Android GPS Data Logger

A. T. Coetzee

Detailed Concept Design

Supervisor:	**Prof. W. Venter**
Date:	**July 30, 2021**
Student Number:	**29982995**

Abstract

Trucking companies require a means to track their employees, ensuring they perform their work as expected. In order to achieve this, a tracking solution is considered making use of ubiquitous smartphone devices with built-in GPS capability. Using sensor data retrieved from smartphones, a solution is considered which intends to generate detailed reports for employers of trucking companies. Such a report is proposed to be displayed to the manager through a web application.

A draft high-level architecture for such a system is postulated and discussed, along with technical requirements, scope definition and deliverables.

CONTENTS

1 Introduction 1

1.1 Purpose of document 1

1.2 Background 1

1.3 Problem Statement 2

1.4 Hypothesis 2

1.5 Project Objective 2

1.5.1 Primary Objective 2

1.6 Anticipated Benefits of Solution 3

1.7 Technical Requirements 3

1.7.1 Requirements 3

1.7.2 Scope Definition 3

1.8 Deliverables 4

1.9 Conclusion 5

2 Literature Review 5

2.1 Internal and External Sensors 5

2.1.1 Internal Sensors 5

2.1.2 External Sensors 6

2.2 Smartphone Application 6

2.2.1 Platform Considerations 6

2.2.2 Development methodology 6

2.3 I/O Server 7

2.4 Database Considerations 7

2.5 Web Application 8

3 Design 8

3.1 System context and base requirements 8

3.2 Contained subsystems and choice of tools 10

3.2.1 Data Model 10

3.2.2 Android Application 11

3.2.3 I/O Server 11

3.2.4 Web Application 11

3.3 Subsystem components 11

3.3.1 Android Application 11

3.3.2 I/O Server 14

3.3.3 MySQL Database 16

3.3.4 Web Application 16

References 18

List of Figures

1 Proposed High Level Architecture 2

2 System Context Diagram 8

3	System Lifecycle - High Level	9
4	Container Diagram - Fleet tracking system	10
5	Life-cycle - Android Application	12
6	Component Diagram - Android Application	13
7	Life cycle - I/O Server	14
8	Component Diagram - I/O Server	15
9	Entity Relationship Diagram	16
10	Component Diagram - Web Application	17

1 Introduction

1.1 Purpose of document

This report documents the contextualization of a problem surrounding the tracking of truckers. The background and problem is considered, and possible solutions with objectives are identified, along with an expected outcome for the potential solution, requirements and scope definition. The design and implementation of a solution are investigated.

1.2 Background

Due to the nature of the trucking industry, it is difficult for company owners to keep track of their employees. Truckers carry out their shifts delivering important cargo to various locations over far distances. As such, it is not practical for employers to track their whereabouts throughout their shifts. This allows truckers the ability to behave undesirably while on the job. Truckers who waste time taking unnecessarily long stops or detours waste company time and money. In addition, some truckers may also be found breaking traffic laws without proper intervention. Such employees are a liability to the reputation and profitability of their respective companies.

The ability to track truckers would provide a potential means to address this issue, by allowing employers to monitor their truckers' location, progress and behavior throughout their shifts. The ability to produce an audit trail detailing the truckers whereabouts during their shifts would allow managers to ensure that work is adequately executed. Such an audit trail would comprise of:

- **GPS coordinates**
  GPS-tracking will allow employers to ensure that truckers are actually traveling to required locations, and doing so via the most effective routes. This also allows employers to ensure no unnecessary detours occur.
- **Altitude**
  An optional parameter which may be useful in some cases.
- **Speed**
  Examining the trucker's speed allows for managers to examine the effectiveness of cargo transportation, and to ensure that traffic laws are generally obeyed.

- **Acceleration**

  A potentially useful parameter which may be used for inferring any dangerous driving behavior.

The ubiquitous nature of cheap, GPS-equipped smartphones provides a potential avenue for realizing such a solution at low cost. In addition, country-wide continuous access to the internet allows for live tracking to be utilized.

1.3 Problem Statement

A smartphone-powered tracking system must be implemented to be used by trucking companies for tracking and logging their trucker's **GPS coordinates, altitude, speed and acceleration**.

In addition, an online interface is required for storing, processing and displaying logged data pertaining to the trucker's location and behavior.

1.4 Hypothesis

An anticipated architecture, as depicted in figure 1, involves the development of a smartphone application capable of interfacing with internal or external sensors, and transferring sensor data through some I/O server into a data store. [1] For the purposes of information presentation, it is proposed that a web server process and serve inferred information to a user via a web application.

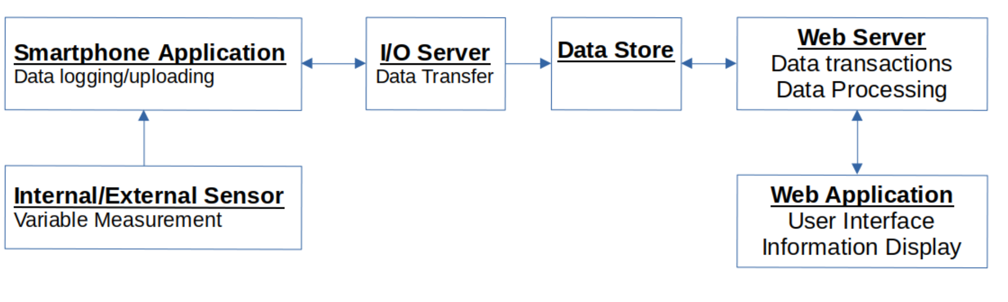

Fig. 1: Proposed High Level Architecture

1.5 Project Objective

1.5.1 Primary Objective: The primary objective in addressing the problem will be the development of detailed reports showcasing the trucker's whereabouts and behavior during their work shifts.

1.6 Anticipated Benefits of Solution

Managers will be able to ensure that their truckers conduct their work efficiently and responsibly. They will then be able to adequately handle truckers who fail to perform as expected.

Managers may also be able to analyze trucker behavior to perform optimizations, potentially allowing for increased efficiency.

