

GNG5140
Design Project User and Product Manual

Open-Sourced Educational Toys – BusyPad 3.0

Submitted by:

Braden Stang, 300426142

Jingxuan Xu, 300450716

Mahmoud Mohammed, 300369733

Youssef Fathi, 300380805

April 3, 2025

University of Ottawa



uOttawa

Table of Contents

Table of Contents	ii
List of Figures	vi
List of Tables	viii
List of Acronyms and Glossary	ix
1 Introduction.....	1
2 Overview.....	2
2.1 Conventions.....	3
2.2 Cautions & Warnings.....	3
3 Getting started.....	5
3.1 Set-up Considerations	5
3.2 User Access Considerations	10
3.3 Accessing the System.....	10
3.4 System Organization & Navigation	13
3.5 Exiting the System	14
4 Using the System	15
4.1 Registering an Account	15
4.2 Logging into Your Account	15
• Parent Account Login.....	16
• Player Account Login.....	16
4.3 Parental Control Function	16

4.4	Playing Games.....	17
	• Playing a Game	17
	• Filtering Games by Category	19
5	Troubleshooting & Support	20
5.1	Error Messages or Behaviors	20
5.2	Special Considerations	20
5.3	Maintenance	20
5.4	Support	21
6	Product Documentation	22
6.1	Hardware System	22
	• BOM (Bill of Materials).....	22
	• Equipment list	22
	• Instructions	23
6.2	3D Design.....	38
	• Equipment list	42
	• Instructions	42
6.3	Software System.....	45
	• Frontend	45
	• Backend.....	48
	System Requirements.....	48
	Installation.....	48
6.4	Clone the Repository	48

• Create Virtual Environment	48
• Install Dependencies	49
• Firebase Setup	49
Running the API	50
Development Mode.....	50
Production Mode.....	50
API Usage	50
Authentication.....	50
Game Management (Admin)	53
Player Access	61
Troubleshooting	63
• Common Installation Issues	63
• Configuration Issues.....	64
• Runtime Issues	65
API Reference	65
• Authentication Endpoints.....	65
• Admin Endpoints.....	66
• Player Endpoints	67
Data Models	68
• Game Model	68
• Admin Model	68

•	Device Model	69
6.5	Cloud Hosting	69
•	Prerequisites	69
•	Instructions	70
6.6	Testing & Validation.....	76
•	Number of Games	76
•	Boot Performance.....	76
•	Games and Performance.....	77
•	APP Performance	78
•	APP Functionality	79
7	Conclusions and Recommendations for Future Work.....	81
7.1	Conclusion.....	81
7.2	Recommendations and Future Work.....	82
8	Bibliography	84
	APPENDICES	85
9	APPENDIX I: Design Files	85

List of Figures

Figure 1: BusyPad 3.0 Device	2
Figure 2: BusyPad 3.0 attached power USB cable	5
Figure 3: Opening the WPA_GUI application.....	7
Figure 4: Clicking the 'Scan' button to search for Wi-Fi networks.....	7
Figure 5: Viewing available networks and signal strength.....	8
Figure 6: Entering the Wi-Fi password (PSK).....	8
Figure 7: Closing the WPA_GUI window after connection.....	9
Figure 8: BusyPad login screen asking for device code after network connection	9
Figure 9: Parent Login Screen: Enter email and password to access parental controls.....	12
Figure 10: Parent Registration Screen: Sign up with email, password, and device code	12
Figure 11: Player Page Login: Enter the device code to access child-friendly content.....	13
Figure 12: Sign-up Page.....	15
Figure 13: Home Page for Parent Account	16
Figure 14: Parental Contral Page	17
Figure 15: Home Page for Player Account	18
Figure 16: Playing Game Page	18
Figure 17: Raspberry Pi Zero 2W Pin Out.....	23
Figure 18: Joystick Connection	25
Figure 19: Final Prototype Component Layout	25
Figure 20: Electrical Decomposition	26

Figure 21: Final Prototype Hardware Implementation	27
Figure 22: Pi Imager Interface	29
Figure 23: Pi Imager Interface	29
Figure 24: PuTTY SSH Client Interface.....	30
Figure 25: Raspberry Pi CLI.....	31
Figure 26: Kiosk Control Panel	32
Figure 27: CURA 3D Design Parameters	39
Figure 28: BusyPad 3.0 CAD Assembly	40
Figure 29: BusyPad 3.0 CAD Assembly Side View.....	41
Figure 30: BusyPad 3.0 Orthographic Sketches	41
Figure 31: 3D Design Print Settings	44
Figure 32: Frontend Structure Overview	45
Figure 33: Cloud Hosting Diagram.....	70

List of Tables

Table 1. Acronyms.....	ix
Table 2: Bill of Materials.....	22
Table 3: Required Components	42
Table 4: Common Installation Issues.....	63
Table 5: Configuration Issues	64
Table 6: Runtime Issues.....	65
Table 7: Authentication Endpoints	65
Table 8: Admin Endpoints	66
Table 9: Player Endpoints.....	67
Table 10: Boot Performance	76
Table 11: Educational Game Performance	77
Table 12: Fun Game Performance	78
Table 13: App Performance	78
Table 14: App Functionality	79
Table 15. Referenced Documents	85

List of Acronyms and Glossary

Provide a list of acronyms and associated literal translations used within the document. List the acronyms in alphabetical order using a tabular format as depicted below.

Table 1. Acronyms

Acronym	Definition
UPM	User and Product Manual
BOM	Bill of Materials
UNSDG	United Nations Sustainable Development Goals
PLA	Polylactic Acid
COTS	Commercial off-the-shelf
USB	Universal Serial Bus
GUI	Graphical User Interface
IP	Internet Protocol
PCB	Printed Circuit Board
GPIO	General-Purpose Input/Output
ADC	Analog-to-Digital Converter
HDMI	High-Definition Multimedia Interface
SSH	Secure Shell)
URL	Uniform Resource Locator
OS	Operating System
CAD	Computer-aided design

1 Introduction

This User and Product Manual (UPM) provides the information necessary for administrative users to effectively use the BusyPad 3.0 and for prototype documentation.

This document represents a comprehensive user manual for the BusyPad 3.0. This document should be used alongside the device to understand and operate the BusyPad effectively. The report is broken down into many key sections to aid the user in the operation of the device. First a general overview and explanation of the device is presented along with added context for understanding the project and the need for such a device to benefit the intended users. In addition, standard industry conventions, cautions and warnings are included to ensure directions are clear and all users remain safe while operating the device.

Next the manual covers startup up and shutdown procedures of the system to ensure optimal loading times and to minimize the power draw from the external power cord. The document will explain considerations to take before setting up the device, and any required access considerations should they be required with the necessary steps to set up the device, access the application and exiting the system should the user be finished with device operations. The report will discuss using the system focusing on specific function and subfunctions of both hardware and all connected components and peripherals, and the frontend and backend software's. This way more technical users will understand the operation of the device should troubleshooting be required.

Next, the manual covers troubleshooting and support. Should any errors occur, the extensive list of support provided in the document should aid users in effectively finding a solution. The software is very robust, however due to the nature of open-sourced software there may be unintended errors that occur. As precautionary measures, we advise in referring to the documentation should any error occur. If problems continue to persist the contacting Flipped Toys for further information may be necessary.

Finally, the manual includes all necessary supporting documentation for the BusyPad3.0. This includes the Bill of Materials, Equipment & Component List, detailed operational instructions, and all testing and validation information for the device. As the device is open-sourced, it is our obligation to provide all information to ensure clarity and transparency to all users intended or otherwise. The hope is to have the next generation learn from the developments made on te BusyPad 3.0 and take their knowledge of engineering and electronics to take the device to another level supported by suggestions of the client Flipped Toys.

2 Overview

The client for the project is Demsey Kirkwood of Flipped Toys. Flipped Toys is a startup based in Ottawa that has the goal of developing educational toys powered by open-sourced software. The flagship product of the company is the BusyPad, a fun handheld toy that can suit the needs of many users through variability as its core functionality. The device allows users to switch between educational and recreational games through an open-sourced platform. The goal of the product and the team's contribution to the project is to provide an engaging educational experience through educational games as a learning medium. This aligns with the UNSDGs of Quality Education, ensuring everyone has access to quality education and Decent Work and Economic Growth ensuring that all individuals have the necessary skills to acquire decent jobs.

Though the BusyPad competes in a competitive market with similar products such as the Amazon Fire Kids 10, Game Activity Pad, iPad, etc. the product sets itself apart due to its many key features. Namely the device with its open-sourced capabilities and use of simple COTS components places it in a price bracket that makes it far more affordable to all families. In addition, the variability of the device gives it far more interesting and engaging educational games than competing products. Finally, the device is designed with sustainability in mind, which is an important factor that new customers often consider when choosing between similar products.



Figure 1: BusyPad 3.0 Device

The BusyPad 3.0 prototype solution can be seen in the Figure above. The device is a simple educational gaming system composed of three key areas: Enclosure, Hardware and Software (frontend and backend).

Enclosure:

The enclosure is a simple two piece shell made of PLA filament, manufactured using the 3D printers available at the MakerSpace. The two shells are designed to fit all peripheral components and to include additional space for running wires and the future inclusion of an internal battery. The design, as requested by the client is similar in form to a Simon Says, offering its own identity.

Hardware:

The hardware is based around the use of a Raspberry Pi Zero 2W, acting as the information processor of the device. Connected to the Zero 2W includes many input buttons and an analog joystick alongside a capacitive 4.3' touchscreen display. The hardware is further described later in the report

Software:

The software is robust and includes both the front end and the back end. In brief, the software has been used to develop the companion app for the device, the optimization and functionality of the games, and the communication between the two. The exact walkthrough of the software is further explained in later sections.

2.1 Conventions

The use of the device is straightforward, however a detailed explanation of the set up of the BusyPad 3.0 and its companion app are discussed in detail in the following section of the report. It must be stated here that the instructions provided do NOT call to direct action, however they guide the user through the operation of the device. By following each step carefully there should be no issues encountered by the users. For context, improvement of the usability of the device was a major accomplishment of the current iteration over previous models.

2.2 Cautions & Warnings

If applicable, identify any cautions or warnings that the user should know about before using the system. If waiver use or copy permissions need to be obtained, describe the process.

The device is compact and robust, however dropping it may result in serious damage to internal components, specifically electrical wires that have been soldered together. Users should take caution when handling the device as they would any other handheld electronic. In addition, the device is powered externally, as such, users must take caution when dealing with electricity and live outlets. Though low powered, the current running through the device could result in harm if improperly plugged into an outlet or cable become loose due to strain on the device. The device should NOT be used for reasons outside of its intended purposes.

3 Getting started

3.1 Set-up Considerations

BusyPad 3.0 is an open-source educational device designed to offer a safe, engaging learning experience for children. The setup instructions below explain how to prepare and use the device.

1. Equipment and Power

BusyPad 3.0 is not battery operated. It must be connected to a stable power source via the attached USB cable. To power the system, connect the USB cable to one of the following:

- A standard wall adapter (5V recommended)
- A powered USB port (e.g., on a computer or charging hub)

Note: The device will not operate without being connected to a power source.



Figure 2: BusyPad 3.0 attached power USB cable

2. Internet Connection

A constant and stable internet connection is required. BusyPad 3.0 uses this connection to:

- Load educational content and updates
- Enable interactive learning features

- Sync data and progress

Be sure the device is within range of a working Wi-Fi network.

3. Physical Configuration and Input/Output Devices

BusyPad 3.0 incorporates a user-friendly physical interface that includes:

- Central Touchscreen Display: The main visual output and interactive touch input.
- Color-coded Physical Buttons and Joystick Controls: For navigation and command selection.
- Built-In Speakers: Provide audio cues and instructional feedback.
- USB Cable Connector: For power and system updates.

4. System Configuration

No additional software installations are required from the user. Once the device is powered and connected to the internet, it will automatically display the home screen and guide you through the remaining setup steps.

Connecting to Wi-Fi Using WPA_GUI

WPA_GUI is a graphical interface used to manage Wi-Fi connections on Linux-based systems. It provides an easy way to scan for available networks, connect to them, and configure wireless settings without using the command line. This tool is included with BusyPad 3.0 to simplify the setup process for users.

For more detailed instructions and troubleshooting, you can refer to the official WPA_GUI documentation at:

https://wiki.archlinux.org/title/Wpa_supplicant

If BusyPad 3.0 is not already connected to a Wi-Fi network when it starts, you will need to use the built-in Wi-Fi configuration tool called WPA_GUI. Follow the steps below to connect:

1. When the system boots, open the application labeled 'wpa_gui'.
2. In the WPA_GUI window, make sure the correct adapter is selected (usually 'wlan0').
3. Click the 'Scan' button to search for available Wi-Fi networks.
4. A new window will appear showing available networks. Select your preferred network and click 'Connect'.

5. If prompted, enter the network password (PSK) in the field provided and confirm the encryption method is set to 'CCMP'. Then click 'OK' or 'Save'.
6. Once connected, the 'Current Status' tab will show 'Completed (station)' and display the IP address assigned to the BusyPad.
7. You can now close the WPA_GUI window using the top right close button.

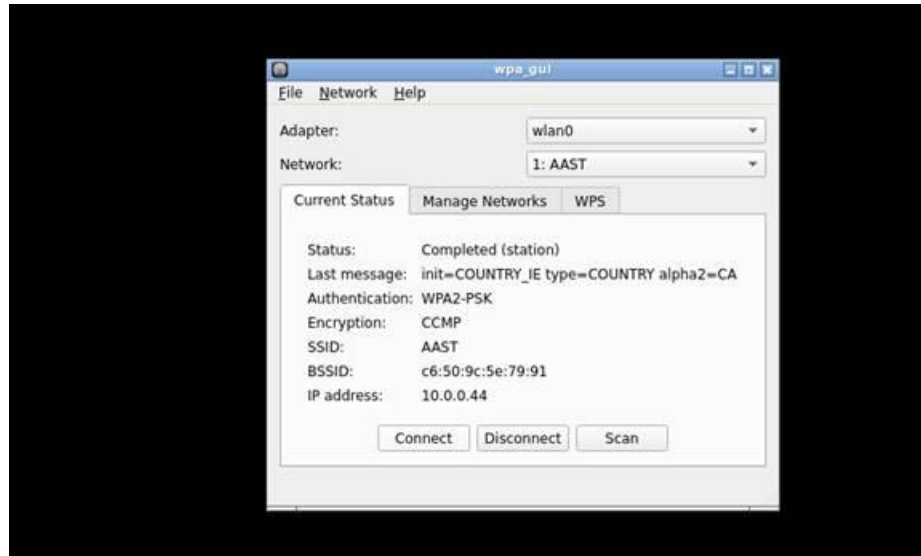


Figure 3: Opening the WPA_GUI application

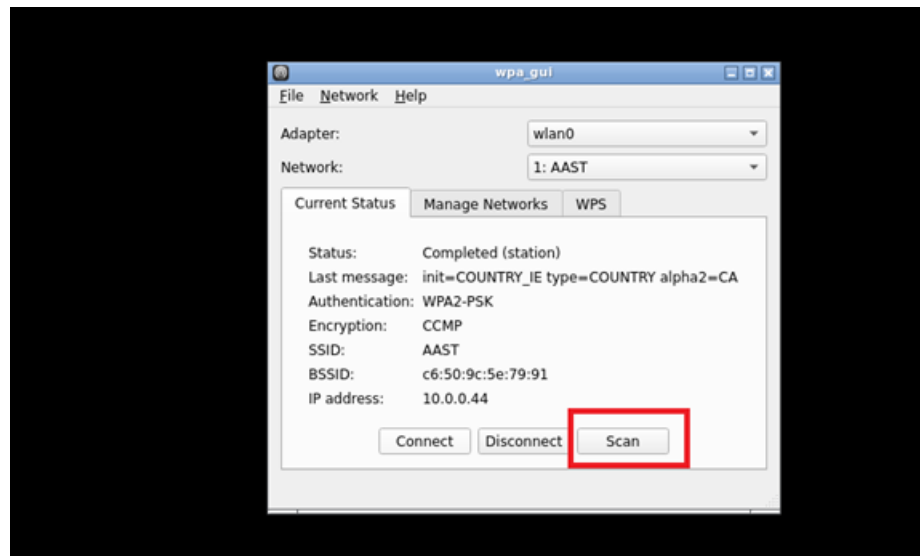


Figure 4: Clicking the 'Scan' button to search for Wi-Fi networks.

