Android GPS Data Logger

A. T. Coetzee

Project Report

Supervisor:	**Prof. W. Venter**
Date:	**October 6, 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

List of Figures

ACRONYMS

app application, 9

CAN Control Area Network, 6 f.

CoAP Constrained Application Protocol, 6

DI Dependency Injection, 8 f.

ECU Electronic Control Unit, 6

GPS Global Positioning System, 1 ff., 6

HTTPS Hypertext Transfer Protocol Secure, 12

HTTP Hypertext Transfer Protocol, 12

iOS iPhone Operating System, 8

I/O Input/Output, 2–5, 9 ff., 14

JSON JavaScript Object Notation, 12

JVM Java Virtual Machine, 8

MVVM Model-View-ViewModel, 9

MVC Model-View-Controller, 11

MQTT Message Queuing Telemetry Transport, 6

NoSQL Not only SQL, 11

ORM Object Relational Mapper, 9

OOP object-oriented programming, 7

OBD on-board diagnostic, 7

RDBMS Relational Database Management Systems, 11

SSL Secure Socket Layer, 11 f., 14

SoC Separation of Concerns, 7, 11

SQL Structured Query Language, 11

SAE Society of Automotive Engineers, 7

TLS Transport Layer Security, 12

UML Unified Modelling Language, 13

XML Extensible Markup Language, 12

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:

- **Global Positioning System (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 Input/Output (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.

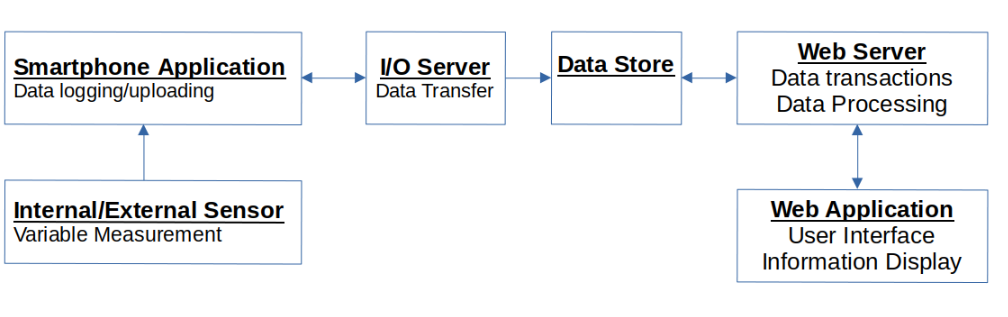

Fig. 1: Proposed High Level Architecture

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).

   a) Trucker identification control must be implemented to ensure that logs sent to the server correspond to a unique trucker. It must not be possible for multiple truckers to assume the same or no identity.
   b) Every 2 minutes, sensor data capturing the **GPS coordinates, altitude, speed and acceleration** must be captured and stored internally on the android device. Data capacity for one continuous week of storage must be possible, to account for connectivity issues.
   c) The application must be able to run in the background, allowing for multitasking.
   d) Sensor data must be uploaded to a central data store, either continuously or on request. This communication must be encrypted for security purposes.

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.

   a) As a dedicated transfer server, it must exhibit high performance, handling multiple requests from the multiple smartphone clients asynchronously.
   b) Trucker logs, received from smartphone clients, must be stored in a central database.
   c) Information about the trucking company must also be sent to 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. The web server is responsible for querying data from the data store, and serving requests to the web application.

4) **Web Server and Web Application**

   The web server must implement back-end business logic and serve pages in the web application. The web application acts as an interface for managers to add truckers and view tracking information about their fleets.

   a) Multiple trucking managers must be able to log in and use the application.
   b) Managers must be able to add multiple truckers to their fleet, including trucker-specific information such as name, and vehicle number.
   c) Managers must be able to view detailed trip information for any adjustable time period. Log data must be processed to determine starting and arrival times for locations traveled to. Statistical information about acceleration and speed should be displayed, including averages, maximums and percentiles.

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. Existing data store providers will be considered.

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. With the hardware pre-existing, most of the design exists in the software domain. Various software tools and methodology are considered.

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 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 fluctuating accuracy or possible performance reduction.

Battery life preservation and reduced performance are often concerns when running computationally heavy daemons (background operating system processes). 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 Unit (ECU)'s 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 Software Architecture

Effective software architecture and design patterns are necessary for writing dynamic, modifiable and modular software.

2.2.1 Separation of Concerns (SoC) and SOLID principles: SoC addresses the need for software to be decomposed into different modular units. Each unit focuses on one main concern, such as data access, authentication, business logic and view rendering. Mixing multiple concerns within one unit leads to code which is less reusable and more difficult to modify. [9]

The 'SOLID' acronym defines a set of guidelines for software design, in object-oriented programming (OOP).

1) **Single Responsibility Principle**
   Classes should have single responsibilities. To achieve this, each responsibility must be implemented in a unique class.
2) **Open/Close Principle**
   Software components such as classes, modules and functions should be open for extension, but closed for modification. That is, classes implementing a modifiable functionality should be extended with interfaces instead of modifying code in the class.
3) **Liskov substitution Principle**
   Objects should be replaceable with derived sub-types without affecting the correctness of the program.
4) **Interface Segregation Principle**

It is better to implement many client-specific interfaces instead of one general-purpose interface. This ensures the interface being implemented only does the minimal that is required.

