RevMetrix Project Proposal, Design, and Schedule

Spring 2025

Teams (include team members and URLs)

Simulation

The Simulation Team for the RevMetrix project, consisting of Andrew Watkins, Carson Mack and Patrick Devine, is responsible for developing the 3D simulation and data visualization components. To improve performance and integration within our .NET MAUI environment, the team is switching from Three.js to OpenTK for 3D rendering and from Chart.js to LiveCharts2 for data visualization. The previous JavaScript-based solutions caused performance issues, slowing down data transfers and rendering. By moving to C#-based tools, simulation can significantly increase speed, reduce lag, and improve compatibility with our existing framework. OpenTK will provide smoother, more efficient 3D rendering, while LiveCharts2 will allow for faster and more interactive data visualization. This transition will enhance simulation accuracy, improve user interaction, and optimize performance for a more seamless experience. In addition, the Unity simulation, developed by the Simulation team of Fall 2023 (Luke Dodson, Bryce Neptune, Enoch Sam, and Ian Viveiros) can be updated, implemented, and shipped into the current ball spinner application. This can be done through a library such as UnityUaal.Maui.

Backend

The Backend team for the RevMetrix project consists of Ryan Curry, Josh Byers, Thomas Bywaters, and Brandon Woodward. This team is responsible for managing cloud infrastructure, maintaining the SQL Server, and developing the API server. The team also designs and

implements data models that process user data efficiently and accurately for the BallSpinner Application. Our primary goals are to enhance the user experience, advance the application's protocol, refine and expand API endpoints, develop efficient methods for interacting with the BallSpinner, and continuously evolve the database schema to meet the needs of our use cases. The Backend team develops the project within the following repositories:

API Server: https://github.com/YCP-Rev-Metrix/BallSpinner-Cloud

BallSpinner Application: https://github.com/YCP-Rev-Metrix/BallSpinner-Application

Ball Spinner Application Frontend

The Frontend team of the RevMetrix project consists of Patrick Devine. The team's responsibilities focus on the maintenance and development of the Ball Spinner Application which was developed in Fall 2024. This team will design and implement pages and changes to the application to the needs of other teams. Additionally, this team will be tasked with improving the GUI of the Ball Spinner application in preparation for the formal showcase at the capstone expo.

Wiki

The RevMetrix wiki is managed by the core team to store and organize all essential information related to the project. This centralized platform includes details about the project's current state, how to contribute, ideas for future developments, assignments, milestones, and tools used throughout the project. The wiki is accessible via two primary methods: through the Help section in the application or by visiting https://docs.RevMetrix.io/.

Ball Spinner Controller

The Ball Spinner Controller team consists of Robert Fields, Brandon Woodward, and Zachary Cox and is in charge of communicating with all of the components in the Ball Spinner System. This includes the Ball Spinner Application, the SmartDot module, and the Ball Spinner's motors and sensors. To achieve communication with the application, the team is also responsible for designing and implementing the Ball Spinner Protocol on the controller side, ensuring that all messages are parsed properly and sent in the expected order. The Ball Spinner Controller team develops the project within the following locations:

Ball Spinner Controller: https://github.com/YCP-Rev-Metrix/BallSpinner-Controller

Protocol: https://docs.google.com/document/d/1l0I801-6IPXqeT2rksSouOzAHo4GqhKl5Tzo1N18tlE/edit?usp=sharing

Ball Spinner

The Ball Spinner team consists of Sam Diskin, Chris Robinson, and David Kyeremeh. The Ball Spinner is responsible for making the physical system that will rotate the bowling ball in three orthogonal directions; It must also accurately measure the movements of the bowling ball externally so the SmartDot has data to compare against. The team will focus on analyzing parts and assembling them into the complete structure that we design.

Mobile Application

Carson, Josh, Zach, Thomas

The Mobile Application Team consists of Carson Mack, Josh Byers, Zachary Cox, and Thomas Bywaters. The application is being developed as a frontend for the user to keep score and data collection. The primary focus for the semesters is building a new application on the latest .NET MAUI (.NET 9) release and creating database connections to the cloud for end users and testers. Our stretch goal is to connect the smartDot module to the Mobile App.

Project Overview

The problem our system addresses is the lack of calibration and testing environments for the SmartDot. The future SmartDot module device and the creation of the testing environment for the future SmartDot module are integral requirements for this project as well. The system will also provide functionality to allow a user to analyze and review previous shot data, enabling them to leverage our platform to evaluate the effects of various factors on a bowling shot. The solution to this problem is the Ball Spinner device. The Ball Spinner device will allow a bowling ball to be placed within its enclosure, where the Ball Spinner Application can then provide physical instructions on how to move the bowling ball through the TCP protocol sent to the Ball Spinner Controller. The purpose of this is to analyze how the SmartDot module reacts to a controlled environment where all input variables are known, so that it can be easily discerned if the sensors on the SmartDot module are not calibrated properly. The solution should include a simulation that displays how the Ball Spinner is physically moving the ball along with a visual representation of what information the SmartDot sensors are conveying, so that the accuracy of the SmartDot module can be observed in real-time. The solution also needs a backing database to store relevant shot parameters as well as the expected vs the actual output for further refined analysis. Since the Ball Spinner device uses physical motors to drive the ball, we must ensure that the instructions we send to the motors move the ball to the specified speed. Since the SmartDot module is still under development, once the Ball Spinner is calibrated, this will also foster a testing environment for Professor Hake to test the module's functionality.

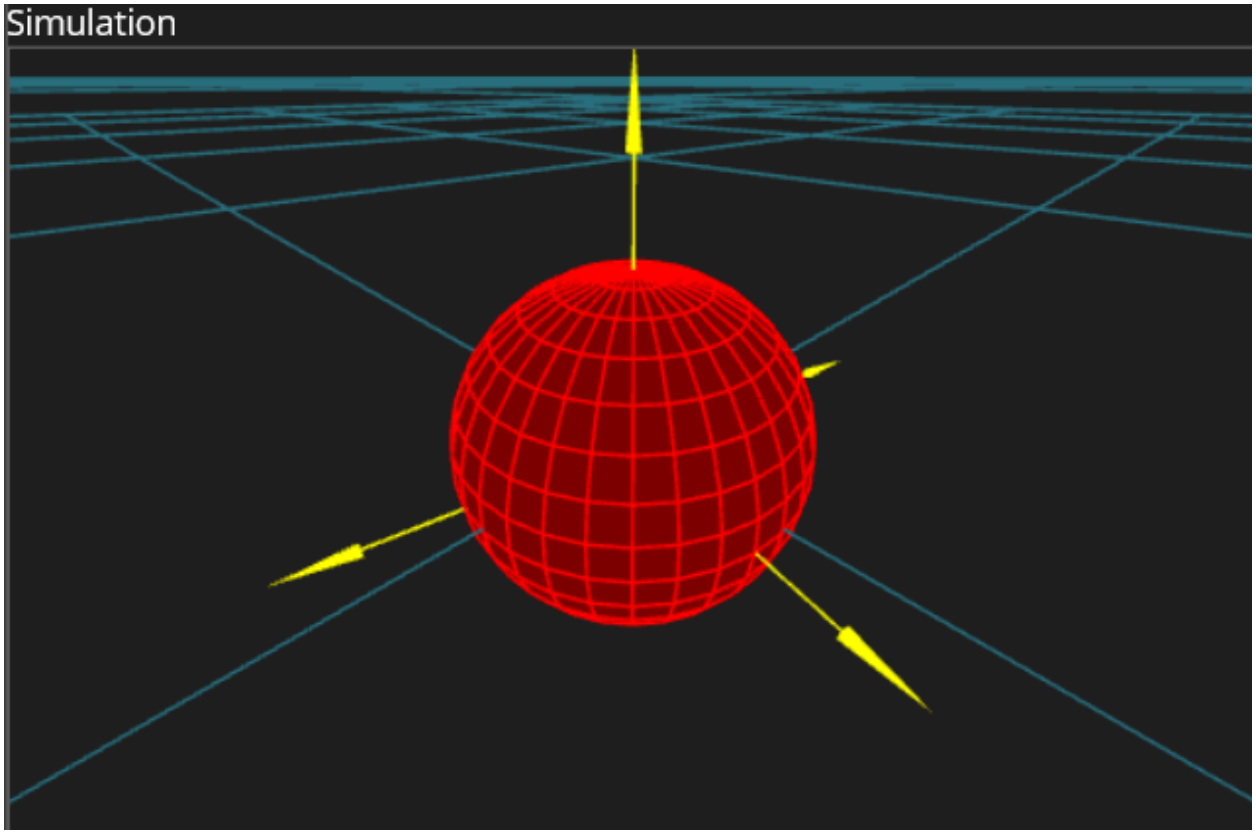
Furthermore, our system also seeks to provide consumers with seamless access to the bowling metrics provided by SmartDot and the various signal processing algorithms developed by Professor Hake to provide bowlers with a simple application to analyze their shots via internal

metrics. This will also incorporate the Ciclopes software within the application to provide external metrics for further analysis. This will take the form of a mobile application, ensuring ease of access to a wide consumer base.

Current State of RevMetrix

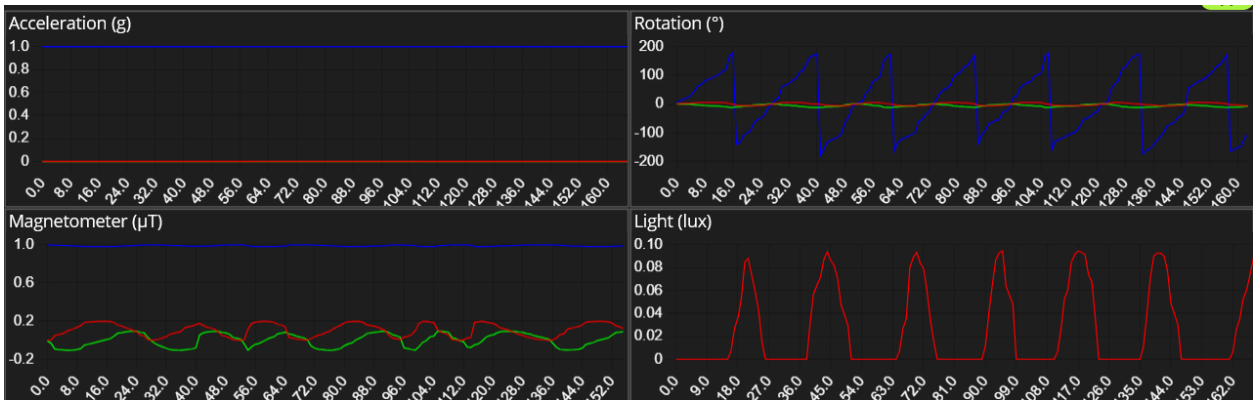
Simulation

To build the simulation, the system was designed to allow the ball's rotation to be controlled in any direction, with clear visuals to show its movement. Three.js, a library for 3D graphics, was used to design and animate the ball. Two methods for controlling rotation were considered: Euler angles and Quaternions. Euler angles seemed easier to understand because they use simple terms like pitch, yaw, and roll to describe the ball's rotation. However, this method encounters a gimbal lock problem, which limits certain movements and can make rotations difficult or impossible when multiple rotations are executed in a certain order. For simple sphere rotation, gimbal lock may not pose an immediate issue if the rotations are limited to basic movements around a single axis. However, when the rotation grows more complex or involves several axes, gimbal lock may limit the sphere's movement, giving unexpected or limited outcomes. Quaternions were chosen instead to avoid this issue. Quaternions allow the ball to rotate smoothly in all directions without constraint. To make the ball's rotation easier to visualize, arrows were added to each side of the ball to indicate its position during the simulation.

