3D architectural model, bridging the communication gap between multiple stakeholders within the life of a project.

This research examines many case studies that illustrate the realization of synchronous, collaborative and dynamic platforms that aim to streamline existing methods of communication. Additionally, the development of progressive web applications (PWA) has broadened the accessibility of de-

localised platforms across a range of digital devices including desktop, mobile and tablet (Birch, 2017). When brought together, de-localised platforms that are built to include PWA capabilities can enable more seamless, collaborative work environments.

More specifically, this project aims to create a dynamic 3D modelling application workflow that interlinks data and geometry. The key focus here will be on the dynamic update of geometry data to a synchronous collaborative web-based platform which will be made accessible on any mobile device. This new digital interface has the potential to create an impact on existing communication in design and architecture, through optimised processes of organising and synchronising 3D architectural model data in real-time.

3. Research Question

Currently, synchronous collaborative work environments exist in 2-D text and image based documents, which can be limiting for architects and stakeholders who work with consolidated 3D models, as it is easier for documentation, fabrication and the comprehension of spatial and organisational relationships. Current multiplayer online games such as Minecraft suggest that client-server gaming and related cloud-based architecture can enable teams to work closely and collaboratively on complex, dynamically changing landscapes (Kim 2002). This research aims to extend and build on these advantages by developing an open-source dynamic PWA, to create a synchronous relationship between data and geometry to consolidate and streamline current methods of communication. Potential increase in collaboration, speed of communication and accessibility to information will influence and reshape contemporary architectural discourse.

In what ways can participating in building open-source and dynamic progressive web applications (PWA) be developed to enable synchronous relationships between data and 3D geometry to optimise communication practices in the AEC industry?

4. Methodology

4.1. ACTION RESEARCH METHODOLOGY

Emerging issues in the design, delivery and research of Information and Communications Technology (ICT) systems and new media applications have established action research as the methodology to not only understand a problem, but to provoke change in the process (Dick et al, 1997). Action

research defines how stages of planning, acting, observing and reflecting are cycled through iterations through the duration of the research. This reflective process is significant in creating knowledge from constructing and evaluating technology artifacts relevant to its associated theories (Purao, 2008).

The overarching methodology driving this research aligns to Stephen Kemmis's Simple Action Research Model as shown in Figure 1 as it seeks change through cycled design iterations.

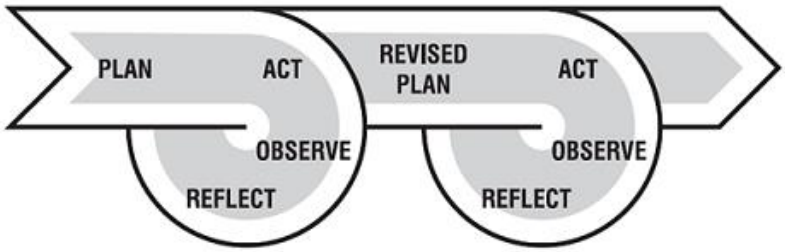


Figure 1. Simple Action Research Model by Stephen Kemmis.

4.2. PROJECT METHOD IN PRACTICE

The following method is planned:

1. From Flux's online database, create a web interface.
2. On the visual dynamic application, a project can be selected with a click. Keys of the project are displayed and is accessible for the user to select to obtain its data.
3. A key that houses geometry can be viewed in the viewport. The meshes in the viewport can be selected. When a specific mesh is selected, it will display a table of attributes.
4. The attributes can be changed within the table which will dynamically alter the geometry and its data in real-time.

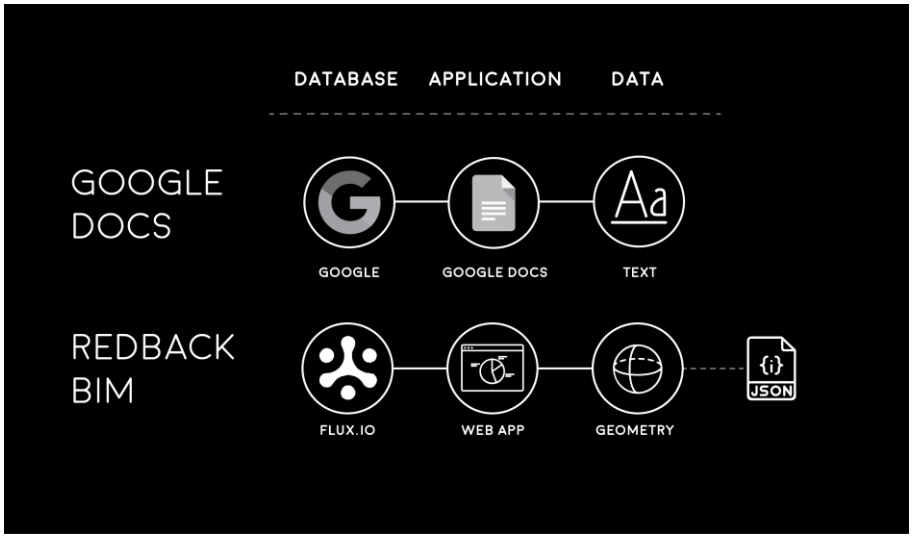


Figure 2. Redback BIM diagram comparative to Google Docs

As a proof of concept, the research will adopt the systematic relationship Google Docs (Figure 2) use by implementing the open-source web platform Flux.io to build the web application. This is based on the hypothesis that:

1. It unlocks existing communication barriers of 3D models through web-based technology
2. CAD and 3D modelling data can be analysed, edited, and documented online
3. Removes the need for localised 3D CAD programs where it can provide potential in producing the same functionalities as that of a 3D modelling software.

5. Background Research

5.1. WEB-BASED APPLICATION PRECEDENTS

Through the process of developing the web application, it is important to frame the project within the context of other dynamic platforms that are used in the AEC industry. These products encapsulate potential qualities and guidelines for what could be achieved in the final objective.

With emphasis on developing open-source synchronous collaborative workspaces for the AEC industry, the following headings were derived along

the first row of the table. More specifically, ‘Documentation' refers to its ability to store and produce 2D architectural drawings used in practice.
'3D / Interactive' is focused on the platform allowing users to build and interact with a 3D model. 'Three.js' is an open-source JavaScript library framework that allows users to build 3D scenes on the web. Finally, platforms that allow collaborative live documents to be hosted on the web, exist under the heading 'Free Collaboration'. As a result, research was conducted where the following table was produced:

TABLE 1. Web-based application precedents.