5) **Dependency Inversion Principle**
   Where possible, it is better to depend on implementable abstractions instead of concretely defined objects. This can be realized by depending on implementable interfaces instead of base classes. This allows classes to be less tightly-bound to a base class, allowing for more modular code.

[10]

2.2.2 Dependency Injection (DI): Often classes require instances of other objects (or dependencies) to perform certain functions. It is wasteful to re-instantiate these objects especially if they are used by other classes. DI provides a means to *inject* an instance of a helper object into a class without explicitly recreating the dependency. [11]

Objects which exist for the lifetime of the application are known as singletons, and the use of singletons is often used with DI.

2.3 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.

2.3.1 Platform Considerations: The two major mobile operating systems are Android (approximately 72.8 % market share) and iPhone Operating System (iOS) (approximately 27.4 % market share) [12]. Android's high market share makes it an attractive option as a target platform for the Smartphone application component of the system.

2.3.2 Development Technologies: 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." [13] 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 [14].

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 [15]. In an ecosystem where hardware used by truck drivers has potential to be slower, cross-platform development is undesirable.

2.3.3 Android - Model-View-ViewModel (MVVM) Design Pattern: Figure 2 depicts the MVVM architecture used in a typical android context. The view (typically activities or fragments in Android) represents the actual rendered output visible to the user. Data displayed by the view is accessed by the view model. The separation of views and view models is necessary for Android applications due to the temporary nature of views. That is, data stored purely in the view component is lost upon re-rendering of the view, while view models hold onto data for longer.[16]

The repository singleton acts as a central holder of application data, which is then accessed by the multiple views. It also interacts with I/O resources such as web resources and database access. Views request data through the repository, and as such shouldn't have direct handles to database connections. [16]

2.3.4 Android - DI with Hilt: Hilt is an android library used for easily implementing DI. It has support for common android components. [17]

2.3.5 Android - Running continuously in the background: Tracking application (app)s need to run continuosly, without forcing the user to keep the app view components open. This can be achieved by implementing the tracking component as a *foreground service*. In this way, the component runs continuously while allowing the user to use other applications.

Users must also be notified of continuously running services for clarity. It is therefore required display notifications about the service. [18]

2.3.6 Android - SQLite database with the Room Object Relational Mapper (ORM): The Room ORM library provides a neat database abstraction layer over SQLite useful for modeling data. SQLite is preferable for android due to its lightweight nature. [19]

The storage capacity of SQLite is basically unlimited. Storage capacity is, however, limited to the storage capability of the smartphone running the application. This makes the use of external storage desirable.

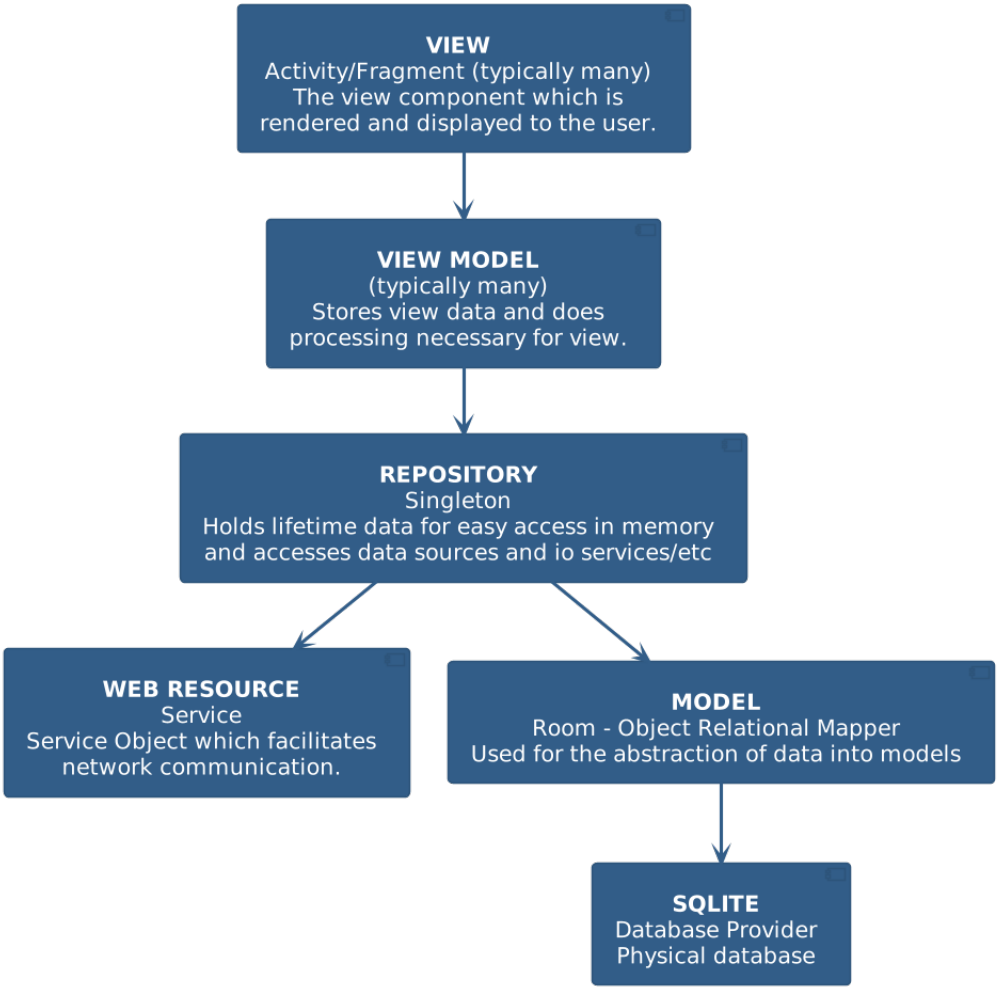

Fig. 2: Android - MVVM Architecture

2.4 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. [20]

2.4.1 Asynchronous I/O: Servers (and many other application) are required to run relatively slow operations; that is communicating over networks and writing to disk. Implementing such functionality synchronously (using blocking calls) leaves functions essentially waiting for data streams to be read, transmitted and written to disk. This is slow and incapable of handling multiple simultaneous connections.