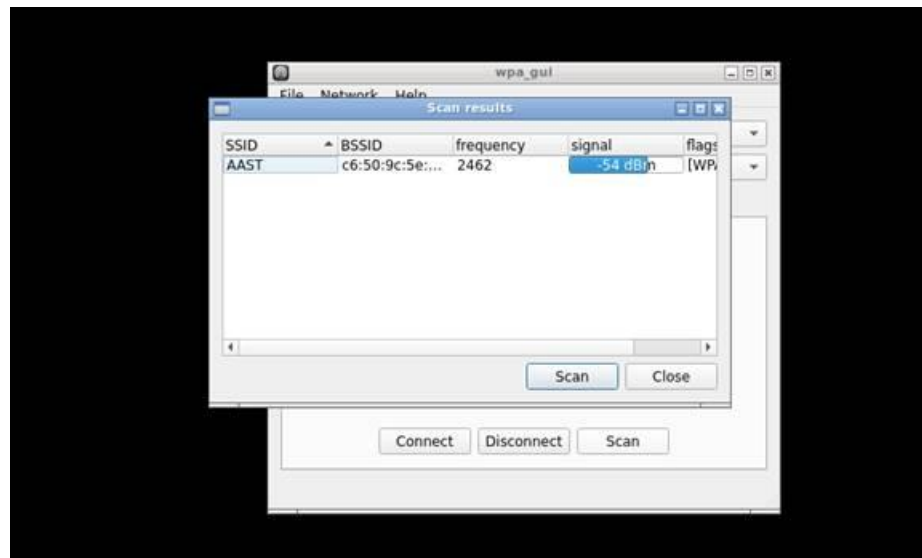


Figure 5: Viewing available networks and signal strength

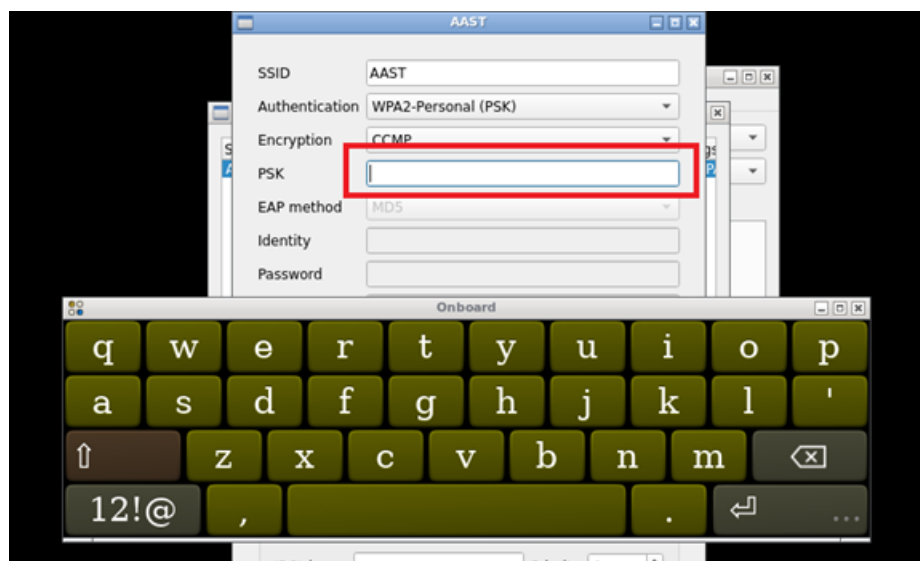


Figure 6: Entering the Wi-Fi password (PSK).

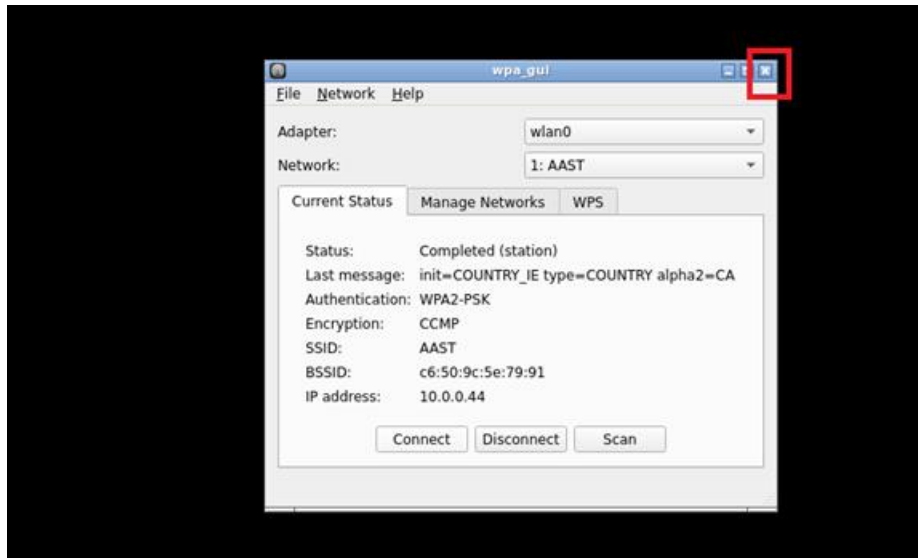


Figure 7: Closing the WPA_GUI window after connection

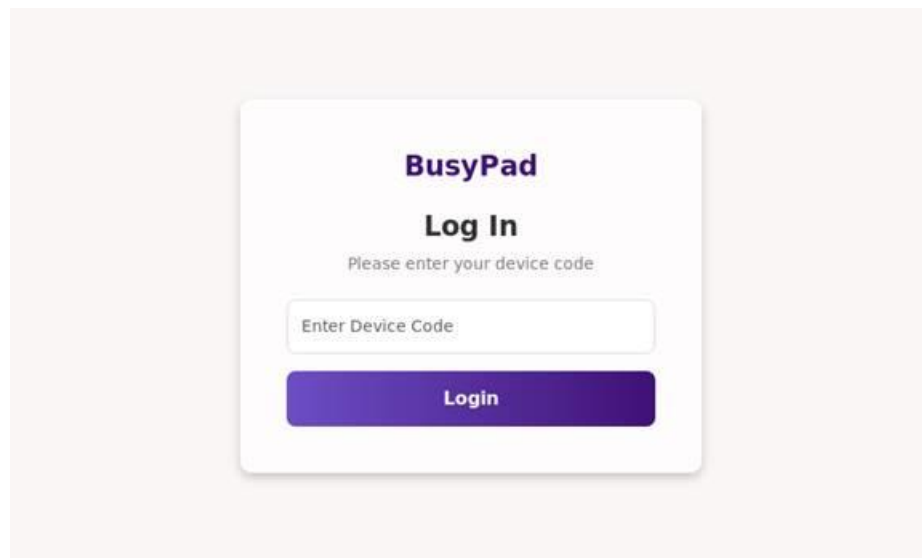


Figure 8: BusyPad login screen asking for device code after network connection

3.2 User Access Considerations

BusyPad 3.0 is designed with two primary access points—the Parent Page and the Player Page—to accommodate different user types and ensure a secure, personalized experience.

1. Parent Page

- **Account Creation:** Parents or guardians create an account with a username and password on the Parent Page.
- **Device Code:** During the account creation process, a device code is generated. This code is required for accessing the Player Page.
- **Parental Controls:** Once logged into the Parent Page, parents can manage various settings, including:
 - Enabling or disabling specific games from appearing on the Player Page.

2. Player Page

- **Access via Device Code:** Child users access the Player Page by entering the device code provided by the parent.
- **Interface for Children:** The Player Page is designed specifically for young learners, displaying age-appropriate games and activities that the parent has enabled.
- **Restricted Permissions:** Child accounts are limited to educational content only and cannot modify system settings or access external websites.

3. Administrator (Device Maintainer)

- **Device Setup and Maintenance:** The administrator (often a parent or guardian) has full rights to update the system, troubleshoot issues, and manage network settings.
- **Authentication:** Access to administrative features requires secure authentication, ensuring that only the designated administrator can make system-wide changes.

This guide ensures that even non-technical users can easily understand the steps to set up BusyPad 3.0, create accounts, and manage the learning environment for children while maintaining safety and control.

3.3 Accessing the System

To access and begin using the BusyPad 3.0 system, follow the instructions below:

1. Power on the device by connecting the USB cable to a wall adapter or USB port.
2. Ensure the device is connected to Wi-Fi. If not already connected, follow the WPA_GUI instructions provided earlier.
3. BusyPad 3.0 has two primary access points: the Parent Page and the Player Page.

Parent Page Access:

- On the login screen, parents must enter their registered email address and password.
- If a parent does not yet have an account, they can click 'Sign Up' to register. The registration form requires:
 - Email address
 - Password and confirmation
 - A device code to link the account to the specific BusyPad unit
- After signing up, parents can log in to manage user settings and parental controls.
- If a parent forgets their password, they can use the 'Forgot Password' link on the login screen to reset it by following the on-screen instructions.

Player Page Access:

- Children access the system using the Player Page by entering the device code provided during parent setup.
- This will load the child-friendly interface with only the enabled games and content.
- Currently, the virtual keyboard functionality is not fully integrated with the application. **As a workaround, users can press right arrow key followed by the 'X' key to automatically input a saved device code.**

The login and registration process is designed to be quick and secure, ensuring that both children and parents can safely access their respective features.

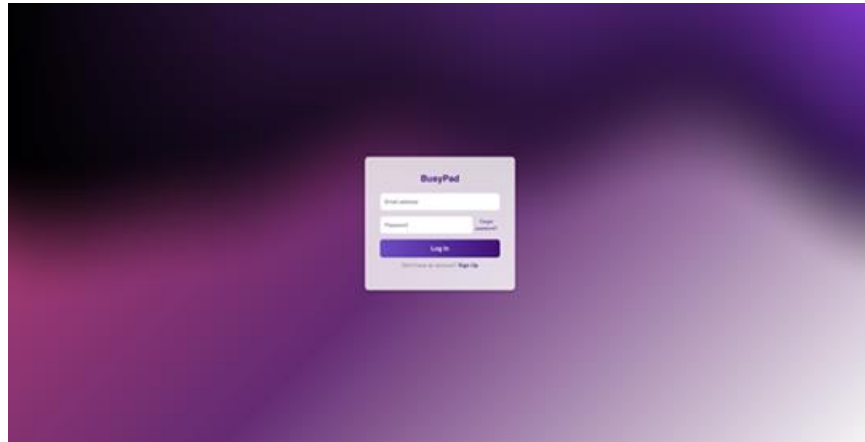


Figure 9: Parent Login Screen: Enter email and password to access parental controls

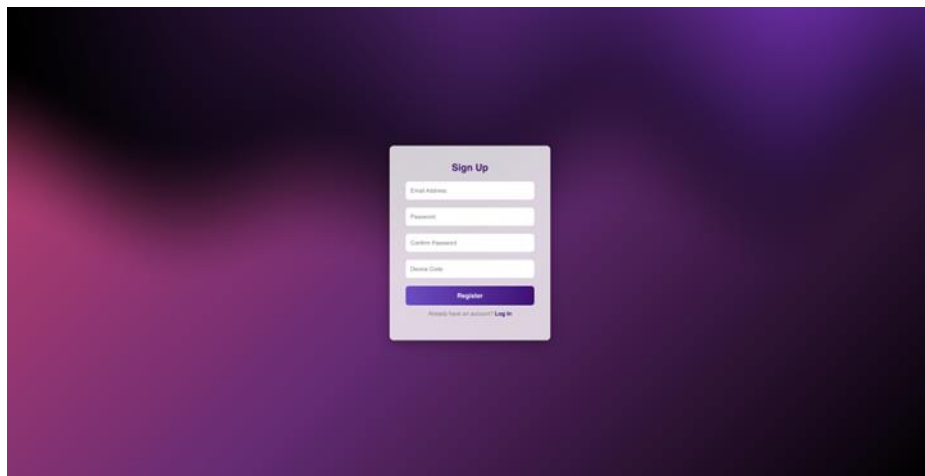


Figure 10: Parent Registration Screen: Sign up with email, password, and device code

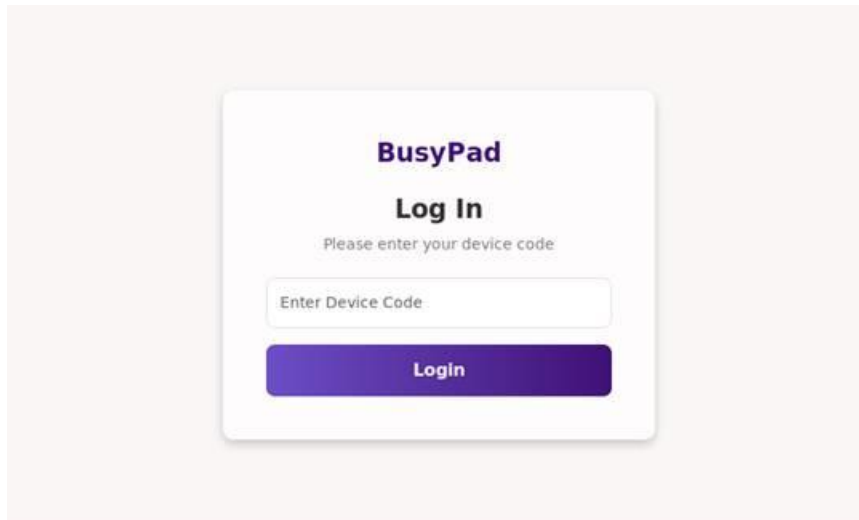


Figure 11: Player Page Login: Enter the device code to access child-friendly content

3.4 System Organization & Navigation

Below is an overview of the system's main pages and how users navigate between key features.