Precedent	Documentation	3D / Interactive	Three.js	Free Collaboration
Autodesk 360 BIM	✓	✓	✓	
Aconex	✓	✓		
Clara.io		✓	✓	
Dropbox	✓			
Flux.io		✓	✓	✓
OnShape	✓	✓		
Playpus		✓	✓	✓
ShapeDiver		✓	✓	

5.2. LITERATURE REVIEW

Given the evolution of the internet to be fast and unpredictable (León and Molina, 2009), industries of gaming and social media have taken advantage of the web’s ability to communicate virtual information between users in real-time. More specifically, French et al. (2016) have described how online multiplayer games such as Minecraft can offer insights into how web-hosted multi-participant environments can be achieved in CAD applications to address collaborative objectives. This suggests great potential for the AEC to further explore wiser applications for ICT to design and planning processes. Exploring opportunities for web-based ways of working is supported by many scholars in the fields of architecture and computer science (Raymond 1999, Graham 2009, León and Molina 2009, Counsell 2012, Marble 2012, Du and Zheng 2014, Xu et al 2016).

To begin, majority of 3D architectural models are built on remote desktop computers, which at times were shared onto local servers. However, this did not, and still does not, achieve a fluid integration of information from the multiple stakeholders who take part in projects (León and Molina, 2009). Therefore, the concept of a “federated single point of truth” is raised to

emphasise the need for a common virtual environment that holds the most current and reliable model defined from all stakeholders (Counsell, 2012, p.510).

However, given that BIM has success in storing model data, practice of web-based 3D visualisation of BIM models has been limited, given the potential value of information exchange to be one that has not been fully realised (Du and Zheng, 2014). In fact, BIM data has been defined as data like any other, where it can be handled in large database management systems (DBMS), which justifies the accessible nature of data to be one that can be transformed for the user's needs (Counsell, 2012). Such implications include the use of Flux Flow, a visual scripting web application that allows users to extract and transform data from properties within a building information model, to be streamlined into a digital spreadsheet readily available for contractors.

Consequently, existing research have focused on comparing existing remote tools to cloud-based platforms to achieve success in working on a project. These include Autodesk Buzzsaw and Dropbox which have presented themselves as alternatives (Counsell, 2012), yet are costly and do not resolve issues of duplication and confliction in file management across teams.

While the majority of studies provide valuable information in relation to the concepts drawn from accessing data on the internet, caution needs to be taken to compare the tools that existing papers have created, such as Vroom (León and Molina, 2009) and the 'IFC to WebGL visualisation of BIM data' (Xu et al 2016). These studies focus on visualisation and interaction with 3D models however do not manage the bi-directional flowing of data between multiple users in real-time. One should not assume that they provide a product that successfully manages the bidirectional change of data between multiple users, but rather proves the ability of presenting a tool that is able to visualise and interact with models on the web. This existing research does not view the web as a space to document and collaborate which is a contributing factor of this thesis. If focus was drawn on such applications, success would remove the boundaries of CAD and BIM programs to be one that could potentially allow digital architectural models to be made purely on the web.

It is therefore advantageous to break down the process into universally applicable processes and logic, to provide a methodology which has the potential to be reapplied using alternative open-source software and programming functions. WebGL (Web Graphics Library) is currently one of the most widely used JavaScript libraries, suitable for rendering interactive 3D models in any compatible web browser without any supplementary plug-

ins (Xu et al, 2016). To appreciate the effects of technological advancements in web-based 3D visualization, we must examine in detail, the different methodologies that have been tried and tested to explore the success of streaming and editing geometry and data on the web.

Current practices within architecture limit the ability to be flexible with the digital tools available (Marble, 2012). The software that architects use are not designed by them, but by computer scientists which makes it quite limiting to them. Consequently the "sheer fact of using architectural software means already to operate like an engineer" (Marble, 2012, p.18). Yet, if the architects understand and even author the tools that influence their process, potential efficiencies could emerge. Scott Marble derives 3 themes that define the integration of digital technologies in practice: Designing Design, Designing Assembly and Designing Industry.

Understanding and questioning the need for synchronous collaborative workspaces fits into Marble's category: Designing Industry. More specifically, scholars of computer science have idealised the need for open-source methodologies in creating successful technologies when no one owns the software. 'Hackers and Painters' is an essay that draws upon the significant connections that creatives such as architects have very similar processes to hackers - which is making. The concepts derived from Paul Graham are focused on understanding how pulling apart (hacking) into software can significantly impact unprecedented discoveries. This can only occur through removing The Cathedral Model (commericalised) to the open-source development model - The Bazaar (Raymond, 1999), which attains full effect of collaboration through being open to input from the public. This concept highlights the success of collaboration to be possible in today's digital age.

6. Case Study (REDBACK BIM)

When developing a 3D modelling web application, the main function is to be able to send and receive data between the frontend (user interface) and backend (database). In relation to Redback BIM, the user must therefore access and update data from a web-based interface, which is then synchronised in Flux. After synchronising the data, Flux can interpret and update the data, which is then sent back to the interface for the user to see.

This paper will look specifically into synchronising edited data back to the web server. There are several steps in this process. The focus will draw upon constructing a scenario for an editable JSON string in association with its geometry.

The following prototypes are available on Github, listed in Appendix A. Referenced Flux documentation is listed in Appendix B and the external online journal aligned to this research is noted in Appendix C.

6.1. COMPARATIVE STUDY BETWEEN DEVELOPING APPLICATIONS USING A-FRAME AND FLUX SDK DOCUMENTATION

6.1.1. Building an A-Frame Scene

Exploration into developing with the existing open-source WebVR JavaScript library, A-Frame was used to define an existing workflow in setting up a 3D architectural model on the web from a localised software. As the first prototype in development, a default sample Revit model was used as a test subject to be exported into a 3D DWG file. As A-Frame requires specific file types to be imported, the DWG files were brought into Rhinoceros to be exported as a series of OBJ files. The steps are as follows:

1. Export the Revit model as a DWG file
2. Modify the settings to export the file as ACIS solids
3. Choose to export it as an AutoCAD 2007 DWG file
4. Open the exported file in Rhinoceros to be exported as .obj files, or any file compatible with A-Frame
5. Store exported files into a folder to reference into A-Frame

The following diagram (Figure 3) derives the WebVR workflow from a localised server to the web:

a. Revit building information model (1)
b. Export to DWG file (A)
c. Open 3D DWG file in Rhinoceros (2)
d. Export to OBJ file (B)
e. Reference OBJ files in A-Frame (3)
f. Model data in JSON format reference into A-Frame (C)