Asynchronous I/O operations enables other processing to continue before a slow I/O operation has completed. This is essential for servers which handle many simultaneous connections. A popular C++ library, *asio* provides asynchronous I/O functionality. [21]

2.5 Database Considerations

Relational Database Management Systems (RDBMS)s 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 Not only SQL (NoSQL) database systems, which are scaleable, efficient and capable for storing large volumes of unnormalized data. [22] [23] [24]

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. [25]

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 [26].

2.6 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 Enterprise Edition (with comparable performance) [27].

2.6.1 MVC design pattern for web applications: A relatively popular design pattern in web development is the MVC architecture. As seen in figure 3, MVC attempts to achieve SoC by separating logic required for viewing, routing and data into separate components.

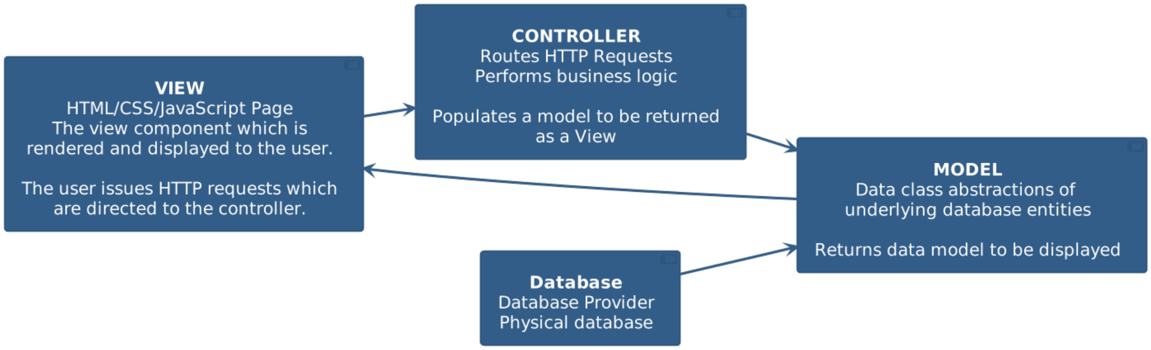

Fig. 3: Web Design Pattern - Model-View-Controller (MVC)

2.7 Secure communication with Secure Socket Layer (SSL)

The use of secure communication over the internet is a modern-day standard. And due to sensitive location data being transmitted, it is necessary to ensure that logs are adequately encrypted.

The SSL protocol is a de facto standard for encrypted communication on the internet. SSL itself is deprecated, and the current standard for encryption is Transport Layer Security (TLS). However, it is common to refer to refer to these related technologies interchangeably, when TLS is the protocol actually in use. [28]

2.8 Serialization and communication protocols

Facilitating communication between two devices requires both devices to use the same protocol. Regardless of this protocol, it is necessary for communication to be encrypted, therefore making use of SSL.

2.8.1 Hypertext Transfer Protocol Secure (HTTPS): HTTPS implements the de facto Hypertext Transfer Protocol (HTTP) protocol over the encrypted SSL protocol. HTTP is an application layer protocol which makes use of standard headers carrying a payload under formalized request types, of which *GET* and *POST* are common. HTTPS is commonly used for web services and websites. [29].

2.8.2 JavaScript Object Notation (JSON) and Extensible Markup Language (XML): XML is a strongly-typed text protocol which can be used for serialization. It follows a tight tagging schema.

JSON is a fast and simple text protocol for serializing objects carrying data. Support for arrays makes JSON reliable for the transmission of many logs. [30]

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.[31]

Low-level abstractions are realized with Unified Modelling Language (UML) diagrams. UML diagrams detail the members and methods belonging to classes, and the relationships between those classes in an object-oriented codebase. [32]

3.1 System context and base requirements

Figure 4 depicts the system context in the problem domain. Project specifications have identified two parties expected to utilize the system - 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. 4: System Context Diagram

The high-level life cycle view of the fleet-tracking system design is depicted in figure 5. 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, including adding trucker identity 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.