1.7 Technical Requirements

1.7.1 Requirements:

1) **Smartphone Application**

   This will be a smartphone application used by the entities being tracked(i.e the truckers). Interface with external sensors must be possible. Every 2 minutes, sensor data capturing the **GPS coordinates, altitude, speed and acceleration** must be captured and stored. After which, this data must be pushed to a central data store via an I/O server through an internet connection.
2) **I/O Server**

   This server will facilitate the transfer of logged data from the Smartphone Application to a central data store, via an internet connection. As a dedicated transfer server, it must exhibit high performance, handling many requests from the smartphone client.
3) **Data Store**

   The data store will be efficient, fast and capable of storing large volumes of data. It should also be capable of adequately interfacing with the I/O server and the Web Server.
4) **Web Server**

   The web server is responsible for querying data from the data store, and serving requests to the web application. In addition, the web server is responsible for processing data into a report format, thereby allowing for important information extraction.
5) **Web Application**

   The Web application acts as a front end. It is responsible for the display of processed trucker information to managers. Reports detailing stopping times, transit times and speeds are displayed here.

1.7.2 Scope Definition: The scope of the problem considered will include

1) **Internal/External Sensor Interface**

   The scope **does not** include the design of sensor circuitry meant to interface with the smartphone. Only configurations capable of interfacing with the smartphone are considered. The smartphone app is not concerned with displaying user reports and statistics. That is left to the web application.

2) **Smartphone Application**

   The smartphone application is purely responsible for logging the appropriate sensor data and transferring the sensor data on through the I/O server. Other measurable variables such as temperature, fuel and pressure are not considered.

3) **I/O Server**

   The I/O server is purely responsible for facilitating the transfer of sensor data from the smartphone to the data store.

4) **Data Store**

   The data store element is purely concerned with the storage of logs, user identity information and providing an interface for the I/O server to query and add records to the store. I will consider existing database providers, and will not be building one from scratch.

5) **Web Server**

   The web server is purely responsible for querying from the data store. In addition it must process records for a shift into a format which highlights activity on a per-day basis.

6) **Web Application**

   The web application's scope entails User Identity validation. Reports must be displayed for an individual trucker's daily activity. That is, a visual map with markers indicating transit times and stationary times at some location.

1.8 Deliverables

The deliverables will require the entire project to function, from the smartphone logging implementation, to the detailed reports available in the web application.

- Smartphone Application and I/O server
- Web Application and web server

1.9 Conclusion

Basic contextualization of the problem has been performed. Low level details, however, have not been considered. Each aspect of the planned architecture may be realized in multiple ways on the low level. Further research and a feasibility analysis are necessary for adequate low level design.

2 Literature Review

This section tackles the investigation of components which make up the proposed high level system depicted in figure 1. There exists a variety of different tools available to realize each system,

2.1 Internal and External Sensors

Effective data logging of acceleration, altitude, location and speed all begin with the quality of measurements being made. Smartphones alone provide a wealth of options. However, external sensors available to the truck operators may also be considered.

2.1.1 Internal Sensors: Most smartphones come well-equipped with a wide variety of on-board sensors, such as global positioning (GPS) sensors, accelerometers, gyroscopes, magnetometers and ambient light sensors, among others [2]. As such, they are capable of inferring a wealth of information related to driving patterns. This includes dangerous driving behavior, for which algorithms have been developed [3].

The variety of on-board sensors provide an adequate means of measuring acceleration and location (and therefore altitude). However, no effective speed sensor exists for smartphone devices. GPS sensors may be used for inferring speed by computing location-time differentials, but with potentially questionable accuracy or possible performance reduction.

Battery life preservation and reduced performance are often concerns when running computationally heavy daemons. Recent efforts in the development and standardization of new, lightweight sensor-probing protocols have been investigated. Namely, Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP), which are targeted at achieving lightweight, low-power performance [4].

2.1.2 External Sensors: The most practical means of utilizing sensors external to the smartphone may be realized through the use of in-vehicle sensors. The Control-Area Network (CAN) bus protocol is a centralized multiplex communication bus standard utilized in many modern vehicles, originally in an attempt to save on copper. The protocol allows for broadcast communication between various electronic control units (ECUs) within a vehicle, all centrally connected to one bus. A priority-based scheme is utilized to ensure the most important units transmit their data packets first, while lower priority units are delayed until a later time when transmission may be uninterrupted. Each packet contains an identifier designating what information is being transmitted, such as wheel speed, temperature, etc. [5]

Assuming that the vehicle has an on-board diagnostic (OBD) connector, communication with a smartphone requires some form of interfacing circuitry. Wireless CAN-to-smartphone interfaces can be most-practically realized via CAN-bus-to-Bluetooth implementations. Such an interface will allow for the smartphone to probe sensor data via the vehicle's CAN bus [6] [7]. The Society of Automotive Engineers (SAE) defines the J1939 standard for CAN-bus communication in the use of heavy-duty vehicles [8], which would be appropriate for the solution.

2.2 Smartphone Application

The smartphone application is responsible for extracting the acceleration, altitude, location and speed data from the sensors and relaying this information to the data store. Certain platform and development design decisions are investigated.

2.2.1 Platform Considerations: The two major mobile operating systems are Android (approximately 72.8 % market share) and iOS (approximately 27.4 % market share) [9]. Android's high market share makes it an attractive option as a target platform for the Smartphone application component of the system.

2.2.2 Development methodology: Native Android development officially supports the Java, Kotlin, C and C++ programming languages. Kotlin, which compiles on the Java Virtual Machine (JVM), has been pushed by Google as their suggested language for app development. Kotlin aims to reduce the verbosity of traditional Java (which was the standard language used for app development), thereby reducing the prevalence of "bad coding practices." [10] It is noted that Java may still be preferable for programmers with prior Java experience, or in cases where more verbosity is preferred. A native C/C++ tool-chain offers finer control of system hardware for potential performance boosts [11].

Cross-platform development presents a popular option for developing applications for both major platforms. Several development frameworks such as Xamarin, Flutter and Apache Cordova allow for cross-platform development, among others. However, cross-platform development does impose potentially reduced performance, according to [12]. In an ecosystem where hardware used by truck drivers has potential to be slower, cross-platform development is undesirable.