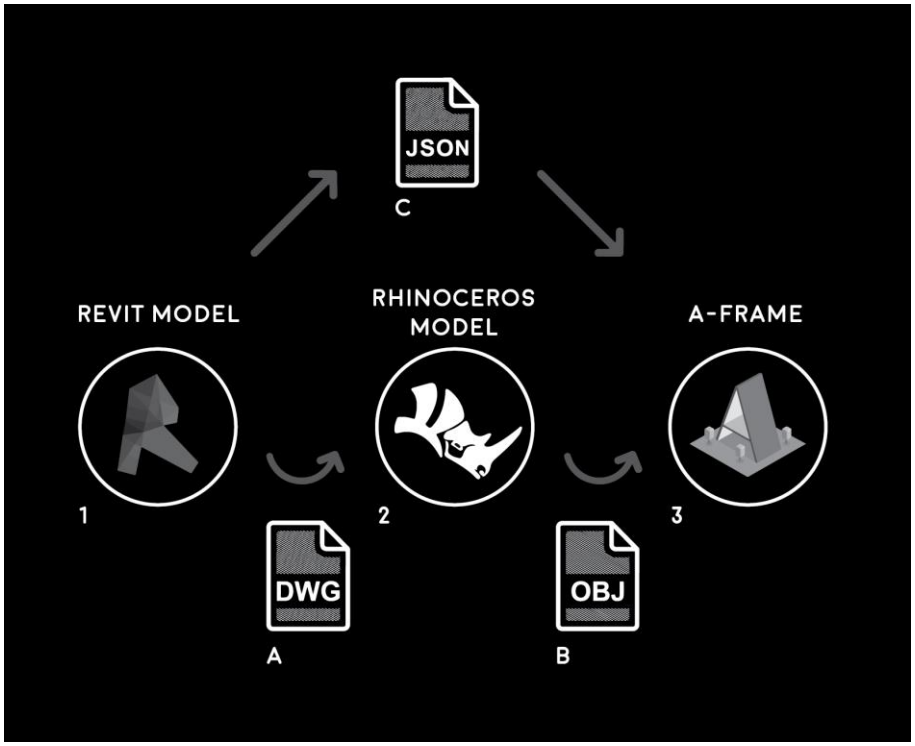


Figure 3. WebVR workflow from Revit to A-Frame.

The way A-Frame works is through referencing model files directly into the HTML code. Each OBJ file is brought into the website through assigning its own <a-asset-item> tag with an ID associated to it (Figure 4). For this reason, each individual file was labelled and tagged manually. This was then followed by creating an <a-obj-model> tag for each asset to be brought into the scene, which was referenced as shown in Figure 5.

```
247 <body>
248
249     <a-scene>
250
251         <!-- ASSETS -->
252         <a-assets>
253             <a-asset-item id="obj-1" src="reference/SAMPLE_C-TOPO.obj"></a-asset-item>
254             <a-asset-item id="obj-2" src="reference/SAMPLE_A-ROOF.obj"></a-asset-item>
255             <a-asset-item id="obj-3" src="reference/SAMPLE_P-SANR-FIXT.obj"></a-asset-item>
256             <a-asset-item id="obj-4" src="reference/SAMPLE_A-GLAZ-CURT.obj"></a-asset-item>
257             <a-asset-item id="obj-5" src="reference/SAMPLE_A-GLAZ-CWMG.obj"></a-asset-item>
258             <a-asset-item id="obj-6" src="reference/SAMPLE_A-GLAZ.obj"></a-asset-item>
259             <a-asset-item id="obj-7" src="reference/SAMPLE_A-FLOR-HRAL.obj"></a-asset-item>
260             <a-asset-item id="obj-8" src="reference/SAMPLE_Q-CASE.obj"></a-asset-item>
261             <a-asset-item id="obj-9" src="reference/SAMPLE_I-FURN.obj"></a-asset-item>
262             <a-asset-item id="obj-10" src="reference/SAMPLE_E-LITE-EQPM.obj"></a-asset-item>
263             <a-asset-item id="obj-11" src="reference/SAMPLE_E-ELEC-EQPM.obj"></a-asset-item>
264             <a-asset-item id="obj-12" src="reference/SAMPLE_A-CLING.obj"></a-asset-item>
265             <a-asset-item id="obj-13" src="reference/SAMPLE_S-STRS.obj"></a-asset-item>
266             <a-asset-item id="obj-14" src="reference/SAMPLE_S-FNDN.obj"></a-asset-item>
267             <a-asset-item id="obj-15" src="reference/SAMPLE_C-BLDG-PADS.obj"></a-asset-item>
268             <a-asset-item id="obj-16" src="reference/SAMPLE_I-WALL.obj"></a-asset-item>
269             <a-asset-item id="obj-17" src="reference/SAMPLE_A-WALL.obj"></a-asset-item>
270             <a-asset-item id="obj-18" src="reference/SAMPLE_A-FLOR.obj"></a-asset-item>
271             <a-asset-item id="obj-19" src="reference/SAMPLE_A-DETL-GENF.obj"></a-asset-item>
```

Figure 4. Line 253 onwards illustrates each OBJ file referenced in A-Frame as an asset.

```
293         <!-- OBJ MODELS -->
294         <a-entity id="target">
295             <a-obj-model change_blue src="#obj-1" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90
    0 -360" color="#424B54" object-info-on-click="color: #7FEFBD">
296             </a-obj-model>
297             <a-obj-model src="#obj-2" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90 0 -360"
    color="#424B54" object-info-on-click="color: #7FEFBD">
298             </a-obj-model>
299             <a-obj-model src="#obj-3" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90 0 -360"
    color="#424B54" object-info-on-click="color: #7FEFBD">
300             </a-obj-model>
301             <a-obj-model src="#obj-4" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90 0 -360"
    color="#424B54" object-info-on-click="color: #7FEFBD">
302             </a-obj-model>
303             <a-obj-model src="#obj-5" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90 0 -360"
    color="#424B54" object-info-on-click="color: #7FEFBD">
304             </a-obj-model>
305             <a-obj-model src="#obj-6" position="5 0 -15" scale="0.001 0.001 0.001" rotation="-90 0 -360"
    color="#424B54" object-info-on-click="color: #7FEFBD">
306             </a-obj-model>
```

Figure 5. Line 295 onwards demonstrates the assets referenced into the scene as an <a-obj-model> to be viewed in the A-Frame scene.

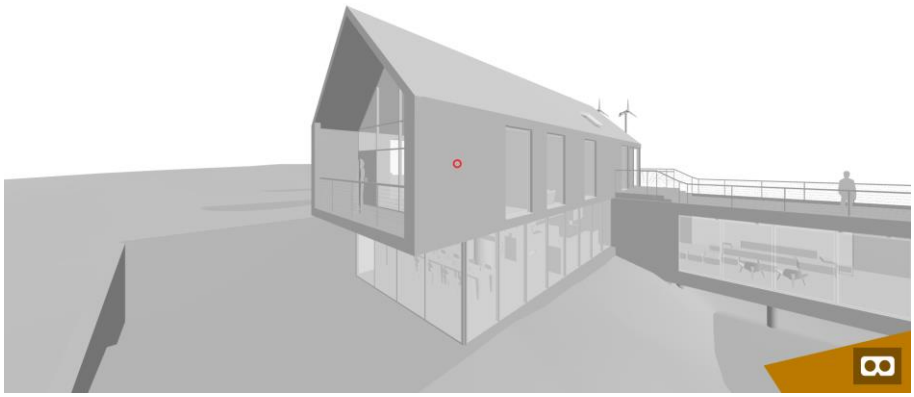


Figure 6. Architectural model A-Frame scene.