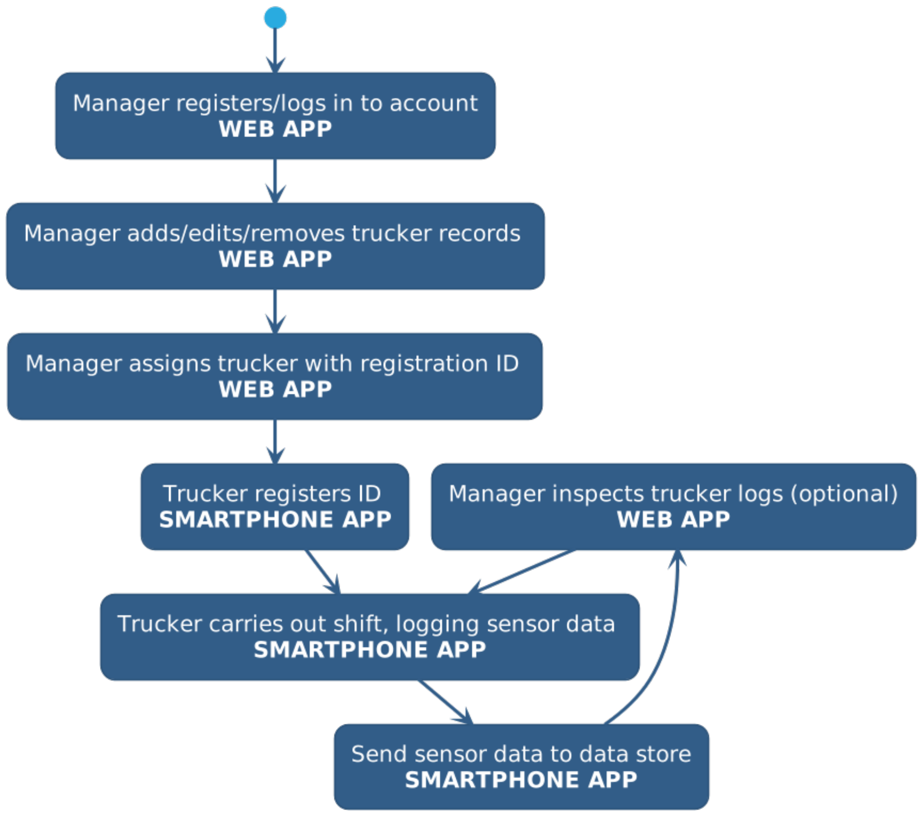

Fig. 5: System Lifecycle - High Level

3.2 Contained subsystems and choice of technologies

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 6. 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 I/O 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 Microsoft's ASP.NET framework fetches the data and allows the fleet manager to view the whereabouts of each member in his/her fleet.

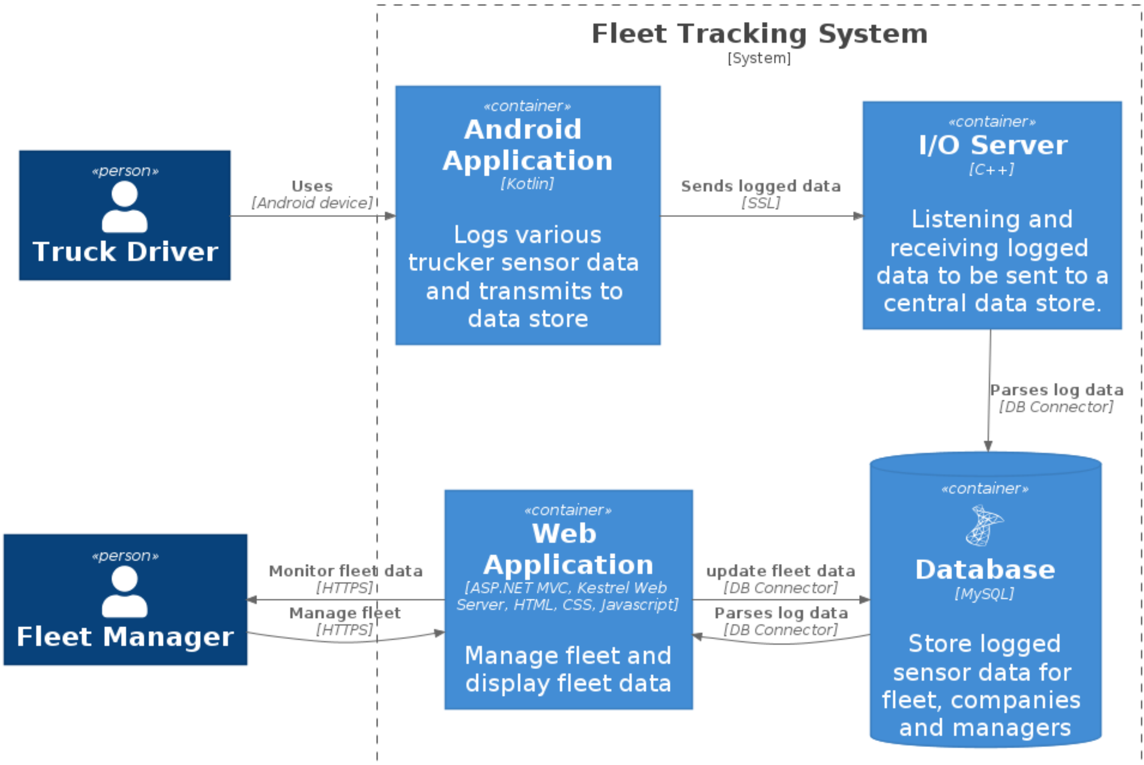

Fig. 6: Container Diagram - Fleet tracking system

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 11. 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 6 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 7. 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 cleared.