1. Parent Home Page (Parent Account)

Parent accounts are directed to the Home page after logging in, where they can view a list of all available games. Games can be filtered by category (e.g., educational or fun). While admin users cannot play games directly from this page, they can review all content available on the platform.

2. Player Home Page (Player Account)

Players are directed to the Player Home after entering a valid device code. This page displays only the games they are authorized to play. Users can filter games by category and launch games directly in full-screen mode.

3. User Setting – Parental Control

Admin users have access to this page by clicking the user avatar in home page. In this page they can control which games are accessible to specific devices or users. Each game has an on/off toggle switch that allows the admin to enable or disable access as needed.

4. About Page

This page provides background information about BusyPad — including its purpose, how it works, and the benefits it offers to young learners and educators.

5. Log out

Click “Log Out” in the top-right corner of the navigation bar to securely sign out of the system and return to the login page.

3.5 Exiting the System

To properly exit or shut down the BusyPad 3.0 system, follow these steps:

1. From the main interface, navigate to the logout option and select it to safely sign out of the current user session.
2. Once you are logged out, disconnect the device from the power source by unplugging the USB cable from the wall adapter or USB port.

This ensures that the device is safely powered down and ready for future use.

4 Using the System

4.1 Registering an Account

Open the BusyPad parent login page and click the Sign-Up link at the bottom to go to the registration page, which is shown in figure.

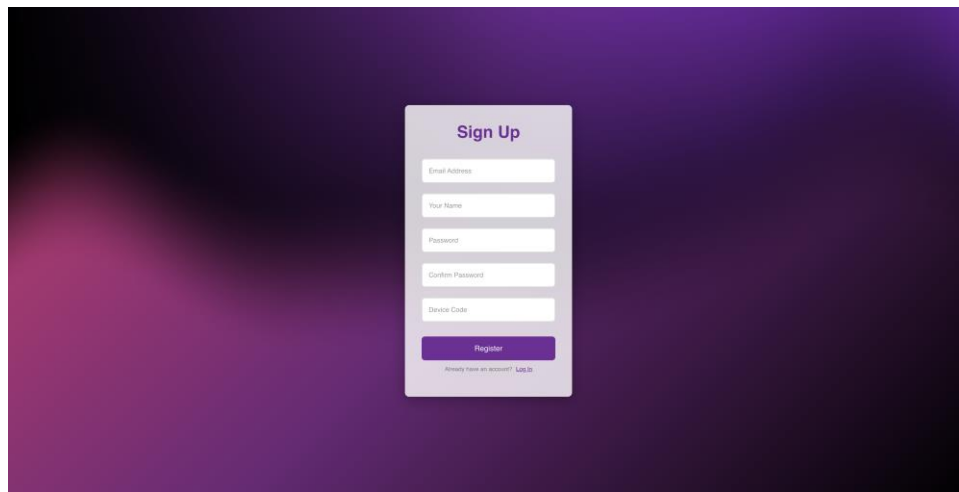


Figure 12: Sign-up Page

There are five input fields on the page, which are:

1. **Email Address:** Enter a valid email address. This will be used for parent account logging in. The system will check if the format is correct (e.g., user@example.com).
2. **Your Name:** Input your display name. This field is required and cannot be left empty.
3. **Password:** Choose a secure password. It must be at least 6 characters long.
4. **Confirm Password:** Re-enter your password to make sure it matches. If the two passwords do not match, an error message will appear.
5. **Device Code:** The Device Code is the unique identifier of the BusyPad device, used for player account login on the device.

All five input fields are required. After filling out all required fields, click the Register button. If there are any issues with your input, the system will display error messages.

4.2 Logging into Your Account

There are two types of login interfaces to support both parents and players. Each type of account has its own purpose and login flow.

- **Parent Account Login**

To log in with a parental account, you need to enter your Email Address and Password. Upon successful login, you will be directed to the Parent Account Home Page. The parent account login page is fully responsive and can be accessed from any device, including mobile phones, tablets, and desktop computers.

- **Player Account Login**

The Player Account must be logged in on the toy device. On the login page, users only need to enter the Device Code. Upon successful login, the system will redirect you to the Player Homepage.

4.3 Parental Control Function

The Parental Control feature allows parents to add or remove games that the corresponding player account can access from their home page.

After logging in, the Parent Account will be directed to the page shown in the figure. In the top-right corner, the user's avatar is displayed. Clicking on the avatar will take the user to the Parental Control page, as shown in the next figure.

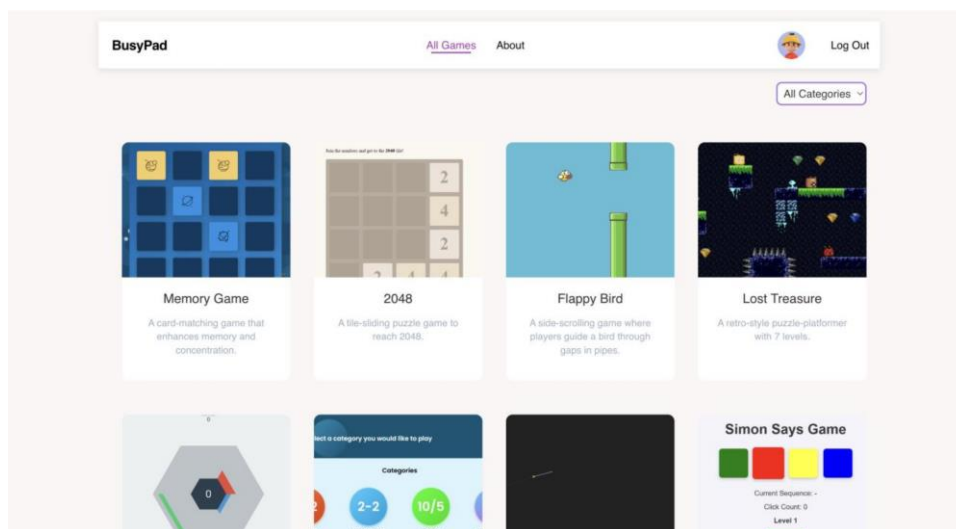


Figure 13: Home Page for Parent Account

On the Parental Control page, all games from the game library are listed. Each game has a toggle switch next to it. By turning the switch on or off and clicking the Save Changes button, the parent can control whether the game is accessible or restricted on the player's home page.

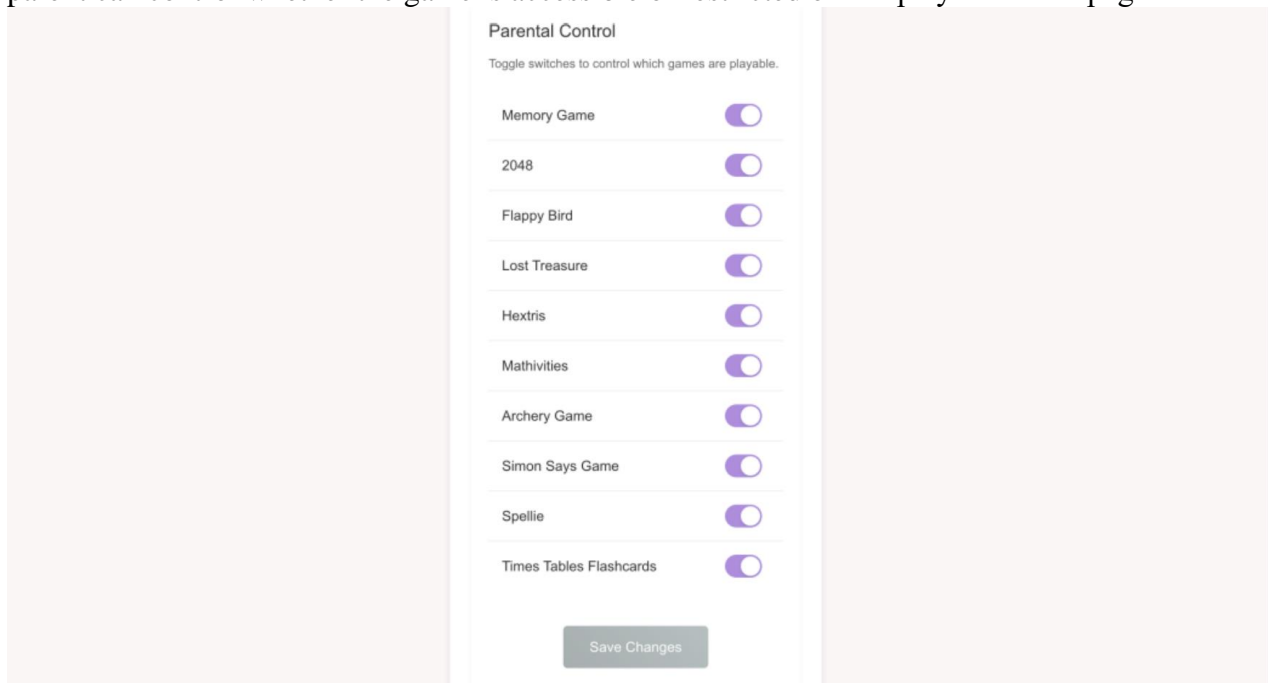


Figure 14: Parental Control Page

4.4 Playing Games

After logging in on the toy device, the Player Account will be directed to the Player Homepage. This page displays only the games that the Parent Account has allowed access to through the Parental Control settings.

- **Playing a Game**

After logging in, the Player user is directed to the homepage, as shown in the figure. On this page, each game is displayed as a card containing the game image, title, and description. Below each game card, there is a "PLAY NOW" button.

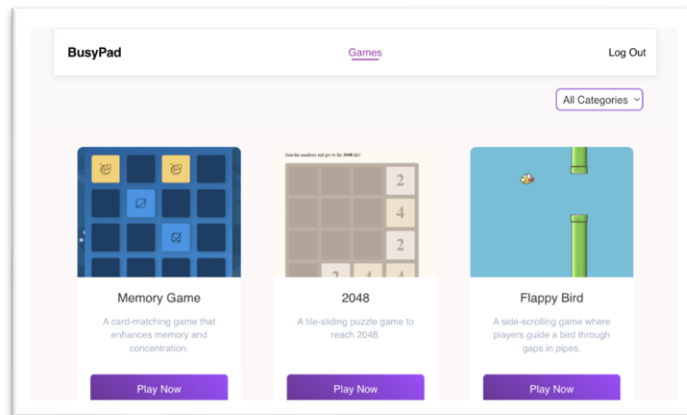


Figure 15: Home Page for Player Account

Clicking the “PLAY NOW” button will take the user to the corresponding game page, as shown in the next figure. The player can use buttons, a joystick, or the touch screen on the toy device to play games.



Figure 16: Playing Game Page

Click “Back to Games” on the game page to go back to the Player’s game list.

- **Filtering Games by Category**

On the Player Homepage, there is a filter dropdown located at the top-right corner of the screen. The filter includes three options: All, Educational, and Fun.

By selecting one of these options, the player can view games that belong to the corresponding category.

5 Troubleshooting & Support

5.1 Error Messages or Behaviors

- If the device fails to connect to Wi-Fi or the on-screen virtual keyboard does not appear when needed, the recommended first step is to restart the system.
- Inconsistent behavior in loading the Player or Parent login screens may be due to poor or lost internet connection.
- If games fail to load or buttons do not respond, ensure that the internet connection is active and the USB power cable is securely connected.
- If the screen remains black after powering on, check the power source and ensure the USB cable is not damaged.

5.2 Special Considerations

Currently, the virtual keyboard functionality is not fully integrated with the application. As a workaround, users can press any arrow key followed by the 'X' key to automatically input a saved device code.

All BusyPad applications and content are hosted on the internet. A reliable and continuous internet connection is essential for accessing games, logging in, and syncing progress. If the system is offline, key features will not be available.

5.3 Maintenance

- Ensure the device is stored in a dry, dust-free environment.
- Clean the screen and buttons gently with a soft, dry cloth.
- Periodically check for firmware or application updates (if update notifications are enabled).
- Verify the USB cable and buttons are functioning properly; replace any damaged hardware as needed.

5.4 Support

If you experience an issue that cannot be resolved through a system restart or checking hardware connections, you may request assistance.

Please contact **Mr. Mahmoud Mohamed** at mmoha409@uottawa.ca

When requesting support:

- Describe the issue clearly
- Include a picture of the problem, if possible
- Mention any error messages or behaviors you observed

This will help the support team provide you with the fastest and most effective assistance.

6 Product Documentation

6.1 Hardware System

- BOM (Bill of Materials)**

Table 2: Bill of Materials

Item #	Code/SKU	Product Name	Unit Cost	Quantity	Total Cost
1	283	Male to Femal Jumper Cables x40	\$ 2.95	1	\$ 2.95
2	2	MCP3008 - 8-channel 10-bit ADC with SPI Interface	\$ 5.95	1	\$ 5.95
3	802	Mini HDMI Plug to Standard HDMI Jack Adapter	\$ 3.45	1	\$ 3.45
4	1019	MicroSD Card - 64 GB - Class - 10	\$ 12.95	1	\$ 12.95
5	CS_PID-3	Mini-HDMI to HDMI cable - Gold Plated - 3Ft	\$ 3.95	1	\$ 3.95
6	1675	Analog 2-axis Thumb Joystick with Select Button	\$ 2.95	1	\$ 2.95
7	505-1	Breadboard Wiring Kit	\$ 8.45	1	\$ 8.45
8	1527-1	Colorful Round Tactile Button Switch Assortment x 15	\$ 8.95	1	\$ 8.95
9	112-1	Raspberry Pi Zero 2 W with Header	\$ 25.45	1	\$ 25.45
10	N/A	Breadboard	\$ -	1	\$ -
11	N/A	4.3 in HDMI LCD 800x480 IPS Capacative Touc Screen	\$ 64.99	1	\$ 64.99
			Total Before Tax:	\$	140.04
			Total Tax:	\$	18.45
			Shipping:	\$	2.00
			Grand Total	\$	160.49

- Equipment list**

1. Soldering Station
2. Wire Cutters
3. Hot Glue Gun
4. Electrical Tape

5. Heat shrink Tubing
6. SSH Client (PuTTY)

- **Instructions**

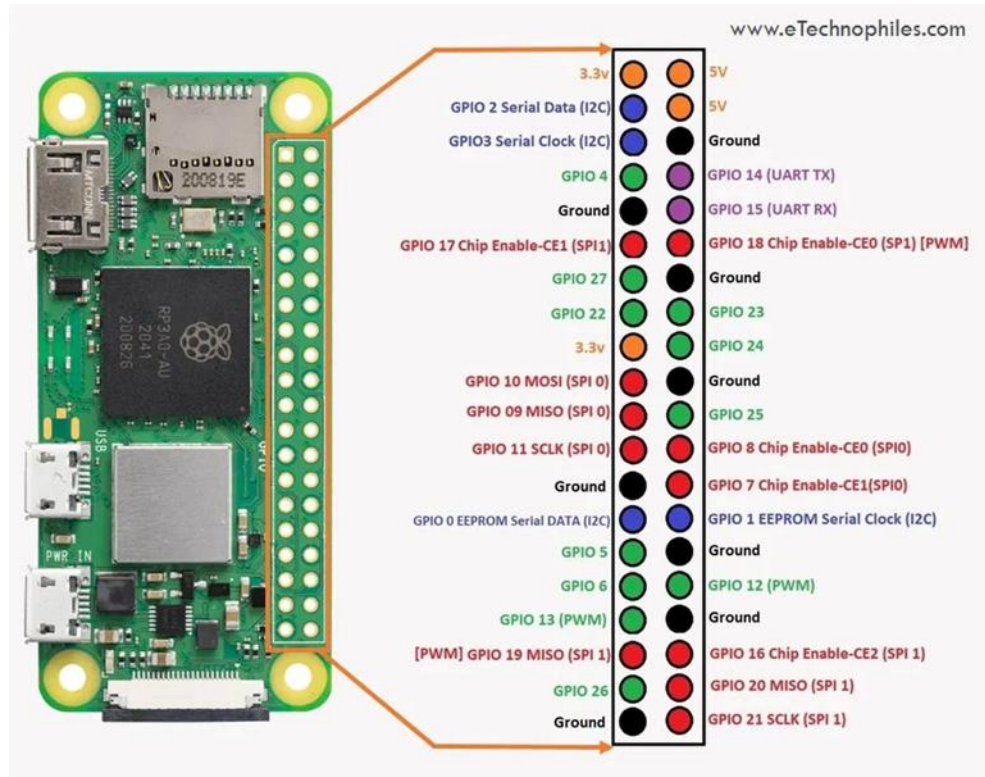


Figure 17: Raspberry Pi Zero 2W Pin Out

The system's hardware components are arranged in a structured manner to ensure efficient signal flow, proper grounding, and readability. From the previous prototype to the current iteration, hardware and electrical connections remain unchanged. This is due to the chosen components working effectively to meet our requirements. For future iterations a custom built PCB and internal power system would be the likely next steps. In addition, design for the implementation of additional input methods or sensors may be considered. This will increase the

longevity of the device and the diversity of available games. The primary components and their connections include:

- External Input Power (5V, 2A) → Supplies power to Raspberry Pi Zero 2 W.
- Raspberry Pi Zero 2 W → Manages GPIO connections and overall processing.
- 4.3 in. Capacitive Touchscreen
- Mini HDMI to HDMI Adapter → Connects Raspberry Pi to the capacitive touchscreen display.