Figure 6 was the result of a 3D scene on the web that allows users to view the space on any mobile device. Now that this was complete, functionality could be added through using JavaScript. More specifically, a JavaScript variable that holds an array of JSON objects (Figure 7) was referenced and aligned to each object's tag by associating the data with the ID of the object. The JSON object was written out manually as each file contained multiple objects. However, this process could also be done through exporting the data from the BIM software to a JSON file, but that would require each individual object to be exported as an .obj file. When an object within the scene is selected, the associated ID of that object is called, and this displays its data within a HTML element situated on the left sidebar (Figure 8).

```
var objInfo = {
      "obj-1": {
         "name": "Topography",
         "cost": 12,
         "material": "Grass",
         "dimensions": "500 x 500"
      },
      "obj-2": {
         "name": "Roof",
         "cost": 55,
         "material": "Corrugated Steel",
         "dimensions": "500 x 5000 each"
      },
      "obj-3": {
         "name": "Fixtures",
         "cost": 100,
         "material": "Fibre glass",
         "dimensions": "1700 x 700"
      },
```

Figure 7. JavaScript variable storing a JSON array for A-Frame scene.

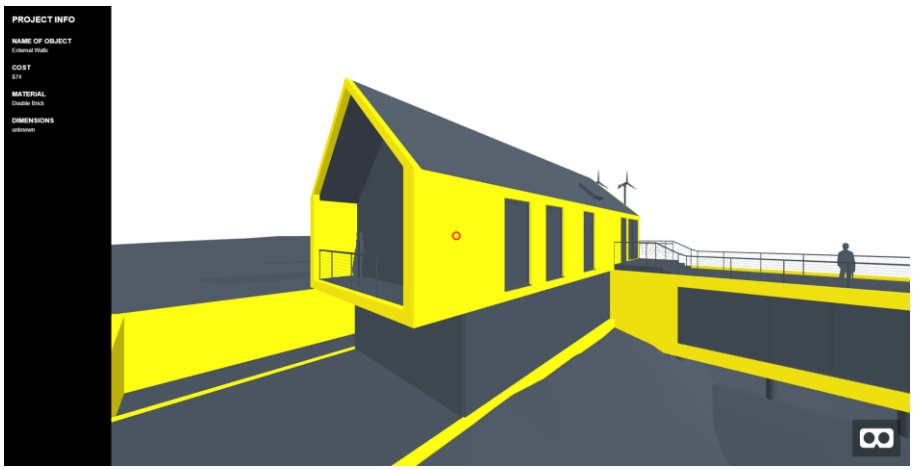


Figure 8. Selected object in A-Frame scene displays data in sidebar.

6.1.2. Building a web application using Flux SDK documentation

In comparison to A-Frame, the desire for accessibility, synchronous data and collaboration requires a platform very similar to Flux.io. Specifically, Flux is a web platform that allows users to send and receive data to and from the web between compatible software. Various types of data can be transferred, from integers, strings and even 3D geometry from a modelling or BIM software to Flux. This was made possible through the many plugins they've developed where the value of the data is stored within your account, in a project and allocated a key. Just like JSON, you have a key and a value {"key": "value"}.

Geometry then becomes the value as expressed in JSON format which is organised by the user for others to access. As a result, building a web application using Flux can open opportunities for creating live 3D documents through a user interface by editing data (in this case, geometry, which is made up of a JSON string).

Following the Flux SDK documentation, the default application was created.

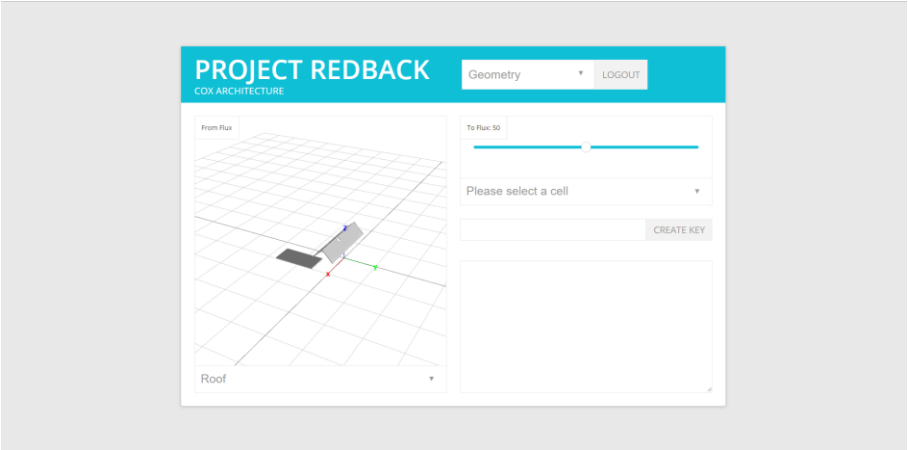


Figure 9. Application developed using Flux SDK Documentation.

6.2. DEVELOPING AN UNDERSTANDING FOR THREE.JS AND FLUX JSON DATA

6.2.1. Three.js JSON and Flux JSON

The process of developing the open-source web application initially suggested a method that would be able to convert Flux JSON to Three.js readable JSON. The next two figures demonstrate the difference between how Three.js and Flux reads JSON data.


```
{
    "metadata":{
        "version":3,
        "type":"BufferGeometry",
        "position":8,
        "generator":"io_three"
    },
    "data":{
        "attributes":{
            "position":{
                "itemSize":3,
                "array":[1,-1,-1,-1,-1,1,-1,-1,-1,-1,1,-1,0.999999,1,1,1,-1,1,-1,1,-1,1,1],
                "type":"Float32Array"
            }
        },
        "index":{
            "itemSize":1,
            "array":[0,1,2,3,4,5,5,6,0,4,1,6,1,3,2,0,3,5,0,6,1,3,7,4,5,4,6,4,7,1,1,7,3,0,2,3],
            "type":"Uint16Array"
        }
    }
}
```


Figure 10. Three.js JSON file of a 2m x 2m cube.

Figure 11. Flux JSON data of a 2m x 2m cube.

It is evident that from these images, Flux reads its own form of JSON data which was made easier to understand for its users by implementing their own

core JSON schema. This problem was therefore removed from the process as the focus of the research was drawn towards creating a synchronous workflow of updating geometry data. For this reason, this issue will be considered later in future projects.

6.2.2. Setting up Flux application

With consideration of following the existing Flux SDK documentation, the realisation to switch from using HTML, CSS, and JavaScript to React was made due to its ability to "…really shine(s) when your data changes over time" (Hunt, 2013), as well as its ability to be developed further as a progressive web application.

In a collaborative effort between this research and Kyle Maxwell, an Engineer from Flux, a sample app was built which facilitated the development of the research outcomes. Figure 12 presents the sample app accessing a user's projects and thus data on a React web application to be displayed within a text area element (Three.js vs Flux JSON) and their Flux Viewport (made using Three.js).