2.3 I/O Server

The I/O server is required for relaying logged data from the smartphone application to the central data store. It must be many clients quickly and efficiently. This server plays a typical server role; In that it must await requests from clients attempting to establish connection for transmitting data.

Implementations for realizing such a server are possible in many programming languages, and almost all top popular programming languages. Generally, for performance-critical applications, C and/or C++ are considered most appropriate. [13]

2.4 Database Considerations

Relational Database Management Systems (RDBMS) are commonly used in for data handling. Typically, for unnormalized complex data, conventional structured query language (SQL) RDBMSs prove inefficient at scale, due to the tendency of modern data catalogues lacking in structure. In addition, relational databases also start to exhibit slower lookup times for immensely large data sets. The solution to this comes in the form of NoSQL (Not only SQL) database systems, which are scaleable, efficient and capable for storing large volumes of unnormalized data. [14] [15] [16]

However, due to the completely uniform structure of the data being stored, an RDBMS would suffice. Numerous high quality RDBMSs, such as MySQL, Microsoft SQL, PostgreSQL, and Oracle Database are available, among others. All options offer relatively efficient performance. [17]

A lightweight caching database is necessary on the client-side for the momentary storage of data which has yet to be transmitted to the server. To this end, SQLite offers a popular solution for smartphone applications [18].

2.5 Web Application

The web application will be used by managers to display daily reports highlighting their truckers' behavior throughout their shifts.

The web application may be easily realized by utilizing pre-existing web frameworks, such as Microsoft's ASP.NET Core and Oracle's Java EE (with comparable performance) [19].

3 Design

This section considers the context in which the problem exists and the design of each system and subsystem necessary to visualize and realize a possible solution to solve the problem.

The nature of the system exists primarily in the software domain. As such, a suitable design architecture is postulated by the C4 model. This model breaks down the system architecture into different layers of complexity, from a generic high-level system overview down to low-level software abstractions.[20]

Low-level abstractions are realized with unified modeling language (UML) diagrams. UML diagrams detail the members and methods belonging to classes, and the relationships between those classes in an object-oriented codebase. [21]

3.1 System context and base requirements

Figure 2 depicts the system context in the problem domain. Project specifications have identified two parties expected to utilze the system - the truck drivers and the fleet managers. Identified requirements on the solution dictate that truck drivers will use an android application to log data on the system. In addition, fleet managers must view the logged data and manipulate their fleets via a web application running in a browser.

Fig. 2: System Context Diagram

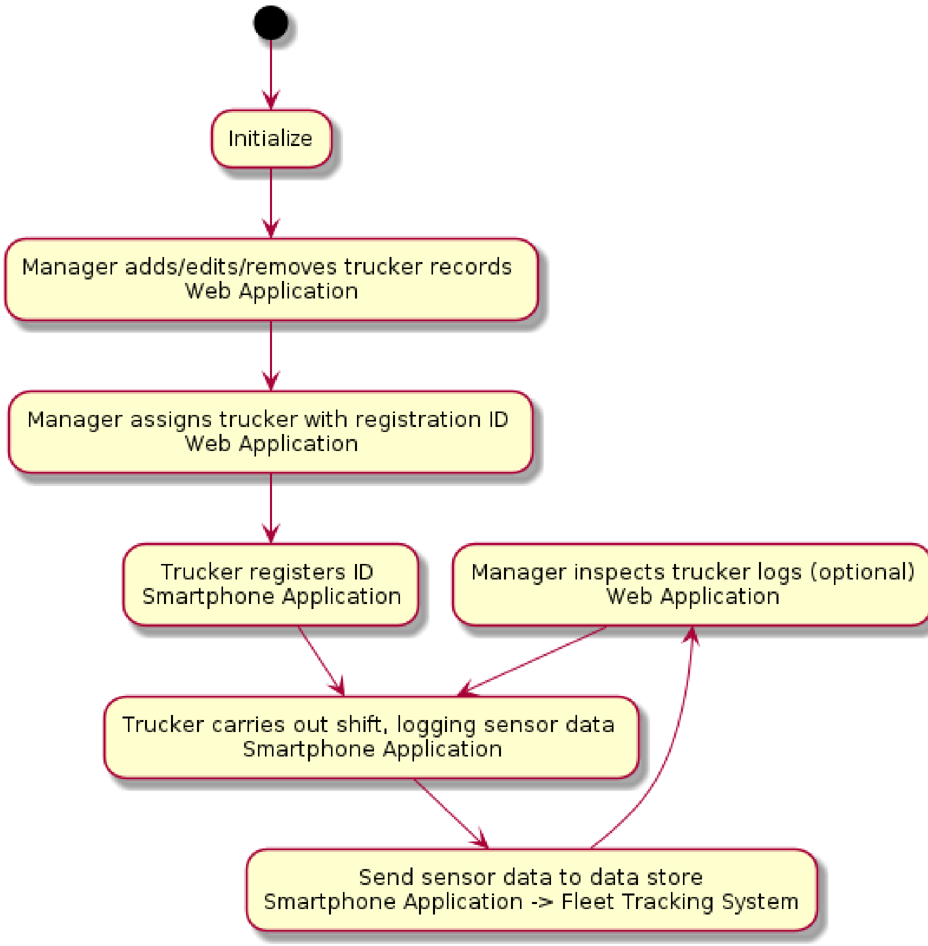

Fig. 3: System Lifecycle - High Level

The high-level life cycle view of the fleet-tracking system design is depicted in figure 3. This life cycle view gives a broad indication of how the system is expected to work for a user. Only front-end components of the system are considered to clarify exactly how users will interact with the system.

Managers are required to perform initial configuration, as well as adding trucker records to a data store. After this, truckers may connect to the system and perform their work while allowing their smartphone applications to track the required sensor data. This data is then relayed to the system, in which managers may analyze and inspect data logs.

3.2 *Contained subsystems and choice of tools*

The second level of the C4 model identifies the choice of technologies to be utilized to realize the fleet tracking system. The fleet-tracking system is divided into mostly-independent containers, as depicted in figure 4. Each container is a standalone process which makes calls to other processes in the system. The main choice of software tools are identified for each container.

