Implementation of backend and web application solution to assist athletic performance measuring through the use of IMU sensors

Athanasios Papageorgiou

Supervising professor: Ioannis Stamelos

This is a masters diploma paper; a part of the evaluation process requirements for the PMS of Interactive Technologies, Aristotle University of Thessaloniki
# Table of Contents

Abstract .................................................................................................................................................. 5

Introduction ......................................................................................................................................... 6

The sport of Bowling .............................................................................................................................. 7

Introduction to Bowling ......................................................................................................................... 7

Bowling basic rules ............................................................................................................................... 8

Challenges in coaching bowling and evaluating athletes ................................................................. 9

Improving coaching Bowling with technology .................................................................................. 10

The Athlos Bowling solution ................................................................................................................ 10

Similar solutions on the market ........................................................................................................... 11

Strike Seeker (strikeseeker.com) ........................................................................................................ 11

Track My Roll (trackmyroll.com) ....................................................................................................... 12

Athlos Bowling: How it compares ...................................................................................................... 12

Introduction to the Technologies ....................................................................................................... 13

Internet of Things (IoT) ..................................................................................................................... 13

IoT Architecture .................................................................................................................................. 13

Pros and Cons of IoT .......................................................................................................................... 14

Wearables ............................................................................................................................................ 15

Big Data ............................................................................................................................................... 16

The Scalability problem ....................................................................................................................... 16
Aristotle University of Thessaloniki

Project Analysis .............................................................................................................. 18

Deep dive into the backend technologies ........................................................................ 18

Flask .............................................................................................................................. 18

Sqlite3 .......................................................................................................................... 21

InfluxDB ....................................................................................................................... 25

Deep dive into web application technologies (frontend) ................................................. 30

HTML5 .......................................................................................................................... 31

Bootstrap ....................................................................................................................... 32

Jinja2 ............................................................................................................................. 34

Deep dive into deployment technologies ..................................................................... 36

Docker .......................................................................................................................... 36

Pm2 ............................................................................................................................... 37

Digital Ocean Droplets .............................................................................................. 38

Build and Deployment ............................................................................................... 39

Project source code .................................................................................................... 39

Project requirements .................................................................................................. 40

Build / Environment setup ......................................................................................... 41

Deploy on local machine ............................................................................................. 43

Deploy with Docker .................................................................................................... 44

Deploy with Gitlab-CI ................................................................................................. 44
Aristotle University of Thessaloniki

Future development analysis ................................................................. 47

Project structure .................................................................................. 47

Flask Blueprints / Swagger ................................................................. 50

.............................................................................................................. 54

Code Documenting .............................................................................. 55

Additions / Improvements .................................................................. 56

Project testing and evaluation ............................................................ 58

Logging .................................................................................................. 58

Testing using Swagger (applies to backend only) ................................ 61

Testing using custom script (applies to backend only) ....................... 62

Project Evaluation Results ................................................................. 63

Conclusion ........................................................................................... 65

References .......................................................................................... 66
Aristotle University of Thessaloniki

Abstract

This paper analyzes the application of Internet of Things (IoT) technologies in athletic performance evaluation (coaching), focusing on the web and server application aspect of technologies. The sport of choice for this analysis is Ten Pin Bowling. To demonstrate IoT application in coaching Bowling, first a desired functionality requirements list was compiled based on bowling coaches’ feedback and a study of the sport’s mechanics in general. Given the feedback and after studying the available IoT technologies on the market, the most fitting ones were selected for use in development. A big focus for this selection was technology scalability and innovation while sticking to the open-source community as much as possible. After the research, we built and extensively documented a web and a server application to serve as part of the frontend and cloud infrastructure respectively of a larger IoT solution called Athlos Bowling. This resulted in a TRL5 software solution that after being put to the test is ready to be used as an assisting tool in coaching bowling. The outcome of this product proves the impact IoT can have on the sport of Bowling and sport science in general.
Introduction

Technology applications have infiltrated most science fields as a means to accelerate research, make experiments more efficient and their findings more accurate. Sports science in particular is a perfect example of a science field being able to achieve huge evolitional leaps through the application of technology in its practices. Even though technology has served sports science since day one of its existence, sudden breakthroughs in technology led us to a peak in sports-technology interaction. Recently, a newly emerged technology called Internet of Things revolutionized the way we apply technology into sports. Using the toolset of the Internet of Things, such as interrelated computing devices (wearables), we are capable of measuring athletic performance in ways not before possible.

In this paper, we will investigate the application of Internet of Things into the sport of Bowling as a means of measuring athletic performance by tracking the athlete's movements. The main purpose of this is to evaluate athletes, track progress and improve the overall athletic performance.

Along with this paper, a software application called Athlos has been developed using the latest of technologies to demonstrate the integration of Internet of Things technologies inside the sport of Bowling.

As far as technical analysis goes and since what is described above is of great complexity, this paper will mainly focus on the technologies used as well as the implementation of the backend side (server) and the Web application while describing on the surface the other architectural levels.
The sport of Bowling

Before we dive into the application developed as part of this thesis it is best to introduce Bowling as a sport for those not familiar with it. We mainly want to focus on the challenges of bowling when it comes to precisely evaluating athletic performance since the developed application tries to help overcome those challenges through technology.

Introduction to Bowling

Bowling is a target sport and recreational activity in which a player rolls or throws a bowling ball toward pins (in pin bowling) or another target (in target bowling). This thesis and the developed application called Athlos focuses solely on pin bowling and in particular on Ten-pin bowling. Ten-pin bowling is a type of bowling in which a bowler rolls a bowling ball down a wood or synthetic lane toward ten pins positioned in a tetractys (equilateral triangle-based pattern) at the far end of the lane. The objective is to knock down all ten pins on the first roll of the ball (a strike), or failing that, on the second roll (a spare).

Behind a foul line is an approach approximately 15 feet (5 m) long used to impart speed and apply rotation to the ball. The 41.5-inch-wide (105 cm), 60-foot-long (18 m) lane is bordered along its length by gutters (channels) that collect errant balls. The lane's long and narrow shape limits straight-line ball paths to angles that are smaller than optimum angles for achieving strikes; accordingly, many advanced bowlers impart side rotation to hook (curve) the ball into the pins.
Bowling basic rules

- The goal is to knock down all ten pins
- Each frame consists of throwing the ball twice to knock down all the pins
- If you knock down all the pins with the first ball, it is called a "strike"
- If you knock down all the pins with the second ball, it is called a "spare"
- Each game consists of ten frames. If you bowl a strike in the tenth frame, you get two more balls. If you throw a spare, you get one more ball.
- Open frames are frames without a strike or spare
- Scoring is based on the number of pins you knock down. However, if you bowl a spare, you get to add the pins in your next ball to that frame. For strikes, you get the next two balls.
- An average of three games is played. You determine a 3-game average by adding all 3 scores and then dividing that number by 3.
- Accurate preliminary scores are essential for fair division.
- Foul line is in effect. If you step over the foul line, any pins knocked down will not count towards your score.
- Ramp Bowling is allowed for those athletes that are not physically capable of rolling a bowling ball. The ramp is for physical disabilities and not to be used to just increase a score.
- Bumpers are not allowed.
Challenges in coaching bowling and evaluating athletes

The investigation performed on the premises of this thesis concluded that coaching bowling and evaluating bowling athletes is a lot of times more challenging than it appears to be. After many interviews with top tier coaches and athlete as well as observation of multiple training sessions it became evident than technology can be the solution to many problems that traditional practices face.

Bowling much like most target-practice activities is a sport where precision and accuracy of movements is very important for the performance of an athlete. Bowling athletes must maintain a very precise composure throughout the whole duration of the roll. Each and every movement an athlete performs has direct impact to the bowling ball and the overall success of the roll, directly affecting the game.