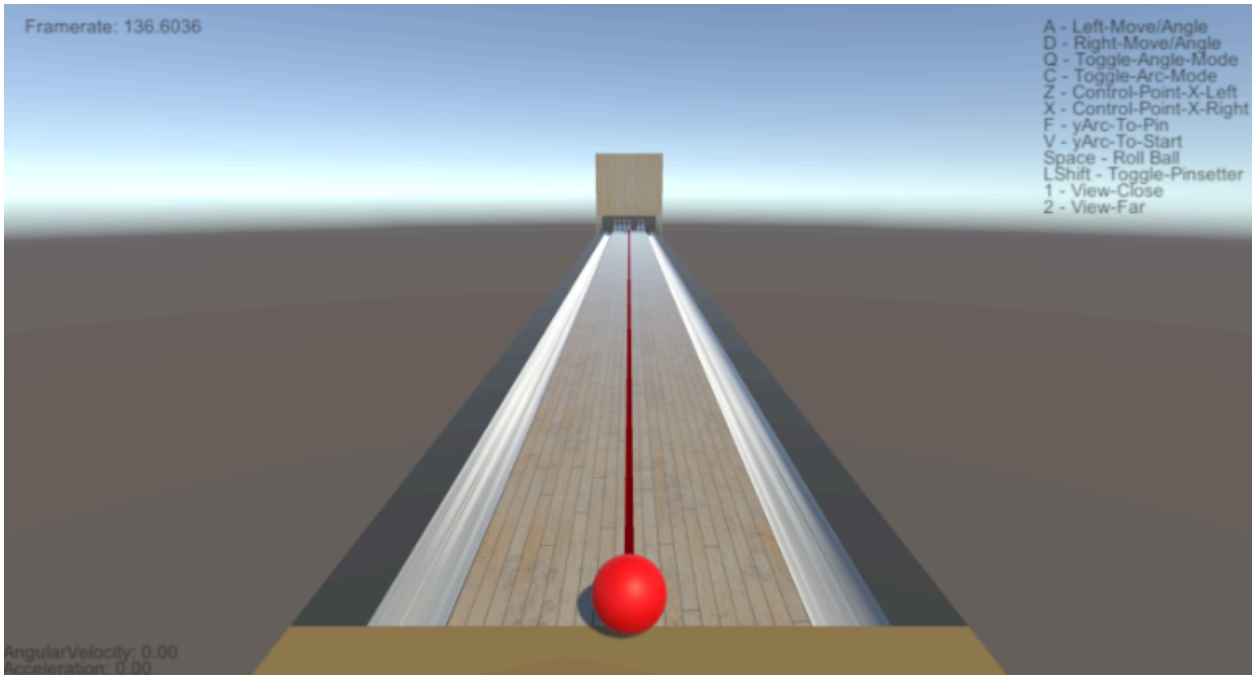


In addition to displaying the ball rotation, the simulation also tracks and displays important data. were created to show the ball's position along the yaw, pitch, and roll axes and data from the magnetometer and light sensors. These graphs are updated in real-time as the simulation is run, providing a clear view of the ball's orientation. The yaw, pitch and roll values are extracted from the quaternion representing the ball’s rotation. This conversion is done by The yaw, pitch, and roll values calculated from the quaternion that represents the ball's rotation. This is done using common math formulas that involve the quaternion's components. To find the yaw, the arctangent of certain components is used, while the pitch is found using the arcsine of a specific value. The roll is calculated with the arctangent of other components. After the yaw, pitch, and roll are calculated, the angles are converted from radians to degrees for easier display.

For example, the rotation graph displayed the ball's position along the three axes (yaw, pitch, and roll). In contrast, the acceleration and sensor data graphs remained constant, showing no significant variation during the simulation. Compared with the emulator, the rotation data closely matched the expected results, while the sensor data highlighted areas for potential improvement.



As for the Unity simulation, the previous team from Fall 2023 was able to accurately recreate the physics of bowling. Their recreation consisted of pins that moved with utmost accuracy to real life, a ball which could have added spin, and the ability to move the ball and change the spin at will. The parameters of the shot cannot be given beforehand, but they are recorded and outputted into a CSV file post-shot. Shown below is their implementation.



Backend

Pertaining to the RevMetrix backend, major foundational progress was made last semester that is important to note in order to understand the work that still needs to be done. The following section will summarize the major accomplishments that are relevant to the future use cases. Firstly, the team made major strides in setting up the application-side TCP protocol that interacts with the Ball Spinner controller as a proxy for sending/receiving information to the Ball Spinner device. From the application, a user can connect to the Ball Spinner controller using its IP address where a handshake protocol is initiated, select a sensor module to connect to, receive data from the sensor module, and send motor instructions that will control how the Ball Spinner's motors behave. The application can process incoming sensor data, transmitting it to the frontend for graphing while also storing it in the database. While this work provides the current team with a lot of the foundational aspects completed, there is still much work to be done. For

instance, the motor instructions are not based on user input, instead they are based on a predefined interval of instructions that repeats. Getting the motor instructions to be based on user input is going to be a major focus for the team this semester as it is integral for the testing and validation of the SmartDot module.

Furthermore, many client-side API methods were developed that allow the user to interact with the RevMetrix database that is running on a DigitalOcean droplet. Some examples of these methods include GetListOfShots(), which enables users to retrieve a list of saved shots along with relevant data for each shot. The application also includes client-side API methods for saving and retrieving bowling balls with their associated data, uploading shot data, and handling user authentication through login and registration. These client-side API methods and their underlying structure facilitate essential user interactions with the database, providing a foundation for the seamless implementation of new API methods planned for this semester.

Next, the application allows users to save shot data locally to remove the need to be connected to the internet to access previously saved shot data. This is facilitated through the use of CSV files where each row stores a data sample obtained from the sensor module while the relevant shot was taking place. Users can also replay sensor data from a local shot on the application's frontend in real-time, synchronized with the timestamps recorded in the sample data. This enables users to reanalyze trends from the shot, though they currently cannot view the input parameters or the specific bowling ball used. While the local replay feature allows users to reanalyze shot data, shots saved to the database cannot currently be replayed or converted into local shots, preventing full offline access to all data. These limitations will be addressed this semester, but the previous work on this feature provides a strong foundation for further improvements.

Additionally, the team has a testing structure in place for the Ball Spinner Application as well as the Ball Spinner Cloud to ensure the integrity of each codebase can be easily maintained through subsequent updates. This consists of a test server on the application that mimics the real API server, so that the client-side API methods can be tested without affecting the database along with unit tests for all of the other essential functionality of the application, like the WriteToTempRevFile class which writes data coming in from the SmartDot module to a CSV file to later be saved/reviewed by the user. Also, for the cloud server codebase the team has a testing structure for the API endpoints as well as the database methods that the API endpoints use to interact with the database. All of these testing components ensure that any future updates made to both codebases can be validated to not interfere with previous essential components.

Not to mention, the team has deployed a Digitalocean droplet that contains the API server as well as the SQL server.

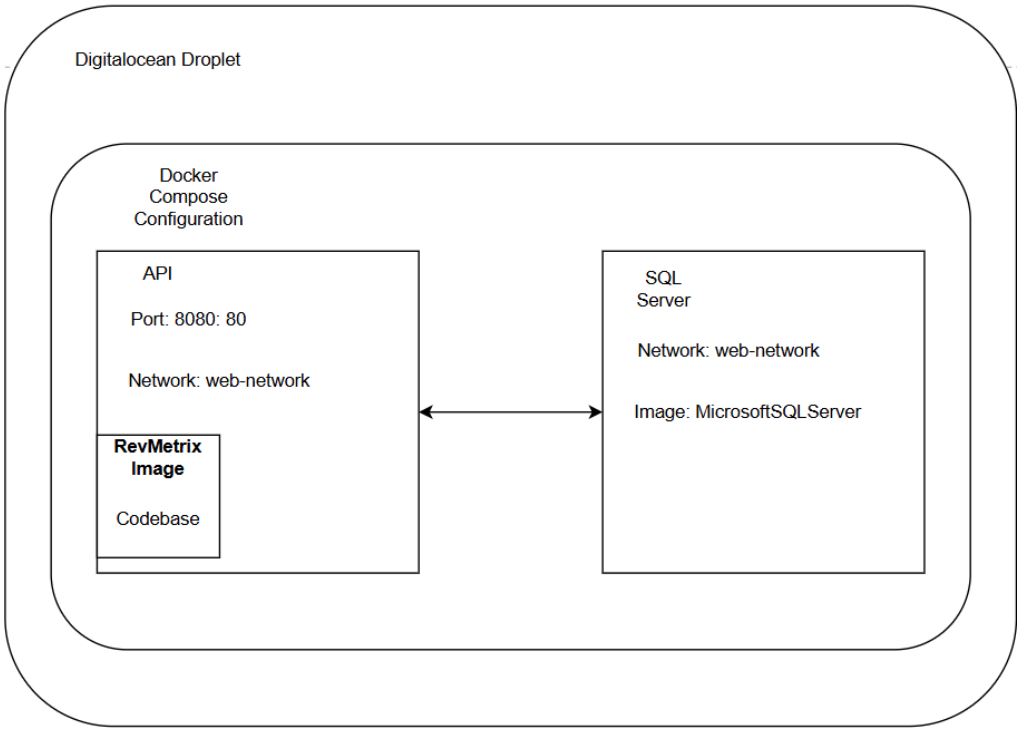


Figure X Current Docker Configuration

Figure X illustrates the team's server configuration for the project. The setup uses Docker to run two containers: one for the API server and another for the SQL server. The API container is built from a Docker image containing the API server code and is mapped to a public domain name via Nginx Proxy Manager. The API communicates with the SQL server container to manage database access for users. While the API server is properly configured to interact with the database, there is a security concern—although the database port is not explicitly exposed, the SQL server remains publicly accessible. This creates a potential vulnerability, as it could be targeted by brute-force attacks. Addressing this issue is a priority for the semester to enhance the security of the server infrastructure.

Moreover, the RevMetrix team has already provided a working CI/CD workflow that allows the automation of the deployment for the API code to the Digital Ocean droplet.

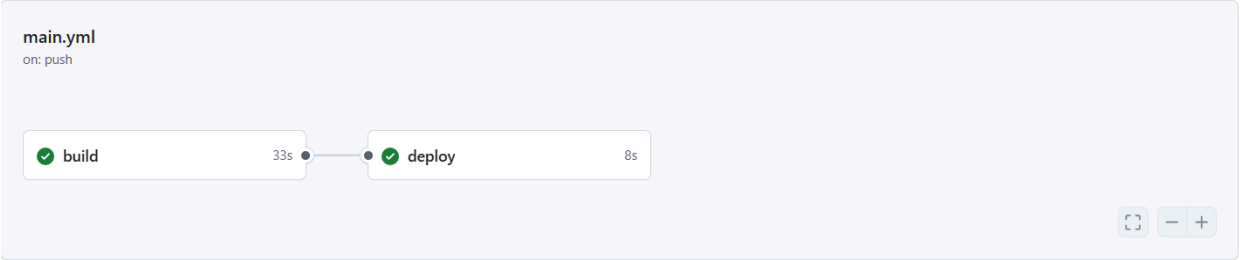


Figure X.A Current CI/CD Pipeline

Figure X.A illustrates the key components of our CI/CD pipeline using GitHub Actions. When new code is pushed to the cloud server codebase, GitHub Actions automatically triggers a workflow to rebuild the Docker image that runs our API server which is the ‘build’ job. If the build is successful, GitHub Actions then establishes an SSH connection to the Digital Ocean droplet hosting the cloud server, describing the ‘deploy’ job. It proceeds to update the API container, ensuring it runs the newly built image, easily deploying the latest changes. However,

what the CI/CD is currently lacking is a ‘test’ job which should only perform the above jobs if all tests pass, ensuring no faulty code is published in the cloud. One of the major tasks of this semester will be fixing this inconsistency.