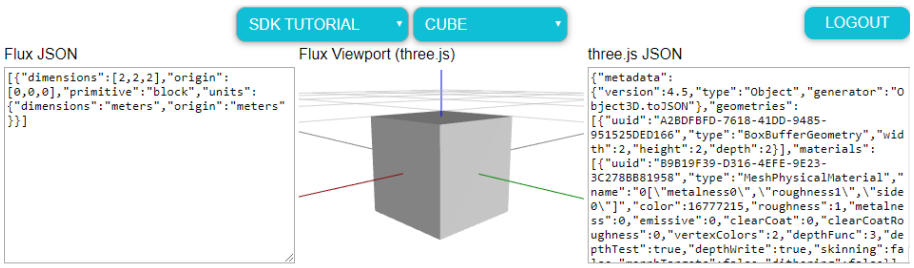


Figure 12. Flux-Three-JSON Sample App.

6.3. ENABLE COMMUNICATION BETWEEN WEB APPLICATION AND FLUX DATABASE

6.3.1. Defining the Workflow

When an architect or stakeholder has existing digital models of a site or building, Flux allows the possibility of sending the geometry to the web for others to access. Using Flux as a database for the web application, the steps are as follows:

1. Existing 3D models are sent to Flux through its compatible plugins
2. Flux stores the geometry and its data in the user's projects and keys
3. Redback BIM accesses the Flux database through JavaScript which parses the JSON into the Flux Viewport and React components

The following are the relational components to the workflow:

A. Database for Flux
B. JSON data (integers, strings, Flux reads as geometry)
C. Three.js (3D viewport)
D. React components

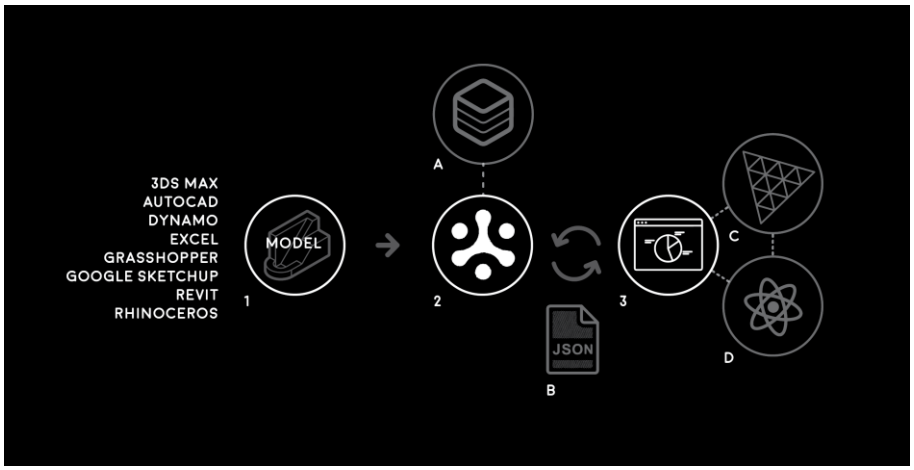


Figure 13. Workflow diagram of Redback BIM.

To enable communication between Redback BIM and Flux Database, the following functions (Figure 14) as documented in the Flux SDK within a

referenced JavaScript file (helpers.js) were used to interact with Flux's database. An example of its use is seen in Figures 15-17.

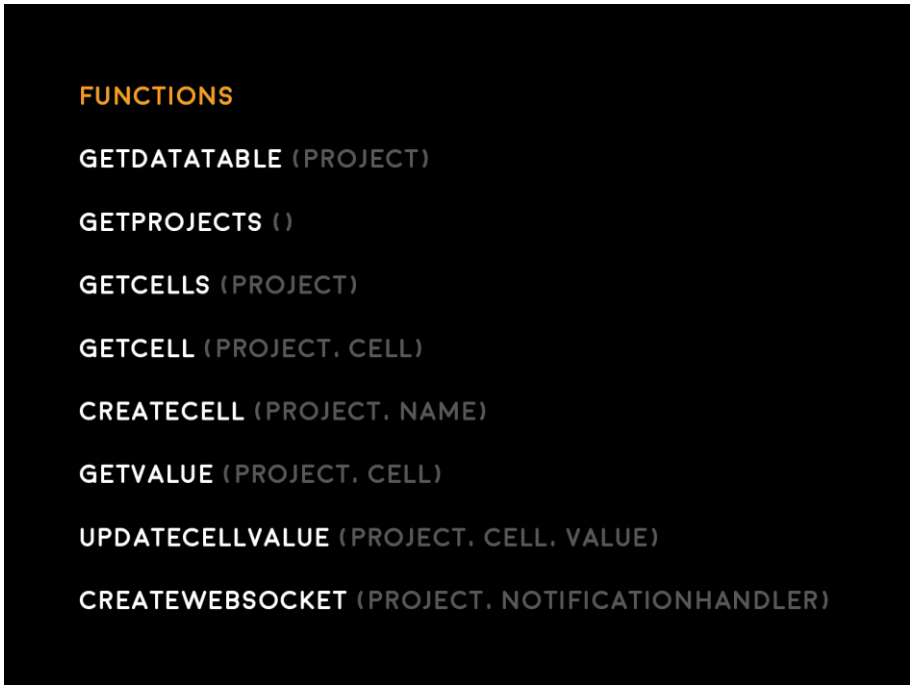


Figure 14. Flux SDK functions as derived in helpers.js.

6.3.2. Updating a key's value

The following steps was introduced on top of the sample application built by Maxwell (Figure 12). To update data in a Flux key, an input value must be obtained. A React form was tested for submission of a slider's input value. Line 218 expresses the value of the key in the application as text. Between lines 220 to 236 is the form that includes: an input slider (lines 221 - 230) which calls the 'keyChange' function whenever the slider changes, an element that displays the input slider's value (line 232), as well as a submit button which calls the function ('updateKey') whenever the button is clicked.

```
<strong>Current Value of key:</strong> {this.state.data}

<form>
    <input
        style={{ width: 250}}
        id="key_slider"
        type="range"
        min="0" max="100"
        step="1"
        defaultValue={this.state.data}
        onChange={this.keyChange.bind(this)}
        ref={(input) => this.input = input}
    />

    <strong>New Value to submit:</strong> {this.state.value}

    <br/>
    <input type="submit" value="Submit" onClick={this.updateKey.bind(this)} />
</form>
```

Figure 15. Rendered Slider React Component.

```
keyChange(event)  {
  this.setState({value:event.target.value});
}
```

Figure 16. KeyChange function is called when React slider is changed.

```
updateKey(e){
    if(this.project != null || this.state.value != null) {
      e.preventDefault();
      this.setState({
        "data": "Loading..."
      });
      helpers.updateCellValue(this.project, this.key, parseInt(this.state.value)).then((cell)=>{
        this.setState.value = cell.value;
        this.updateViewport(this.value);
        this.setState({
          "data": parseInt(this.state.value)
        });
    });
  }
}

```

Figure 17. UpdateKey function is called when React form is submitted.