Generally, some of the very important factors when executing a roll are the following:

- Footwork (number of steps, position of legs, tempo etc.)
- Arm movement (the way the ball is held, angles, wrist movement etc.)
- Speed (stepping speed, arm speed)
- Coordination (ball release, foot slide etc.)

With decisive factors such as the above, one can very easily conclude that even a tenth of a second deviation (in ball release time for example) can lead into a completely different result in the roll and this is why coaching bowling to athletes can be a challenge.

For a coach, effectively measuring and evaluating all of the mentioned variables is very difficult since everything takes place in such a small timeframe. Also, the position of the athlete when playing the sport (facing towards the lane) makes it difficult for the coach to keep the athlete
inside the proper field of view to ensure every body movement is visible. Sure, some coaches with a lot of experience can empirically evaluate their athletes but in the end, the more accurate the measuring data the more easily it is to evaluate the rolls and dictate appropriate corrections.

Athletes on the other hand find it difficult to understand and visualize their mistakes. Without quantitative data, roll correction instructions from coaches are vague and hard to understand especially -as already mentioned- when most of the coaching feedback is about timing and coordination down to the millisecond.

**Improving coaching Bowling with technology**

Based on the feedback received, a complete solution was designed to assist both coaches and athletes. The solution’s name is Athlos Bowling and is part of an up and coming series of solutions brand named Athlos - [name of sport] that uses similar technology to the one that is going to be described in this thesis in hopes of assisting coaches and athletes of different sports.

**The Athlos Bowling solution**

As stated above, Athlos Bowling is a complete solution providing all the tools required to be considered a state-of-the-art application. It includes two hardware sensors, a mobile application, a web application (accessible by browser) and a server-side infrastructure to manage all of the data involved.

The goal of Athlos Bowling application is to:

- Gather precise movement data of rolls executed by the athlete
- Evaluate gathered data based on bowling sport-science and kinesiology
Aristotle University of Thessaloniki

- Produce statistics of an athlete’s progress and overall progression
- Provide a platform for coaches and athletes to share their data and interact

The primary purpose of this complete solution is to make bowling coaches and athletes training workflows much more accurate, efficient and easy. Hopefully solving all of the issues and challenges both parties have been facing so far.

**Similar solutions on the market**

There are already some products on the market advertised as tools to assist coaches and athletes. As a reference, two of the most promising ones will be presented. The first one being a more professional solution while the second one a more enthusiast oriented one.

**Strike Seeker (strikeseeker.com)**

Strike seeker is a video feed-based solution that records the athlete rolls and analyses them. It is aimed mainly towards coaches that want a clear video representation of the athletes roll from multiple angles and with rich data information to go along. It requires proprietary camera gear and processing units. A big part of its marketing qualities is:

- It can work as a self-coaching solution
- Gives the option to compare results between rolls
- Fast data manipulation (exporting, sending, analyzing)

Pricing is set at 4799 US dollars as of publishing of this paper.
Track My Roll (trackmyroll.com)

Track my roll began as a crowdfunded solution. It is a smartphone application that uses the phone’s built-in camera to capture the bowling lane. The captured video is then analyzed by the application producing useful data metrics regarding the roll. Its popularity can be attributed to the following capabilities:

- Simple to setup and use
- No special equipment
- Easy to learn and navigate

The application is priced at 19.99 US dollars as of publishing of this paper.

Athlos Bowling: How it compares

Many technology companies are interested in the sport of bowling and in particular assisting with coaching it. As a result, many attempts have been made at technology solutions that serve this purpose, many of those having successfully reached the market.

Most of these solutions however have weak points that deter many users from investing in them. The most advanced solutions seem to rely on really expensive and professional equipment while the enthusiast one’s lack on functionality and features. Another problematic aspect of most solutions on the market is the approach of evaluation. As it can be seen from both of the examples presented above, most solutions focus on studying the bowling ball to evaluate the roll and player at the same time. Little to no effort goes towards measuring and analyzing the athlete’s movement and technique.

The player-oriented approach that Athlos Bowling introduces is what sets it apart from other solutions and promises better results in training and coaching bowling.
Introduction to the Technologies

Before we perform a deep dive into the architecture and development process of Athlos Bowling as well as the product itself it is important to introduce the foundation of technologies this project is based on. Understanding those technologies will make clear how and why this application can be considered innovative.

Internet of Things (IoT)

Let’s start with the definition first.

The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

IoT Architecture

An IoT ecosystem consists of web-enabled smart devices that use embedded processors, sensors and communication hardware to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analyzed or analyzed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices -- for instance, to set them up, give them instructions or access the data.
The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.

In our particular case, the “thing” is the athlete wearing a pair of sensors constantly transmitting readings regarding movement and posture to the smartphone device. This device (acting as an IoT gateway) analyzes the data locally and in turn is connected to the internet communicating readings data to the server over network. The server stores those data and helps visualize everything through the frontend web app while keeping track of all connected devices. This way, data can be shared among athletes and coaches as well as accessed from anywhere in the world.

**Pros and Cons of IoT**

Some of the advantages of IoT include:

- Ability to access information from anywhere at any time on any device;
- Improved communication between connected electronic devices;
- Transferring data packets over a connected network saves time and money;
- Automating tasks helps improve the quality of a business’ services and reduces the need for human intervention.

Some disadvantages of IoT include:

- As the number of connected devices increases and more information is shared between devices, the potential that a hacker could steal confidential information also increases;
- Enterprises may eventually have to deal with massive numbers -- maybe even millions -- of IoT devices and collecting and managing the data from all those devices will be challenging.
Aristotle University of Thessaloniki

- If there’s a bug in the system, it’s likely that every connected device will become corrupted;
- Since there’s no international standard of compatibility for IoT, it’s difficult for devices from different manufacturers to communicate with each other.

**Wearables**

Let’s start with the definition.

Wearables are electronic technology or devices incorporated into items that can be comfortably worn on a body. Wearable devices are used for tracking information on real time basis. They have sensors that can measure day to day activity and sync them with mobile devices or laptop computers. After the invention of smartphones, wearable electronics are the next big innovation in the world of technology.

Wearable devices such as activity trackers are an example of the Internet of Things, since "things" such as electronics, software, sensors, and connectivity are effectors that enable objects to exchange data through the internet with a manufacturer, operator, and/or other connected devices, without requiring human intervention.

For the Athlos Bowling project, two wearable devices are being used. Both of those devices have sensors capable of determining orientation and speed/acceleration thus being able to track and trace movement. For optimal accuracy, one sensor is placed on the athlete’s throwing arm (worn like a watch) and the other on the opposite side foot (clips on shoe). Following the key essence of wearables, the Athlos Bowling wearables produce real time data with great accuracy and transmit it to the user’s smartphone through Bluetooth.
Big Data

Big data is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software. Data with many cases (rows) offer greater statistical power, while data with higher complexity (more attributes or columns) may lead to a higher false discovery rate. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. Big data was originally associated with three key concepts: volume, variety, and velocity. When we handle big data, we may not sample but simply observe and track what happens. Therefore, big data often includes data with sizes that exceed the capacity of traditional software to process within an acceptable time and value.

In the case of Athlos Bowling, big data exists in the form of roll records. The backend databases store all of the data metrics collected by the sensors of multiple users. This adds up to great numbers of records. Also, each record contains a lot of information in the form of tags regarding the user information, roll information as well as readings. A batch of records is stored for each roll. Through those records an evaluation of every roll can be produced which in turn gets stored on a different table.

The Scalability problem

Scalability is the ability of a device to adapt to the changes in the environment and meet the changing needs in the future. It is essential feature of any system which has the capability to handle the growing amount of work. It is a desirable attribute of a system or a network whose
lack can cause a poor system performance and the necessity of reengineering of the whole system.