Truckers will make use of an android data-logging application to fetch the various sensor data, and securely transmit this data via an SSL connection. The IO server, implemented in C++, will listen for multiple asynchronous connections from the android application and relay the data to a MySQL database. A web application, realized with the ASP.NET framework fetches the data and allows the fleet manager to view the whereabouts of each member in his/her fleet.

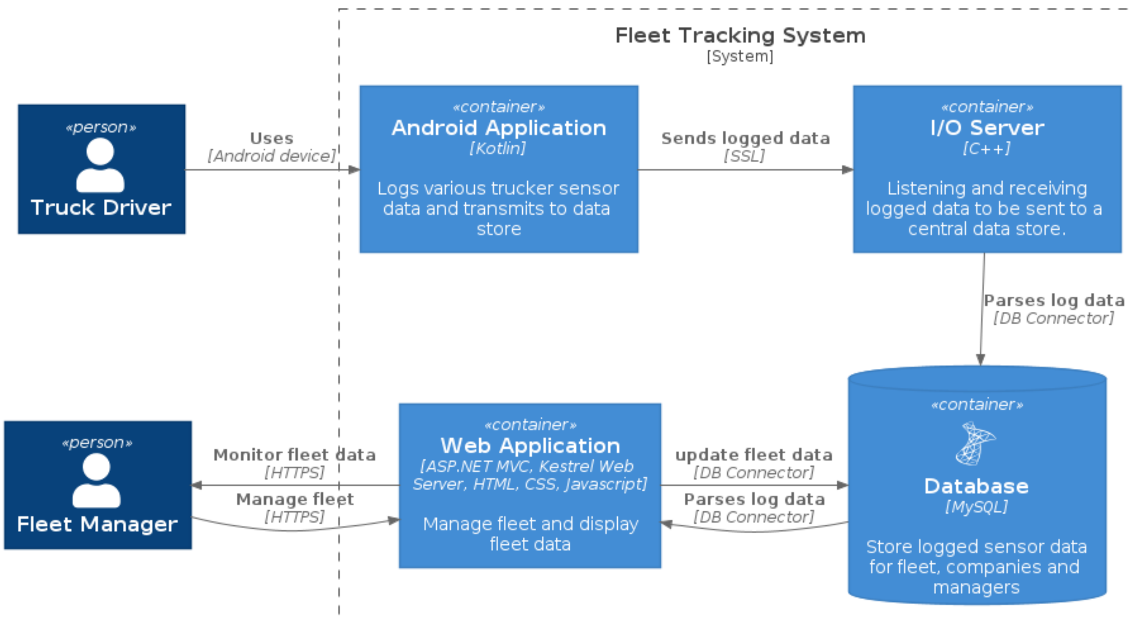

Fig. 4: Container Diagram - Fleet tracking system

3.2.1 Data Model: The entire system revolves around the effective abstraction and manipulation of logged fleet data. MySQL is chosen as the RDBMS to realize a relational database model, as it is high performance and reliable. Other database systems offer comparable performance, but MySQL is chosen for familiarity.

The relational model is depicted in figure 9. The model is designed to allow one company to have many managers and truckers. Each trucker can have many logs.

3.2.2 Android Application: Kotlin is the language of choice to write the android application due to its simplicity and mainstream Google support.

Truckers must receive an initial code from their managers' to register their devices. Sensor readings are taken every two minutes, and stored into a lightweight database. Finally, a connection is attempted with an I/O server. If available, the database contents is emptied into via the I/O server to the central system database.

3.2.3 I/O Server: C++ is chosen for the I/O server, due to its high performance capabilities. The I/O server needs to listen and allow multiple asynchronous connections, during which log data is transmitted to the database.

3.2.4 Web Application: The model-view-controller (MVC) architecture will be realized with the Microsoft ASP.NET framework. This architecture allows for separation between business-logic, data models and viewing logic. This is necessary to ensure that code related to displaying data is not mixed with code used for core logic, thereby separating and modularizing the functionality of different components in the system.

3.3 Subsystem components

Each container in figure 4 is subdivided into several core software components necessary to achieve the desired outcomes. This is depicted through container diagrams, which makes up the third level of the C4 model.

3.3.1 Android Application: The expected life cycle of the Android application is depicted in figure 5. Initially, a check is performed to confirm that the trucker ID is in the central database, and is not duplicated. If this ID is not valid, the trucker must request a valid ID from the fleet manager.

After this, the usual logging process is continued. Data is polled from the available sensors and stored in a local database. A connection is attempted with the I/O server and the local database entries are transmitted to the server. Upon successful transmission, the local database is emptied.

However, if a connection fails, the local database is not cleared. This process loops continuously loops every two minutes.