Control Inputs

- Joystick:
 - X-axis → ADC → GPIO (e.g., GPIO17)
 - Y-axis → ADC → GPIO (e.g., GPIO27)
- Buttons:
 - Button 1 → GPIO (e.g., GPIO5)
 - Button 2 → GPIO (e.g., GPIO6)
 - Button 3 → GPIO (e.g., GPIO7)
 - Button 4 → GPIO (e.g., GPIO8)
 - Button 5 → GPIO (e.g., GPIO9)
 - Button 6 → GPIO (e.g., GPIO10)
 - Button 7 → GPIO (e.g., GPIO11)
 - Button 8 → GPIO (e.g., GPIO12)

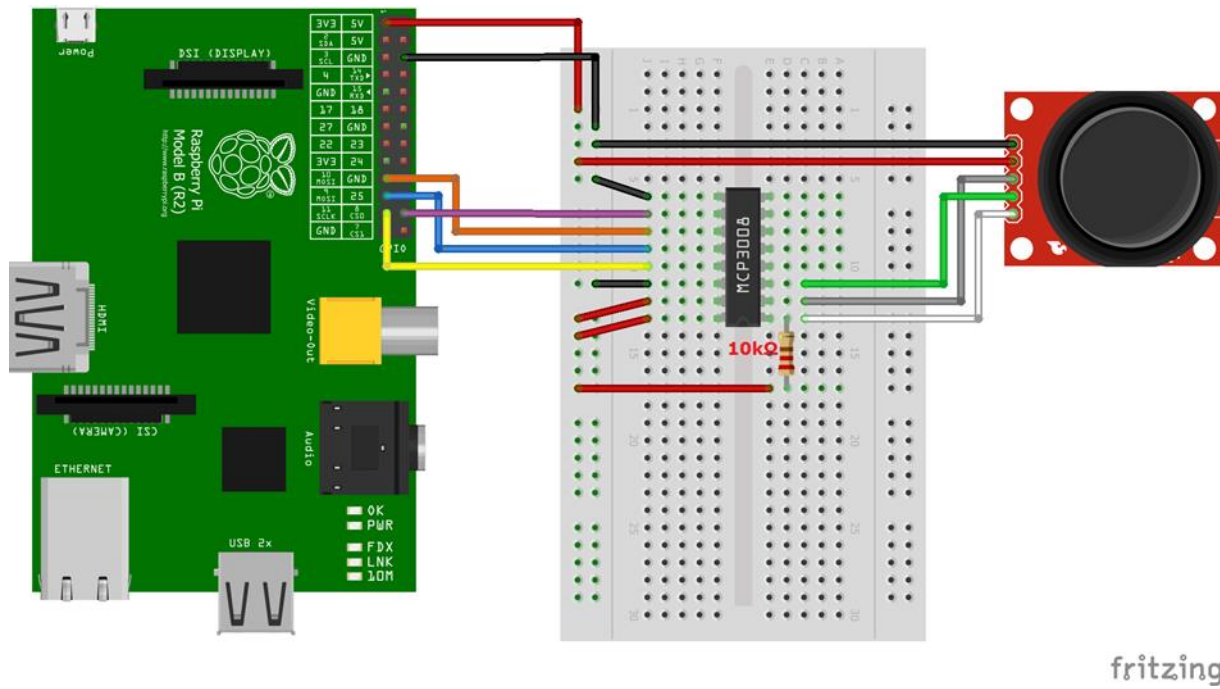


Figure 18: Joystick Connection

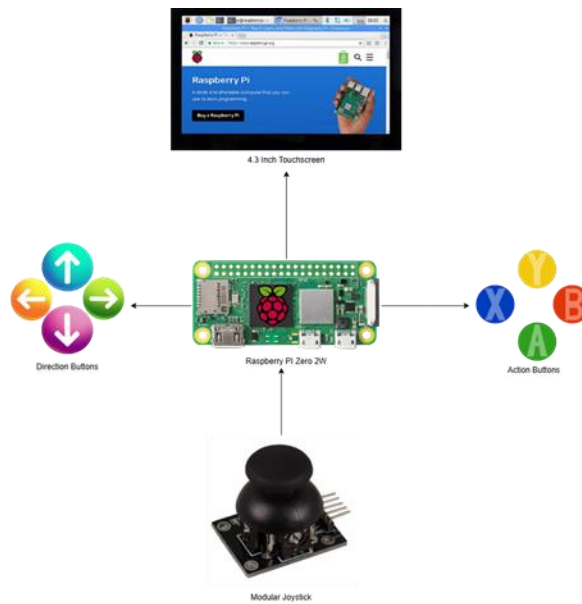


Figure 19: Final Prototype Component Layout

Dedicated ground connections for ADC, buttons, and joystick to ensure stable operation and minimize electrical noise. The layout is further detailed in the accompanying flow diagram for all the major electrical connections:

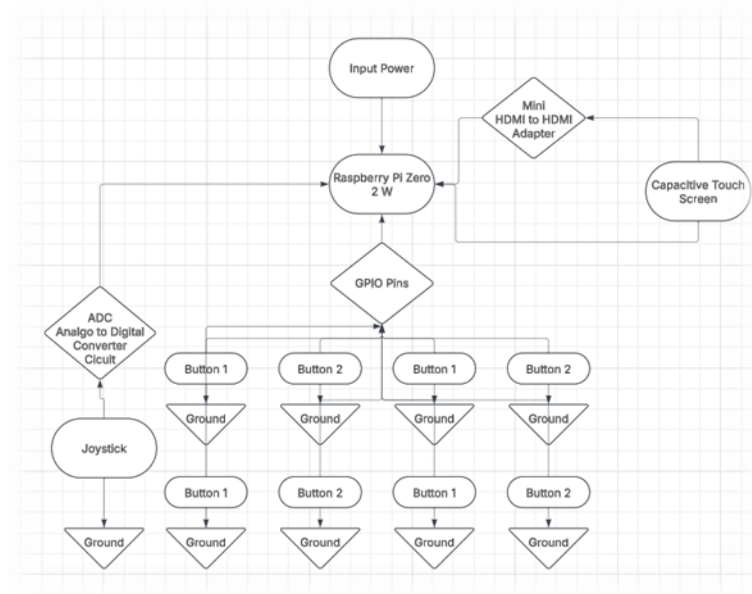


Figure 20: Electrical Decomposition

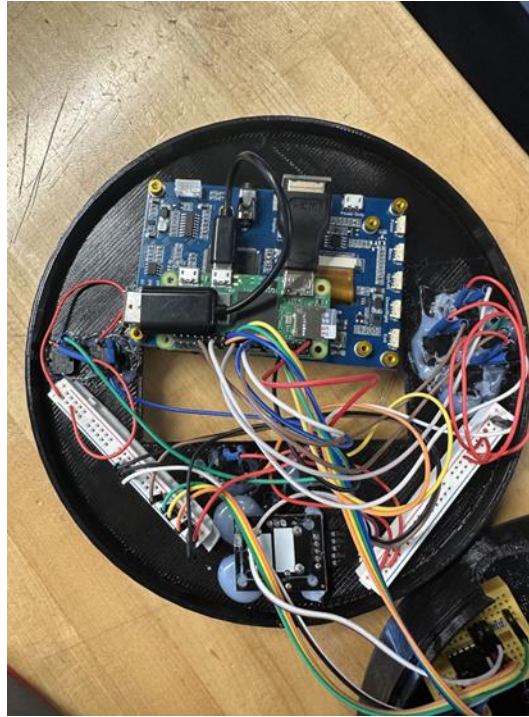


Figure 21: Final Prototype Hardware Implementation

The Raspberry Pi Zero 2 W is equipped with multiple input controls, including 8 buttons, a joystick, and touchscreen to facilitate user interaction. Each button is directly connected to the Raspberry Pi's General-Purpose Input/Output (GPIO) pins and configured as keyboard inputs through the Pi's settings. One of these buttons executes a script that enables backward navigation in a web browser. Each button features a pull-down resistor configuration, ensuring proper signal readings and avoiding floating GPIO states. The wiring structure includes:

- One end of each button is wired to a dedicated GPIO pin.
- The other end is connected to the ground to complete the circuit.

To maintain system stability and ensure reliable performance, the following considerations have been implemented:

- Proper Grounding: Ensures signal integrity and prevents floating states in GPIO connections.

- **Pull-down Resistors:** Used on buttons to avoid unintended signal activation.
- **Voltage Compatibility:** Ensures ADC voltage levels are compatible with the Raspberry Pi's operating range.
- **Signal Noise Reduction:** Shielded cables are recommended for analog signal connections to minimize interference.

This structured configuration and detailed layout provide a well-organized and professional approach to designing an interactive Raspberry Pi-based control system.

Setting Up the Raspberry Pi Zero 2W for Kiosk Mode:

Step 1: Install Raspberry Pi OS Lite (Legacy 32-bit Bullseye)

1. Download **Raspberry Pi OS Lite (Legacy 32-bit Bullseye)** from the official Raspberry Pi website.
2. Use **Raspberry Pi Imager** to flash the OS onto an SD card.
3. Before writing the image, configure the following settings in Raspberry Pi Imager:
 - a. **Set a username and password** for SSH access.
 - b. Enable SSH from Services
 - c. **Enter Wi-Fi credentials** (SSID and password) to enable network access after installation.

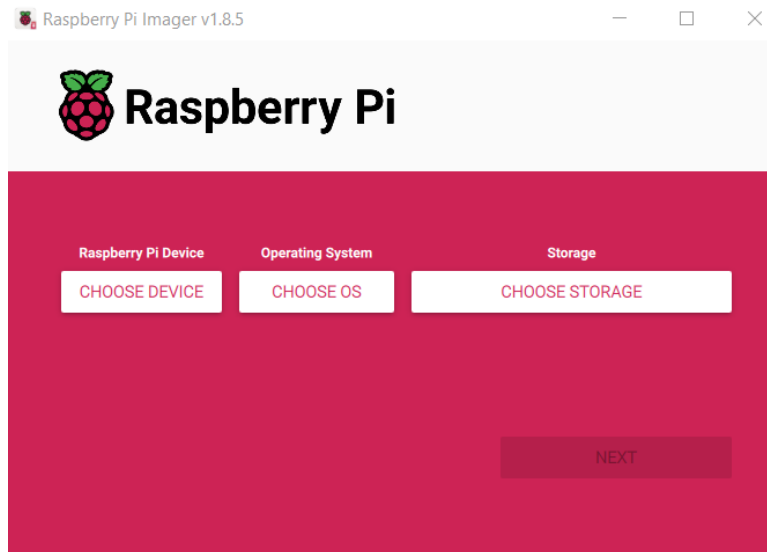


Figure 22: Pi Imager Interface

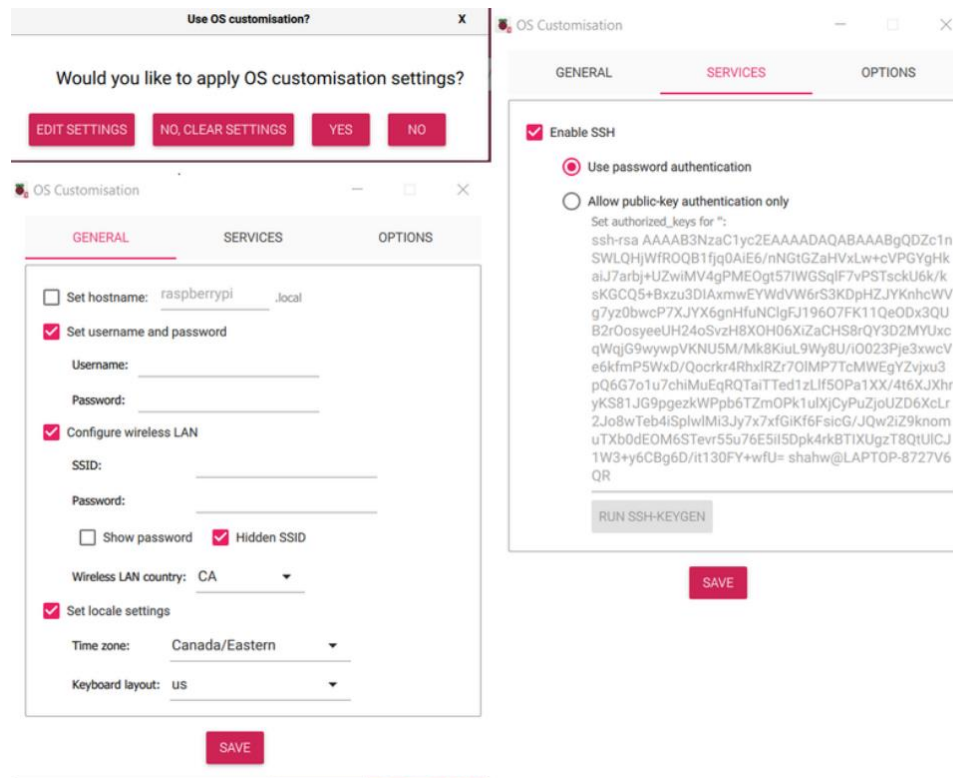


Figure 23: Pi Imager Interface

4. Download PuTTY SSH client

- a. Check the router page to identify Raspberry Pi IP and Make sure you remember the IP
- b. Put Raspberry Pi IP in the Hostname field and press open