Athlos Bowling is a Project where scalability has been the main focus since the beginning of its development cycle. With multiple users and sensors as well as a web application constantly requesting sessions and reading/writing big amounts of data, it is very important that the whole system is built around flexibility and scalability.
Aristotle University of Thessaloniki

Project Analysis

As stated in the introduction, this paper focuses on technical aspects of the backend and Web application side of the Athlos Bowling Project.

In this section of the paper we will deep dive into the project’s architecture and solutions used throughout the backend and web application development. Since many of the available technologies can satisfy a single requirement each choice will be compared to an equally viable alternative one as part of the substantiation. Pros and cons will be provided as well for the more controversial choices.

Deep dive into the backend technologies

What is backend? Essentially everything that happens of the server side and not on the browser (in case of the Athlos Bowling web application) or the mobile / sensor. In the case of Athlos Bowling the server has to perform the following tasks (high level):

- Store/serve registered user data
- Store/serve roll information (both raw and evaluated)
- Calculate basic data statistics

Flask

Flask is a Free and Open Source Python web framework built with a small core and easy-to-extend philosophy. It is classified as a microframework because it does not require particular tools or libraries. It doesn’t have a database layer or provisions for form validation and makes use of extensions. In the context of Athlos Bowling server, Flask is tasked to provide API endpoints for the Web application as well as the Mobile application to be able to perform queries
to the databases. Those queries are either user related (registration, login etc.) or roll related (save roll, find roll etc.). There is also some semi-complex logic coded behind some of the API endpoints namely the APIs responsible for producing statistics.

**Pros**

- Extremely flexible
- Minimalist without sacrificing power
- Simple to learn and use
- Routing URLs is easy
- Small core and easily extensible

**Cons**

- Not asynchronous-friendly
- Limited support and documentation
- Lack of database/ORM/forms
- Truly limited in features

As it can be concluded by the pros stated above, Flask is a perfect fit for the architecture of Athlos Bowling server. Its simplicity in learning and implementation helps a lot when dealing with small development teams and tight timeframes. Also, Athlos Bowling server is very API dense, so the simplicity in URL routing helps a lot with the flexibility and ease of development. Moving on to the cons, thankfully none of them has any real impact on the project. Lack of database/ORM/forms goes hand-in-hand with the loose and flexible character of the database of choice influxDB (influxDB analysis will follow up on a later section of the paper). In the cases where form validation is needed, Flask extensions (fields of flask_restplus) were used to
facilitate the functionality. The limitation in features mentioned was no issue whatsoever. On the contrary it encouraged the use of only the necessary extensions making the codebase easier to read and work on.

**Scaling Flask**

There is common misconception that Flask is hard to scale with. In the case of Athlos Bowling server, this is no true. A lot of well-known backend infrastructures with high-traffic use Flask effectively to scale, some of the best examples are Pinterest and LinkedIn.

Here are the practices followed that allow Athlos Bowling server project to scale:

- By keeping the logic complexity low, Flask is more than capable of handling huge amount of traffic and requests (as it will be displayed in testing).
- Flask multithreading is configured and applied.
- The project has been implemented around the use of Flask blueprints technology which helps with load balancing and also means that splitting the project to different microservices is also a viable option.

Scaling with Flask can extend the framework itself by the following practices if it is required in the future:

- Use auto scaling server instances (example. AWS EC2)
- Configure proxy load balancers/workers (options for nginx, apache2)
- Split blueprints into microservices inside different server instances
Alternative to Flask: Django

For the project of Athlos Bowling, Django framework was also considered by the design team. Even though Flask and Django share more similarities than differences in terms of functionality and design here are some key focuses between the two:

<table>
<thead>
<tr>
<th>Django</th>
<th>Flask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>Speed</td>
</tr>
<tr>
<td>Browsable API</td>
<td>Support for NoSQL</td>
</tr>
<tr>
<td>Periodic and regular releases</td>
<td>Minimal complexity</td>
</tr>
<tr>
<td>Rigid application structure</td>
<td>Absolute minimalism</td>
</tr>
<tr>
<td>Functional admin panel</td>
<td>No ORM, easily connected with extensions</td>
</tr>
<tr>
<td>Lots of batteries</td>
<td>Debugger embedded in a browser</td>
</tr>
<tr>
<td>Huge community</td>
<td>Short and simple code among other Python web skeletons</td>
</tr>
<tr>
<td>Huge third-party application support</td>
<td>Descriptive and elaborative documentation</td>
</tr>
</tbody>
</table>

To conclude, Django is a full stack framework, whereas Flask is a micro and a lightweight framework. Besides being the community favorite, Flask was chosen because of its simplicity both in function and development. Many of the more advanced features Django had to offer were deemed unnecessary and increased complexity without justification. Overall, Flask is a better fit for this project at its current stage. However, since both are python-based transitioning from one to another in the future if necessary should be no difficult task.

SQLite3

SQLite3 is a compact free database used to easily create and use a relational type database. Though SQLite3 is not a full-featured database, it supports a surprisingly large set of the SQL standard, and is ideal for those just starting to learn SQL as well for developers that need a simple database engine to plug into their applications.
Aristotle University of Thessaloniki

In the context of Athlos Bowling backend, sqlite3 is used as a relational database which stores the registered users’ information. This includes personal information regarding the user (first name, last name, user type, password hash) as well as friendships between users. To achieve optimal results in performance and code readability user information and friendships are split and stored into two different database instances.

**Pros**

- SQLite is a very light weighted database so, easy to use.
- It is almost 35% faster than File system.
- It only loads the data which is needed, rather than reading the entire file and hold it in memory.
- Doesn’t need to install and configure it. Just download SQLite libraries in your computer and it is ready for creating the database.
- It updates your content continuously so, little or no work is lost in a case of power failure or crash.
- SQLite is portable across all 32-bit and 64-bit operating systems
- It can be used with all programming languages without any compatibility issue.

**Cons**

- SQLite is used to handle low to medium traffic HTTP requests.
- Database size is restricted to 2GB in most cases.

Because of the roll data manipulation being the main focus of the system rather than the user management, sqlite3 was chosen for the task of managing user-related data. The ability to develop fast with very low dependencies and recourses is very important on a big and time
Aristotle University of Thessaloniki

constrained project like Athlos Bowling. Besides that, sqlite3 helps set the basic schema of a relative database which can be easily upgraded to a more production-friendly solution in the future.

**Scaling sqlite3**

Even though sqlite3 is not built with scalability in mind, a few practices were followed on the code and logic side of the project in order to ensure that the user and friendship databases were capable of lasting through a lot of traffic without the need to upgrade to a different relational database in the near future. Transactions are very few and limited as well as the writes on the database. Database writes (which are the more taxing tasks) take place only upon user registration friendship creation which means very far in between. Reads are no issue even with a big user base, being more tolerant by design and benefitting by caching. Reads happen on user profile fetching. Lastly, user validation for the Web application takes place separately on the frontend itself as showcased in the web app deep dive section of the paper.

**Alternative to Sqlite3: MySQL**

MySQL is an open source SQL relational database management system that’s developed and supported by Oracle. It can be considered the most popular options among relational databases. Some of the key features of MySQL include:

- Ease of Management – It is pretty easy to download and use the software.
- High performance – It provides you fast loading utilities with different memory cache.
Aristotle University of Thessaloniki

- Scalable – With MySQL, you can scale anytime you like. It is really easy to create data warehouses including an enormous amount of data.

- Compatibility – MySQL is compatible with all modern platforms like Windows, Linux, Unix.

- Performance – MySQL gives you high-performance results without losing essential functionality.

- Complete Data Security – Only the authorized users can access the database. Complete security for the data.

- Low Cost – It is free to use.

- Memory Efficiency – MySQL has low memory leakage.