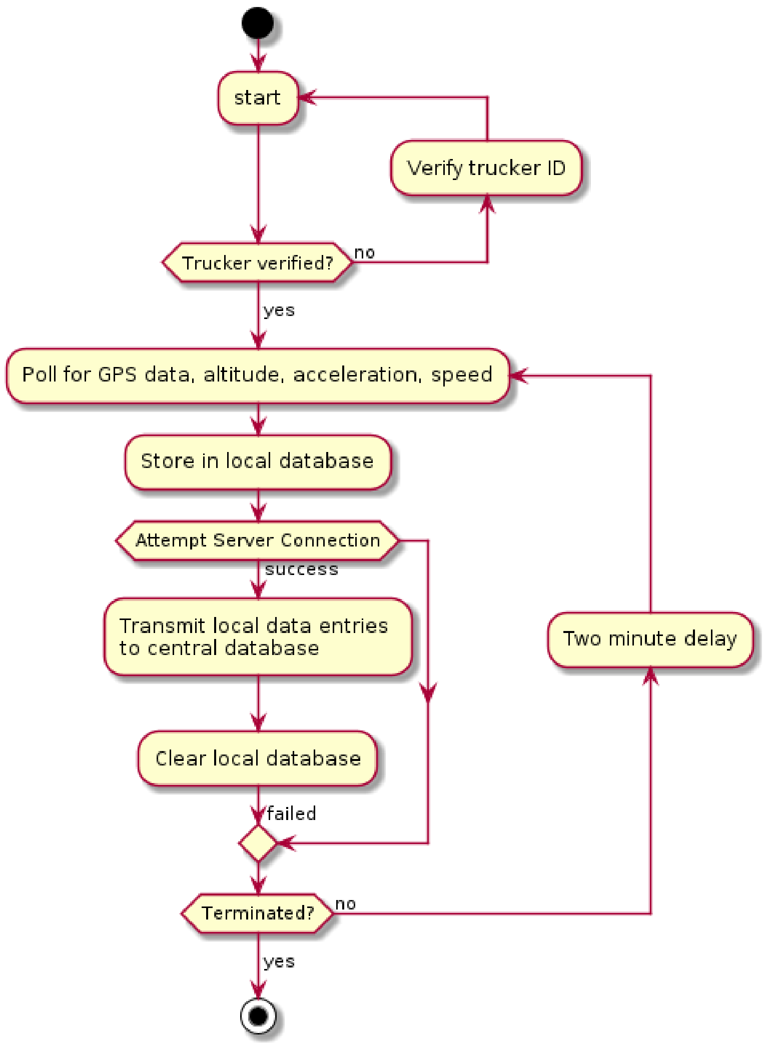

Fig. 5: Life-cycle - Android Application

Figure 6 depicts the system components necessary to realize the required functionality. The logging controller collects sensor information and interacts with the local database and central I/O server. Various Android APIs are accessible and expose Location and sensor information to the Logging controller.

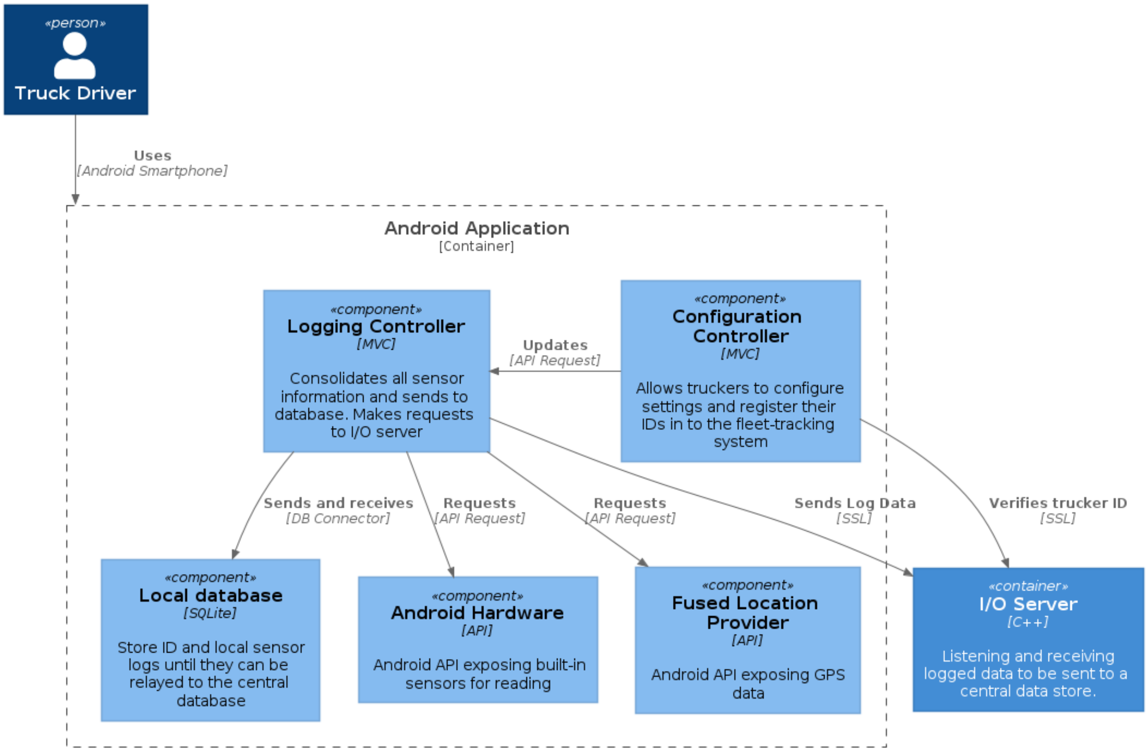

Fig. 6: Component Diagram - Android Application

3.3.2 I/O Server: The typical life cycle view of the I/O server is depicted in figure 7. A secure connection must be made due to the sensitive nature of GPS data.

The I/O server contains a request handler to process the request sent by the android client. It then calls the specific function as determined and parses the appropriate data via the JSON format.