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

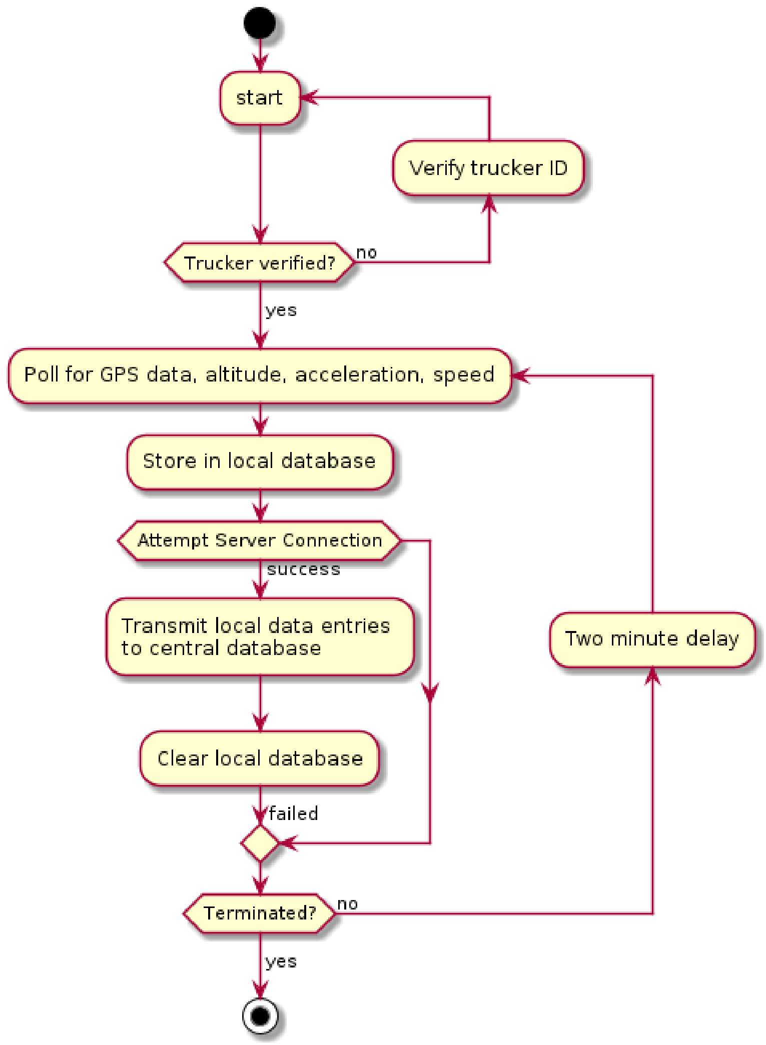

Fig. 7: Life-cycle - Android Application

Figure 8 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.

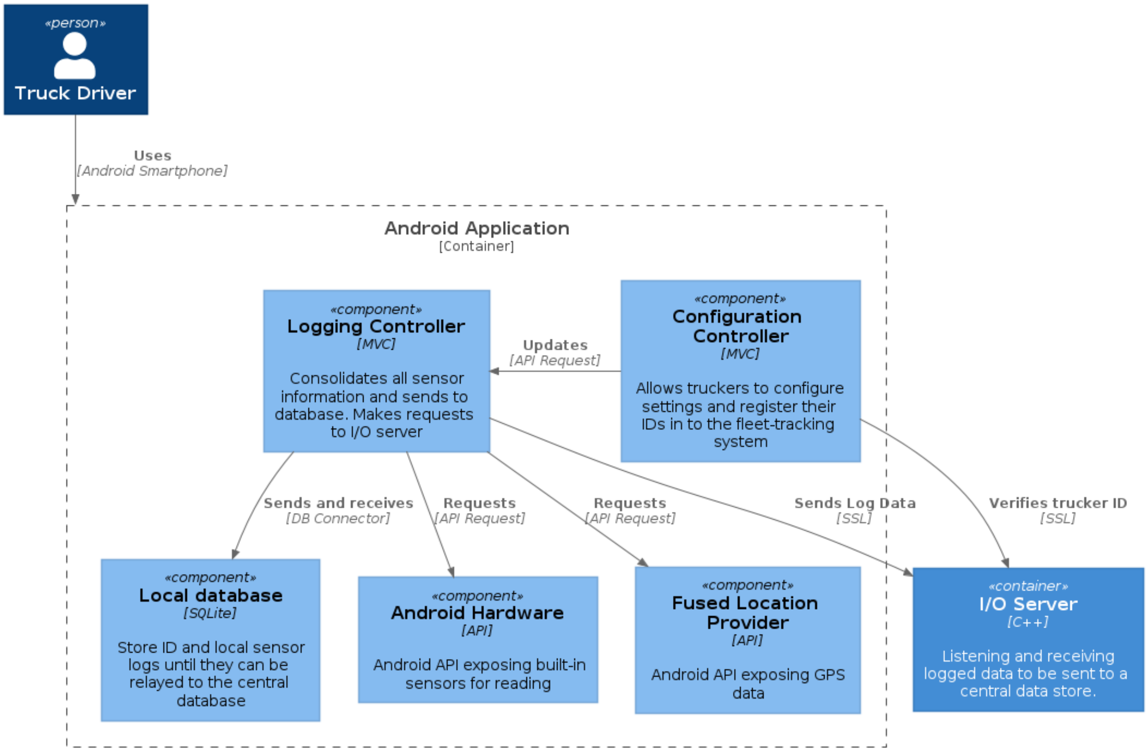

Fig. 8: Component Diagram - Android Application

3.3.2 I/O Server: The typical life cycle view of the I/O server is depicted in figure 9. 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.