To conclude, MySQL is a better long-term solution to Athlos Bowling project because of its scalability along with performance. However, as long as the project remains of low complexity in terms of its user management infrastructure and moderate number of users, going with sqlite3 will pose no issues. On the contrary, sqlite3 is easy to develop with, has the basic required functionality and options to fully satisfy project needs and can be considered as a stepping stone for future migration to a fully-fledged MySQL system.
InfluxDB

InfluxDB is the open source time series database that is part of the TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) stack. It is designed to handle high write and query loads and provides a SQL-like query language called InfluxQL for interacting with data. Some of the key features of InfluxDB are:

- High performance. InfluxDB is a high-performance data store written specifically for time series data. It allows for high throughput ingest, compression and real-time querying. InfluxDB is written entirely in Go and compiles into a single binary with no external dependencies. It provides write and query capabilities with a command-line interface, a built-in HTTP API, a set of client libraries (e.g., Go, Java, and JavaScript) and plugins for common data formats such as Telegraf, Graphite, Collectd and OpenTSDB.

- SQL-like queries. InfluxDB works with InfluxQL, a SQL-like query language for interacting with data. It has been lovingly crafted to feel familiar to those coming from other SQL or SQL-like environments while also providing features specific to storing and analyzing time series data. InfluxQL supports regular expressions, arithmetic expressions, and time series-specific functions to speed up data processing.

- Down sampling and data retention. InfluxDB can handle millions of data points per second. Working with that much data over a long period can lead to storage concerns. InfluxDB automatically compacts data to minimize your storage space. In addition, you can easily down sample the data; keeping high-precision raw data for a limited time and storing the lower-precision, summarized data for much
longer or until the end of time. InfluxDB has two features that help to automate the down sampling and data expiration processes — Continuous Queries and Retention Policies.

Athlos backend uses InfluxDB for storing and visualizing the sensor readings and evaluated rolls. InfluxDB is at the center of Athlos Bowling backend and everything is implemented around it. Given the fact that Athlos Bowling is built to serve multiple concurrent users, each one having two sensors attached amounts to a potential huge number of data.

**Pros**

- Tailored-made for Time Series data: InfluxDB is designed to handle time series data more efficiently. It has impressive write and read throughputs.
- Own infrastructure or cloud services. In other words, InfluxDB offers various solutions to match every potential business need.
- Holistic Solution: InfluxDB is designed to work perfectly along with the rest of the Influx Data ecosystem (all in one package).
- Various Input Plugins: InfluxDB does not limit itself to one or two input methods, but it offers various input plugins free of charge.
- Grafana Support: Grafana is the go-to software for time series analytics, with well over 100,000 active installations. Grafana has introduced a plugin for InfluxDB as a data source for their analytics dashboards.
- Extensive Programming Languages Support: InfluxDB offers support for various programming languages, including, but not limited to: .Net, Java, Perl, PHP, Python, R, Ruby, Scala and more.
• SQL-like Query Language: InfluxDB comes with an SQL-like query language, InfluxQL, which means it does not have a steep learning curve and is easier to write and understand by non-tech people as well compared for instance to OpenTSDB which does not provide such query language.

• Continuous Query Support: “Continuous Queries (CQ) are InfluxQL queries that run automatically and periodically on real-time data and store query results in a specified measurement.”

• Down sampling (roll-up) of commonly-queried, high granularity data to a lower granularity. Queries on data with lower granularity require fewer resources and are faster than queries with higher granularity.

• Easy Installation: InfluxDB compiles into a single binary file with no dependencies, which makes it extremely easy to install and have it up and running. Auto-Expiration: Time series data may become less relevant or even useless depending on the application as time goes by. InfluxDB with the use of Retention Policies enables automatic expiration of stale data.

• Unlimited fields: The new storage engine of InfluxDB, TSM Tree is columnar format, which means that the number of fields does not affect querying performance in a negative way.

• Built-in Web Administrator Interface: Like SQL, InfluxDB provides a built-in online interface for users who are not comfortable with command line interfaces and would prefer a more intuitive solution.
• Extensive Documentation: Influx Data provides an extensive documentation guide for InfluxDB from installation to complex queries and schema optimization.

**Cons**

- Clustering for scalability purposes is a Closed-Source Feature solution.
- Community Issues: Given that InfluxDB is a relatively new product, its community is again relatively small compared to other making troubleshooting of problems through community really difficult.

**Scaling InfluxDB**

InfluxDB is by design build to withstand huge amounts of data reads and writes rendering it a perfect option for the Athlos Bowling project. Scaling in the future will not be an issue with the implemented setup. There are far more complex implementations compared to Athlos Bowling on the production field using InfluxDB with no issues. However, even if user traffic gets blown out of proportion, there are a few closed-source options Influx Data provides to help with scaling, namely a technology called clustering and also managed services. Lastly as a free alternative many developer choose to use shards technology as a way to improve scalability.

**Alternative to InfluxDB: Amazon Timestream**

Amazon Timestream is the Amazon’s take on time series databases. It is brand new, recently got out of BETA and takes full advantage of Amazon’s multiservice ecosystem. Its main features compared to other solutions is the Built-in Analytics, adaptive query processing engine which is optimized for a wide variety of time intervals such as milliseconds, microseconds, and
nanoseconds. Finally, the serverless capabilities allowing for complete automation of scaling and easy configuration.

To conclude, InfluxDB is the overall better option for managing huge amounts of roll data. Architecturally going with a document-based database solution rather than a relational one is a one-way road because of the quantity of data cycled inside the Athlos Bowling infrastructure and the need to scale in the future. Compared to other solutions, InfluxDB got preferred because it strikes a perfect balance between features, technology and cost to performance ratio. While Amazon Timestream is a worthy opponent, it is strictly attached to Amazon, has no-free tier options and -being brand new- has not been on the market long enough to confirm its stability and reliability.
Deep dive into web application technologies (frontend)

What is frontend? Front-end refers to the client side or the web design in the web industry. Essentially, it has to do with everything that the end user visually sees and interacts with. In the case of Athlos Bowling the frontend consists of two solutions. One is the mobile application and the other is the web application. In this paper we will focus on the web application and its technologies. As the name suggests, Athlos Bowling web application runs in the web browser and it is accessed by a URL. The target audience for this web application is of course bowling athletes and coaches that are interested in the solution.

The functionality the web app provides is the following:

- Allows users (athletes, coaches) to register an account with the Athlos Bowling solution in order to authenticate themselves and access their profile and data.

- Allows users to view their bowling activity in the form of evaluated rolls and overtime statistics.

- Allows users to befriend other users that they are interested in (could be coaches but also other athletes) and access their profiles (bowling activity, statistics) as well as other personal information (contact info, location).

With the expected functionality explained and in mind we can proceed with the technologies being used to power everything.
**HTML5**

HTML5 is a programming language whose acronym stands for Hyper Text Markup Language. It is a system that allows the modification of the appearance of web pages, as well as adjusting their appearance. It also used to structure and present content for the web.

Apart from HTML5, there are other languages that are necessary to give format and interactivity to a site, but the basic structure of any page is first defined in the HTML5 language. In the case of Athlos Bowling web app, Bootstrap and Jinja2 are the supplementary language assisting with the interactivity and both are going to be analyzed later in this paper. Everything displayed on the Athlos Bowling web app is defined by HTML5. It is also important to note that in this section of the paper there is no need to be discussed. Web applications are served on the user’s machine browser and are strictly dependent on client resources. Here are some pros and cons of HTML5;

**Pros**

- Supports the latest versions of major browsers
- Allows simple interactivities without external plugins
- Is compatible with mobile browsers
- Takes less time and efforts to develop with
- Supports offline access

**Cons**

- May not support old browsers
- Restrictions on multimedia
- Inconsistencies in delivery (renders differently on different browsers)
- Security concerns with web apps
Alternative to HTML5: None

HTML5 has not alternatives. HTML in general has always been the leading programming language for coding everything web-related. An argument could be made about choosing a previous version of HTML as an alternative (HTML4 for example) however the pros of going with the latest version completely overrule any other argument.