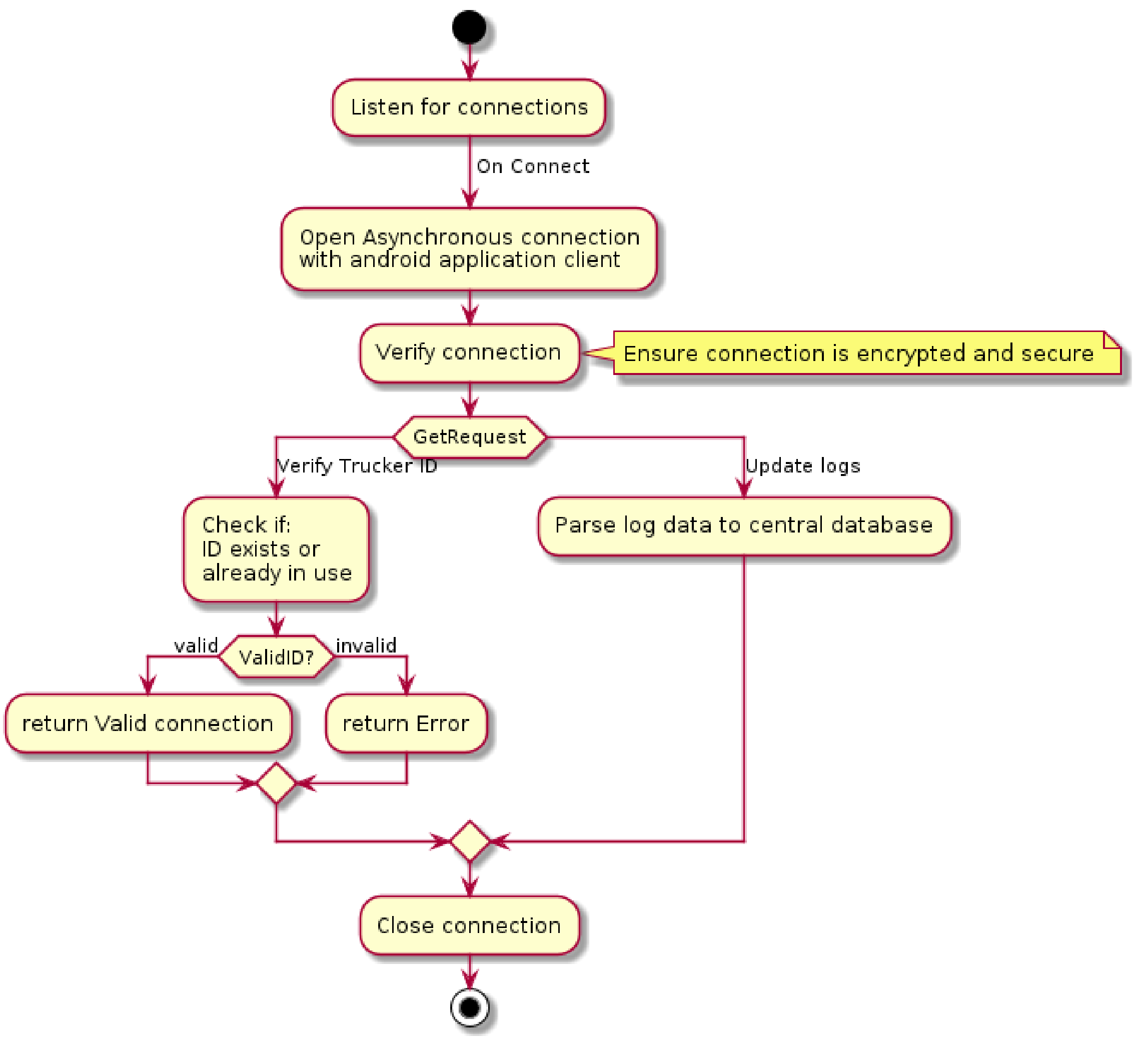

Fig. 7: Life cycle - I/O Server

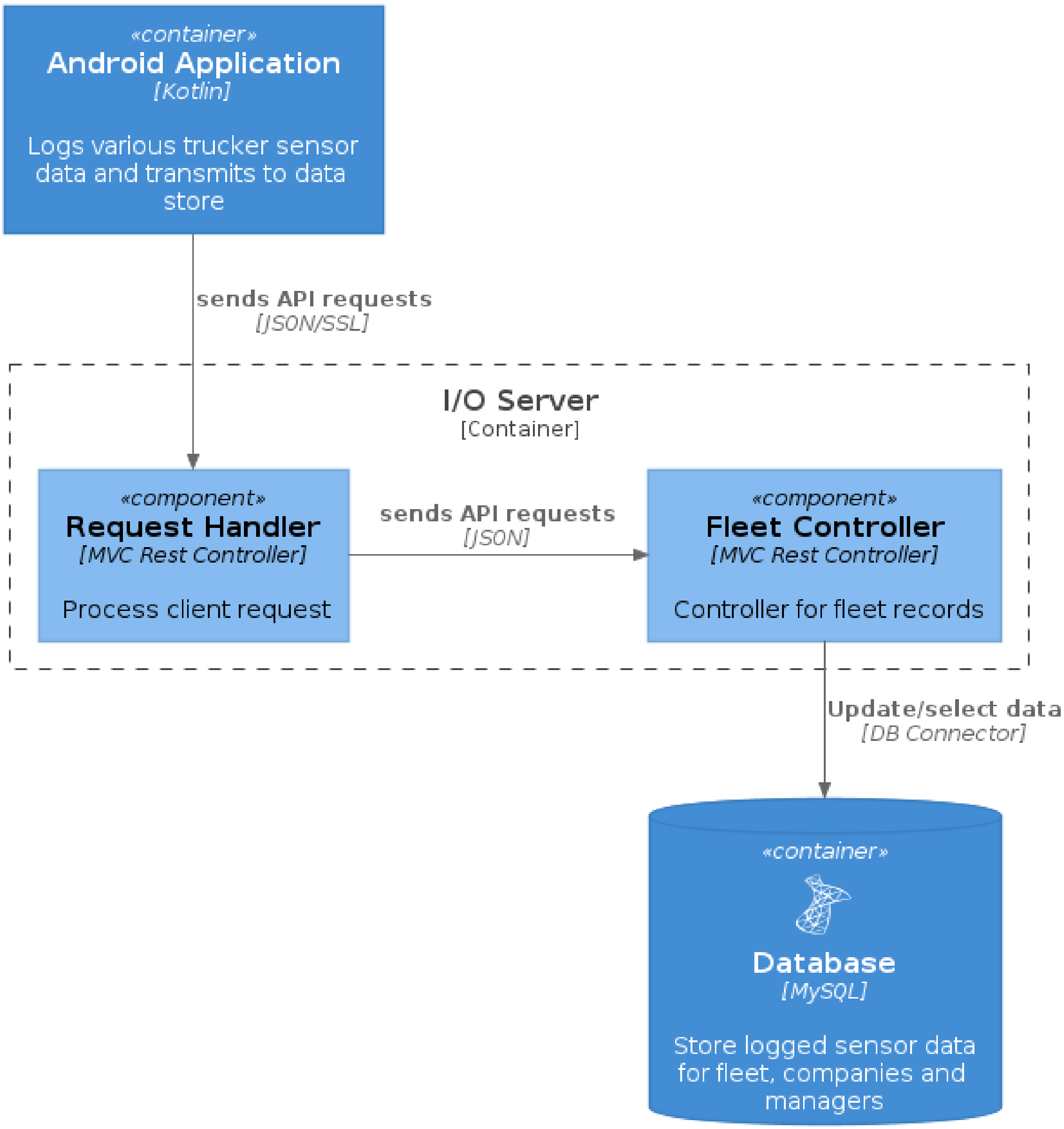

Fig. 8: Component Diagram - I/O Server

Figure 8 depicts the component make-up of the I/O server. The codebase clearly contains these low-level abstractions.