The ‘keyChange’ function (Figure 16) is called to change the state of the slider’s input value. The submit button calls the ‘updateKey’ function (Figure 17), which in line 184 requires a conditional of a selected project and key to allow for submission. The input value of the slider is then pushed to Flux where the function ‘updateCellValue’ is called from helpers.js. The state of the input value becomes the value in Flux’s database (line 190), which is then synchronised in the web application’s state (line 193) and viewport (line 191).

6.3.3. Access Geometry and Data via mouse click on Mesh for its JSON data

Developing on from Maxwell’s open-source repository and using the Flux-Viewport documentation, the following lines of code were added to allow users to engage with geometry within the viewport. Therefore, each time a geometry is selected with a click action, the object is highlighted and the JSON string that makes up the geometry is printed in the console as shown in Figure 20.

```
62
63    setViewport(div)
64    {
65      if (div == null || this.viewportDiv != null) return;
66      this.viewportDiv = div;
67      this.div = div;
68
69      // Set up the FluxViewport in it's container
70      var token = helpers.getFluxToken();
71      this.vp = new FluxViewport(this.div, {
72        projectId: this.project.id,
73        token: token,
74        selection: FluxViewport.getSelectionModes().CLICK
75      });
76      // var sphere = JSON.parse(this.state.data);
77      // this.updateViewport(sphere);
78
```

Figure 18. Line 74 was added to select and highlight geometry in Flux viewport.

```
80
81      this.vp.addEventListener(FluxViewport.getChangeEvent(), function(e) {
82        var selectedGeometry = e.selection;
83        if(selectedGeometryObject !== null && selectedGeometryObject !== undefined) {
84          var myData = JSON.parse(JSON.stringify(selectedGeometryObject.userData.data));
85          console.log(myData);
86        } else {
87          console.log("Nothing")
88        }
89      });
90  }
91
```

Figure 19. Line 81 establishes an addeventlistener in the viewport to detect a selection to display the geometry's JSON string value.

```
{"dimensions":[1,2,3],"origin":[0,0,0],"primitive":"block"}        App.js:135
>
```

Figure 20. Data of geometry selected printed in console.

6.3.4. Additional Features

Further research into the Flux Viewport documentation has collated additional functions and features. These of which can do very similar tasks as that in localised 3D modelling programs. Additional research can introduce the implementation of these functions in future projects.

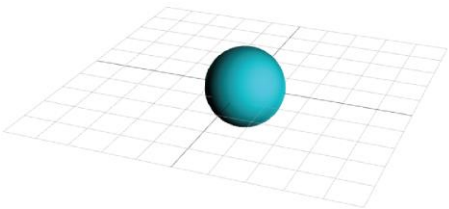
Figure 21. Additional functions in Flux Viewport Documentation for future implementation.

6.4. OBTAIN GEOMETRY DATA FOR EDITING IN WEB APPLICATION

6.4.1. JSON Datatable via mouse click

As geometry is made up of a JSON string, the next step was to attempt to generate a datatable that would be editable for stakeholders to view and update. Using pure JavaScript (Figure 23), the JSON string of a selected geometry (Figure 20) was obtained and extracted to generate a HTML table.

The following table in Figure 22 illustrates datatable of a selected sphere. Due to the structure of a geometry's JSON in Flux to include nested data (as shown on the right of Figure 22), the datatable cannot preview the data within its allocated cell. The areas highlighted blue emphasise the nested JSON as '[object Object]' with an additional problem that does not allow users to edit the HTML element.



Data Table

attributes	origin	primitive	radius	units
[object Object]	0,0,0	sphere	12	[object Object]

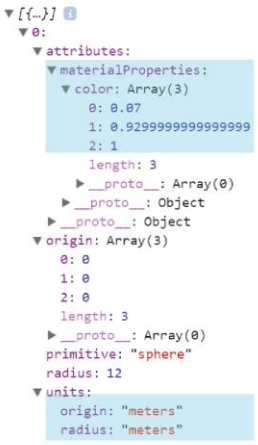


Figure 22. Datatable previewed in application when geometry is selected. Right is JSON parsed into the console. Below left is the resulting HTML datatable.

```
81  this.vp.addEventListener(FluxViewport.getChangeEvent(), function(e) {
82    var selectedGeometry = e.selection;
83    for(this.x in selectedGeometry) {
84      var selectedGeometryObject = selectedGeometry[this.x];
85    }
86
87    if(selectedGeometryObject !== null && selectedGeometryObject !== undefined) {
88      var myData = JSON.parse(JSON.stringify(selectedGeometryObject.userData.data));
89
90      INSTANT DATATABLE VIA MOUSE CLICK (DOES NOT WORK ON NESTED JSON)
91
92      var myData = JSON.parse("[" + JSON.stringify(selectedGeometryObject.userData.data) + "]" );
93      var col = [];
94      for (var i = 0; i < myData.length; i++) {
95          for (var key in myData[i]) {
96              if (col.indexOf(key) === -1) {
97                  col.push(key);
98              }
99          }
100     }
101
102     // CREATE DYNAMIC TABLE.
103     var table = document.createElement("table");
104
105     // CREATE HTML TABLE HEADER ROW USING THE EXTRACTED HEADERS ABOVE.
106
107     var tr = table.insertRow(-1);                   // TABLE ROW.
108
109     for (var i = 0; i < col.length; i++) {
110         var th = document.createElement("th");      // TABLE HEADER.
111         th.innerHTML = col[i];
112         tr.appendChild(th);
113     }
114
115     // ADD JSON DATA TO THE TABLE AS ROWS.
116     for (var i = 0; i < myData.length; i++) {
117
118         tr = table.insertRow(-1);
119
120         for (var j = 0; j < col.length; j++) {
121             var tabCell = tr.insertCell(-1);
122             tabCell.innerHTML = myData[i][col[j]];
123         }
124     }
125
126     // FINALLY ADD THE NEWLY CREATED TABLE WITH JSON DATA TO A CONTAINER.
127     var divContainer = document.getElementById("showData");
128     divContainer.innerHTML = "";
129     divContainer.appendChild(table);
130
131     console.log(myData);
132   } else {
133     console.log("Nothing")
134     $( "#showData" ).empty();
135   }
136 });
137 }
138
```

Figure 23. Lines 83 – 129 outlines the JavaScript used to produce a HTML datatable with a selection of a geometry in Redback BIM.

6.4.2. Open-source React JSON Tree Component

There are a multitude of open-source components available for use when developing applications. Research into finding an alternative solution was derived to resolve the inability to edit the datatable or view nested JSON which appears as [Object Object].