Bootstrap

Bootstrap is one of the most popular web development frameworks out there, and it is used for developing highly-responsive projects in HTML, CSS, and JavaScript. Essentially, what Bootstrap does is reduce the time needed to get a modern website up and running. It includes design templates for every aspect of a site, from typography to buttons or image carousels. Bootstrap can be quickly integrated with other platforms or even different frameworks. In the case of the Athlos Bowling web app, Bootstrap plays a very big role. It is responsible for the majority of the aesthetic the application has to offer especially the layout and grid as well as the icons and images. Here are some of the pros and cons of using Bootstrap;

Pros

• Great standardized platform with all the basic styles and components needed (layout grid, panels, tables, buttons, modals, form styles, etc) to build on which saves a lot of time.

• Support for all major browsers and fixes CSS compatibility issues.

• Consistent UI that looks good out of the box.
• Lightweight and customizable (on their site or via less and sass) files so you can include only what you use.
• Designed with responsive structures and styles for mobile devices.
• Several JavaScript plugins included using jQuery.
• Icons included (now as a font file for infinite scaling).
• Good documentation and community support.
• Lots of free and professional templates, themes and plugins.

**Cons**

• Styles are verbose and can lead to lots of HTML output that's not perfectly semantic.
• JavaScript is tied to jQuery (it's by the far the most common JavaScript library though and the plugins can just as easily be left unused).
• Can require lots of overriding styles or rewriting their files if you have lots of customizations or want to deviate from Bootstrap structure.
• Websites can start to look the same if you don't customize the styles and colors much.

**Alternative to Bootstrap: Pure CSS**

Pure CSS is very similar to the Bootstrap framework. It is much more barebones and lightweight containing only basic functionality. It is considered liberating since you can pick and choose from as needed. That said, it does come with a fair amount of buy-in and learning curve needing to learn a new (slightly different) way of doing things as opposed to the norm Bootstrap has enforced. While the Athlos Bowling web app could be benefited by a simpler framework
such as Pure CSS, the justified popularity and community around Bootstrap as well as the flexibility it offers made it to be a clear choice.

Jinja2

Jinja2 is a Python template engine used to create HTML, XML or other markup formats that are returned to the user via an HTTP response. Template engines allow developers to generate desired content types, such as HTML, while using some of the data and programming constructs such as conditionals and for loops to manipulate the output. Template files that are created by developers and then processed by the template engine consist of prewritten markup and template tag blocks where data is inserted. In the case of Athlos Bowling web app, Jinja2 is responsible for programmatically populating the pages of the web app with the user requested information. This includes iterating through query data and populating table lists as well as user information. Here are some of the pros and cons of Jinja2;

Pros

- Flexibility. Jinja templates are very flexible in terms of what they can contain, supporting concepts like macros and more Python-like constructs.
- Asynchronous execution. Jinja templates support asynchronous execution, which allows backing tasks to run their course -without holding-back templates- and later reconvene with templates when finished.
- Speed and performance. Jinja compiles template source code to Python bytecode when it's first loaded, so the template is only parsed once, resulting in better runtime performance. In addition, Jinja also supports the option of ahead-of-time compilation, which can also result in better performance.
**Cons**

- Little to no third-party package support. It can lead to difficulties and confusion when template customization is required.
- New concepts in terms of logic. As every concept is new and will require some practice.

**Alternative to Jinja2: Cheetah**

There are many alternative templates engines that are much more powerful and flexible compared to Jinja2. Most of them however are written in JavaScript programming language. For the Athlos Bowling project in general the main focus was to keep the codebase as homogenous as possible. This meant that since the server code was in Python, the template engine was chosen to be in python as well. An alternative to Jinja2 while still using python in its core is Cheetah. Cheetah is an open source project with increasing popularity and very similar to Jinja2 in terms of practices. Jinja2 was picked over Cheetah for the huge community support and extensive documentation it excels in compared to Cheetah.
Deep dive into deployment technologies

Besides the technologies regarding the product itself mentioned so far, there are also technologies used that have to do with the successful deployment of the product (both server and web application).

Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Docker configuration files have been written for both the server application as well as the Web application of Athlos Bowling project. The base image used for the docker container is ubuntu for compatibility and support reasons. Using docker is optional but makes it very easy to deploy the application fast without worrying about dependency installation and compatibility. It can also help with scaling either application by deploying different docker instances and serving them from different infrastructures.
Pm2

PM2 is a free and open source production process manager for Node.js and recently Python/Flask applications with a built-in load balancer. It allows to keep applications alive forever, to reload them without downtime and to facilitate common system admin tasks. Pm2 is installed as a npm package of node 8 or above. Some of the key features pm2 is advertised with are:

- Behavior configuration
- Source map support
- Container integration
- Watch and reload functionality (max memory reloads)
- Log management
- Monitoring of application
- Module system

In the case of Athlos Bowling backend, pm2 makes sure that both the web application as well as the server application run at all times even in cases of internal error for a near 100% uptime. As it is going to be explained later pm2 plays a big role in the pipelines as well. Additionally, it makes it very easy to keep track of the health and functionality of both application as well as debug any possible issues because of its very efficient logging capabilities (pm2-logger).
Digital Ocean Droplets

Digital Ocean Droplets are Linux-based virtual machines (VMs) that run on top of virtualized hardware. Each Droplet created is a new server that can be used, either standalone or as part of a larger, cloud-based infrastructure. Digital Ocean, Inc. is an American cloud infrastructure provider headquartered in New York City with data centers worldwide. Digital Ocean provides developers cloud services that help to deploy and scale applications that run simultaneously on multiple computers. As of January 2018, Digital Ocean was the third-largest hosting company in the world in terms of web-facing computers.

As of the time of publishing this paper, for testing purposes Athlos Server and Web application are deployed on a single Digital Ocean droplet. The droplet of choice is the entry tier droplet with 1 GB of memory, a single core CPU, 1 TB of transfer and 25 GB of SSD storage running Ubuntu Server.
Build and Deployment

In this section of the paper, instructions for building and deploying both applications (server, web app) are presented. The analysis is purely on a technical level meaning that everything demonstrated can be directly applied. Also, it is important to note that demonstrated steps are subject to change as the project evolves in the future.

*Terminal commands demonstrated in this section are verified to be working on a UNIX based system. Some of the commands may be applicable in terminals of other systems however most of the time there might be an alteration in syntax. For non-UNIX systems it is advised to search the system's equivalent commands.*

Project source code

Source code for both projects is uploaded to a Gitlab repository. Gitlab was selected over GitHub because it provides many DevOps tools for free while maintaining the same core functionality as GitHub.

*Gitlab repository for backend (athlos-backend)*

- https://gitlab.com/pathanasi/athlos-backend

*Gitlab repository for frontend (athlos-frontend)*

- https://gitlab.com/pathanasi/athlos-frontend

*Gitlab repository for miscellaneous tools (athlos-tools)*

- https://gitlab.com/pathanasi/athlos-tools

At the time of compiling this paper, all three projects are set to private. This means that access to the code is through invite-only. To request access to the projects, refer to; pathanasi@csd.auth.gr
Beside the Gitlab website, after access has been granted to the projects, the code can be cloned through the use of git client (https://git-scm.com/) by running the following commands from a terminal;

# athlos-backend
git clone https://gitlab.com/pathanasi/athlos-backend.git

# athlos-frontend
git clone https://gitlab.com/pathanasi/athlos-frontend.git

# athlos-tools
git clone https://gitlab.com/pathanasi/athlos-tools.git

**Project requirements**

In this section of the paper all the requirements needed for both projects to run are presented along with the where to find them. (Information also included inside README.MD of each project). Some of the deployment methods may depend on additional requirements but since those are optional and not directly affiliated with the ability of the project to run, they are going to be presented on their corresponding deployment analysis in this paper.