Ball Spinner Application Frontend

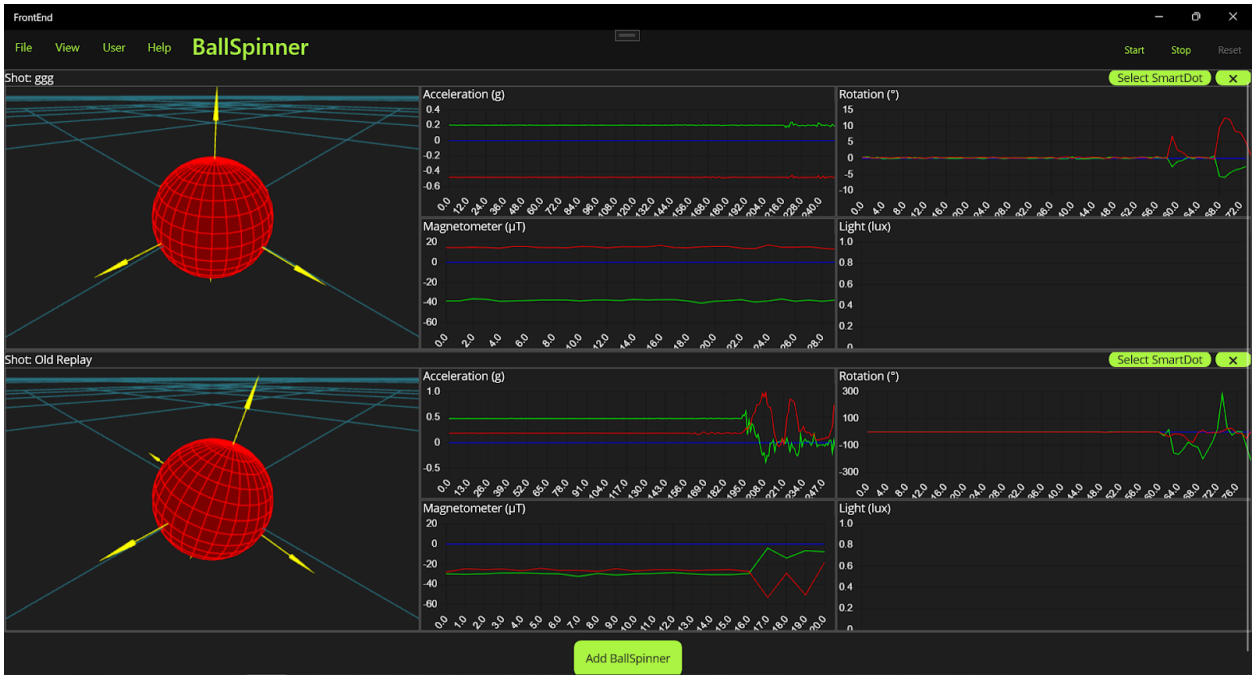


Figure X.Y.Z

The Frontend of the RevMetrix Ball Spinner consists of a .NET MAUI application and associated pages of the BallSpinner Application. The application consists of one main page as well as several smaller pages. The main page serves as a main output display for the sensor data received from the SmartDot sensors as well as a link to the application’s other pages. Figure X.Y.Z displays the main page as well as two simulations. The Add BallSpinner button at the bottom of the figure opens the Connect Ball Spinner page which allows a user to either add a new simulation or connect to a Ball Spinner Controller, both of which will appear on the main

page. Upon successfully connecting to a Ball Spinner Controller, the user will have the opportunity to select which SmartDot to use from a list of options provided by the Ball Spinner Controller.

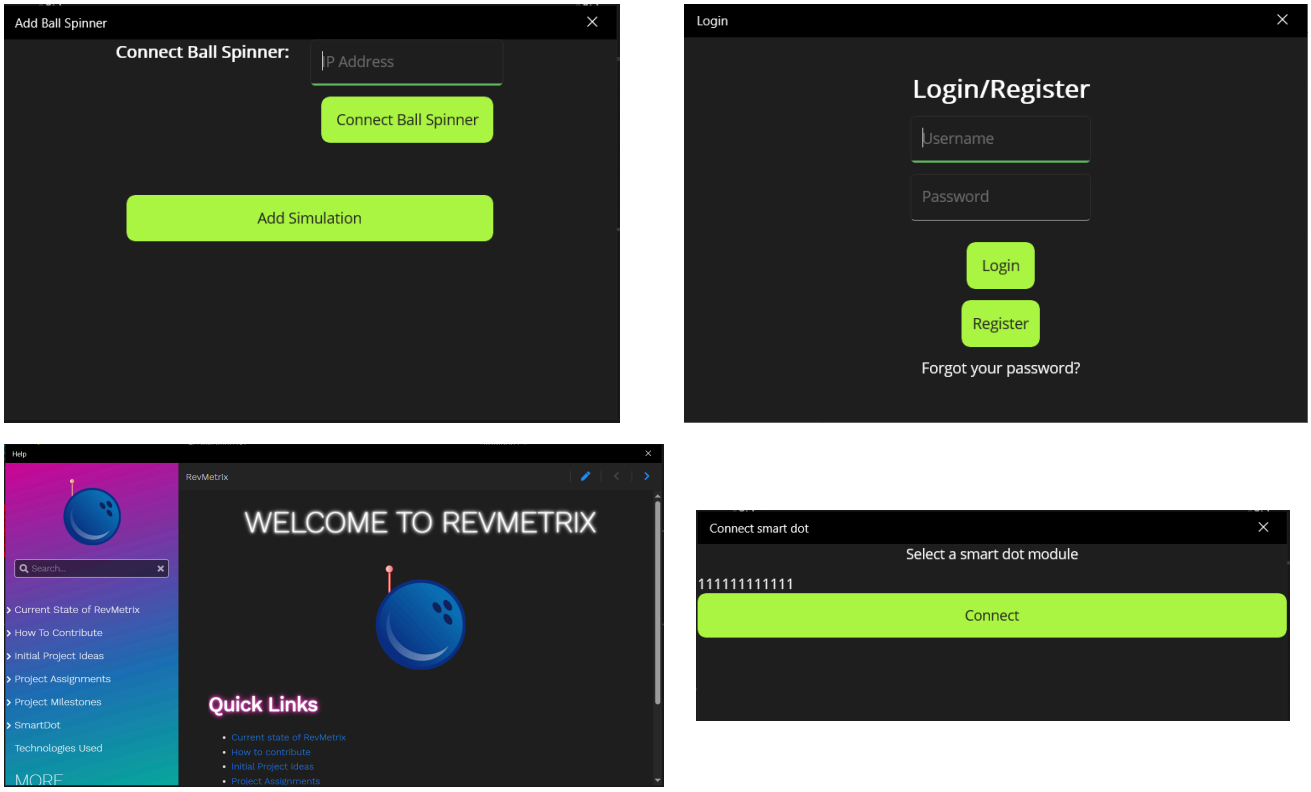


Figure A.B.C - Connect Ball Spinner Page (Top Left), Login Page (Top Right), Help Page (Bottom Left), Connect SmartDot Page (Bottom Right)

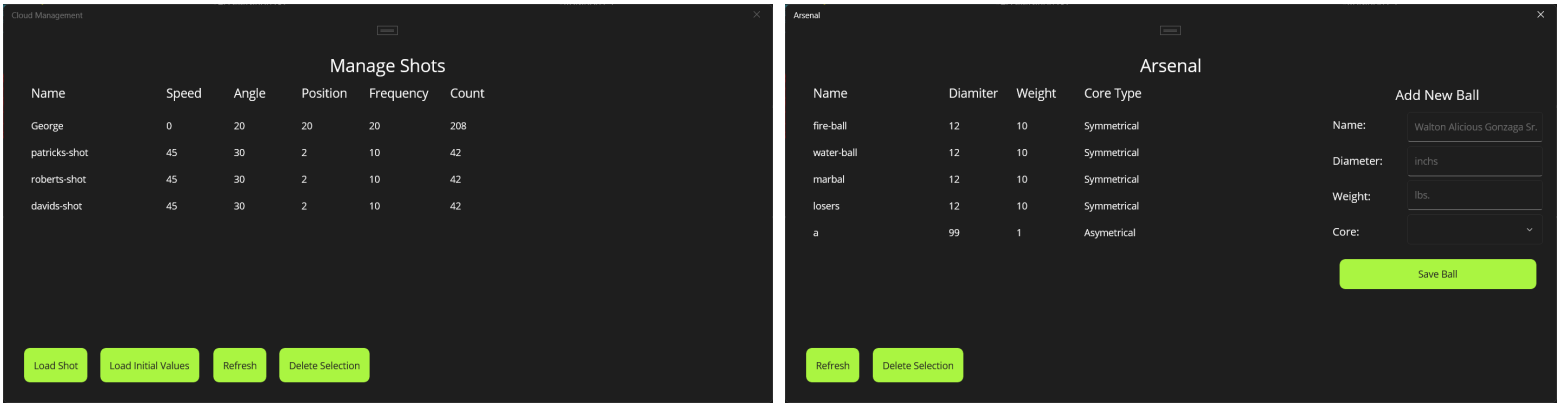
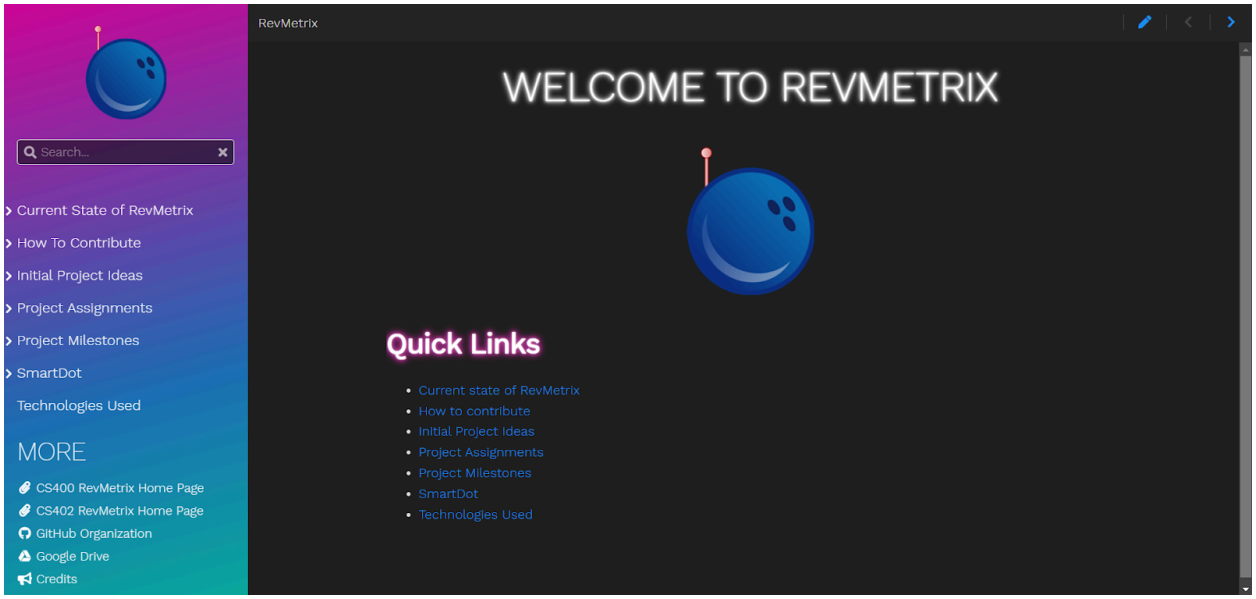


Figure H.I.J - Cloud Management Page (Left), Arsenal Page (Right)

The options bar shown at the top of Figure X.Y.Z allows users to access the other pages available in the application and some additional functions. The File tab allows users to save and load simulations that they have run, as well as open the New Shots page to prepare a new simulation. As of the current implantation of the project, the New Shots page can be opened, but is primitive in design and missing functionality. The User tab allows users to open the Login page (Figure A.B.C) and the Cloud Management and Arsenal pages (Figure H.I.J). The Help tab simply opens the wiki within the application, and at the moment there are no available options under the view tab. The Start and Stop buttons in the top right of Figure X.Y.Z function as controls for the simulation, starting and stopping respectively.

Wiki

The wiki is flushed out for the **2023-2024** RevMetrix team. It has details for the state of the project, development environment instructions, repository links, actual assignments, and a lot of useful links near the bottom of the menu.

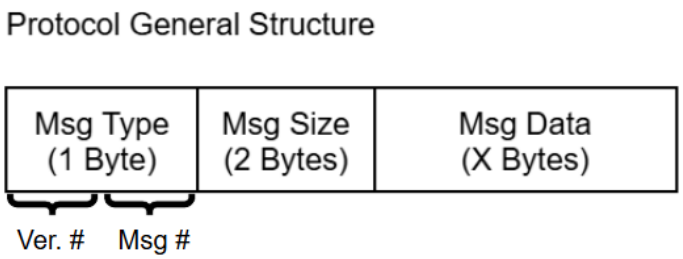


As for the 2024-2025 team, the wiki was not updated in the Fall of 2024. The wiki github for the frontend is accessible from the RevMetrix github.