3.3.3 MySQL Database: The MySQL database is driven by MySQL server. A relational data structure is utilized, as shown in figure 9. Relational modelling allows for logical structuring and integrity of the data.

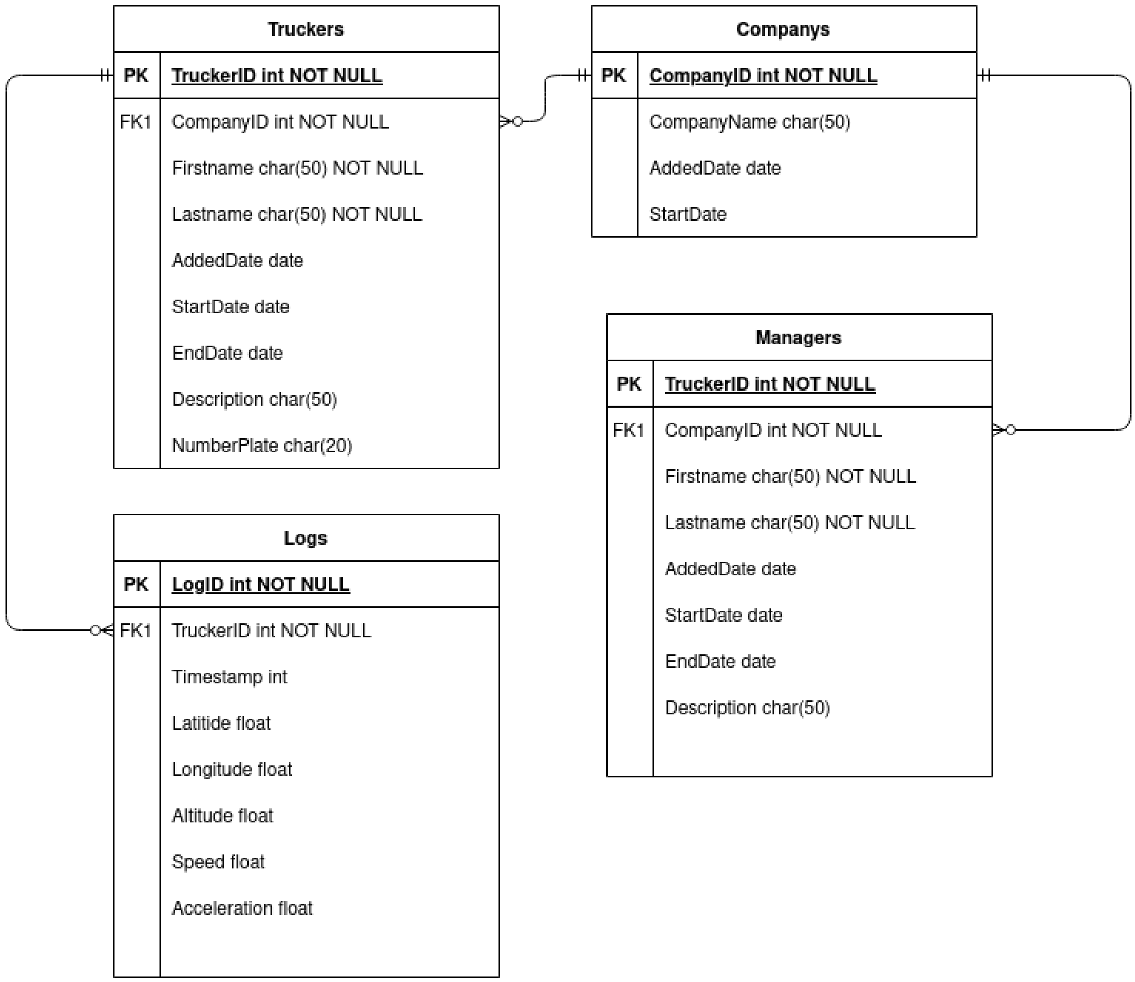

Fig. 9: Entity Relationship Diagram

3.3.4 Web Application: The web application is modeled with the view-model-controller design pattern, which allows separation of UI logic from business logic. Two view models are considered to allow for managers to sign in and manage their fleets. Controllers handle core logic behind the presentation of data. The central database is also accessed by the controller. Figure 10 depicts the architectural arrangement of the web application.