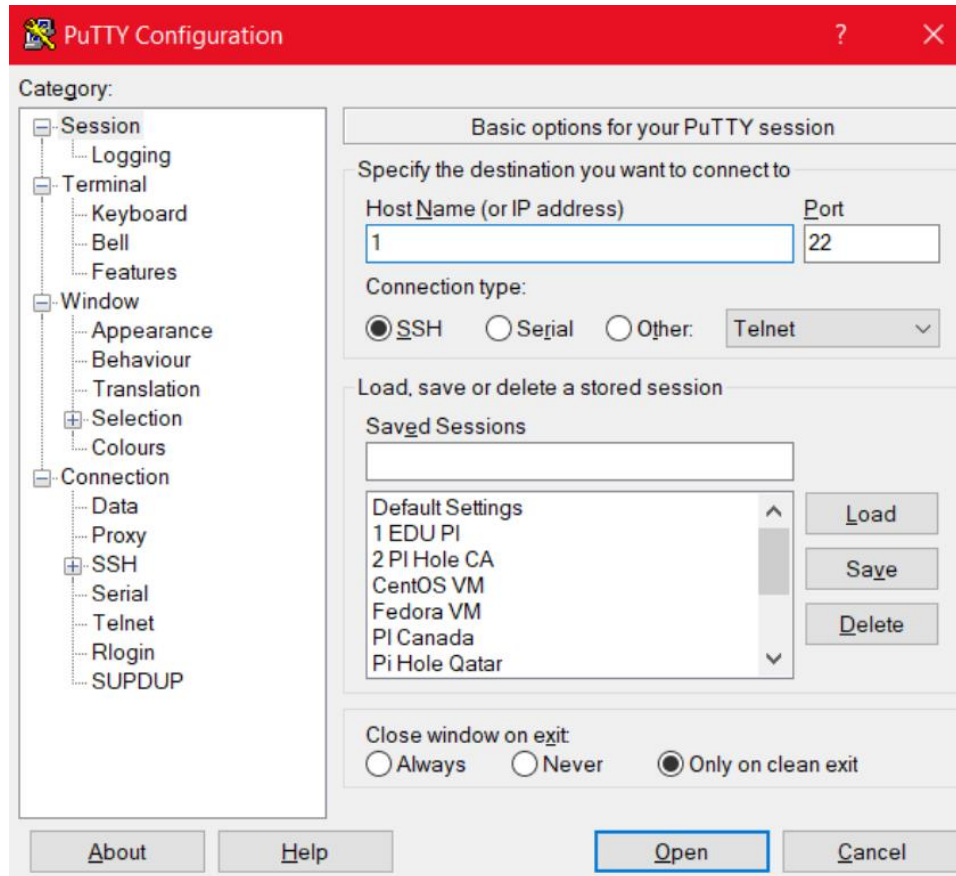
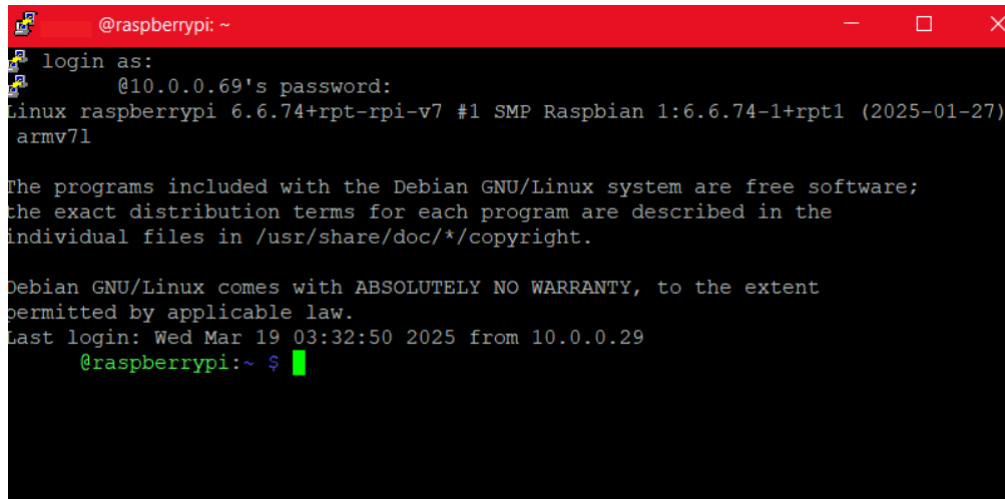


Figure 24: PuTTY SSH Client Interface



```
@raspberrypi: ~
login as:
@10.0.0.69's password:
Linux raspberrypi 6.6.74+rpt-rpi-v7 #1 SMP Raspbian 1:6.6.74-1+rpt1 (2025-01-27)
armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar 19 03:32:50 2025 from 10.0.0.29
@raspberrypi:~ $
```

Figure 25: Raspberry Pi CLI

Step 2: Install Pi-Kiosk

1. Clone the **Pi-Kiosk GitHub repository** and install the kiosk application:

```
sudo apt install git -y
```

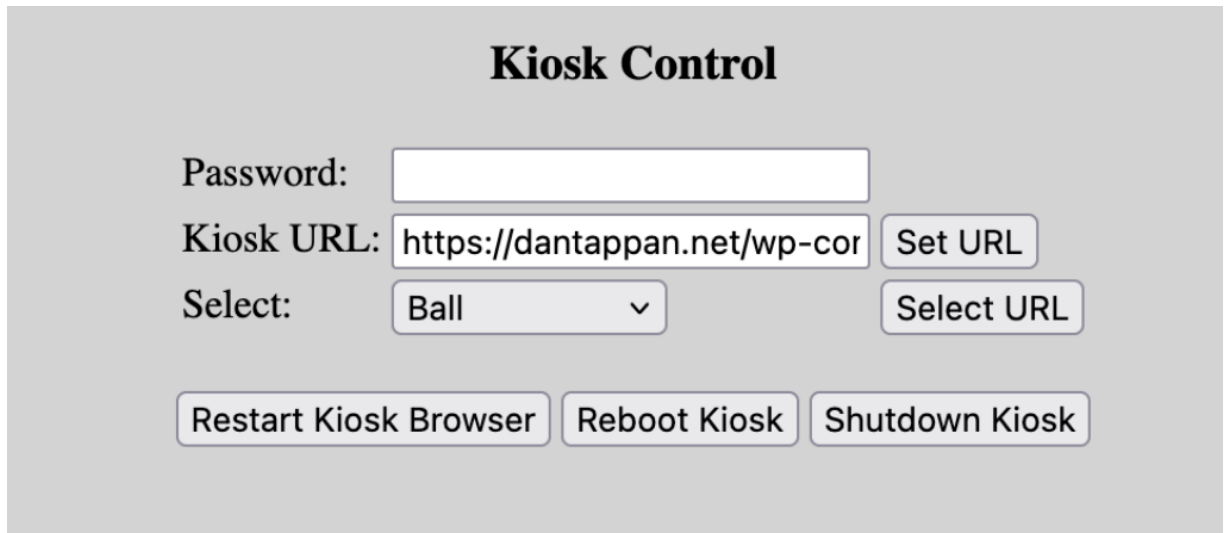
```
git clone https://github.com/DanTappan/Pi-Kiosk
```

```
cd Pi-Kiosk
```

```
./install.sh [ --browser browser ]
```

2. The **Pi-Kiosk installer** automatically configures **Openbox**, a lightweight window manager, instead of a full desktop environment to minimize resource usage.
3. During installation, you can choose to install **Midori**, a lightweight browser capable of running JavaScript applications.
4. At the end of the installation, **set a password** to access the kiosk control panel.

5. Access the kiosk control panel at the Raspberry Pi's IP on your local network to set the URL of the app



The image shows a web interface titled "Kiosk Control" on a light gray background. It contains several input fields and buttons. At the top, the title "Kiosk Control" is centered in a bold, black, sans-serif font. Below the title, there are three rows of controls. The first row has the label "Password:" followed by a white text input field. The second row has the label "Kiosk URL:" followed by a white text input field containing the URL "https://dantappan.net/wp-cor" and a button labeled "Set URL". The third row has the label "Select:" followed by a dropdown menu showing "Ball" with a downward arrow, and a button labeled "Select URL". At the bottom of the interface, there are three buttons: "Restart Kiosk Browser", "Reboot Kiosk", and "Shutdown Kiosk".

Figure 26: Kiosk Control Panel

Step 3: Configure Automatic Browser Launch and Network Check on Boot

Using the Openbox autostart File

- The `/etc/xdg/openbox/autostart` file is responsible for running startup scripts when the Openbox window manager starts. This allows us to:
- Disable screen blanking and power management to keep the screen active.
- Run the network-check script to detect internet connectivity and prompt the user with Wi-Fi settings if needed.
- Launch the browser in fullscreen mode upon startup.
- Enable a physical back button for navigation.

Editing the Autostart File:

1. Open the autostart file in a text editor:
`sudo nano /etc/xdg/openbox/autostart`

2. Modify the file to include the following:

```
# Disable any form of screen saver / screen blanking / power management
xset s off
xset s noblank
xset -dpms
# Allow quitting the X server with CTRL-ALT-Backspace
setxkbmap -option terminate:ctrl_alt_bksp
# Change directory to the user's home folder
cd /home/edu
# Start the virtual keyboard in the background
onboard &
# Run the network check script to detect Wi-Fi status
./network-check.sh &
# Enable the physical back button functionality
./midori_back2.py &
# Set up port forwarding from port 80 to 8000 for the kiosk control page
sudo iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8000
# Start the kiosk browser application in an infinite loop to restart it if it crashes
while true; do
    # Launch the kiosk script
    ./kiosk.py
done
```

3. Save the file (CTRL + X, then Y, then Enter).

Step 4: Automate Wi-Fi Setup if No Internet is Detected

Network Check Script (network-check.sh)

This script **continuously checks for an internet connection** and launches the Wi-Fi configuration tool (**wpa_gui**) if no network is available.

1. Open the script file for editing:

```
sudo nano /home/edu/network-check.sh
```

2. Add the following script:

```
#!/bin/bash
while true; do
    # Check if Google is reachable
    if ping -c1 google.com >/dev/null 2>&1; then
        # Kill wpa_gui and onboard if internet is available
        pkill wpa_gui
        pkill onboard
        exit 0
    else
        # Start wpa_gui if not already running
```

```

if ! pgrep -x "wpa_gui" >/dev/null; then
    wpa_gui &
fi
fi
# Wait 5 seconds before retrying
sleep 5
done

```

3. Save the file and make it executable:

```
chmod +x /home/edu/network-check.sh
```

This ensures that the Wi-Fi setup tool will launch only if the Raspberry Pi does not have internet access.

Step 5: Configure GPIO Buttons as Keyboard Inputs

To allow physical buttons to function as keyboard keys, modify the **/boot/config.txt** file.

1. Open the file for editing:

```
sudo nano /boot/config.txt
```

2. Add the following lines to configure **GPIO pins** as keyboard inputs:

```

# Setup Arrow Keys
dtoverlay=gpio-key,gpio=12,active_low=1,gpio_pull=up,keycode=105 # Left Arrow
dtoverlay=gpio-key,gpio=1,active_low=1,gpio_pull=up,keycode=108 # Down Arrow
dtoverlay=gpio-key,gpio=25,active_low=1,gpio_pull=up,keycode=103 # Up Arrow
dtoverlay=gpio-key,gpio=24,active_low=1,gpio_pull=up,keycode=106 # Right Arrow
# Setup Control Keys
dtoverlay=gpio-key,gpio=23,active_low=1,gpio_pull=up,keycode=57 # Spacebar
dtoverlay=gpio-key,gpio=18,active_low=1,gpio_pull=up,keycode=45 # X Key
dtoverlay=gpio-key,gpio=15,active_low=1,gpio_pull=up,keycode=63 # F5 Key

```

3. Save the file and reboot the Raspberry Pi:

```
sudo reboot
```

Step 6: Enable a Physical Back Button for Navigation

The **back button script** (**midori_back2.py**) allows a physical button connected to **GPIO 14** to act as a "Back" button in the browser.

1. Open the script file for editing:

```
sudo nano /home/edu/goBack/midori_back2.py
```

2. Add the following Python code:

```
#!/usr/bin/env python3
from gpiozero import Button
import os
button = Button(14, pull_up=True)
def go_back():
    # Send Alt+Left to navigate back
    os.system('midori -e tab-next')
    os.system('midori -e tab-previous')
    os.system('midori -e go-back')
button.when_pressed = go_back
while True:
    button.wait_for_press()
```

3. Save the file and make it executable:

```
chmod +x /home/edu/goBack/ midori_back2.py
```

This script listens for a button press and **sends a "Back" command** to the Midori browser.

Step 7: Setting Up a Joystick

1. Enable SPI for Joystick Support

a. Open the Raspberry Pi configuration tool:

```
sudo raspi-config
```

b. Navigate to **Interface Options** → **SPI** and **enable SPI**.