A JSON Tree component created by Mac Gainor was implemented in Redback BIM. Line 351 outlines the properties required, where the source (src) of data is the value of the key selected, which is piped through to generate the JSON Tree. When changes were made, the props (onEdit, onAdd, onDelete) are called, which fire the function 'this.editAttribute' as shown in Figure 25.

```
348
349             <h3>Data Table</h3>
350             <div id="showData" className="showDataTable" ></div>
351             <ReactJson src={object} onEdit={this.editAttribute} onAdd={this.editAttribute}
                onDelete={this.editAttribute} theme="base01" collapsed={1} iconStyle="circle" />
352
```

Figure 24. React JSON Tree Component.

```
308
309 editAttribute = (object) => {
310   // console.log(updated_src)
311
312   console.log(JSON.stringify(object.updated_src));
313   var hello = JSON.parse(JSON.stringify(object.updated_src));
314   if(this.project != null || this.state.value != null || object.updated_src !=null) {
315     this.setState({
316       "data": JSON.stringify(hello),
317       "object": JSON.stringify(hello)
318     });
319     // console.log("hello world")
320     helpers.updateCellValue(this.project, this.key, hello).then((cell)=>{
321       this.setState.value = cell.value;
322       this.updateViewport(JSON.parse(this.state.data));
323       this.setState({
324         "data": this.state.data
325       });
326     });
327   };
328 }
```

Figure 25. Function that is called when the React JSON Tree Component is edited.

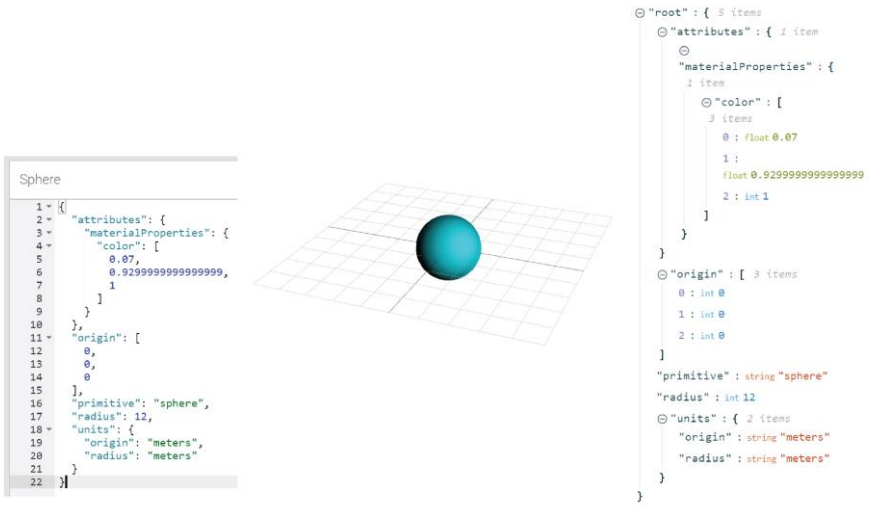


Figure 26. Left is the JSON as shown in Flux. Right is the React JSON Tree component with the geometry JSON string parsed into for editing.

Redback BIM allows a selected key to push its data into the JSON Tree component, where changes made to it are pushed back to Flux, updating the database and the web application interface, which is available as an open source project on Github.

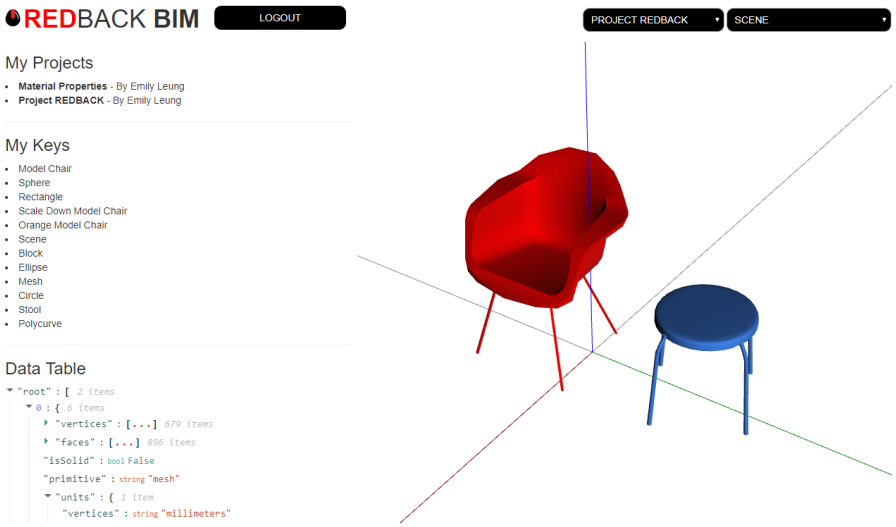


Figure 27. Redback BIM Flux Web Application.

7. Significance of Research

The paper concludes and argues that a direct link between 3D models and the web would benefit all stakeholders of a project. Based on the research of academic papers and action research practice, Redback BIM has shown potential to succeed as a step forward as a tool and workflow to enable synchronous and collaborative workspaces in an AEC industry context.

Referencing existing web-based synchronous collaborative workspaces such as Google Docs has contributed to defining a systematic workflow for such programs to be implemented within an architectural context, and achieve the same success in removing communication barriers. Existing communication barriers between coordinating a 3D architectural model can be resolved through breaking down the accessing, embedding and updating of geometry data to be done solely on the web. This paper has demonstrated the case by interconnecting JSON data to client-server interfaces for architects, designers and the multitude of stakeholders to communicate effectively in one common space. More specifically, the realisation of this research can be applied as users of the web application can gain access to live information that previously required use of other tools to communicate with. Using this web-based tool in architectural practice can cut costs in software licensing, hosting of private servers and save time by instantaneously tracking, updating and communicating changes to a 3D model. By implementing Redback BIM in the industry, multiple stakeholders can converse in a live dialogue within their 3D architectural models and achieve a design process that allows informed decisions to be made.

Open-source software has already achieved free manifestation to allow the public to further develop tools that are otherwise limiting for their users. By implementing a combination of open-source libraries and software currently available, the possibilities to build and further develop custom programs can generate unprecedented results. Users of architectural, computer science and creatives can freely use Redback BIM and even contribute to a community of like-minded computational designers, architects and tinkerers of 3D modelling applications. Thus, many projects can stem from this one framework as this project also offers scalability as an open-source application.

8. Evaluation

This project focused heavily on interdisciplinary learning, bridging areas of computer science and architecture into one platform. Overall, success was seen in creating a synchronous, collaborative web application which enables multiple stakeholders to feed, update and extract data between a database and

a user interface. The outcome to successfully enable a user to access and update geometry data through a web application, was done through updating the geometry's JSON string on a database (Flux.io) hosted on the web. This product demonstrates a new automated means of delivering synchronised models that provide real-time feedback and changes. This project provides a centralised model that is accessible anywhere, which has the power to revolutionise the industry by streamlining previous practices. Whilst the success of the case study produced a synchronous collaborative workspace on a web-based platform, there are many constraints and limitations that should be addressed in the future.