Protocol

The protocol structure is designed to package messages between the Ball Spinner Controller and the Ball Spinner Application. These messages are sent through Transmission Control Protocol (TCP) to take advantage of its error detection and error correction. The structure of the protocol can be split into three separate stages: Setting up connection to the application, setting up connection to the SmartDot module, and sending data from the smartDot modules, motors, and auxiliary sensors in real-time data.

The general structure of the protocol message can be seen in below:



Protocol General Structure

The Msg Type is a byte representing which specific message is being transmitted. The byte is made of 2 bit fields, the version number (2 bits) and message number (6 bits). The version number adds future-proofing for additional versions implementing a different general message structure. The current version number is set to 0 for all current messages, and while the message number usually relates to the sequence order in which the messages are expected to be passed, case-specific messages (i.e., error messages) will start with the highest message number and

decrement. The Msg Size is a 2 byte integer representing the size of the Msg Data. The max size of any of the messages comes from the SMARTDOT_SCAN message, which, in its worst case, can have a Msg Size of 261 bytes.

To initialize the connection between the application and the Ball Spinner Controller, the application sends the APP_INIT Message, consisting of a randomly generated byte acting as a token to establish the connection. Ball Spinner Controller responds with the APP_INIT message by resending the byte. This exchange confirms on both ends that their receiving end is a device that understands the protocol while also providing a way to determine an error in transmission. The application then asks for the Name of the Ball Spinner Controller through the BSC_NAME_REQ in which the Ball Spinner Controller responds by sending the parameter in ASCII format. Below is the layout of the protocol messages related to setting up connection to between the application and the Ball Spinner Controller:

1. Setup Connection Messages

a) APP_INIT:

Msg Type (0x81)	Msg Size (0x0001)	Random Start Byte (1 Byte)

b) APP_INIT_ACK:

Msg Type (0x82)	Msg Size (0x0001)	Repeated Random Byte (1 Byte)

c) BSC_NAME_REQ:

Msg Type (0x83)

d) BSC_NAME

Msg Type (0x84)	Msg Size (2 Bytes)	ASCII NAME (Up to 255 Bytes)

Figure 2.4.2
Setup Connection Protocol Layout

To initialize the connection between the application and the SmartDot module, the application sends the SMARTDOT_SCAN message with a MAC address of 00:00:00:00. The Ball Spinner Controller will then perpetually send scanned SmartDot device name and MAC addresses in the SMARTDOT_SCAN message. When the user selects the SmartDot module to connect to, the application resends with the SMARTDOT_SCAN message with the selected name and MAC address. From there, the Ball Spinner Controller sends the ABBREVIATED_SMARTDOT_CONNECTED, which resends the MAC address of the connected SmartDot module. Below is the layout of the protocol messages related to setting up connection to the SmartDot:

2. Setup SmartDot Connection

a) SMARTDOT_SCAN

Msg Type (0x85)	Mess Size (2 Bytes)	BLE MAC Address (6 Bytes)	Name in ASCII (Up to 255 Bytes)

b) ABBREVIATED_SMARTDOT_CONNECTED

Msg Type (0x86)	Msg Size (1 Byte)	BLE MAC Address (6 Bytes)

The final stage of the Ball Spinner Protocol starts with the application sending the ABBREVIATED_MOTOR_INSTRUCTIONS message. The ABBREVIATED MOTOR_INSTRUCTIONS consists of an integer correlating to the voltage to be applied to each motor. The Ball Spinner Controller uses the first arrival of this message as a start message for the SmartDot to turn on and start taking data. After receiving the first ABBREVIATED_MOTOR_INSTRUCTIONS message, the Ball Spinner Controller will continuously send the SD_SENSOR_DATA message for each of the various sensors. Each SD_SENSOR_DATA message will contain a byte that will store the type of sensor (using an ASCII character for each sensor). Along with that, it will contain a byte for the numbered sample count for that sensor, a float that stores the elapsed time (4 bytes), and finally a float for each

Cartesian value data (4 bytes each) for a total of 23 Bytes for the sensor data. Every time a sensor receives an out-of-order frame, the Application will send a rejection message with the number required to resend. Finally, when the roll is finished, the Application will send the Stop message which will just be an Error message with the error code correlating to the stop function.

3. Sending Run Data

a) START_SMART_DOT

Msg Type (0x84)	Msg Size (1 Byte)	Sensor Sample Rates + Range 5 Bytes

b) MOTOR_INSTRUCTIONS

Msg Type (0x87)	Msg Size (0x0003)	Motor1 Speed (1 Byte)	Motor2 Speed (1 Byte)	Motor3 Speed (1 Byte)

c) ABBREVIATED_MOTOR_INSTRUCTIONS

Msg Type (0x88)	Msg Size (0x0003)	Prime Motor Voltage (1 Byte)	Sec Motor 1 Voltage (1 Byte)	Sec Motor 2 Votlage (1 Byte)

d) SD_SENSOR_DATA

Msg Type (0x8A)	Msg Size (0x0013)	Sensor Type (1 Byte)	Sample Count (3 Byte)	Time Stamp (4 Byte)	“X-axis” data (4 Bytes)	“Y-axis” data (4 Bytes)	“Z-axis” data (4 Bytes)

Ball Spinner Controller

The Ball Spinner Controller currently consists of a Raspberry Pi 4, a 7-inch display, and an electrical circuit to power the Ball Spinner. A 25V power supply is used to power the entire system, with different DC Buck Converters to power the Pi and the secondary motors that run at 5V and 12V respectively. The circuitry is housed on a breadboard sitting on top of the power supply inside a 3D-printed housing. The housing consists of a slit to slide the screen into, a hole to feed the wires for the motors through, and a detachable lid to have open access to the internal components. The figure below shows the current prototype of the Ball Spinner Controller.

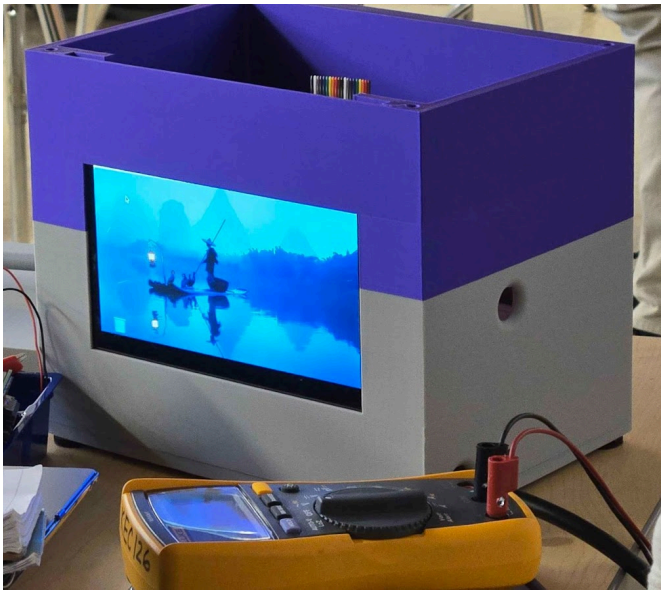


Figure 3.2.1: Ball Spinner Controller Prototype

The BSC is currently able to interface with the motors by using Pulse Width Modulation (PWM) of the GPIO pins of the Pi connected to transistors that connect to the power of the motor. Changing the duty cycle of the PWM wave directly affects the speed of the motor. The BSC plans to communicate with the SmartDot module through Bluetooth Low Energy (BLE), so while the module is still in development, the BSC communicates through BLE with the MetaMotion Module S (MMS). The MMS is a third-party 9 Degree of Freedom (DoF) module also equipped with a light sensor with its API to retrieve data from the device. The BSC communicates with the application through Wi-fi through TCP packets sent through port 8411 (chosen as the port resembles the word “BALL”).

The software is run on the Raspberry Pi 4, all written in Python. and must be started manually through the command line. From there, the BSC displays its IP address for the application to connect to. Once connected, the device performs an additional custom handshake

and then listens for all incoming messages from the application. When the Application prompts for the list of SmartDot devices, the BSC starts scanning for Bluetooth devices with the UUID matching that in its stored list. This parses the Bluetooth devices of nearby electronics and only displays local MMS modules currently. From there the user asks to connect, and the BSC pairs with the MMS. When the START button is pressed on the application, the BSC sets the sample rates of the accelerometer, magnetometer, and gyroscope to 100Hz, 25Hz, and 100Hz respectively. Data begins being taken from the MMS, and each packet received from the MMS is parsed and sent to the application. When motor instructions are received by the application, the BSC runs a calculation on what to set the Duty Cycle to based on the voltage sent from the message. When the STOP button is pressed, the motor voltage is set to 0, stopping the movement of the bowling ball.

Embedded in the BSC is a SmartDot Emulator, designed to allow developers to test the application without the need for the MMS module. This emulator reads data from a predefined CSV file, and based on the sent sample rate of the user, sends each value of the specific modules at that speed. To connect to the SmartDotEmulator, the BSC must be set in Debug mode, which sends a MAC address of 11:11:11:11:11:11 to the application.

Another feature for the developers of the BSC is a Command Line Interface (CLI), used to interact with each mechanical device manually. This allows users to connect to the MMS to toggle the lights on the chip and control each of the 9DOF modules separately. To connect to the motors, the CLI allows the developer to manually set the GPIO that the motor is connected to and manually change the duty cycle of the PWM signal to that GPIO port.

Ball Spinner

The BallSpinner currently is a set of rollers on a wooden platform. The BallSpinner allows for two orthogonal directions. The bowling ball sits on the rollers spinning in one direction. The motor stand is next to the BallSpinner to enable the motor to drive the wheel to remain in contact with the bowling ball. However, the coefficient of friction between the rollers and the bowling ball is high, limiting the ability to spin the bowling ball. With this current design, we cannot ensure that the bowling ball has no slippage, which would affect the data sent to the Raspberry Pi. The wheel used by the motor to interface with the bowling ball currently works for testing, but the surface material of the wheel still restricts the bowling ball from spinning due to the high coefficient of friction between the wheel and the bowling ball.

Current Ball Enclosure

A Carter brand 30V DC shunt motor was used for testing. The shunt motor's high-speed axle could meet the wheel speed requirements but was unable to spin the bowling ball when applied. The motor does not have enough torque to drive the bowling ball to the desired speeds due to the coefficient of friction between the rubbery surface of the wheel and the bowling ball.

The motor stand is currently manufactured out of PLA to hold the motor steady for testing. The base of the motor stand is shaped asymmetrically for balance, ensuring that the stand does not tip over. However, the wheel narrowly reaches the bowling ball due to the positioning of the motor stand and the BallSpinner. The motor does not smoothly fit into the motor stand which may cause cracks.