2. Install Required Python Libraries

a. Install Python and the spidev library to communicate with the joystick via SPI:

```
sudo apt install python3 python3-pip -y
```

```
pip install spidev
```

3. Create and Run the Joystick Script

a. Open a new script file:

```
sudo nano /home/edu/joystick/arrow-joystick.py
```

b. Add the following code:

```
#!/usr/bin/python
import spidev
import os
import time
```

```

import subprocess

# SPI setup
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1000000
# Define Axis Channels
swt_channel = 0
vr_x_channel = 1
vr_y_channel = 2
# Time delay for reading values
delay = 0.1
# Joystick thresholds
threshold_high = 700
threshold_low = 300
# Function for reading the MCP3008 channel
def readChannel(channel):
    val = spi.xfer2([1, (8+channel) << 4, 0])
    data = ((val[1] & 3) << 8) + val[2]
    return data
# Track current state
current_keys = set()
# Endless loop
while True:
    # Determine position
    vr_x_pos = readChannel(vr_x_channel)
    vr_y_pos = readChannel(vr_y_channel)
    swt_val = readChannel(swt_channel)

    # Determine which keys should be pressed
    keys_to_press = set()

    # X-axis (left/right)
    if vr_x_pos > threshold_high:
        keys_to_press.add("Right")
    elif vr_x_pos < threshold_low:
        keys_to_press.add("Left")

    # Y-axis (up/down)
    if vr_y_pos > threshold_high:
        keys_to_press.add("Down")
    elif vr_y_pos < threshold_low:
        keys_to_press.add("Up")

    # Release keys that are no longer being pressed
    for key in current_keys - keys_to_press:
        subprocess.run(["xdotool", "keyup", key])

    # Press keys that aren't already pressed
    for key in keys_to_press - current_keys:

```

```

        subprocess.run(["xdotool", "keydown", key])

# Update current keys
current_keys = keys_to_press

# Output for debugging
print("VRx: {} VRy: {} SW: {} Keys: {}".format(
    vr_x_pos, vr_y_pos, swt_val, list(current_keys)
))

# Wait
time.sleep(delay)
swt_val = readChannel(swt_channel)

# Determine which keys should be pressed
keys_to_press = set()

# X-axis (left/right)
if vr_x_pos > threshold_high:
    keys_to_press.add("Right")
elif vr_x_pos < threshold_low:
    keys_to_press.add("Left")

# Y-axis (up/down)
if vr_y_pos > threshold_high:
    keys_to_press.add("Down")
elif vr_y_pos < threshold_low:
    keys_to_press.add("Up")

# Release keys that are no longer being pressed
for key in current_keys - keys_to_press:
    subprocess.run(["xdotool", "keyup", key])

# Press keys that aren't already pressed
for key in keys_to_press - current_keys:
    subprocess.run(["xdotool", "keydown", key])

# Update current keys
current_keys = keys_to_press

# Output for debugging
print("VRx: {} VRy: {} SW: {} Keys: {}".format(
    vr_x_pos, vr_y_pos, swt_val, list(current_keys)
))

# Wait
time.sleep(delay)

```

- c. Save and make it executable:
`chmod +x /home/edu/joystick/arrow-joystick.py`
- d. Run the script:
`sudo python3 /home/edu/joystick/joystick.py`
- e. Add this line to `/etc/xdg/openbox/autostart`
`cd /home/edu/joystick`
`sudo ./arrow-joystick.py`

6.2 3D Design

The physical housing of the BusyPad 3.0 prototype is a two-piece solid shell made of PLA filament with layer height set to 0.4mm to ensure acceptable precision in the produced piece. The set parameters and rendering of the component for 3D printing in the CURA software can be seen in the following figure:

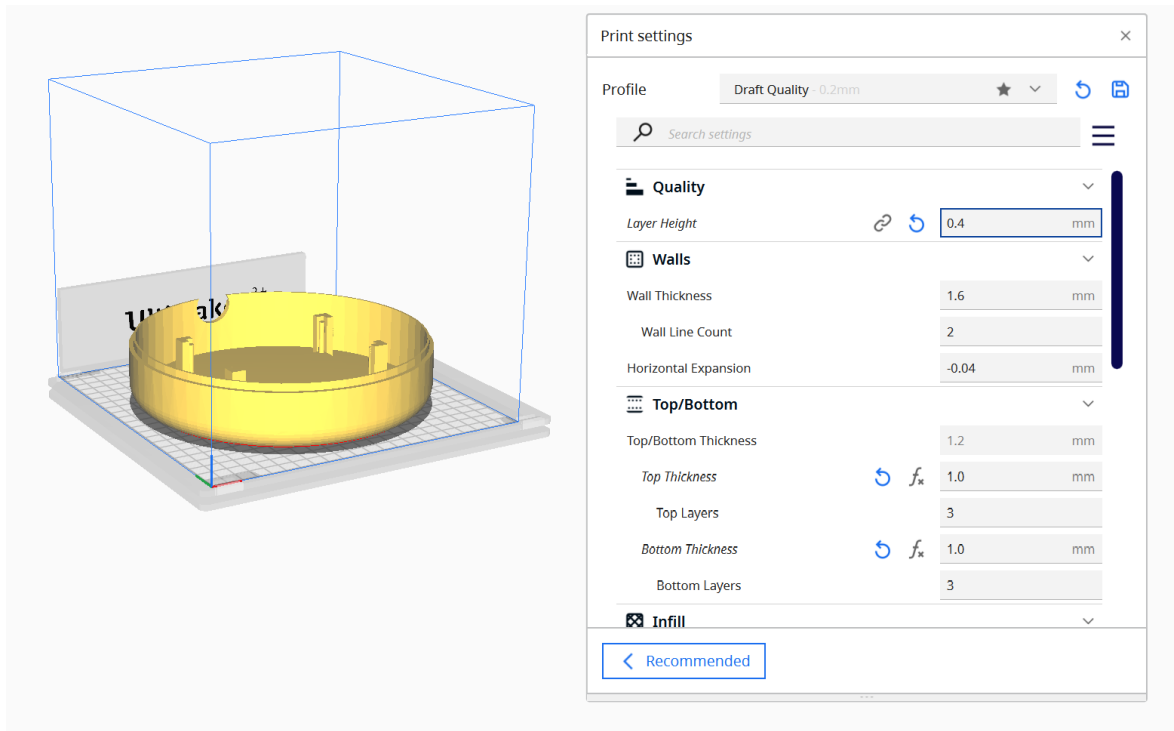


Figure 27: CURA 3D Design Parameters

It is suggested in future iterations to use alternative manufacturing processes or the use of different filaments to provide greater detail (especially with the incorporation of rounded fillets) and to promote sustainability. For example, resin molding may be the optimal solution as it will allow production of multiple batches of BusyPad enclosures at one time. It must be noted that the client would have been satisfied with an enclosure made of light wood through laser cutting, however as mentioned the rapid prototyping was necessary for ensuring all tolerances were met.

The top shell is a thin curved piece with a height of 10 mm and depth of 8mm. It includes extruded cuts to allow for opening and attachment of the eight buttons, swappable joystick, and capacitive touchscreen display. The previous version had 4 auxiliary buttons underneath the screen and d-pad installation location. Changes were made to have the 4 buttons on the left side perform the d-pad function (up/down, left/right) to simplify the design. Removal of 2 buttons on the right side was done to limit unnecessary components. Another change was to increase the overall size of

the prototype, changing the diameter from 180mm to 200mm. This necessary change gives more space in the enclosure to fit all necessary electronics and cables.

The bottom plate is a thicker shell at 40mm height and 35mm depth. It incorporates an offset ridge to allow attachment between the two sections with a thickness set to 1.5mm. A raised shelf is built into the base with 4 circular pegs to support the screen (or in the future, incorporation of a flat lithium-ion battery). A drilled cutout of ½” was added for the external power cable. This iteration incorporates a through hole to fit the cable dimensions. The CAD assembly of the BusyPad 3.0 is on the next figure:

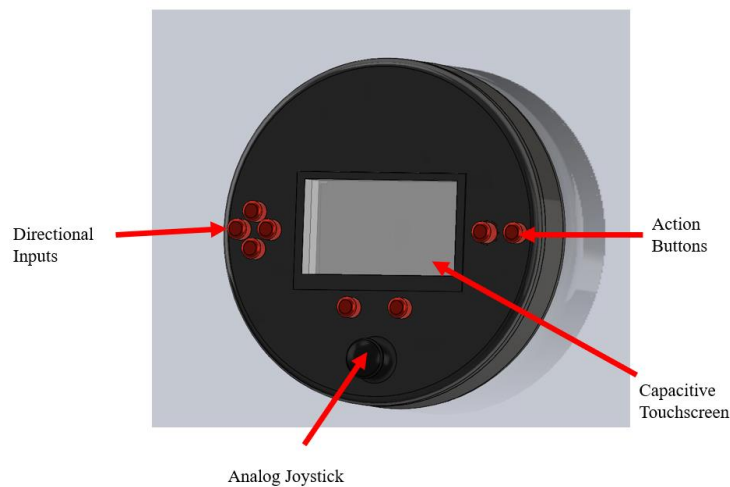


Figure 28: BusyPad 3.0 CAD Assembly

The initial prototype attempted to use PLA screws to attach both shells, however this did not work so a simple resting design that makes use of gravity to secure both components will be used for the updated prototype. This way the top shell can be easily removed for troubleshooting the electrical components or mechanical input methods.

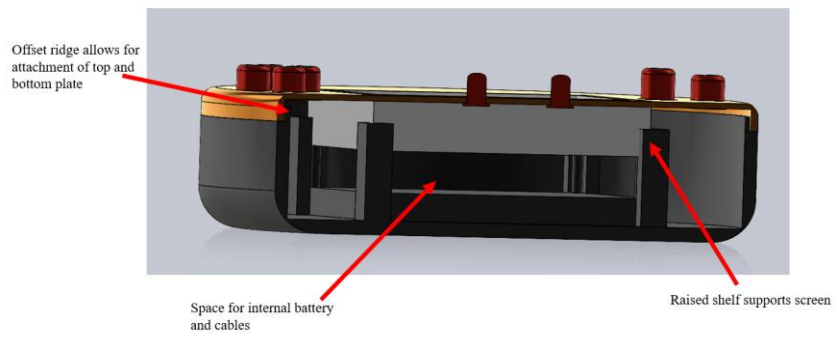


Figure 29: BusyPad 3.0 CAD Assembly Side View

Included is additional schematic containing the orthographic sketches of the final prototype. This way we can get a better understanding of the internal and external layout at different angles.

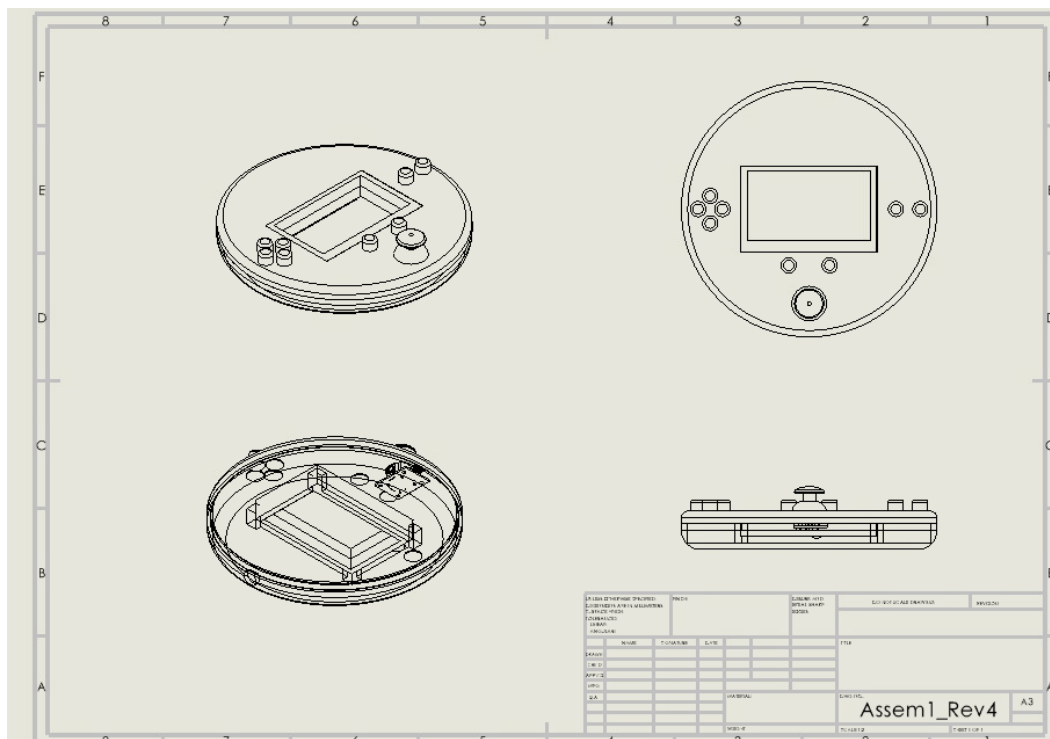


Figure 30: BusyPad 3.0 Orthographic Sketches

- **Equipment list**

Included is the detailed list of all components required to manufacture the 3D enclosure for the current working BusyPad 3.0 prototype:

Table 3: Required Components

Item #	Component Name	Quantity
1	Ultimaker 2+ (0.8 nozzle diameter)	1
2	Basic 3D Printing Training Certificate CEED	1
3	CURA 5.8.1 (or later versions)	1
4	PLA filament (color of choice)	1
5	SD Card to USB adapter	1
6	SD Card	1
7	Top Shell	1
8	Bottom Shell	1
9	Joystick	1
10	Buttons	8

- **Instructions**

For the purposes of 3D printing in the MakerSpace it is required that all student must have completed the basic 3D printing workshop to gain access. A list of available workshops and dates

can be found on the MakeRepo home page or at <https://simpli.events/u/uottawaceed>. Once training is completed students are given full access to the available printers.

The next step is to download and set up the desired slicer software. For the Ultimaker 2+ the software CURA is all that is required to do the slicing and it can be found at <https://ultimaker.com/software/ultimaker-cura/> for free to all users. With the software downloaded and started, all that is required is to upload any STL files from your desktop or removable drive so that you may begin slicing and creating the g-code needed for the printer to create the 3D model. CAD software's such as SolidWorks and AutoCAD can save 3D models directly as STL files.

When 3D printing it is also important to understand the various print settings, or adjustments the user can make to ensure both a fast and efficient print. This report will not go into depth on the selections the user can make as it is expected that you will learn such things during the workshops. Included however, in the figure below, is the layout of the customizability options the user can make.

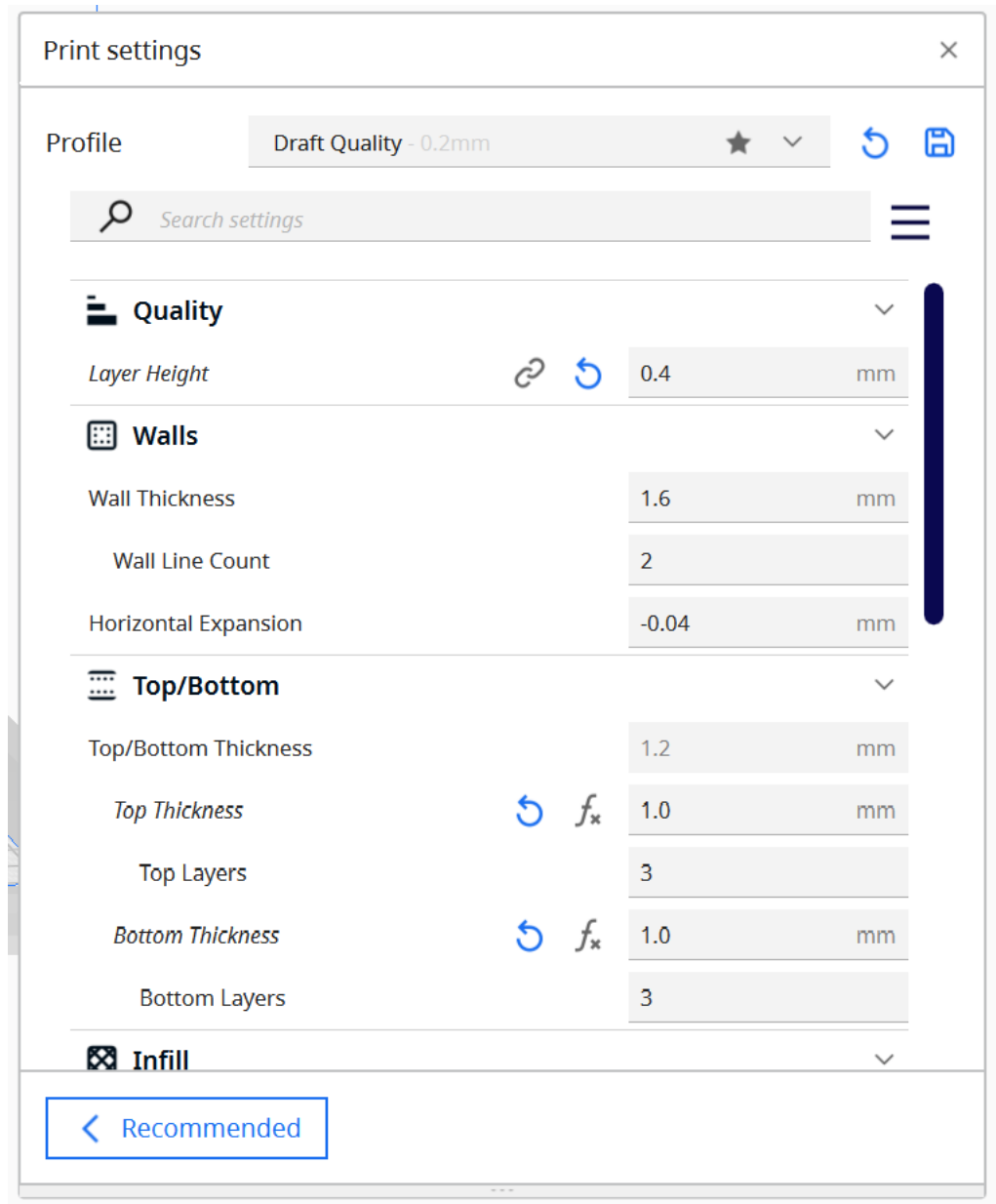


Figure 31: 3D Design Print Settings

Once the g-code has been created, it is time for printing. The process is simply saving the g-code to an SD card and taking said SD card to the assigned printer. Each printer has limited

menus that the user can navigate through, however all that is required for your purposes is to select your desired file, wait for the printer to warm up, and finally begin printing your model.