**Base Requirements for both athlos-backend and athlos-frontend**

- Programming Language: Python3.6 (get from https://www.python.org/downloads/release/python-360/)
- Package manager: Pip3 (already included in the Python standard library)
**Additional packages required for athlos-backend**

- Additional python packages (obtained by pip):
  
  Flask
  connexion
  flask_restplus
  pyopenssl
  influxdb
  schema

**Additional packages required for athlos-frontend**

- Additional python packages (obtained by pip):
  
  flask
  flask_login
  flask_migrate
  flask_wtf
  flask_sqlalchemy
  flask_bcrypt
  gunicorn

**Build / Environment setup**

In contrast to most other languages, Python auto compiles (builds) itself to *.pyc* upon code execution so there is no need for specific actions to build the applications. However, the installation of dependences is done manually through the use of pip3.

It is advised that the additional packages required for each project are installed inside a virtual environment as to better keep track of them -separately for each project- as well as housekeeping reasons (keep the global package directory clean).

Here are the steps of setting up the environment and installing dependencies;
Setup environment for athlos-backend

# Create virtual environment to house dependencies
virtualenv -p python3 venv_athlos_backend

# Enter the environment (activation)
source venv_athlos_backend/bin/activate

# Install additional packages (dependencies)
pip3 install -r requirements.txt

# Setup flask env type to dev (omit for production setup)
export FLASK_ENV=development

# Initialize the application main
export FLASK_APP=app.py

# Initialize the databases (will do nothing if databases already exist)
flask initdb
Setup environment for athlos-frontend

# Create virtual environment to house dependencies
virtualenv -p python3 venv_athlos_frontend

# Enter the environment (activation)
source venv_athlos_frontend/bin/activate

# Install additional packages (dependencies)
pip3 install -r requirements.txt

# Setup flask env type to dev (omit for production setup)
export FLASK_ENV=development

# Initialize the application main
export FLASK_APP=run.py

Deploy on local machine

Deploying both applications locally on a user machine is very easy. Steps documented in the previous section (Build / Environment setup) need to be applied first. After that, from within the corresponding project directory run the command;

# Run the application on specified port
flask run --port=5001

Note: if both projects (athlos-backend, athlos-frontend) are run on the same machine specified, port needs to be different between the two projects or else there will be a port allocation error.
Aristotle University of Thessaloniki

**Deploy with Docker**

As it was analyzed extensively on the deployment technologies section, one of the three available deployment options for both the athlos-backend and athlos-frontend is Docker. To make use of this method, Docker needs to be installed on the target machine.

(Docker can be found here: [https://docs.docker.com/install](https://docs.docker.com/install))

For both projects, Dockerfile configurations have been written that exist inside the root directory of each project. To deploy using docker, enter the project’s root directory and run the commands;

```bash
# Build container
docker build . -t athlos-backend (or athlos-frontend)

# Run
docker run -d -p 4444:4444 athlos-backend (or athlos-frontend)
```

Same port rules apply here as with the local machine deployment option.

**Deploy with Gitlab-CI**

Gitlab-CI configuration files have also been written for both projects (athlos-frontend, athlos-backend) and are located inside the root directory of each project (gitlab-ci.yml). In case of athlos-backend and athlos-frontend, Gitlab-CI was mainly configured and aimed to automate the project’s deployment to production. For this reason, it was also paired with Pm2 for the benefits analyzed on the technology analysis section of this paper. With this combination there is zero downtime during deployments.

(Instructions to install and configure Pm2 can be found here: [https://pm2.keymetrics.io/docs/usage/quick-start](https://pm2.keymetrics.io/docs/usage/quick-start))
Each time there is a commit to one of the specified branches (master branch or development branch), a pipeline inside a GitLab runner is triggered that performs the following steps:

1. Installs all the required dependencies for the pipeline
2. Enters the production server (droplet)
3. Creates a backup of the databases
4. Cleans the project directory
5. Transfers the newly committed codebase
6. Restores databases
7. Installs dependencies
8. Restart the application with Pm2

To personalize/change the target deployment server go to Gitlab > athlos-backend > Settings > CI/CD > Variables. Here, the following variables can be configured:

- `PROD_SERVER_IP`=(ip goes here)
- `PROD_SERVER_PASS`=(ssh password goes here)
- `PROD_SERVER_PORT`=(port goes here)

If there is a need to change the variable names, it is important to also update them inside the configuration file (gitlab-ci.yml).

To include additional pipelined that trigger the pipeline process append the following format of code inside the gitlab-ci.yml;
deploy-{env name goes here}:
  stage: deploy
  allow_failure: false
  before_script:
    - export FLASK_ENV="{flask env goes here}"
    - export SERVER_IP=$PROD_SERVER_IP
    - export SERVER_PORT=$PROD_SERVER_PORT
    - export SERVER_PASS=$PROD_SERVER_PASS
<<: *deploy_script_definition
environment:
  name: {env name goes here}
when: manual ← remove this for auto deploy only:
  - {branch name goes here}
Future development analysis

This section of the paper is dedicated to aiding with any future development of athlos-backend and athlos-frontend. For this purpose, all of the coding practices followed will be analyzed (for project homogeneity) as well as tools that aided the development process. Also, functionalities that were not implemented because of time constrains will be presented.

Project structure

Project structure was designed with code readability and modularity in mind. As it is going to be evident by the project tree analysis, code with similar functionality is grouped together. Filenames are kept simple and fully descriptive of the functionality of code contained. CamelCase is being used throughout the project both in filenames as well as functions and variables.

Athlos-backend directory tree

```
├── src
│   ├── converter
│   │   ├── {converter related files}.py
│   │   └── handler
│   │   └── {handler related files}.py
│   │   └── model
│   │       └── {model related files}.py
│   │   └── repository
│   │       └── {repository related files}.py
│   │   └── service
│   │       └── {service related files}.py
│   └── utils
│       └── {utils related files}.py
```
- **Handler.** This folder contains all the code for the REST API routes. Each filename depicts the functionality of the code inside following this format; ____Handler.py For example UserHandler.py contains code for all of the routes that have to do with the user such as registration, session etc.

- **Service.** This folder contains the code logic (if required) behind the API calls. In similar fashion to handlers, filenames depict the functionality of the service code; ____Service.py For example UserService.py contains all code regarding the logic behind user API calls.

- **Model.** This folder contains everything that has to do with fixed schemas including validators. For example, LoginValidators.py contains the fixed schema regarding the payload which is required for a user to login.

- **Converter.** This folder contains all of the object conversion functions. It follows the same name principal as the other folders; ____Converter.py For example RollConverter.py converts the roll payload object to an enriched object (containing user id) that can be stored inside Influx DB.

- **Repository.** This folder contains all of the queries to sqlite3 and Influx DB. Filenames inside have the format; ____Repository.py For example UserRepository.py contains all of the queries for querying users inside sqlite3 database.

- **Utils.** This folder contains all of the helper code that does not fit inside any other project structure category. For example Toolbox.py contains a function that converts epoch timestamp to date-Time format.
Athlos-frontend directory tree

Athlos-frontend structure is much more complex compared to athlos-backend because it includes multiple programming languages (html, CSS, JavaScript, python) as well as a plethora of external modules and libraries. For this reason, everything external that is auto-generated and not implemented by the developer will be omitted from the analysis. A more conservative project structure visualization of the frontend is this;

```
app
│   ├── converter
│   │       ├── {converter related files}.py
│   │   └── request
│   │       ├── {request related files}.py
│   │   └── service
│   │       ├── {service related files}.py
│   │   └── static
│   │       ├── assets
│   │       │       ├── css
│   │       │       ├── fonts
│   │       │       ├── img
│   │       │       └── js
│   │       └── templates
│   │               └── utils
│   │                   └── {utils related files}.py
```

- **Converter.** Much like athlos-backend, inside converter folder of athlos-frontend exist functions related to object conversion. For example, RequestConverter.py includes functions that convert provided variables to HTTP request payloads.

- **Request.** Request folder contains all of the request that the frontend performs to the backend API in order to fetch the data to be visualized.