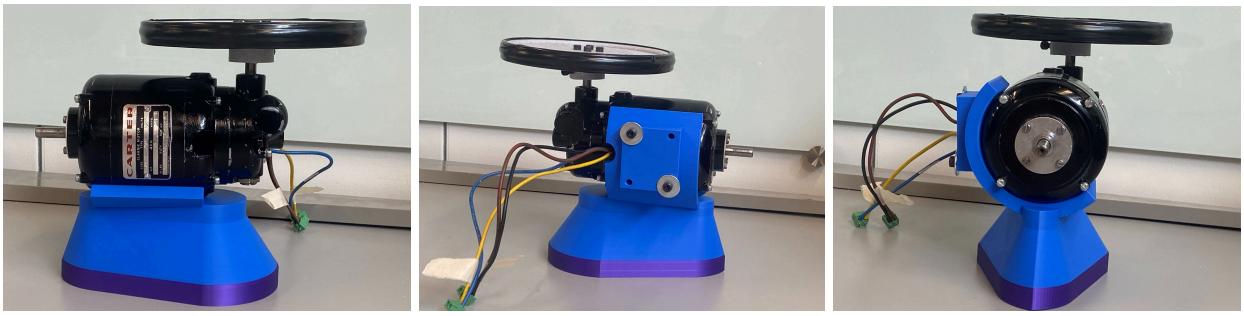


Figure ### Current Motor Stand with Motor Mounted

Mobile Application

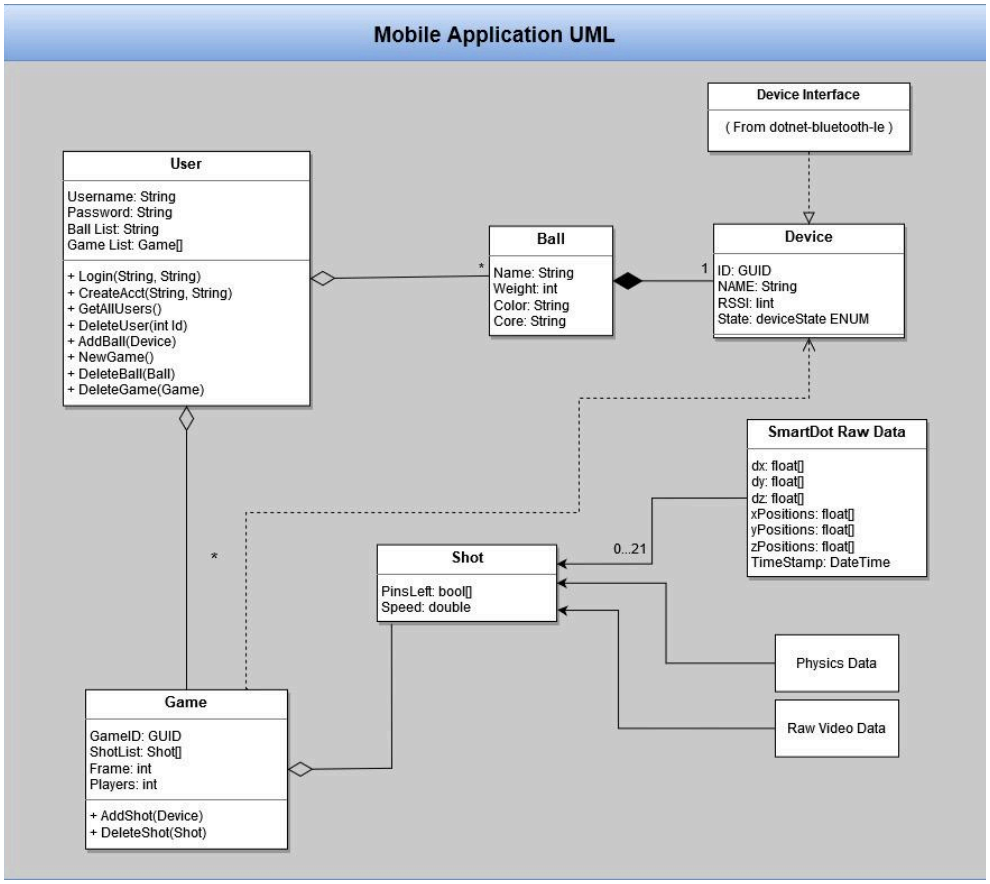


Figure 4.10: Mobile Application UML Diagram

The previous iteration of the mobile application was developed on a now out-of-date version of .NET MAUI as well as add-ons that no longer receive software support. As a result, the mobile application team has decided to rebuild from scratch with the additional knowledge of the Ball Spinner testing equipment. The previous team worked on the Local Database, SmartDot Communication, Data Graph, File Reader, and Frontend Interface. In the Frontend, they had designed pages for Login, Home, Bluetooth, Scan, Graph, Create Account, Simulation, Camera, Games, Ball Arsenal, and Notes. They had a variety of doneness on these Pages and features.

Figure 4.11 below shows the schema of the local database. Whenever a user signs in or creates an account, their information is sent to the local database including their username, User

ID, time of last login, and Ball-list. The Ball-list deserializes into individual Ball objects using a JSON reader. Each bowling ball has information such as the Ball ID, weight, color, and specified device, which is set to Null by default.

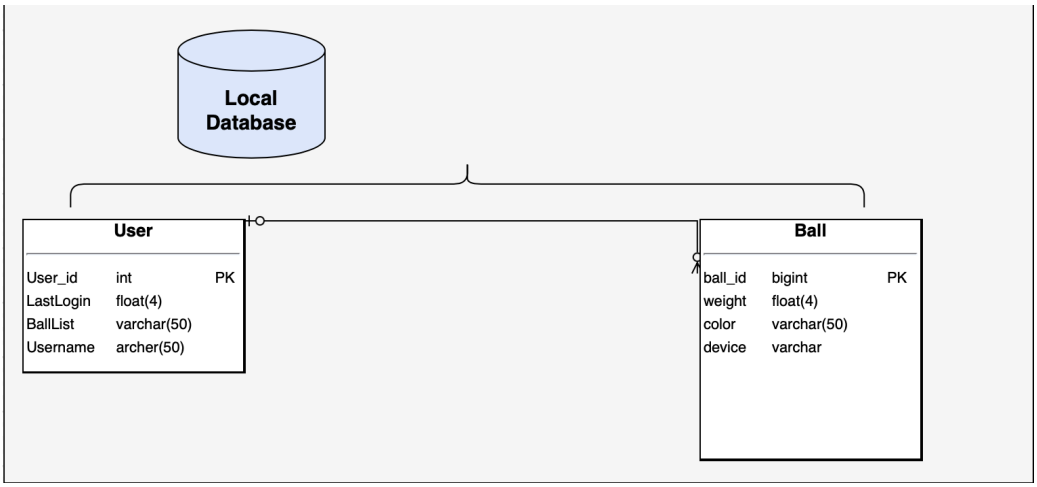


Figure 4.11: Local Database Schema

As shown in Figure 4.13, the login page features text entry fields for a username and password, as well as a create account option for first-time users. This page interacts with the web server via the web API to verify that users are using the correct credentials before allowing a sign-in.

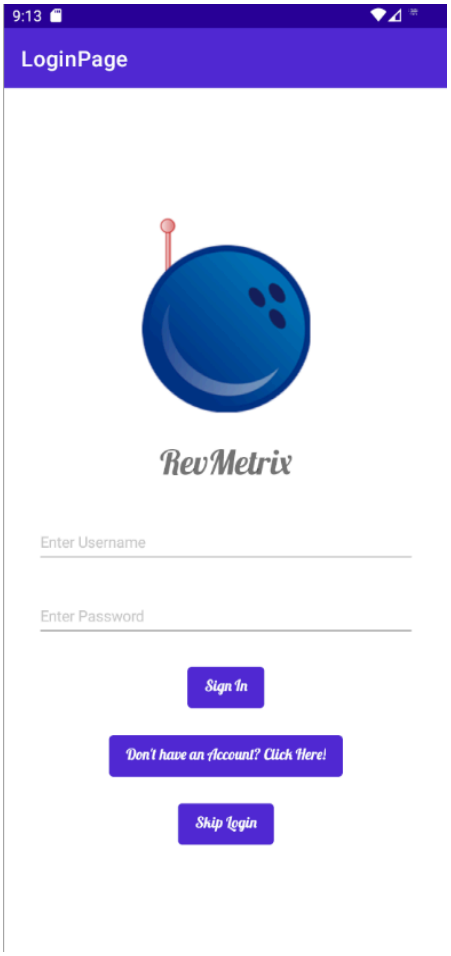


Figure 4.13: RevMetrix Login Page

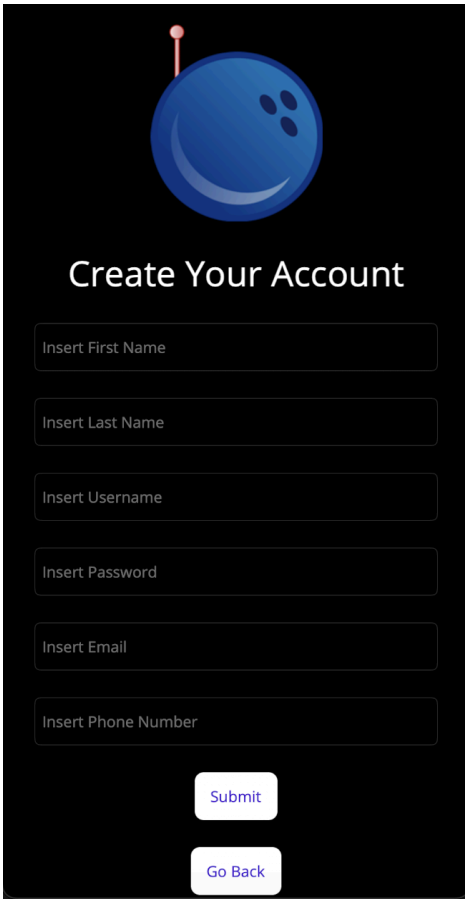


Figure 4.20: Create Account Page

The Games Page is for users to track their shots during each game. Users can start a game and mark each frame as either a strike, spare, or miss. These options are presented as buttons seen in the figures below. On this page, SmartDot data will be matched to its frame of origin, allowing users to track their stats throughout the game. The page shown in Figure 4.23 displays the user's game, keeping track of the score of each shot, and frame score, and having shortcut buttons for strikes, spares, fouls, and gutterballs.

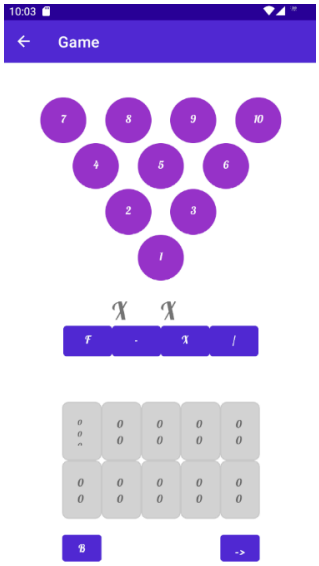


Figure 4.23: Game Page

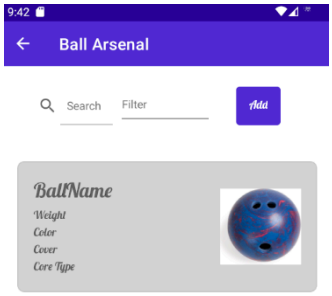


Figure 4.24: Ball Arsenal Page

Requirements (include how each will impact current design)

Simulation

Andrew, Carson, Patrick

The Simulation Team is responsible for making the 3D simulation and data charts work smoothly in the RevMetrix project. To get ready for the Capstone Expo and meet client needs, the team need to make a few big changes:

- The team is replacing Three.js (a JavaScript tool) with OpenTK (a C# tool) to make the simulation faster and smoother. This change will reduce lag and help everything work better inside .NET MAUI.
- Since Chart.js is a JavaScript tool, it was slowing things down. The team are switching to LiveCharts2, which works better with C#. This will make data charts update faster and work more smoothly with the rest of the app.
- The user should not get stuck or experience bugs while using the simulation. The team will add tools to help us find and fix problems faster.
- Users need a way to save and share their simulation data. The team will make sure data can be stored locally and uploaded to the cloud if needed.

Additionally, the Unity simulation could use some major overhauls.

- First, it needs updates to include parameters that can be imported from the Smartdot module, so the simulation will show accurately what the shot will look like on a real lane.
- The simulation needs to be embedded into the ball spinner application, so that the user can see the results of their fabricated shot on the lane.
- Outputting data needs to be changed from a CSV file to the database. If it is redundant information, such as information that would be measured through the Smartdot module, it should output the data locally, as to affirm correctness.

Backend

The backend, encompassing the Cloud infrastructure, the API server, and backend logic for the Ball Spinner application will need to implement the following requirements in order to ensure a successful demonstration at the Capstone Expo and meet client expectations:

Milestone 1

- The SQL server security must be updated to prevent external brute-force attacks, impacting local API development and testing since they rely on the cloud database. Additionally, the team must remove sensitive environment variables from the codebase.
- There must be a local implemented in the application to replace the local shot feature. At this point, it will serve as a cache and will not be able to be accessed without connecting to the cloud database.
- The input parameters and the variables associated with our kinematic model for motor instructions must be determined.
- Sample rate and range configuration for sensor modules should be saved in the database.