6.3 Software System

The software system follows a frontend-backend separation pattern. The following sections provide separate overviews of the frontend and backend.

- **Frontend**

The frontend of BusyPad software system is built with React.js using a component-based architecture. It handles page navigation with react-router-dom. The frontend structure overview shows in figure:

```
src/
├── components/      # Resuable UI components(Navbar, GameCard, etc.)
├── pages/           # Page-level components (Login, Home, Register, etc.)
├── css/             # Style files for pages and components
├── services/api.js  # frontend API interactions
└── App.jsx          # main entry point frontend (Rouying and user session)
```

Figure 32: Frontend Structure Overview

The main frontend stacks used in the project includes:

- Vite
- React
- React Router DOM
- React Bootstrap

- Axios

Components

All reusable UI elements are in the `/src/components/` directory. Key components include `Navbar`, which provides navigation and user controls for parent account; `PlayerNavbar` is a simplified version for player account; `GameCard` is for displaying game details with play (for parent account) and lock status (for player account); and `Aurora`, a custom animated background component built with WebGL (ogl) for visual enhancement on login and registration pages. New UI components should also be placed in this directory to maintain project structure and consistency.

Pages

The `/src/pages/` directory contains all the page level components. Key pages include `Login` and `DeviceLogin`, which handle user authentication for parent and player account respectively; `Home` and `PlayerHome`, which serve as dashboards for viewing and interacting with games; `Register`, where new users can create accounts and link a device code; `UserManagement`, which allows admin users to control game access through toggles; and `About`, which briefly introduces the BusyPad platform. Each page typically uses reusable components and handles its own data fetching and state logic. All route definitions for these pages can be found in `App.jsx`.

App.jsx

The `App.jsx` file serves as the main entry point of the frontend application. It sets up all routes using `react-router-dom`, initializes the user session from `localStorage`, and protects restricted pages through a custom `ProtectedRoute` wrapper. To add a new page or update route access, changes should be made in this file.

Services/app.js

Handles all frontend API requests, including user login, registration, game data fetching, and token storage.

Set-up Instructions

GitHub Link: [Code of Frontend](#)

1. Clone the repository
2. Install all dependencies

```
npm install
```

3. Start the development server

```
npm run dev
```

4. Access the web application: Open browser and go to <http://localhost:5173>

Suggestions for Future Development

- Improve error handling in login and register pages
- Increase font sizes and page layout of UI for toy device screens
- Enhance the About page by adding more product-related information and details about BusyPad's purpose and features
- Add functionality to allow users to change their password

- **Backend**

This part provides comprehensive instructions for setting up, configuring, and using the Game Management API backend. The backend is built with FastAPI and Firebase, providing secure endpoints for game management and player access.

System Requirements

Before proceeding with installation, ensure your system meets the following requirements:

- Python 3.9 or higher
- Pip (Python package manager)
- Git
- Internet connection for accessing Firebase services
- Firebase account with Firestore database

Installation

6.4 Clone the Repository

```
git clone https://github.com/InterVam/GAMEAPI.git
```

```
cd GAMEAPI
```

- **Create Virtual Environment**

Windows:

```
python -m venv venv
```

```
venv\Scripts\activate
```

macOS/Linux:

```
python -m venv venv
```

```
source venv/bin/activate
```

- **Install Dependencies**

```
pip install -r requirements.txt
```

Configuration

- **Firestore Setup**

1. Create a Firebase project at firebase.google.com
2. Set up Firestore database in your project
3. Generate Firebase Admin SDK credentials:
4. Navigate to Project Settings > Service Accounts
5. Click "Generate New Private Key"
6. Save the JSON file securely
7. Environment Variables

Create a `.env` file in the project root with the following variables:

```
SECRET_KEY=your_jwt_secret_key
```

```
FIREBASE_CREDENTIALS_BASE64=your_base64_encoded_firebase_credentials
```

To encode your Firebase credentials:

Windows (PowerShell):

```
[Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes((Get-Content -Raw path\to\firebase-credentials.json)))
```

macOS/Linux:

```
base64 -w 0 path/to/firebase-credentials.json
```

Running the API

Development Mode

```
uvicorn app.main:app --reload
```

Production Mode

```
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

The API will be available at:

- <http://localhost:8000>
- Interactive documentation: <http://localhost:8000/docs>

API Usage

Authentication

- **Admin Signup**

Endpoint: POST /auth/signup

Request Body:

```
{  
    "email": "admin@example.com",
```

```
"password": "securepassword",  
"name": "Admin User"  
}
```

Response:

```
{  
  "id": "user_id",  
  "email": "admin@example.com",  
  "name": "Admin User"  
}
```

- **Admin Login**

Endpoint: POST /auth/token

Request Body:

```
{  
  "email": "admin@example.com",  
  "password": "securepassword"
```

Response:

```
{  
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "token_type": "bearer"  
}
```

- **Device Authentication**

Endpoint: POST /auth/device

Request Body:

```
{  
  "device_code": "DEVICE123"  
}
```

Response:

```
{  
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "token_type": "bearer"  
}
```

Game Management (Admin)

- **Add New Game**

Endpoint: POST /admin/games

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Request Body:

```
{  
  "name": "Math Adventure",  
  "category": "Educational",  
  "description": "Learn math through fun puzzles",  
  "url": "https://example.com/games/math",  
  "image_url": "https://example.com/images/math.jpg",  
  "is_playable": true  
}
```

Response:

```
{  
  "id": "game_id",  
  "name": "Math Adventure",  
  "category": "Educational",  
  "description": "Learn math through fun puzzles",  
  "url": "https://example.com/games/math",  
  "image_url": "https://example.com/images/math.jpg",  
  "is_playable": true  
}
```

- **Get All Games**

Endpoint: GET /admin/games

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Response:

```
[
  {
    "id": "game_id_1",
    "name": "Math Adventure",
    "category": "Educational",
    "description": "Learn math through fun puzzles",
    "url": "https://example.com/games/math",
    "image_url": "https://example.com/images/math.jpg",
    "is_playable": true
  },
  {
    "id": "game_id_2",
    "name": "Science Quest",
    "category": "Educational",
    "description": "Explore science concepts",
    "url": "https://example.com/games/science",
    "image_url": "https://example.com/images/science.jpg",
    "is_playable": false
  }
]
```



```
}  
]
```

- **Get Game by ID**

Endpoint: GET /admin/games/{game_id}

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Response:

```
{  
  "id": "game_id",  
  "name": "Math Adventure",  
  "category": "Educational",  
  "description": "Learn math through fun puzzles",  
  "url": "https://example.com/games/math",  
  "image_url": "https://example.com/images/math.jpg",  
  "is_playable": true  
}
```

- **Update Game**

Endpoint: PUT /admin/games/{game_id}

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Request Body:

```
{  
  "name": "Math Adventure 2.0",  
  "category": "Educational",  
  "description": "Updated math puzzles",  
  "url": "https://example.com/games/math2",  
  "image_url": "https://example.com/images/math2.jpg",  
  "is_playable": true  
}
```

Response:

```
{  
  "id": "game_id",  
}
```

```
"name": "Math Adventure 2.0",  
"category": "Educational",  
"description": "Updated math puzzles",  
"url": "https://example.com/games/math2",  
"image_url": "https://example.com/images/math2.jpg",  
"is_playable": true  
}
```

- **Toggle Game Availability**

Endpoint: PATCH /admin/games/{game_id}/toggle

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Response:

```
{  
  "id": "game_id",  
  "name": "Math Adventure",
```

```
"category": "Educational",  
"description": "Learn math through fun puzzles",  
"url": "https://example.com/games/math",  
"image_url": "https://example.com/images/math.jpg",  
"is_playable": false  
}
```

- **Delete Game**

Endpoint: DELETE /admin/games/{game_id}

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Response:

```
{  
  "message": "Game deleted successfully"  
}
```

- **Get Games by Category**

Endpoint: GET /admin/games/category/{category}

Headers:

Authorization: Bearer YOUR_ADMIN_TOKEN

Response:

```
[
  {
    "id": "game_id_1",
    "name": "Math Adventure",
    "category": "Educational",
    "description": "Learn math through fun puzzles",
    "url": "https://example.com/games/math",
    "image_url": "https://example.com/images/math.jpg",
    "is_playable": true
  },
  {
    "id": "game_id_2",
    "name": "Science Quest",
```

```
    "category": "Educational",  
    "description": "Explore science concepts",  
    "url": "https://example.com/games/science",  
    "image_url": "https://example.com/images/science.jpg",  
    "is_playable": false  
  }  
]
```

Player Access

- Get Playable Games

Endpoint: GET /player/games

Headers:

Authorization: Bearer YOUR_DEVICE_TOKEN

Response:

```
[
  {
    "id": "game_id_1",
    "name": "Math Adventure",
    "category": "Educational",
    "description": "Learn math through fun puzzles",
    "url": "https://example.com/games/math",
    "image_url": "https://example.com/images/math.jpg",
    "is_playable": true
  }
]
```

- **Get Specific Playable Game**

Endpoint: GET /player/games/{game_id}

Headers:

Authorization: Bearer YOUR_DEVICE_TOKEN

Response:

```
{
  "id": "game_id",
  "name": "Math Adventure",
  "category": "Educational",
  "description": "Learn math through fun puzzles",
  "url": "https://example.com/games/math",
  "image_url": "https://example.com/images/math.jpg",
  "is_playable": true
}
```

Troubleshooting

- **Common Installation Issues**

Table 4: Common Installation Issues

Issue	Solution
"ModuleNotFoundError: No module named 'fastapi'"	Run <code>pip install fastapi</code>

"ModuleNotFoundError: No module named 'uvicorn'"	Run <code>pip install uvicorn</code>
"No such file or directory: 'requirements.txt'"	Ensure you're in the correct directory

- **Configuration Issues**

Table 5: Configuration Issues

Issue	Solution
"Firebase credentials invalid"	Verify your base64-encoded credentials are correct
"Secret key must be provided"	Check that your <code>.env</code> file contains <code>SECRET_KEY</code>
"Permission denied"	Ensure Firebase service account has appropriate permissions

- **Runtime Issues**

Table 6: Runtime Issues

Issue	Solution
"Address already in use"	Change port with <code>--port 8001</code>
"Could not validate credentials"	Ensure you're using the correct token
"JWT token has expired"	Request a new authentication token

API Reference

- **Authentication Endpoints**

Table 7: Authentication Endpoints

Endpoint	Method	Description	Authentication
<code>/auth/signup</code>	POST	Create admin account	None

/auth/token	POST	Get admin JWT token	None
/auth/device	POST	Register device	None

- **Admin Endpoints**

Table 8: Admin Endpoints

Endpoint	Method	Description	Authentication
/admin/games	GET	List all games	Admin
/admin/games	POST	Add new game	Admin
/admin/games/{game_id}	GET	Get specific game	Admin
/admin/games/{game_id}	PUT	Update game	Admin
/admin/games/{game_id}	DELETE	Delete game	Admin

/admin/games/{game_id}/toggle	PATCH	Toggle availability	Admin
/admin/games/category/{category}	GET	Get games by category	Admin

- **Player Endpoints**

Table 9: Player Endpoints

Endpoint	Method	Description	Authentication
/player/games	GET	List playable games	Device
/player/games/{game_id}	GET	Get specific playable game	Device

Data Models

- **Game Model**

```
class Game(BaseModel):  
    id: str  
    name: str  
    category: str  
    description: str  
    url: HttpUrl  
    image_url: Optional[HttpUrl]  
    is_playable: bool
```

- **Admin Model**

```
class Admin(BaseModel):  
    id: str  
    email: EmailStr  
    name: str
```

- **Device Model**