- **Service.** Similarly, to backend once again, service folder contains the logic behind API calls performed on the web application this time. Filename format is the same;
Service.py For example PageContentService.py contains the logic behind the calls requesting a specific page, providing the data-to-be-displayed.

- **Static/assets.** This folder contains the majority of the design elements of the web application. Namely, the CSS folder contains all of the CSS code, the fonts folder contains all of the font configurations used across the frontend, img contains all of the images, js contains all of the JavaScript code and lastly, scss contains all of the bootstrap code.

- **Templates.** This folder contains all of the HTML code. (Filenames represent the corresponding page the HTML code refers to index.html for the index page profile.html for the profile page etc.)

- **Utils.** This folder contains all of the code that its functionality cannot be categorized within any other directory. Toolbox.py for example contains a function that translates HTTP responses from an object to response code and body.

**Flask Blueprints / Swagger**

Two very important development decisions made during designing of the project that affect the way and future of the development process are the use of Blueprints and Swagger, two packages offered by Flask. Blueprints were also discussed on the project technologies section of the paper as a means of amplifying scalability of the backend while Swagger is going to be discusses later as a means of testing the backend.
Blueprints (applies to backend only)

Flask uses a concept of blueprints for making application components and supporting common patterns within an application or across applications. Blueprints can greatly simplify how large applications work and provide a central means for Flask extensions to register operations on applications. A Blueprint object works similarly to a Flask application object, but it is not actually an application. Rather it is a blueprint of how to construct or extend an application.

Blueprints in Flask are intended for these cases:

- Factor an application into a set of blueprints. This is ideal for larger applications; a project could instantiate an application object, initialize several extensions, and register a collection of blueprints.
- Register a blueprint on an application at a URL prefix and/or subdomain. Parameters in the URL prefix/subdomain become common view arguments (with defaults) across all view functions in the blueprint.
- Register a blueprint multiple times on an application with different URL rules.
- Provide template filters, static files, templates, and other utilities through blueprints. A blueprint does not have to implement applications or view functions.
- Register a blueprint on an application for any of these cases when initializing a Flask extension.

Blueprints are highly recommended to be used during future development. While Blueprints can help with many things current project implementation utilizes them mainly for entry-point splitting of the API. Each handler file inside handler directory amounts to a single Blueprint. More specific, handler/RollHandler.py contains the blueprint for the /rolls API entrypoint while
Aristotle University of Thessaloniki

handler/UserHanlder.py contains the blueprint for the /user API entrypoint. In terms of python code, it is translated into this;

```python
# RollHandler.py
rollHandlerBlueprint = Blueprint('roll', __name__)
api = Api(rollHandlerBlueprint, version='1.0', title='Athlos Server API - Roll Endpoints', description='Endpoints - /: Registers a new roll

# UserHandler.py
userHandlerBlueprint = Blueprint('user', __name__)
api = Api(userHandlerBlueprint, version='1.0', title='Athlos Server API - User Endpoints', description='Endpoints - /n register: Registers a new user
/login: Authorizes user

# app.py (where blueprints are declared)
app.register_blueprint(UserHandler.userHandlerBlueprint, url_prefix='/user')
app.register_blueprint(RollHandler.rollHandlerBlueprint, url_prefix='/roll')
```

**Swagger (applies to backend only)**

Swagger is a set of rules (in other words, a specification) for a format describing REST APIs. It is an open-source project highly regarded by the community developers and use on many large-scale projects. Swagger has been a great tool for the development of the backend because it helps with the designing, building, documenting, and testing the RESTful API being developed.

Here is a screenshot of the Swagger-UI tool landing page depicting all of the available endpoints provided by the backend along with the Models;
### Athlos Server API - User Endpoints

**Endpoints**
- register: Registers a new user
- login: Authorizes user

**Default Namespace**

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/friends/add</td>
</tr>
<tr>
<td>DELETE</td>
<td>/friends/delete</td>
</tr>
<tr>
<td>GET</td>
<td>/friends/list</td>
</tr>
<tr>
<td>GET</td>
<td>/friends/pending</td>
</tr>
<tr>
<td>GET</td>
<td>/friends/requests</td>
</tr>
<tr>
<td>GET</td>
<td>/friends/statistics</td>
</tr>
<tr>
<td>POST</td>
<td>/login</td>
</tr>
<tr>
<td>GET</td>
<td>/profile</td>
</tr>
<tr>
<td>POST</td>
<td>/register</td>
</tr>
<tr>
<td>POST</td>
<td>/{user}/friends/add</td>
</tr>
<tr>
<td>DELETE</td>
<td>/{user}/friends/delete</td>
</tr>
<tr>
<td>GET</td>
<td>/{user}/friends/list</td>
</tr>
<tr>
<td>GET</td>
<td>/{user}/friends/pending</td>
</tr>
<tr>
<td>GET</td>
<td>/{user}/friends/requests</td>
</tr>
<tr>
<td>GET</td>
<td>/{user}/friends/statistics</td>
</tr>
<tr>
<td>GET</td>
<td>/{user}/profile</td>
</tr>
</tbody>
</table>

**Models**

- Register
- Login
- AddFriend
- DelFriend

*Image 1. Backend Swagger-UI for /user entrypoint*
Image 2. Backend Swagger-UI for /roll entrypoint
Code Documenting

In a project that is aimed to scale in terms of functional complexity it is very important that it is documented as best as possible. Both athlos-backend and athlos-frontend use documentation comment stubs in the reStructuredText format. It is highly recommended for future developers to document every function and class created and explain through extensive commenting where complex logic is involved. Here is an example of comment stub used inside UserRepository.py to explain a query-function functionality;

```

Registers user inside db
:param username: Username of user
:type username: String
:param password: Password Hash
:type password: String
:param firstName: User's first name
:type firstName: String
:param lastName: User's last name
:type lastName: String
:param email: User's email
:type email: String
:param role: User's role (0 = coach, 1 = athlete)
:type role: String
:return:
:rtype: String

```

As it is going to be analyzed in the next section, the use of reStructuredText format can also work in conjunction with documentation automating and design Tools.
**Additions / Improvements**

Even though a major part of the envisioned functionality made it to the deliverable, there are a few things that because of the short timeframe could not be included into the implementation and will be good to be included in the future. The first one is an error tracking tool and the second one is a project documentation tool.

**Sentry**

Sentry is an application monitoring and error tracking software. Sentry provides self-hosted and cloud-based error monitoring that helps all software teams discover, triage, and prioritize errors in real-time. Among many programming languages, Sentry provides SDK support for python and more specifically, the Flask framework. In a project like Athlos Bowling that include multiple architectural levels and an API dependent infrastructure, monitoring of the whole application is very important. Especially when could-be-breaking changes to the code are deployed, Sentry can help with pinpointing errors and debugging.

(Sentry for Flask can be found here: https://sentry.io/for/flask)

**Sphinx**

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, it is open source and licensed under the BSD license.

It was originally created for the Python documentation, and it has excellent facilities for the documentation of software projects in a range of languages. Properly formatted and centralized documentation is very important for any project that aspires to scale and exchanges
Aristotle University of Thessaloniki

developer hands frequently. This is why Sphinx is a very important addition for future development. The following features of Sphinx should be highlighted:

- Output formats: HTML (including Windows HTML Help), LaTeX (for printable PDF versions), ePub, Texinfo, manual pages, plain text
- Extensive cross-references: semantic markup and automatic links for functions, classes, citations, glossary terms and similar pieces of information
- Hierarchical structure: easy definition of a document tree, with automatic links to siblings, parents and children
- Automatic indices: general index as well as a language-specific module-indices
- Code handling: automatic highlighting using the Pygments highlighter
- Extensions: automatic testing of code snippets, inclusion of docstrings from Python modules (API docs), and more
- Contributed extensions: more than 50 extensions contributed by users in a second repository; most of them installable from Pipi
- Sphinx uses reStructuredText as its markup language, and many of its strengths come from the power and straightforwardness of reStructuredText and its parsing and translating suite, the Docutils.