Milestone 2

- The user experience should be smooth with minimal to no bugs.
- Motor instructions should be able to be generated.
- Shot data should be related to the specific ball data that was used for the shot.
- Shot data should be related to the specific sensor used for the shot. This should also include a date and time that the shot took place, and comments allowing the user to describe their shot.

- The CI/CD must be updated to ensure all pushed code is validated to work before being published. This will add a ‘test’ job to the github action deployment code.
- The local database should be fully integrated in the application, and any feature relating to local shots from last semester should now use the local database instead of CSV files.

Milestone 3

- The application must efficiently generate motor instructions and have the ability to send them to the Ball Spinner Controller. These instructions should accurately reflect bowling physics to ensure a valid SmartDot testing environment.
- The database must support shot types beyond ‘Simulated Shots’ by integrating the 2023-2024 RevMetrix schema. This allows Professor Hake to access all relevant data from one database, with minimal impact except for added foreign keys in the user table.
- There should be resources available to save auxiliary sensor data associated with a shot. This should also apply to locally saved shots.
- There must be a cloud backup feature that ensures that both the local and cloud databases are in sync. This will allow the use of the local database without being connected to the cloud.
- An interface should be created for the mechanical team to calibrate the Ball Spinner, allowing retrieval of auxiliary sensor data and direct motor control. It should display both actual sensor data and expected behavior through interpolation.

Ball Spinner Application Frontend

The majority of the application was designed and implemented in the previous semester. However it was implemented in a state of the project where continued development on other aspects of the project may require significant additions or changes to the application. Accordingly, application frontend development of the Ball Spinner application will focus on support for other teams and improvements to the existing application.

Milestone 1

- Finalize SmartDot Settings Page - A new page created to allow a user manually select sample rate and range for the SmartDot sensors. This will be accessed through a button attached to each Ball Spinner output.
- Local Database - The backend addition of a local database will require changes to C# side of the cloud management page and arsenal page to draw from the local database when a user is not logged in.
- Offline Work - Allowing a user to use the application without the need to login. The login page will still open on login, but currently it cannot be closed, and all buttons and views are disabled unless the user is logged in.
- Begin Migration to .NET 9.0 - To avoid the continued development of the application from becoming hindered by an outdated or unsupported framework it will be necessary to update from .NET 8.0 to .NET 9.0. At the moment, it is unclear how drastic the changes to the application will have to be once it is moved to .NET 9.0.

Milestone 2

- Revise Initial Values Page - The initial values page will be changing from a series of text inputs to an adjustable graph for defining the curve of the bowling ball traveling down a lane. This will require a LiveCharts2 view, so the page content will need to be updated and a viewmodel created.
- Update ShotsViewModel - The changes to simulation and graphing libraries will require significant changes to the viewmodel that implements the outputs on the main page.
- Full transition to .NET 9.0

Milestone 3

- Initial Values Page - Significant changes will have to be made to the current initial values page to allow it to be functional.
- Cleanup and Tuning - This will be a process worked on over the course of the semester and involves bug fixes and improvements to GUI of the application to improve usability. This will mostly not involve changing the structure of the application, though it may also include cleanup of the application's code.

Wiki

- Maintain and update the RevMetrix Wiki
- Ensure information is accurate and current
- Assign sections for updates based on project progress
- Reorganize content for clarity and ease of use

Ball Spinner Controller

For Milestone 1, the team plans on having the Ball Spinner Controller send data from all 9DOF modules and the Light Sensor from the SmartDot module. This adds interfacing with the ambient light sensor on the MetaMotion module, the last sensor anticipated to be implemented on the SmartDot module. Along with the previous task is sending configuration settings between the Ball Spinner Application and the Ball Spinner Controller. This will remove the predetermined configuration embedded in the Ball Spinner Controller and allow the user to set these parameters of the SmartDot. At this point, the team will ensure all protocol messages that are designed to allow the user to connect to the SmartDot, start the motors, and take sensor data will be implemented on the Ball Spinner Controller side. The final task for this milestone is to have a preliminary design of the Graphical User Interface (GUI) to display, giving users easier access to the current state of the Ball Spinner Controller.

For Milestone 2, the Ball Spinner Controller will be equipped with amperage readers so the user has feedback on the power distribution of electronic components like the motors. The new primary motor should be in-house by this point, so the Ball Spinner Controller will be able to interface with it along with its encoder. This will allow the user to once again control the primary access of the Ball Spinner, and move the bowling ball at speeds unobtainable from the previous state of the project. All additional changes to the protocol will also be implemented, and all error messages will be designed for further implementation. Finally, designs for both the final housing and the Printed Circuit Board (PCB) will be finalized to be ready to be shipped for the Capstone Expo.

For Milestone 3, the Ball Spinner Controller will be able to interface with the secondary motors, giving the user control over the three orthogonal axes of the Ball Spinner. For feedback, the Ball Spinner Controller will be able to interface with the auxiliary motors of the Ball Spinner Controller, sending this information to the Application. At this point, the controller will have implemented all foreseen error messages. Depending on lead times, the PCB will be implemented along with the new housing of the Ball Spinner Controller.

For the final Presentation, optimization to the protocol messages will be to send sensor data from the SmartDot module. This will allow for lessening the amount of unneeded data sent to the application and improve the performance of both sides of the socket. At this point, all documentation will be completed and accessible to developers with proper installation instructions, protocol setup, and the steps on how to add interface capabilities with the SmartDot.

Ball Spinner

The Ball Spinner must be able to rotate a bowling ball to RPMs that match a real bowler's shot data. It needs to replicate the axis of rotation's lateral and vertical drift to mimic lane behavior. It must also measure the bowling ball's speed externally, so the SmartDot has data to compare against. To ensure accuracy, the system must have no slippage.

Milestone 1

- Design parts interfacing with motor
- Design motor mounts
- Completed Bill of Materials

- Design ball enclosure
- Select motor
- Spec Power supply

Milestone 2

- Prototype major spin axis
- Machine necessary parts
- Model Ball enclosure
- Model parts interfacing with motors
- FEA & verification of designs
- Magnetic shielding for motors

Milestone 3

- Machine necessary parts
- Plug bowling ball holes

Mobile Application

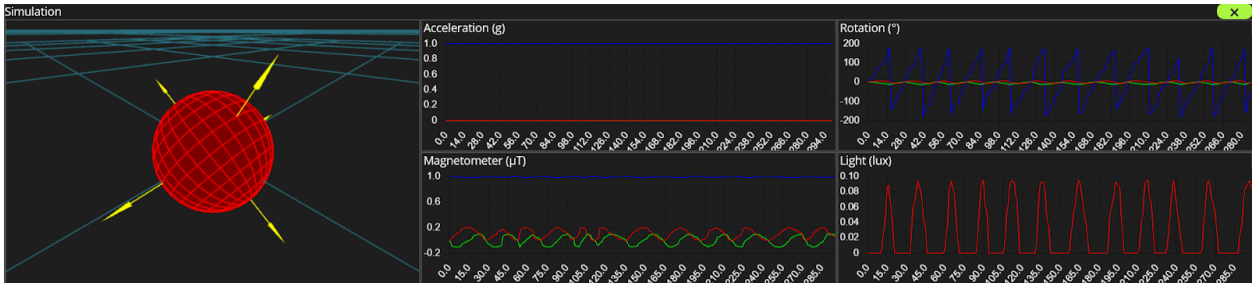
The mobile application must be able to collect user data regarding pins knocked down in a shot, and associate it with a bowler, ball, game, session, and event. While game, session, and event are not core functionality for the data analysis of the RevMetrix project, the utility in organizing the project's database (and pleasing our client) will be useful.

Group Proposal

Simulation

The simulation team has decided to switch from Three.js and Chart.js to OpenTK and LiveCharts2 for better speed and performance. Our current setup, which runs on JavaScript, has been too slow when transferring data and rendering 3D graphics. Since the team are working inside a .NET MAUI environment, it makes sense to use C#-based tools that integrate better with our project. OpenTK and LiveCharts2 will allow us to take full advantage of C#'s performance while keeping everything in the same development framework.

One of the main reasons for this switch is that C# runs faster than JavaScript, especially when handling complex data and 3D objects. OpenTK will make our 3D simulations smoother and more efficient, improving rendering times and reducing lag. LiveCharts2 will help us create better charts with real-time updates and more customization options. Since both of these tools are built for C#, they will work much better with our .NET MAUI setup, making development easier and more efficient.



Backend

The Backend team proposes major updates should be applied to the Ball Spinner Application, the Ball Spinner Cloud, and the server configuration both locally and on the Digital Ocean droplet to ensure all of the backend requirements are met.

Firstly, the motor instruction protocol will be developed to provide a stable testing environment for the SmartDot module. The development of this protocol will require users on the frontend to determine a start point, an end point, and an inflection point on a bezier curve. This bezier curve will allow the user to map out a specific path with respect to RPM over time. From these points the backend team will generate an array of time points and an associated RPM that will represent the motor instructions to send to the Ball Spinner Controller. The motor instructions will also require communication between the Ball Spinner Controller team and the backend team to establish a ramp-up protocol to ensure that the motor is set to its required initial RPM before the shot commences. Once this is determined, a structure for the protocol will be developed, where it will then be implemented in code and sent to the physical Ball Spinner device. Once instructions are sent to the Ball Spinner, any inconsistencies and issues with the instructions will be flushed out through the physical testing of their accuracy.

Next, the server configuration should be altered to provide enhanced security, particularly for the SQL server. For this issue, the team proposes that the SQL server should no longer be publicly accessible, instead it should only communicate with the API server through a Docker network. Doing so will ensure that the API server can still interact with the database, but anyone outside of the network will not be able to log in to the SQL server, preventing a brute force attack. This will have a ripple effect on our local API server implementation that the team uses for development and testing, and for this issue the team proposes that the configuration for a

local docker SQL server that contains the same structure as the cloud database be implemented. This way, the team can ensure that the exact same SQL server version and configuration is used, synchronizing local API development with the API server. With this proposed fix, the environment variables used for SQL server connection in the cloud's codebase will need to updated to point to the local SQL server, where the password will selected to one that is not relevant to any of our systems, and any account that uses the current password that is exposed within the environment file need to be changed. Additionally, relating to the database, the database schema will be updated to be more modular, including merging it with the database schema worked on by last year's team.

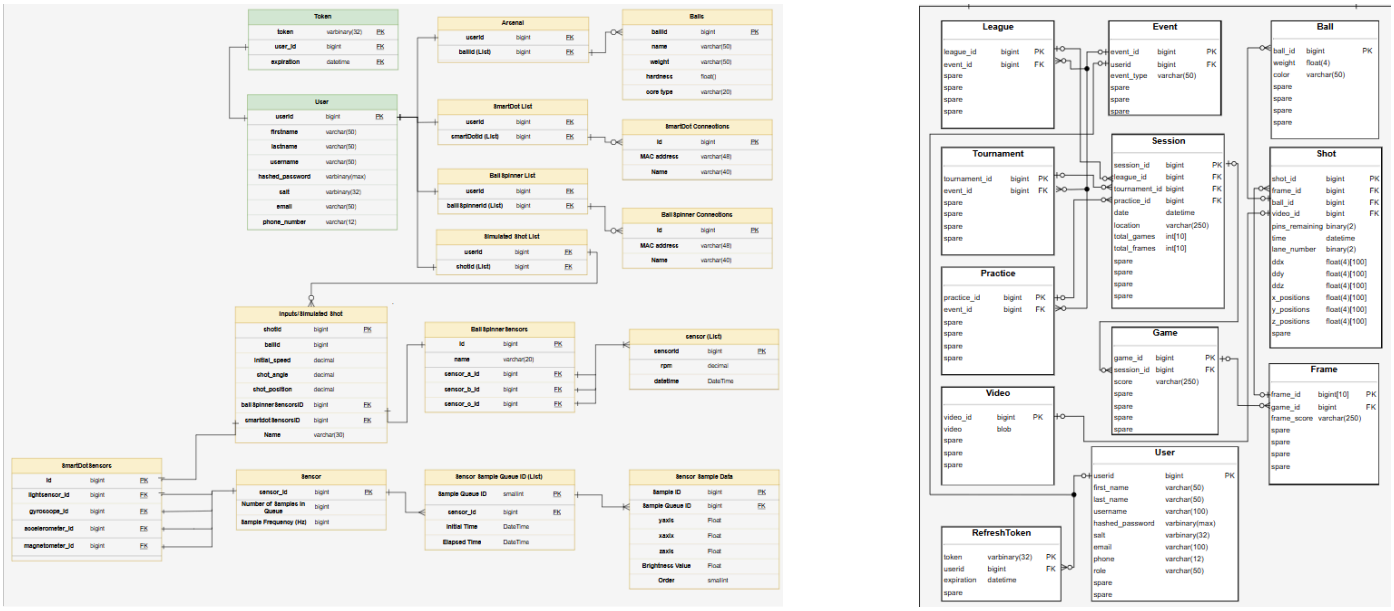


Figure A.B: Current and Former Database Schemas to Merge

As shown in figure A.B both database schemas are user oriented, so all that needs to be done is to recreate the tables and relationships from the old database schema and incorporate them into the new schema. This will allow the old relational schema to coexist with the new schema allowing for the storage and retrieval of simulated shots and regular shots.