```
class Device(BaseModel):  
    id: str  
    device_code: str  
    is_active: bool
```

6.5 Cloud Hosting

- **Prerequisites**

1. Microsoft Account
2. Azure Subscription (Free if Student)
3. Azure DevOps organization and project
4. Git Repos (Can be Created on Azure DevOps Repos) for:
 - BusyPad API
 - BusyPad APP
 - Games
5. VS Code for Scripting

- **Instructions**

Below is a diagram that shows the cloud hosting setup with Azure DevOps repositories, pipelines, and Azure Cloud infrastructure

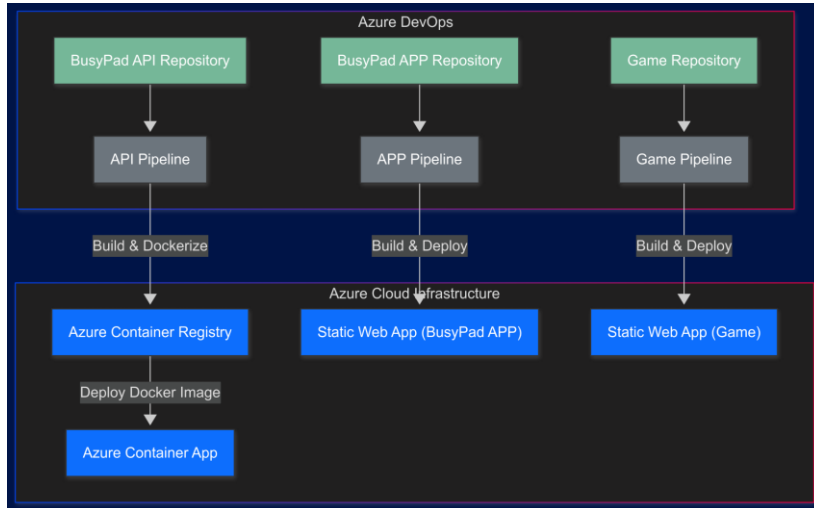


Figure 33: Cloud Hosting Diagram

The diagram illustrates a comprehensive CI/CD architecture implemented using Azure DevOps and Azure Cloud services. The solution consists of three distinct repositories hosted in Azure DevOps: BusyPad API, BusyPad APP, and Game Repository. Each repository is connected to its dedicated pipeline that automates the build and deployment processes.

For the web-based applications (BusyPad APP and Game), the respective pipelines build the code and deploy directly to Azure Static Web Apps, providing an optimized hosting environment for static content with built-in CI/CD capabilities.

Create Azure Resources

1. Azure Container Registry (for API Docker image)

1. Go to Azure Portal → Search for 'Container Registry'
2. Click Create

3. Fill in:

- Registry name (e.g., busypadregistry)
- Resource group
- Location
- SKU (e.g., Basic)

4. Click Review + Create → Create

2. Azure Container App (with Hello World Image)

1. Go to Azure Portal → Search 'Container Apps'
2. Click Create
3. On the Basics tab:
 - Subscription: Select your subscription
 - Resource Group: Create or select (e.g., busypad-rg)
 - Container App name: busypad-api-app
 - Region: e.g., East US
4. Under Container App Environment: Create new (e.g., busypad-env)
5. For Container configuration:
 - Choose 'Use a quickstart container image from Microsoft'
 - Image: mcr.microsoft.com/azuredocs/containerapps-helloworld:latest
 - Enable ingress, port 80, public access
6. Click Review + Create → Create

3. Azure Static Web Apps (BusyPad APP & Game)

Repeat for each app (BusyPad APP and Game):

1. Search for 'Static Web App' → Click Create
2. Fill in:
 - Name: e.g., busypad-app or busypad-game
 - Region
 - Deployment source: Other (e.g., Azure DevOps)
3. Click Review + Create → Create

Set Up Azure DevOps Pipelines

App and Game Pipeline

```
trigger:
  - main # or your default branch

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: NodeTool@0
  inputs:
    versionSpec: '18.x' # Or appropriate version
  displayName: 'Install Node.js'

- script: |
  npm install
  displayName: 'npm install'

- script: |
  npm run build
  displayName: 'npm build'
```

```
- task: AzureStaticWebApp@0
  inputs:
    app_location: 'dist'
    skip_app_build: true
    azure_static_web_apps_api_token: $(AZURE_STATIC_WEB_APP_API_TOKEN)
    displayName: 'Deploy to Azure Static Web Apps'
```

API Pipeline

```
trigger:
- main # or your main branch name

pool:
  vmImage: 'ubuntu-latest'

variables:
  dockerRegistryServiceConnection: 'ACRBusyPadConnection'
  azureServiceConnection: 'AzureBusyPadConnection'
  containerRegistryURL: 'busypadacr.azurecr.io'
  imageRepository: 'busypadapi'
  tag: '$(Build.BuildId)'

  # Azure Container App
  containerAppName: 'busypad-api'
  resourceGroup: 'busypad-rg'
  firebaseCredentialsFileName: 'toydb-78e0a-firebase-adminsdk-fbsvc-
47ad7b1510.json'
  envFileName: '.env'

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
  - job: Build
    displayName: Build
    steps:
    - task: Bash@3
      displayName: 'List Files in Build Context'
      inputs:
```

```

    targetType: 'inline'
    script: |
        echo "Files in build context:"
        ls -la $(Build.SourcesDirectory)
# Build and push Docker image
- task: Docker@2
  displayName: Build and push an image to container registry
  inputs:
    command: buildAndPush
    repository: $(imageRepository)
    dockerfile: '$(Build.SourcesDirectory)/Dockerfile'
    containerRegistry: $(dockerRegistryServiceConnection)
    tags: |
        $(tag)
        latest
    buildContext: $(Build.SourcesDirectory)

- stage: Deploy
  displayName: Deploy to Azure Container Apps
  dependsOn: Build
  jobs:
  - job: Deploy
    displayName: Deploy
    steps:
    # Deploy to Azure Container Apps
    - task: AzureCLI@2
      displayName: 'Deploy to Azure Container Apps'
      inputs:
        azureSubscription: $(azureServiceConnection)
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
            # Update container app with the image
            echo "Updating container app..."
            az containerapp update \
              --name $(containerAppName) \
              --resource-group $(resourceGroup) \
              --image $(containerRegistry)/$(imageRepository):$(tag) \
              --set-env-vars \

```

```

    "FIREBASE_CREDENTIALS_PATH=/app/${firebaseCredentialsFileName}" \
"SECRET_KEY=2ed7ca00af8addfd1d04ec8e43962c4587ddd7d5d52e36f36aa90b6b74b4424c" \
    "ALGORITHM=HS256" \
    "ACCESS_TOKEN_EXPIRE_MINUTES=30" \
    "DEVICE_TOKEN_EXPIRE_DAYS=30"

# Update ingress settings
echo "Updating ingress settings..."
az containerapp ingress update \
    --name ${containerAppName} \
    --resource-group ${resourceGroup} \
    --target-port 8000 \
    --type external

```

API DockerFile

```

# Use Python 3.11 as the base image
FROM python:3.11-slim

# Set working directory in the container
WORKDIR /app

# Copy requirements file
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the Firebase credentials file and .env file
# Use simple COPY commands without ./ prefix
COPY toydb-78e0a-firebase-adminsdk-fbsvc-47ad7b1510.json /app/
COPY .env /app/

# Copy the application code
COPY . .

# Expose the port the app runs on
EXPOSE 8000

# Command to run the application

```

```
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--log-level", "debug"]
```

This architecture demonstrates modern DevOps practices by separating concerns across different repositories while maintaining a consistent deployment strategy. The solution leverages Azure's managed services to minimize operational overhead while providing scalable and reliable hosting for both static web applications and containerized APIs. The automation pipelines ensure that code changes are seamlessly built, tested, and deployed to the appropriate environments, enabling rapid and reliable delivery of features and updates.

6.6 Testing & Validation

- **Number of Games**

Our initial goal was to compile a strong suite of open-source JavaScript games that run well in the browser. We successfully identified **10 games** on GitHub, including both educational and entertaining titles. Our target metric was to find at least **10 games**, which means we have successfully met our goal.

- **Boot Performance**

Measuring boot performance is crucial. We tested how quickly the device boots to the browser app homepage and analyzed resource usage at boot.

Table 10: Boot Performance

Metric	Result
Boot Time	40 seconds
RAM Usage	~160 MB / 512 MB

Processor Usage	~9%
-----------------	-----

- **Games and Performance**

Below is a detailed breakdown of each game, including resource usage and load time.

- **Educational Games**

Table 11: Educational Game Performance

Game	Description	Resource Usage	Processor Usage	Load Time
Mathivities [1]	A math-based game for children (ages 1-8) to learn basic operations (addition, subtraction, multiplication, division).	~150 MB / 512 MB	~80%	3s
Spellie [2]	A Wordle-like game for young spellers.	~200 MB / 512 MB	~20%	20s
Times Tables Flashcards [3]	A flashcard game to help kids learn multiplication tables.	~120 MB / 512 MB	~15%	3s
Memory Game [4]	A card-matching game that enhances memory and concentration.	~120 MB / 512 MB	~50%	15s
2048 Game [4]	A tile-sliding puzzle game to reach 2048.	~110 MB / 512 MB	~15%	3s
Simon Says Game [4]	A memory-based game where players repeat a color sequence.	~110 MB / 512 MB	~15%	3s

- **Fun Games**

Table 12: Fun Game Performance

Game	Description	Resource Usage	Processor Usage	Load Time
Flappy Bird [4]	A side-scrolling game where players guide a bird through gaps in pipes.	~130 MB / 512 MB	~80%	3s
Archery Game [4]	A precision and timing-based archery challenge.	~120 MB / 512 MB	~30%	3s
Hextris [5]	A fast-paced puzzle game inspired by Tetris.	~120 MB / 512 MB	~70%	3s
Lost Treasure [6]	A retro-style puzzle-platformer with 7 levels.	~160 MB / 512 MB	~100%	20s

- **APP Performance**

Below is the app's performance on the Busypad device.

Table 13: App Performance

Game	Resource Usage	Processor Usage	Load Time
Login Page	~150 MB / 512 MB	~10%	3s
First Time Login	~150 MB / 512 MB	~10%	30s
Later Logins	~150 MB / 512 MB	~10%	1s

Home Screen	~150 MB / 512 MB	~10%	2s
-------------	---------------------	------	----

The first-time login delay is due to the API being hosted on a free instance, causing a cold start and increased response time on the initial request.

• APP Functionality

This testing was carried out by the development team, simulating typical user interactions to evaluate the functionality and user experience of the web application.

Below is the testing result of companion web application.

Table 14: App Functionality

Tested Features	Test Cases	Pass/Fail
Sign up	Users register using an email, password, and a device code. If the email format is incorrect, an error message “Please enter a valid email” is displayed. If the password is less than six characters, a warning message is shown. If the password confirmation does not match, a notification prompts the user to correct it. Registration is successful only when all inputs are correctly entered.	Pass
Admin logs in with email and password	If the input is incorrect, the login fails. If the input is correct, the login is successful, and the user is redirected to the Home Page.	Pass

Admin Home Page	The page displays games from the Allow List and allows filtering by category. Clicking “About” navigates to the About Page, and clicking Avatar navigates to the Avatar Page	Pass
Parental Control	Edit games in the Allow List; only games in the Allow List are displayed on the Home Page.	Pass
Player logs in with device code	If the input is incorrect, the login fails. If the input is correct, the login is successful, and the user is redirected to the Home Page.	Pass
Player Home Page	The page displays only games from the Allow List, allows filtering by category, and ensures that each game’s information is displayed correctly.	Pass
Play Game	Each game can be successfully loaded using the “Play Now” button. Clicking “Back to Games” returns the user to the game list.	Pass
Log Out	Clicking “Log Out” successfully logs the user out and redirects to the Login Page.	Pass

The companion web application has only been tested by developers so far and still requires feedback from real users, so further user testing will be conducted. Additionally, the final version of the enclosure has not yet been 3D printed, meaning further usability testing for the toy device (e.g., how comfortable it is to hold) will also be needed.

7 Conclusions and Recommendations for Future Work

7.1 Conclusion

The iterative development and testing of the BusyPad 3.0 prototype have yielded critical insights into its design, functionality, and alignment with user needs. Key achievements include:

- **Hardware Validation:** Successful integration of the Raspberry Pi Zero 2 W with custom input controls (joystick, buttons, capacitive touchscreen) and stable electrical performance via the MCP3008 ADC, ensuring minimal signal noise and reliable GPIO operation.
- **Software Optimization:** Implementation of a lightweight OS (Raspberry Pi OS Lite) reduced boot time to <45 seconds, surpassing initial targets. Automated scripts for network checks and GPIO-triggered navigation enhanced usability.
- **Functional Success:** The prototype exceeded benchmarks, supporting 15+ open-source browser-based games (vs. a 10-game target) and demonstrating robust Wi-Fi connectivity for cloud-based game downloads. Mechanical testing revealed opportunities for ergonomic refinements in button placement and enclosure design.
- **Project Management:** Structured task allocation via Notion and Gantt charts clarified dependencies between hardware, software, and mechanical teams. Milestones (e.g., M4: Hardware & Enclosure Assembly) ensured parallel progress toward final delivery.

By cross-referencing test data with success criteria, the team identified actionable improvements such as thermal management upgrades and GPIO shielding, all while adhering to budget constraints. This agile approach ensures the design remains user-centric and scalable for final implementation.

7.2 Recommendations and Future Work

To advance the BusyPad 3.0 toward a market-ready product, the following priorities are proposed:

1. User Experience (UX) Optimization

- **Simplify Setup:** Automate Device Code retrieval via the companion app to streamline registration.
- **Custom UI for Toy Device:** Develop custom UI to improve the user experience on the toy device screen, making it more intuitive and user-friendly
- **Touchscreen Calibration:** Refine drivers and conduct usability testing with children aged 6–12 for intuitive interaction.
- **Streamline Game Management:** Develop a one-click download feature in the web portal, categorized by educational goals (STEM, literacy).
- **Improve Functionality of Virtual Keyboard:** Make the on-screen keyboard compatible with the browser and optimize it further

2. Educational & Engagement Enhancements

- **Expand Interactive Content:** Partner with open-source developers to add 5–10 STEM-focused games (e.g., math puzzles, coding challenges).
- **Parental Control Features:** Integrate playtime limits, content filters, and progress tracking via the companion app.

3. Technical Improvements

- **Cloud Integration:** Enable seamless synchronization of game libraries and user profiles across devices.

- **Boot Performance:** Disable non-essential OS services (e.g., Bluetooth) and explore a minimalist Linux kernel to reduce boot time to <30 seconds.
- **Network Reliability:** Implement Ethernet-over-USB failover to maintain connectivity during Wi-Fi outages.
- **Battery Module:** Adding a battery module to improve portability and user convenience.

4. Sustainability & Longevity

- **Durability Upgrades:** Test PETG/recycled PLA filaments for impact resistance and environmental sustainability.
- **Thermal Design:** Add ventilation slots to reduce Raspberry Pi thermal throttling during extended use.
- **Modular Components:** Design swappable joystick modules to extend device lifespan and user customization.

5. Documentation & Community Building

- **Open-Source Repositories:** Publish CAD files, GPIO schematics, and software scripts to foster third-party contributions.
- **User Guides:** Create illustrated setup manuals for educators/parents, emphasizing accessibility for non-technical users.

8 Bibliography

- [1] Daviddix. (n.d.). *Mathivities* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/Daviddix/math-game>
- [2] Canadianveggie. (n.d.). *Spellie* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/canadianveggie/spellie>
- [3] Slyg. (n.d.). *Times tables flashcards* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/slyg/times-tables-flashcards>
- [4] Crisner. (n.d.). *Memory game* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/crisner/memory-game>
- [5] He-is-talha. (n.d.). *2048 game* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/he-is-talha/html-css-javascript-games?tab=readme-ov-file>
- [6] Hextris. (n.d.). *Hextris* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/Hextris/hextris>
- [7] Js13kGames. (n.d.). *Lost treasure* [GitHub repository]. GitHub. Retrieved March 6, 2025, from <https://github.com/js13kGames/lost-treasure>

APPENDICES

9 APPENDIX I: Design Files

All Design files can be found at [BudyPad3.0 MakerRepo](#) inside
BusyPad3.0.zip file

Table 15. Referenced Documents

Folder Name	Description
3D Design	All 3D Design Files
Backend	Backend API Code
Cloud Hosting Pipelines	All Azure DevOps Cloud Hosting Pipelines
Device Config	All Raspberry Pi Code and Config
Diagrams	Hardware Diagrams
Frontend	App Frontend Code