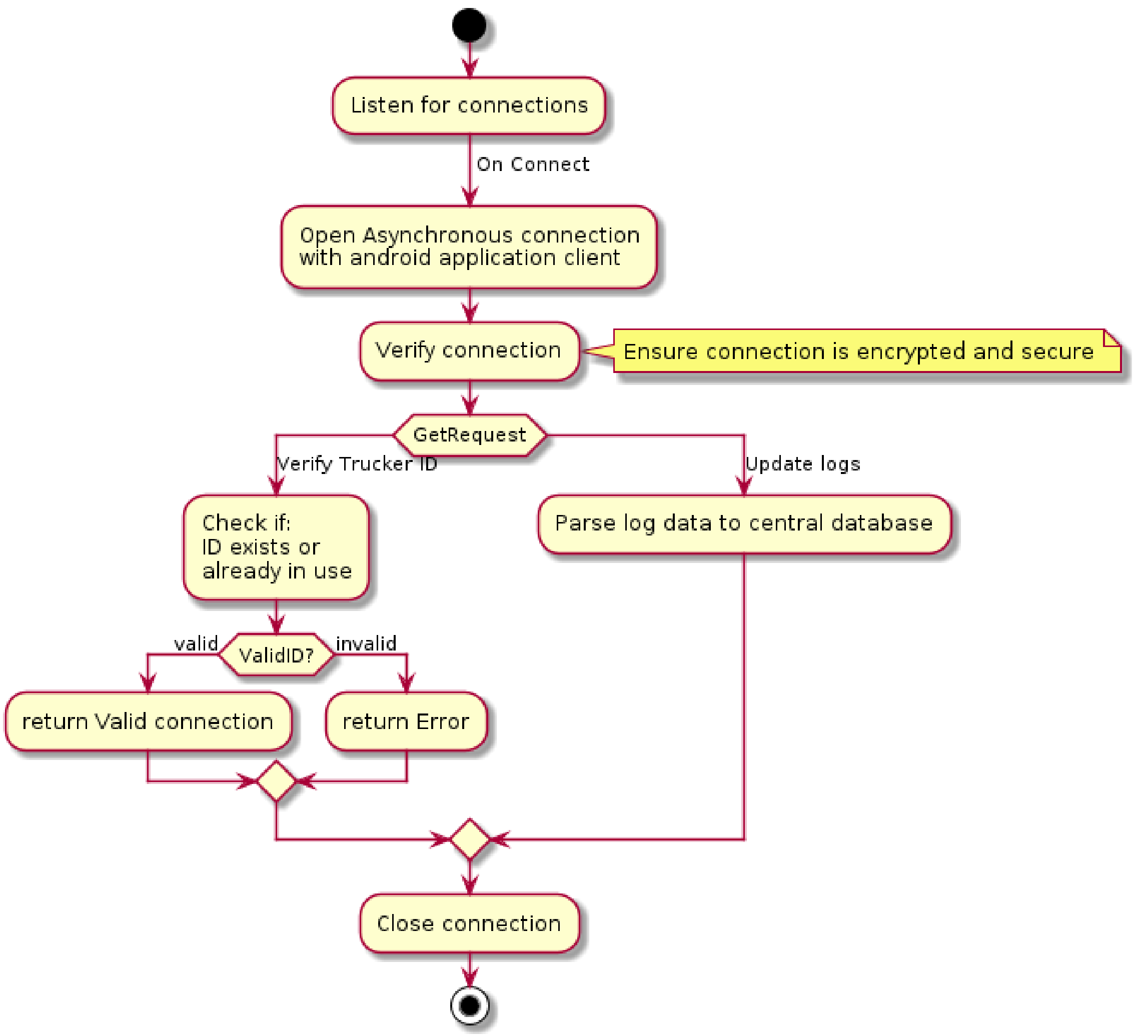

Fig. 9: Life cycle - I/O Server

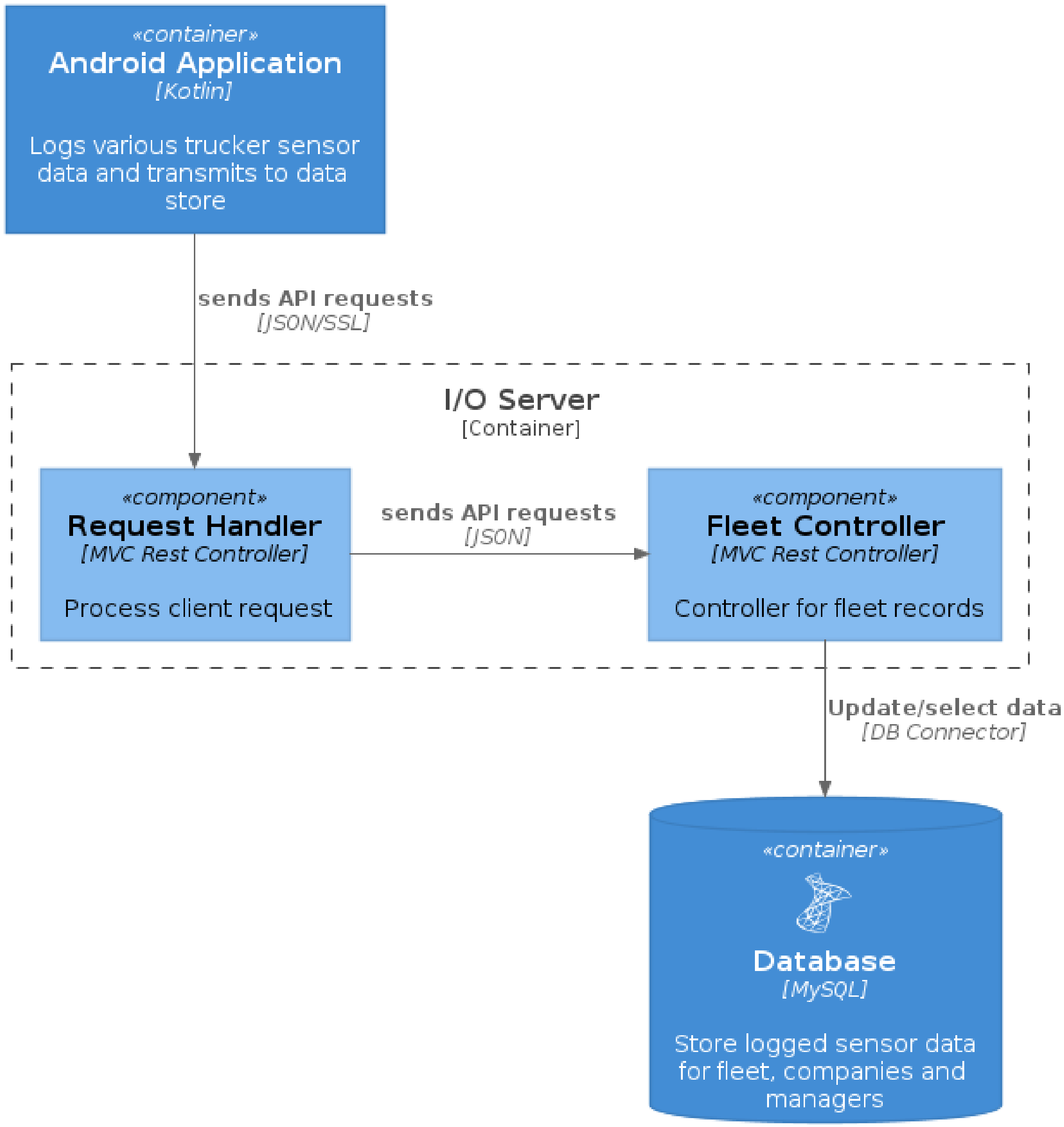

Fig. 10: Component Diagram - I/O Server

Figure 10 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 11. Relational modelling allows