First, development of open-source applications requires highly detailed and understandable documentation to maintain success in user application. Many of the React components that did not consider the need for clear documentation, was unable to be implemented within Redback BIM. However, the case for open-source components can be difficult to maintain as they can sometimes be deprecated for further use and development.

Another aspect to consider when developing open-source architecture-centric tools is the knowledge that is required to build such tools. Success of these tools are heavily reliant on basic knowledge and experience with web development languages (HTML, CSS, JavaScript, React, Three.js), 3D modelling filetypes (obj, json, stl), data structures (JSON) and data management concepts.

In relation to Redback BIM, as an application that seeks to provide a platform for storing, accessing, updating and tracking changes made to a 3D model on the web, it lacks the desired user-friendly interface that a typical localised 3D modelling software has achieved. Precedent web applications such as Clara.io prove the ability to attain this interactivity between the model by providing a user interface like that of a 3D modelling platform. Redback BIM at this stage does not provide interactivity on the geometry to change the form but does this through editing its JSON value in a React component. For this reason, the application is only able to bring 3D model data together from external programs to allow for integration of models that can be edited. It also can update parameters but has yet to include the option of creating geometry directly in the platform which will be included later down the track.

With the emphasis on building applications, an area that is yet to be fully implemented is to justify the use of Redback BIM in practice within the industry through user testing scenarios. Given that an objective of this

research was to enable PWA capabilities onto Redback BIM, the application is yet to be published for mobile device access and user testing. When this goal is achieved, it can also quantify performance issues and qualitatively define the success of removing existing communication barriers in the process.

However, these factors can be improved with further development and iterations. This case study should be considered as a first prototype in the development of a complex relationship between web-based tools and data within architectural models. With more research and time invested in the progression of Redback BIM, it would assist in expanding the usability, efficiency and flexibility as an open-source tool between multiple stakeholders of a project.

9. Conclusion

Research into scholars within the field of architecture and computer science has defined data as an important asset which should be taken advantage of in the AEC industry. Therefore, this research explored and developed using existing open-source WebVR and database technologies (A-Frame, Three.js and Flux.io) in order to contribute to building a collaborative web application, where synchronous relationships were generated between data and 3D geometry. With the introduction of Redback BIM, limitations such as the inability to create geometry with a user-friendly interface will require additional research. Future research will also be implemented to justify the success of Redback BIM in practice within the industry through user testing scenarios. However, by bridging the areas of architecture and computer science, the ability to collaboratively organise and synchronise changes made to a 3D architectural model in real-time. This successful integration is a step closer to building live architecture-centric applications on the web, contributing to the potential to optimise communication practices and focus our attention to solving real world problems.

Acknowledgements

This research was done in conjunction with Cox Architecture and the University of New South Wales Built Environment Faculty. The authors would like to thank Kyle Maxwell, the engineer from Flux.io for their valuable help in making Redback BIM possible.

References

Aguirre León, E.A. and Molina, M.R., 2009. New Interfaces, new scenarios. Vroom n. 0: Vroom n. 0: The emerging potential of collaborative 3D web platforms. SIGraDi 2009 sp.

Birch, S. 2017. Progressive Web Apps: Great Experiences Everywhere (Google I/O '17). [video] Available at: https://youtu.be/m-sCdS0sQO8 [Accessed 11 Aug. 2017].

Counsell, J. "Beyond Level 2 BIM, Web Portals and Collaboration Tools," 2012 16th International Conference on Information Visualisation, Montpellier, 2012, pp. 510-515.

French, D., Stone, B., Nysetvold, T., Hepworth, A. and Edward Red, W. 2016. Collaborative Design Principles From Minecraft With Applications to Multi-User Computer-Aided Design. Journal of Computing and Information Science in Engineering, 16(2), p.021006.

Gainor, Mac, 2017, Github, accessed 19.10.17, < https://mac-s-g.github.io/react-json-view/ >

Graham, P. 2009. Hackers & Painters. Sebastopol: O'Reilly Media, Inc.

Hearn, Gregory N. and Foth, Marcus. 2005. Action Research in the Design of New Media and ICT Systems, in Kwansah-Aidoo, Kwamena, Eds. Topical Issues in Communications and Media Research, pages pp. 79-94. Nova Science.

Hunt, P., 2013, Why did we build React, accessed 13.09.17, < https://reactjs.org/blog/2013/06/05/why-react.html >

Juan, Du & Zheng, Qin. 2014. Cloud and Open BIM-Based Building Information Interoperability Research. Journal of Service Science and Management. 07. 47-56.

Marble, S. 2012. Digital workflows in architecture. Basel: Birkhäuser.

Maxwell, Kyle, 2017, Github, accessed 31.08.17, < https://github.com/flux-labs/flux-three-json >

Norouzi, N., Shabak, M., Embi, M. R. B. and Khan, T. H., 2014. The Architect, the Client and Effective Communication in Architectural Design Practice. Procedia - Social and Behavioral Sciences, vol. 172, 635-642.

O'Brien, R. 1998. An Overview of the Methodological Approach of Action Research, Accessed, 19.10.17, < http://www.web.net/~robrien/papers/arfinal.html >

Purao, S. 2008. The overlaps between Action Research and Design Research. [online] Slideshare.net. Available at: https://www.slideshare.net/SandeepPurao/draricis2005 [Accessed 1 Nov. 2017].

Raisbeck, Peter, 2016, Knowledge Future: Future Proofing the Architectural Firm, accessed 16.10.17, < http://aca.org.au/article/knowledge-futures-future-proofing-the-architectural-firm >

Raymond, E. 1999. The cathedral & the bazaar. Sebastopol: O'Reailly Media, Inc.

Sung-Jin Kim, F. Kuester and H. K. Kim, 2002. "A global timestamp-based scalable framework for Multi-player Online Games," Fourth International Symposium on Multimedia Software Engineering. Proceedings., 2002, pp. 2-10.

Xu, Zhao & Zhang, Yang & Xu, Xiayan. 2016. 3D visualization for building information models based upon IFC and WebGL integration. Multimedia Tools and Applications.

Appendices

APPENDIX A

TABLE 2. Prototype Github repositories.

Prototype / Documentation	**Github Repository**
A-Frame Scene	https://github.com/emilyyleung/revit_obj_170821
Sample App Flux-Three-JSON by Kyle Maxwell	https://github.com/flux-labs/flux-three-json
React-json-view Component By Mac Gainor	https://mac-s-g.github.io/react-json-view/
Redback BIM	https://github.com/emilyyleung/Redback_BIM

APPENDIX B

TABLE 3. Flux SDK application documentation.

Documentation	**Website**
Flux App SDK Documentation	https://flux.gitbooks.io/flux-javascript-sdk/content/
Flux Viewport Documentation	https://flux-viewport-reference.herokuapp.com/index.html

APPENDIX C

TABLE 4. Online process journal.

Online Process Journal	**Website**
Project Redback Medium Blog	https://medium.com/scratch-pad-3201-3202