Furthermore, the following performance issues with the Ball Spinner application will be addressed to provide users with a smoother experience: The SmartDot data coming in from the Ball Spinner Controller will be written to a memory mapped file instead of a CSV file for faster, less expensive write speeds, where the file can then be written to the hard drive if the user chooses the save the shot locally.. The issue resulting in packet loss for received TCP packets from the Ball Spinner Controller will be resolved so that no valuable data is in transmission. The bug that results from being able to create two Ball Spinners that are named the same thing and causes the application to crash will be resolved in order to ensure all of the current edge cases are addressed.

The local shot feature will be updated in the following ways: A local database will be implemented on the Ball Spinner Application that exactly resembles the cloud database structure and contains all of the same data as the cloud does. The specific database engine will still need to be determined, but most likely it will be MSSQL server in order to match that of the cloud. The system will function as if it is always connected to the cloud database, ensuring a seamless experience even when the user is offline. This eliminates the requirements related to the local shot feature, while still allowing users to save shots locally and access them without the internet. It also allows for locally saved shots to contain foreign keys containing input data like input parameters and bowling ball data relating to the shot. All uses for local shots will need to be removed from the application codebase, while any useful features implemented for local shots, like the shot replay feature, will need to be implemented for shots stored in the local database. An important consideration for this addition is implementing a cloud backup feature. This will ensure that any data saved to the local database while offline is automatically synchronized with the cloud once an internet connection is restored. There will also need to be local API endpoints

that get called whenever data is stored to ensure the local database is in sync with the cloud database. Also, there will be local endpoints that will be called when offline as the team needs endpoints for retrieving local data as well.

Final considerations for the application backend include: updating the CI/CD to test all code pushed to the cloud API codebase and only deploy it to the Digital Ocean droplet if all tests pass. Updating the database schema/API endpoints to relate the specific bowling ball used for a shot, as well as being able to store auxiliary sensor data associated with a shot in the database. The team will also relate shot data to the specific sensor module used, which will also include a date and time entry and any relevant comments relating to the specific shot. Furthermore, the team will also need to save specific sensor module configuration related to a shot (i.e. sample rates and ranges) to ensure all relevant information related to data collection is documented. Implementing the backend methods and application side protocol for the Ball Spinner calibration page. This page will be accessible only to mechanical team members with the appropriate permissions. It will enable them to control all three motors on the Ball Spinner by setting an initial condition, an end condition, and a time differential. The system will then use auxiliary sensor data to show how the Ball Spinner actually responds, while interpolated data will illustrate the expected response over time, providing a real-time method of determining calibration. The team will implement this feature for as many motors as the mechanical team can get up and working to send auxiliary data. However, this feature will require a significant commitment from other team members. Given our current workload, implementing the calibration page would be beneficial but not essential.

Ball Spinner Application Frontend

This semester's development for the Ball Spinner Application Front End of the application will focus on improvements to the existing system and adding new features and pages based on the needs of other teams. Additional features and pages will include tasks such as implementing the initial values page and developing a page to allow users to configure sample rate and range of a connected SmartDot Sensor. Other improvements to the application will involve bug fixing for any problems that arise during development and visual improvements such as improving the visibility or allowing a user to customize parts of the GUI. The ultimate goal of the Front End Team is to improve the stability, usability, and visibility of the application in preparation for the upcoming Capstone Expo.

Wiki

The wiki team will be responsible for not only fixing the information that is currently on the wiki, but also keeping the wiki up-to-date with current information. This means that the entire time the RevMetrix team is working, the wiki team will update the wiki with current, correct information about each team and their activities. The wiki team will have to communicate with the other teams in order to update the information as accurately as possible.

Protocol

The protocol will continuously be changed throughout the course of the project, and the first proposed change is to remove all abbreviated messages. These were made as temporary

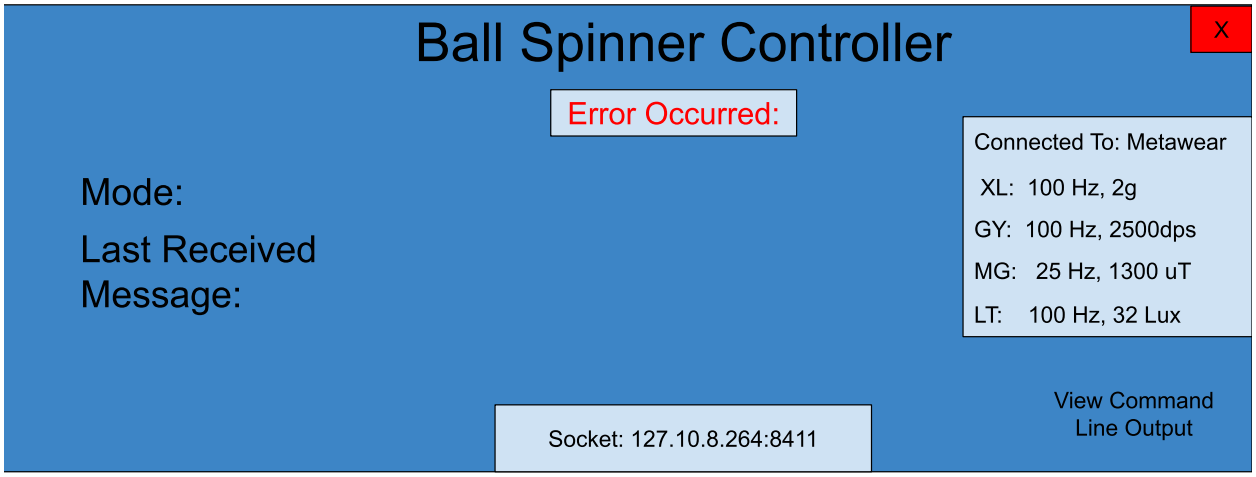
substitutes for more in-depth protocol messages that were unable to be implemented based on the progress of the project. Those protocol messages will be implemented, along with a revisal of the naming convention for each protocol message to remove the ambiguity of the message's purpose. This naming convention includes adding directional communicators to the beginning of each message, describing both which device sends each message and which device receives each message.

Error messages will also be implemented, accounting for all found edge cases that cause either the application or ball spinner controller to crash. This currently include out of order messages, unable to connect to the SmartDot when prompted, attempting to connect to the SmartDot multiple times, or if the TCP socket was closed unexpectedly. All of these changes and additional messages will be properly documented, describing the expected data in each byte and any intention behind some of the variable-size messages.

Ball Spinner Controller

Several further implementations will be made on the Ball Spinner Controller, starting with removing functionality of the abbreviated messages and implementing the messages those were substituting for. The Ball Spinner Controller will expand on communication with the MetaMotion module by adding the ability to read the ambient light sensor and send its received data along with the data from the other 9Dof modules. Further communication will be implemented with the mechanical components, being able to control the new motors along with the auxiliary sensors that will be implemented on the Ball Spinner. To display important

information that the Ball Spinner Controller easier, a Graphical User Interface (GUI) will be designed to read the current state of the controller, the latest incoming message from the application, the information relating to the connected SmartDot, and its IP address. A mockup for the Ball Spinner Controller GUI can be seen below



Other aesthetic changes to the Ball Spinner controller include designing a Printed Circuit Board (PCB) for the Ball Spinner Controller that will house the electronic components instead of the breadboard. The board will still host all power distribution components needed for all of the electronic devices in the Ball Spinner. This design change also demands a redesign of the housing for the controller, keeping the same features by adding more security to the internal components and “plug and play” ports for the external devices that need to connect with the PCB.

Ball Spinner

The ball spinner team will move to a new design to minimize slippage. This design will use polyurethane in a clamp to grasp the bowling ball. The polyurethane in the clamp has a high

enough coefficient of friction to ensure that the bowling ball has no slippage. The polyurethane also has a hardness harder than the material of the bowling ball to combat the compressive and shearing forces so that the clamps do not wear. That clamp will either be spun directly by a motor or through a gear or pulley system. The team has calculated that 35 lbs of compressive force will create enough static friction force to ensure the bowling ball will not slip or shear within the system throughout the testing process. Once everyone is satisfied with the performance of this first rotation direction, additional orthogonal directions will be added. The clamp system will be hoisted on a platform with other motors set to it. One will be built to rotate the platform, while the other will tilt the platform.

Mobile Application

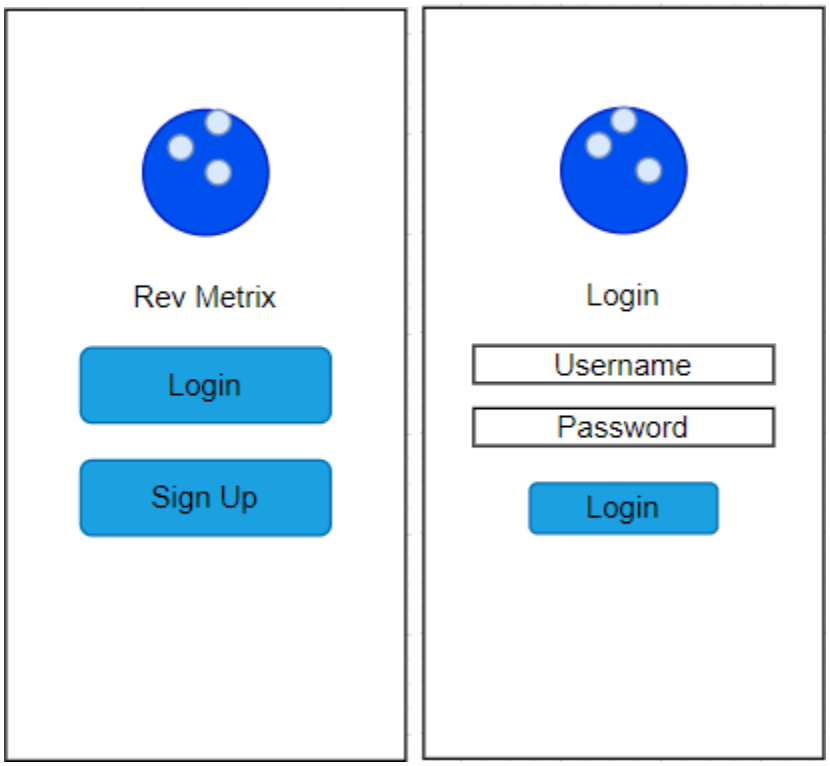
The mobile application will consist of pages for user account management, a list for the user's games, shot data entry, and BlueTooth connection management. However, as user accounts are not critical to the core function of data collection, this page will be left as a nonfunctional framework.

The Main page will provide navigation to access every other feature of the application. It will only allow access to certain pages if the user has logged in. The user will be able to login or sign up to start. Then they will be able to track their games and connect the smart dot module.

The Bluetooth page will allow the user to connect a smart dot module to their phone. This will be used to show analytics taken directly from their bowling ball.

The games list and shot data entry pages will be able to store and retrieve data from the device's local database. The user will be able to select an existing game to view past shots, or create a new game to record new shot data. Shot data will include the pins left standing, if the

shot was the first or second ball, and which ball was used. This can also be associated with a particular event, or left unassociated if it was free practice.