for logical structuring and integrity of the data.

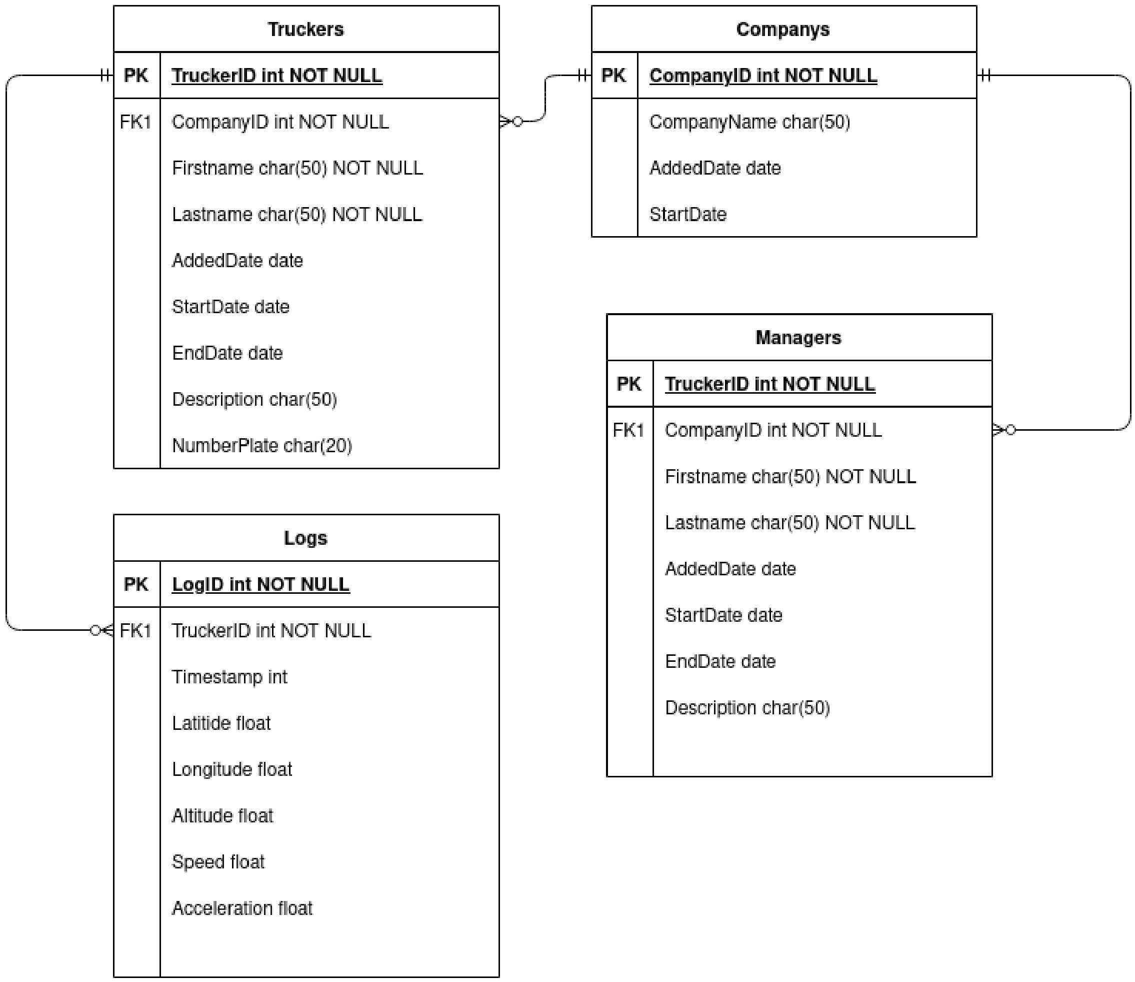

Fig. 11: Entity Relationship Diagram

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 12 depicts the architectural arrangement of the web application.