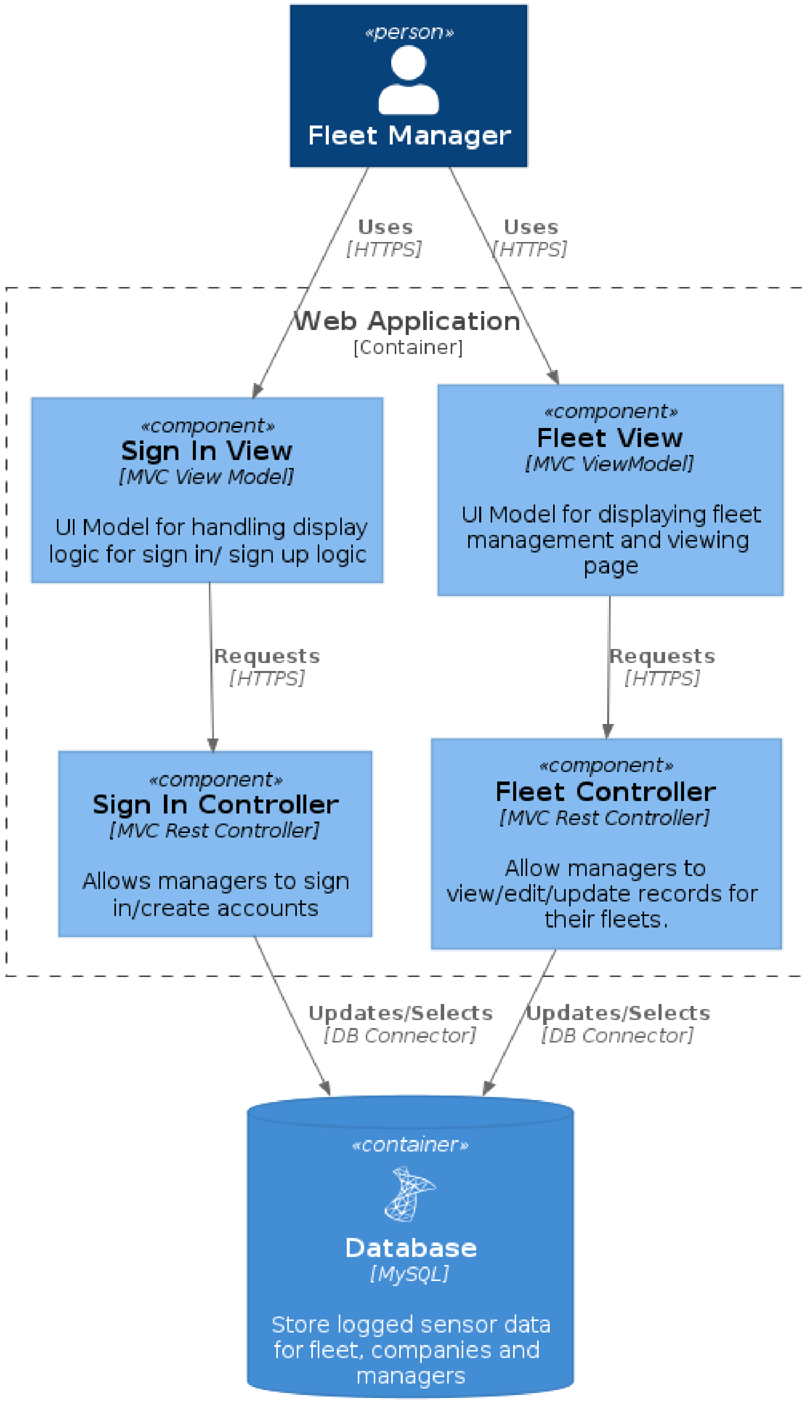

Fig. 10: Component Diagram - Web Application

REFERENCES

[1] M. Bertocco, F. Ferraris, C. Offelli, and M. Parvis, "A client-server architecture for distributed measurement systems," *IEEE transactions on instrumentation and measurement*, vol. 47, no. 5, pp. 1143–1148, 1998.

[2] S. Majumder and M. J. Deen, "Smartphone sensors for health monitoring and diagnosis," *Sensors*, vol. 19, no. 9, p. 2164, 2019.

[3] F. Li, H. Zhang, H. Che, and X. Qiu, "Dangerous driving behavior detection using smartphone sensors," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1902–1907.

[4] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*. IEEE, 2013, pp. 1–6.

[5] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011, p. 20.

[6] C. Campolo, A. Iera, A. Molinaro, S. Y. Paratore, and G. Ruggeri, "Smartcar: An integrated smartphone-based platform to support traffic management applications," in *2012 first international workshop on vehicular traffic management for smart cities (VTM)*. IEEE, 2012, pp. 1–6.

[7] O. Walter, J. Schmalenstroeer, A. Engler, and R. Haeb-Umbach, "Smartphone-based sensor fusion for improved vehicular navigation," in *2013 10th Workshop on Positioning, Navigation and Communication (WPNC)*. IEEE, 2013, pp. 1–6.

[8] M. R. Stepper, "J1939 high speed serial communications, the next generation network for heavy duty vehicles," SAE Technical Paper, Tech. Rep., 1993.

[9] "Mobile operating system market share worldwide." [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[10] M. Flauzino, J. Veríssimo, R. Terra, E. Cirilo, V. H. Durelli, and R. S. Durelli, "Are you still smelling it? a comparative study between java and kotlin language," in *Proceedings of the VII Brazilian symposium on software components, architectures, and reuse*, 2018, pp. 23–32.

[11] D. Kwan, J. Yu, and B. Janakiraman, "Google's c/c++ toolchain for smart handheld devices," in *Proceedings of Technical Program of 2012 VLSI Technology, System and Application*. IEEE, 2012, pp. 1–4.

[12] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, "An empirical

investigation of performance overhead in cross-platform mobile development frameworks," *Empirical Software Engineering*, vol. 25, pp. 2997–3040, 2020.

[13] J. O. Ogala and D. V. Ojie, "Comparative analysis of c, c++, c# and java programming languages," *GSJ*, vol. 8, no. 5, 2020.

[14] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "Nosql databases: Critical analysis and comparison," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. IEEE, 2017, pp. 293–299.

[15] M. A. Qader, S. Cheng, and V. Hristidis, "A comparative study of secondary indexing techniques in lsm-based nosql databases," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 551–566.

[16] G. Ongo and G. P. Kusuma, "Hybrid database system of mysql and mongodb in web application development," in *2018 International Conference on Information Management and Technology (ICIMTech)*. IEEE, 2018, pp. 256–260.

[17] W. Truskowski, R. Klewek, and M. Skublewska-Paszkowska, "Comparison of mysql, mssql, postgresql, oracle databases performance, including virtualization," *Journal of Computer Sciences Institute*, vol. 16, pp. 279–284, 2020.

[18] S. Bhosale, T. Patil, and P. Patil, "Sqlite: Light database system," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 4, pp. 882–885, 2015.

[19] K. Kronis and M. Uhanova, "Performance comparison of java ee and asp. net core technologies for web api development," *Applied Computer Systems*, vol. 23, no. 1, pp. 37–44, 2018.

[20] A. Vázquez-Ingelmo, A. García-Holgado, and F. J. García-Peñalvo, "C4 model in a software engineering subject to ease the comprehension of uml and the software," in *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2020, pp. 919–924.

[21] M. Petre, "Uml in practice," in *2013 35th international conference on software engineering (icse)*. IEEE, 2013, pp. 722–731.