Foreseeing the use of Sphinx in the future is also why reStructuredText format was chosen for the class and function commenting inside the source code as demonstrated in the Code Documenting section of this paper.
Aristotle University of Thessaloniki

**Project testing and evaluation**

In this section, methods used for testing and evaluated (functionality wise) athlos-frontend and athlos-backend will be presented. Since both athlos-frontend and athlos-backend are Flask applications in their core, most of the testing revolves around proper API call execution. Behind every call there is vulnerable code logic that it is very important to be tested and evaluated for its performance. In both application cases the method used for verifying performance and functionality is logging. This is why a lot of emphasis has been placed in proper logging practices. Additionally, popular tools like Swagger have been used for manual testing as well as custom coded scripts that stress test the project by aggressively generating data for the endpoints to consume.

**Logging**

**In-code prints**

When evaluating the functionality of code, proper logging is very important to ensure that code behaves in an expected manner. Flask itself (for both the athlos-frontend and athlos-backend) by default when run in debugging mode:

```python
# Enable debugging logs in flask
Export FLASK_DEBUG = 1
```

prints on the terminal console and the browser itself (if calls performed from there) additional information regarding the code functionality as well as any errors. This is especially helpful when debugging new code additions since it gives a clear view of the actions the code performs and any errors that may exist. Here is a preview of the logs that Flask provides when debugging is enabled;
Along with the automated logging that Flask provides, inside the code, strategic log prints have been placed that aid with the logging of the code logic. Here is an example of such print inside the handler (UserHandler.py) blueprint;

```python
# User registration endpoint
@api.route('/register')
class Register(Resource):
    @api.expect(registerValidator)
    def post(self):
        print("[API] Got user registration request ")
        username = api.payload['username']
```

In the case of athlos-frontend that also serves GUI, Flask offers in-browser logs for ease of debugging. The Image below demonstrates some of the logs provided directly by Flask to the browser when there is an error occurrence;
Aristotle University of Thessaloniki

**SyntaxError**

File "/Users/anon/PycharmProjects/athlos-backend/app/views.py", line 161
print 'dasfd'

SyntaxError: Missing parentheses in call to 'print'. Did you mean print('dasfd')?

---

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object

Brought to you by DON'T PANIC, your friendly Werkzeug powered traceback interpreter.

*Image 3. Flask werkzeug in-browser logging*

---

**Pm2 logging**

Logging and debugging when both applications (athlos-backend, athlos-frontend) are deployed on a server (droplet) is assisted by the build-in feature of Pm2 called pm2 logs. To
access pm2 logs simple SSH into the deployment server and run the following command;

```bash
# Access pm2 logs
(sudo) pm2 logs
```

Here is a previous of the expected output;

```
2|app | [API] Got GET user rolls request
2|app | [API] Got GET Rolls Statistics request
2|app | [API] Got user profile request
2|app | User FOUND for username sakispap21
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:23] "GET /swaggerui/swagger-ui.html HTTP/1.1"
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:23] "GET /swaggerui/swagger-ui.html HTTP/1.1"
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:23] "GET /swaggerui/swagger-ui.html HTTP/1.1"
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:24] "GET /swagger.json HTTP/1.1"
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:24] "GET /swagger.json HTTP/1.1"
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:27] "GET /user HTTP/1.1" 308 -
2|app | 46.176.123.227 -- [15/Feb/2020 23:14:27] "GET /user/ HTTP/1.1" 200 -
```

Because both athlos-backend and athlos-frontend are managed by Pm2, pm2 logs show logging for both apps in conjunction. In green athlos-backend logs are displayed and in red, athlos-frontend logs are displayed. This means that any action performed that included functionalities for both frontend and backend can be very easily traced.

**Testing using Swagger (applies to backend only)**

Besides documenting Rest APIs, Swagger has built-in tools for manual testing. To test an endpoint, simply access the Swagger landing page on any browser and select the endpoint of interest. A new dialogue window will appear, by clicking the “Try it out” button, the API call payload can be configured and executed by clicking the “Execute” button as demonstrated below;
Testing using custom script (applies to backend only)

Alongside the development of the main project, a testing tool was developed in parallel to test the backend infrastructure. This tool serves both as a functionality test as well as a stress-testing tool. This tool has the form of a python script and can be found in the corresponding repository (athlos-tools). Refer to Project source code section of this paper for more information of repository location. This tool has a single dependency of python3.6 to run which should be already available since it is the same version as the rest of the project itself.
Aristotle University of Thessaloniki

Before running the script (generate_roll_data.py), first it has to be edited to provide information regarding the target server as well as testing condition parameters. After opening the script with the editor of choice, these are the variables that need to be edited:

```
# Target server URL
URL = “http://athlos.ddns.net”

# Target server PORT
PORT = 5000

# Test username (this user needs to be registered)
username = “mockname”

# Test password
password = “mockpwd”

# Generation interval (seconds) – low increases stress
Interval = 0.5
```

After parameters inside the script are setup, the script is ready to be executed;

```
# Execute testing script
python3 generate_roll_data.py
```

To expected behavior of the script (for the default variable values) is to emulate the generation of rolls on behalf of the provided user. All roll variables inside the payload are set to random but are kept within the allowed ranges. The frequency of set call is dictated by the interval specified. Emulation will stop once the script is terminated. Multiple instances of the script can be run for different users to increase stress.

**Project Evaluation Results**

The project was continuously tested and evaluated throughout the majority of the development cycle with the methods analyzed above. Both the core of the Athlos Bowling
Aristotle University of Thessaloniki

application (backend and frontend) as well as its ability to scale were put to the test and verified. Tests were conducted in controlled yet realistic environments.

Besides code testing and evaluation, end-to-end testing was also performed utilizing all architectural levels of the application including the Android application along with the sensors. Subjects (athletes, coaches) were instructed how to use the application and performed all basic workflows intended for the (full functionality display).

Empirically, the athlos-backend and athlos-fronted individually as well as a whole solution can be rated as TRL6 technology on the Technology Readiness Level scale.

![TECHNOLOGY READINESS LEVEL (TRL)](Image 6. Technology Readiness Levels)
Conclusion

To conclude, Internet of Things technologies can influence greatly sport science. By using technologies that are either open-source or free such as python-Flask and InfluxDB along with HTML and Bootstrap, we were able to build a scalable and reliable cloud solution that when combined with supplementary technologies (sensors, smart devices) forms a complete IoT ecosystem that can have great impact in the way coaching bowling is performed.

Athlos Bowling frontend and backend, by storing, manipulating and visualizing sensor data gathered by bowling athletes can help coaches of the sport be more efficient in directing and monitoring their athletes. This all comes down to providing the ability to branch out from empirical evaluation and additionally include precise arithmetic data. Pairing this with computer evaluation processes and the result is a very strong assistive tool.

It is also important to summarize that this thesis also invested a lot on the development process of said software. Best practices were investigated and selected, structuring the project in a way that can be furtherly expanded and developed. Finally, by using testing methods and creating supporting ones where needed the software produces was verified for its reliability and scalability before being delivered to the end user.
References

https://pypi.org/project/influxdb/
https://sentry.io/about/
https://getbootstrap.com/
https://affinitybridge.com/blog/pros-and-cons-docker
https://aws.amazon.com/timestream/
https://www.educba.com/django-vs-flask/
https://www.influxdata.com/blog/getting-started-python-influxdb/
https://Trackmyroll.com
https://Strikeseeker.com
https://Influxdb.com
https://aws.amazon.com
https://Palletsprojects.com
https://Sqlite3.com
https://Mysql.com
https://Pm2.com
https://Python.org
https://Swagger.io
https://Docker.io
https://Gitlab.com