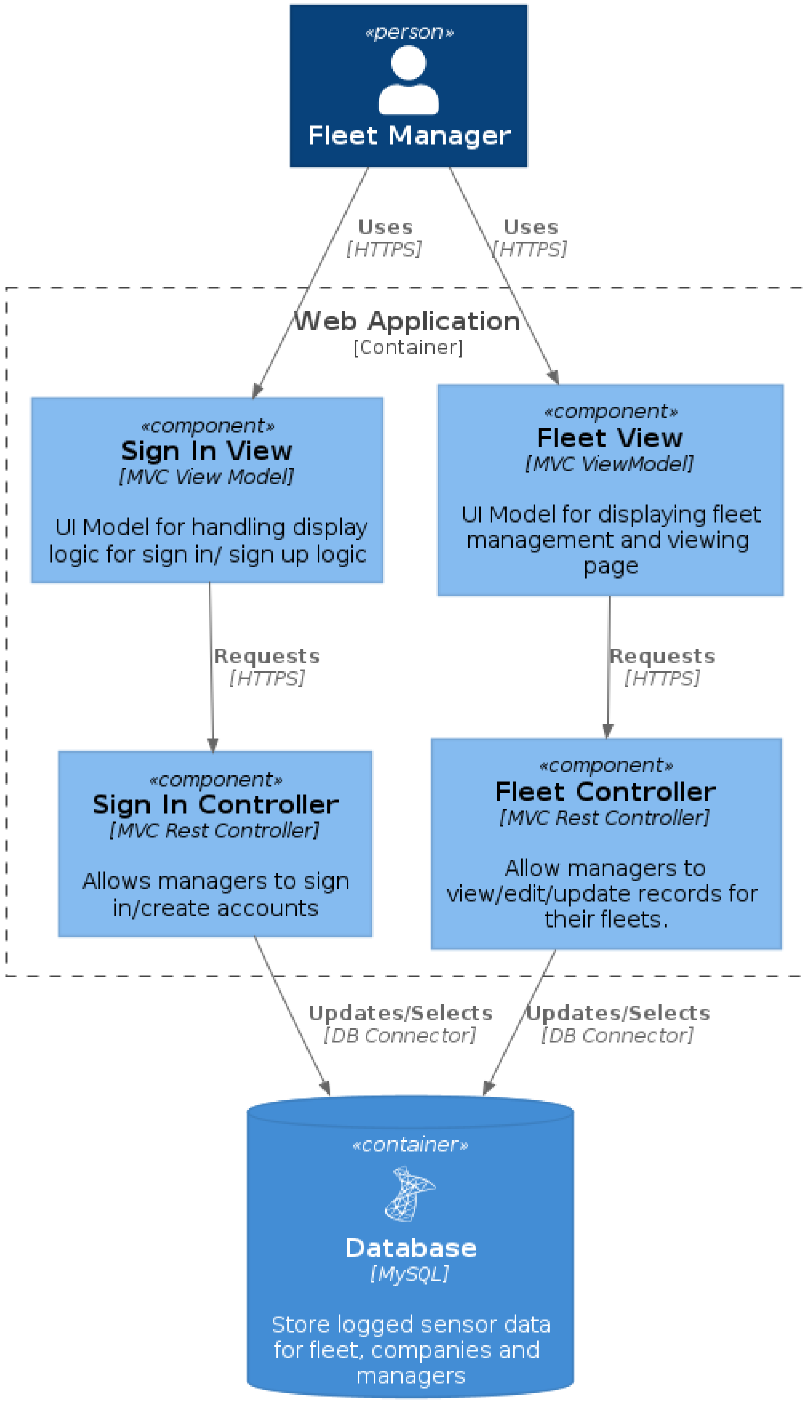

Fig. 12: Component Diagram - Web Application

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] K. Gamage, "Separation of concerns for web engineering projects."

[10] E. Chebanyuk and K. Markov, "An approach to class diagrams verification according to solid design principles," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2016, pp. 435–441.

[11] Z. A. Kocsis and J. Swan, "Dependency injection for programming by optimization," *arXiv preprint arXiv:1707.04016*, 2017.

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

[13] 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.

[14] 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.

[15] 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.

[16] "Guide to app architecture." [Online]. Available: https://developer.android.com/jetpack/guide

[17] "Dependency injection with Hilt." [Online]. Available: https://developer.android.com/training/dependency-injection/hilt-android

[18] "Foreground services." [Online]. Available: https://developer.android.com/guide/components/foreground-services

[19] "Save data in a local database using Room." [Online]. Available: https://developer.android.com/training/data-storage/room

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

[21] W. Anggoro and J. Torjo, *Boost. Asio C++ network programming*. Packt Publishing Ltd, 2015.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] K. Hickman and T. Elgamal, “The ssl protocol,” 1995.

[29] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol–http/1.1,” 1999.

[30] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of json and xml data interchange formats: a case study.” *Caine*, vol. 9, pp. 157–162, 2009.

[31] 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.

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