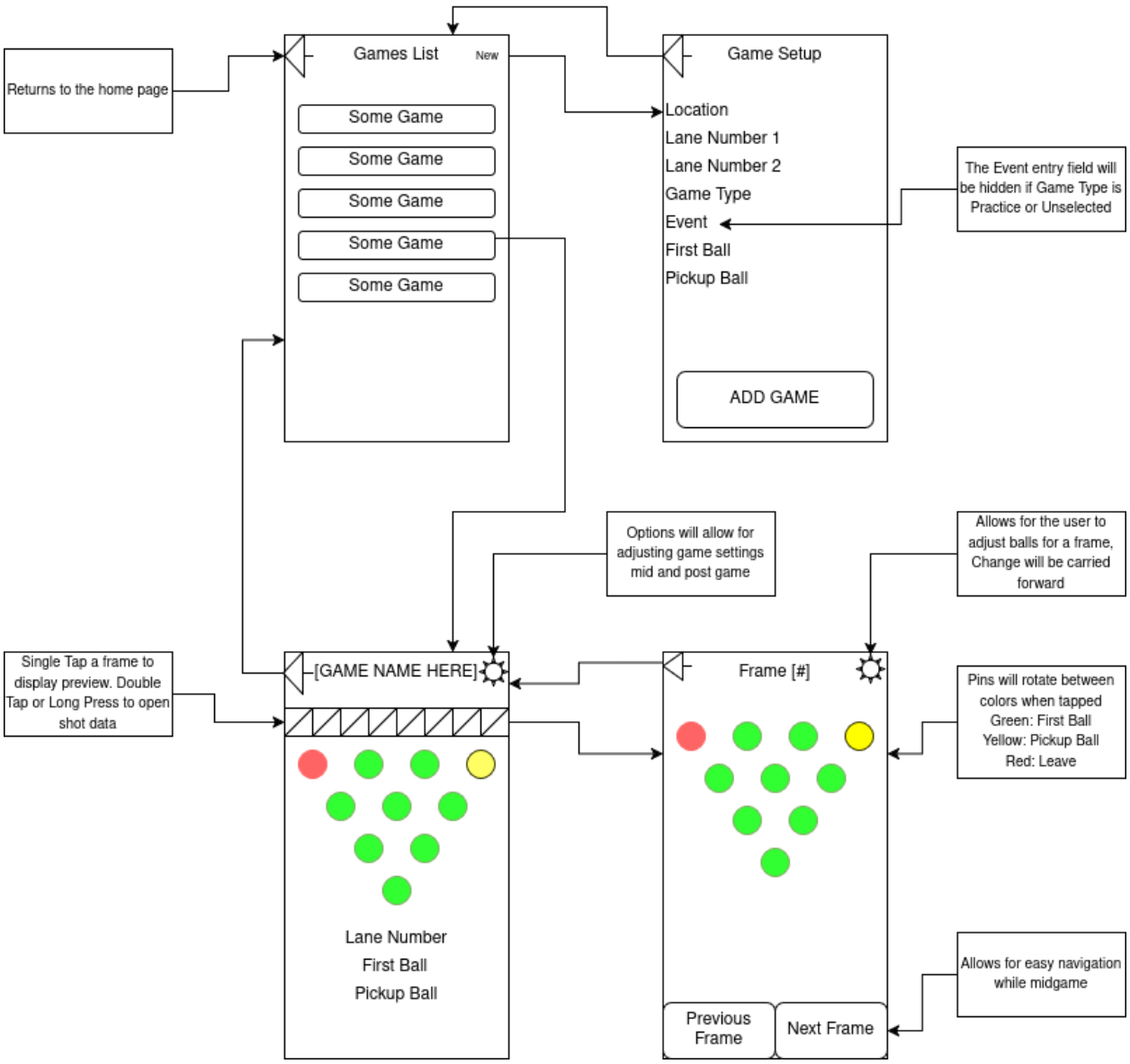


Figure #X: Mockup and Navigation for the Game UI

The local database will act as an intermediary between the application and the cloud database. Its primary functions are to cache data prior to making cloud inserts to allow for fixing user error, and to allow for offline functionality of the app.

Technology Stack (include justification)

Simulation

- .NET MAUI – This is the main framework the team uses to build the app. It allows the simulation to work on Windows, macOS, iOS, and Android using one codebase. This makes it easier to develop and keeps everything in one place.
- OpenTK (Replacing Three.js) – The team is switching from Three.js (a JavaScript tool) to OpenTK, which is written in C#. This makes the 3D simulation run much faster because the team doesn't have to translate JavaScript into C#.
- LiveCharts2 (Replacing Chart.js) – Instead of using Chart.js (which is slow because it's written in JavaScript), the team are switching to LiveCharts2, which runs in C#. This makes our charts update faster and work better with the rest of the program.
- Unity - This is used for the accurate simulation of the shot. The current implementation of the Unity simulation can be found in the Technical Report from Fall 2023. The team will be implementing this simulation into the ball spinner application for accurate shot simulation.

- UnityUaal.Maui - This is a library that allows Unity to embed into MAUI programs. Through UnityUaal, the team will be able to embed the accurate shot simulator into the application.

Backend

- .NET & .NET MAUI – Chosen for its rich development tools, cross-platform capabilities, and seamless integration with APIs, making it ideal for both the Ball Spinner application and the API server.
- Docker – Provides isolated, consistent environments for development and deployment, ensuring the API and database run reliably across different machines without complex setup.
- Digital Ocean – Offers scalable, cost-effective cloud infrastructure with easy deployment and management, making it a practical choice for hosting the API and database securely.
- GitHub Actions – Enables automated testing, building, and deployment within the same repository, ensuring smooth CI/CD workflows without needing external tools.

Ball Spinner Application Frontend

- .NET & .NET MAUI - Application framework using C# and .XAML. It was chosen to allow for development of an application for Windows, Mac, iOS, and Android.
- OpenTK - Part of the new simulation implemented by the simulation team, used for 3D ball simulation.
- LiveCharts2 - Part of the new simulation implemented by the simulation team, used for live charts to display output data received from the controller.

Wiki

- Hugo - a fast and flexible static site generator. Hugo allows the team to create and manage documentation efficiently, ensuring that updates are quick and easy to implement.
- Docker - Docker helps keep the environment consistent across different systems, making it easier to manage updates and ensure reliability.

Ball Spinner Controller

- Raspberry Pi 4: Used to interface between the Ball Spinner Application, the motor, and the MetaMotion S. Data and commands can be sent between these three technologies..
- Python: Programming Language with many libraries that support the Raspberry Pi 4.
- TINA Cloud: Design and simulate electronic schematics to improve design efficiency. Helps save money and hours preventing manual building/testing.

- MetaMotion S: Acts as a SmartDot module. This module allows us to connect via bluetooth to our Raspberry Pi. We can then send data (from the accelerometer, gyroscope, magnetometer, and ambient light sensor) via Bluetooth from the MetaMotion S to the Raspberry Pi.

Ball Spinner

- Onshape: Browser run, cloud-based, 3-D modeling software allowing real-time collaboration. The software is able to export in file types that can be read by SolidWorks and 3-D printers.
- SolidWorks: 3-D modeling software that allows for CAD and finite element analysis (FEA) on modeled parts. G-code can also be generated via Solidworks and downloaded to a USB drive.

Mobile Application

- Android/ IOS Emulators: Virtual representations of mobile devices that allow developers to test their apps on different screen sizes and operating system versions without needing a physical device.
- .NET MAUI: A framework that lets you create apps for mobile and desktop across multiple platforms. You can use C# and XAML to develop apps that run on Android, iOS, macOS, and Windows.
- SQLite: An open-source, embedded database management system. It’s used in many applications, including mobile operating systems, desktop applications, and web browsers.

- Microsoft SQL Server: Microsoft's relational database management system. The system's function is to manage multiple databases. It provides a suite of tools that help to build, change, and manage the data.

Technical Challenges

Simulation

Andrew, Carson, Patrick

- Moving from JavaScript (Three.js and Chart.js) to C# tools (OpenTK and LiveCharts2). This means the team has to rewrite parts of the simulation, which takes time and testing.
- Our project runs inside .NET MAUI, and needs to make sure OpenTK and LiveCharts2 work without crashing.
- Picking up Unity means more on the team's plate in terms of research. Also, since the simulator was dropped after Fall 2023, there is some catching up to do.
- UnityUaal.Maui is new, unfamiliar, and necessary if the simulator is to be integrated into the current ball spinner application.

Backend

- Compiling the kinematic model that will represent the motor instructions: There are many variables and factors that are in play when a bowling ball moves down a lane. However, isolating the most necessary factors along with simulating the linear velocity, the angular velocity, the angular momentum, and using this to determine overall displacement of the ball represents a unique challenge.
- Updating CI/CD to run automated tests before deployment: Configuration Github actions to run our automated tests will be quite difficult, especially considering it will need

access to a database that contains the same schema as the RevMetrix database. This will most likely be facilitated through a database container that will need to be configured for Github Actions to access, however configuring the container within the Github Actions framework, providing the container our database schema, and configuring the API code to connect to this specific container instance while the CI/CD is testing the code will be challenging.

- Creating application-side interface for the calibration of the Ball Spinner motors: Specifically displaying the auxiliary sensor output in real-time and simultaneously displaying interpolated output will be quite challenging.

Ball Spinner Frontend

The majority of technical issues that are anticipated to arise surrounding updating the framework from .NET 8.0 to .NET 9.0. If there are significant enough changes between the new and old versions of .NET, there will have to be major revisions done to the front end of the application. Outside of this change, development on the front end is predicted to proceed smoothly.

Wiki

A major challenge with the wiki is that it is incredibly out of date. Setting up development environments was difficult for many of the new team members due to outdated information. Every page on the wiki will require an update. The pages themselves are not difficult to update, but the sheer amount of pages and information in need of updating is daunting.

Another challenge the wiki team will face is that the last time the wiki was updated was Spring 2024. This means that those who were working on the wiki have graduated/aren't working on the project anymore. Everyone on the wiki team has no experience updating the wiki, and will have to set up the development environment, as well as familiarize themselves with the frameworks.

Ball Spinner Controller

Technological challenges exist in the implementation of new protocol messages, as this requires a complete understanding of the packet order for both this and the Backend Team. This includes knowledge of the Endian of any sent datatypes, expected message orders, and how to maximize the information sent in each package. Additionally, creating a strong naming convention that easily communicates the intention behind each protocol message for future developers of the project is difficult to accomplish.

Other expected challenges include interfacing with the several anticipated sensors. Besides the SmartDot module, there is an expectation of at least 2 different additional sensors, both of which will require their means of data extraction, requiring downtime to understand their communication protocols and interface scheme.

Finally, errors are continuously found throughout the design process, some of which cause the Ball Spinner Controller to crash unexpectedly. There will be challenges in catching all edge cases to make sure that the Ball Spinner Controller is unable to crash, and that error messages are sent to the application when these issues occur.

Ball Spinner

Bowling balls are not a uniform sphere. They also have a significant amount of mass and inertia. This combination makes controlling its spin more difficult than initial estimations. Another aspect of difficulty is attributed to the target speeds, to account for the extremes, the ball spinner is being designed to manipulate a 16 lbs bowling ball at 800 RPM. To securely hold a bowling ball and prevent slippage also requires a considerable amount of force; By our calculations, if we use polyurethane (with a shore hardness between 70A-90A), we need to apply over 35 lbs of force. The ball will have a large amount of momentum as it goes through testing. A solid structure will be necessary to reduce vibrations.

The motors also generate a significant magnetic field, which will be picked up by the magnetometer inside the SmartDot. To mitigate this issue, we must develop some kind of magnetic shielding. The other external issue with the motors is wire management; As many parts of the ball spinner will be rotating and moving, we need to ensure no wires get pinched or twisted.

Mobile Application

- Restarting the mobile app from scratch due to the former app being severely outdated. The team will have to use the old repository as a reference.
- Learning .NET MAUI
- Connecting the local database to the cloud database.
- Learning Bluetooth Connectivity.

Schedule

https://docs.google.com/spreadsheets/d/1XV9flG-RVYokWHDQ5p4dRcY8xWQZqy82d4whpSQuybg/edit?gid=0#gid=0