WebGIS Integrated Information Management System for Effective Administrative Functions

Poi, N.,[1] Jana, S. K.,[1*] Nerit, L.,[2] Sekac, T.[1] and Pai, A.[1]
[1]School of Surveying and Land Studies, The PNG University of Technology, Lae, Morobe, Papua New Guinea
E-mail: nebarepoi22@gmail.com, sujoy2007@gmail.com,* tingneyucsekac@gmail.com, andrew.pai@pnguot.ac.pg
[2]School of Mathematics and Computer Science, The PNG University of Technology, Lae, Morobe, Papua New Guinea, E-mail: lenz.nerit@pnguot.ac.pg
**Corresponding Author*

DOI: https://doi.org/10.52939/ijg.v21i6.4241

Abstract
WebGIS integrated information management system provides a powerful platform for effective spatial data management, decision making, greater control of operational requirements, and service delivery functions. This research is dedicated to developing web-based Real-time Integrated Information Management System (RIIMS) incorporating various elements of GIS and information communication technology (ICT) software applications through WebGIS services over the Web-Technologies. A locational based integrated information system is developed with defined set of interface implementation specifications that implemented with diverse WebGIS technologies. RIIMS application is developed and integrated with related location-based web service applications to generate up to date and reliable information. The platform enables administrators and decision makers to visualize real-time geospatial data patterns, streamline real time public service delivery functions, monitor infrastructure developments and enable effective resources management functions. It is a web-based integrated software application with user authentication and functionalities that connected with computer SQL database system. The functionality of the system provides easy access to geospatial information and seamlessly deals with locational information stored in a database, execute simple and complex geo-spatial analysis over the internet and return messages that contain geospatial information to user community.

Keywords: ArcGIS, APIs, Integration, MVC, RIIMS, WebGIS Services

1. Introduction

Spatially connected web services through application-to-application interface are self-contained, self-describing, modular applications that can be published located and invoked across the Web and perform functions that can be anything from simple requests to complicated functionality processes. The advanced development in internet related technologies extend Location-based services in a new horizon as boldly pointed out by scholars [1][2][3][4][5][6][7][8][9][10][11] and [12], that locational based web services provide easy access to geospatial information and seamlessly deals with geospatial information stored in a database, execute simple and complex geospatial analysis over the internet and return messages that contain geospatial information to user community. Web-based real time locational data accessibility and availability of adequate calculating power remain a long-term bottleneck of geospatial sciences. The researchers, [6][13][14][15][16] and [17] have labeled data accessibility and data reusability as the most important forces that drive current geospatial information research and technological revolution in recent years. One possible direction for large data processing is through web-based data access [18] and server-based geospatial data processing [19] and [20]. The Web based geographically connected computer data display geographically referenced information attached to a location, such as latitude and longitude, or street location including features like road, rivers, building, boundaries and so forth. All governments, both at the national and local levels, hold considerable quantities of geospatial information and location data, for example database of schools and school performance, flood risk data and so forth. However, this information is often not current, shared and or of sufficient quality for effective decision-making.

This research is focusing on developing extendable web-based Real Time Integrated Information Management System (RIIMS) that support multiple Application Programming Interface (APIs) integration which will provide a bonding pathway connecting geospatial system and information communication (ICT) system application through web services. Furthermore, the technologies of Web Service, which is supported by the protocols of Extensible markup language (XML), Geography Markup Language (GML) or Scalable Vector Graphic (SVG) are best suited options to link in the RIIMS software system development for data sharing and exchange among different users. The processing services in and out GIS environment and geospatial data from different formats, platforms and types can be easily accessed under the protocols of XML, GML and SVG, [21]. That makes web-based locational information system more robust in terms of its communication with a variety of users.

The Real Time locational Information (RTLI) enables the system functionalities through simple JavaScript APIs and HTML technology which makes the system go live online disseminating real time up to date and reliable information for user community. The advance in Web GIS programming is obviously involves creating, extending, utilizing, Web GIS or web mapping solutions to solve specific problems, build complete applications, or consume or produce data and geospatial processing services [22] and [23]. Obviously, Web GIS programming nowadays has become a commonly mandatory skill set in many organizations with the expansion of the Internet and availability of Web GIS Options. The RIIMs Application is therefore integrated with various ICT APIs and GIS APIs which provides access to geospatial information and seamlessly deals with geospatial information stored in a database, execute simple and complex geospatial analysis over the internet and return messages that contain locational information to user community. The objectives of a Web GIS integrated information system are to provide broader access to real-time data, reduce system costs, simplify operations for users, and optimize efficiency by leveraging the internet and spatial data. It also aims to facilitate better cross-platform capability and collaboration while making spatial information accessible to a wider audience, including non-technical users.

2. Software Used and Methodology

The method in which this research is carried out is basically by designing and developing a software application utilizing the 3-tier; client and server based architecture (Figure 1).

In the three-tier architecture, it utilized visual studio version 2022 from Microsoft.NET. 2022. The Visual Studio 2022 is used in this research as it comes with a built-in model, view and Control (MVC) project template to make it well-suited and easy to manageable in the integrated development**.** MVC is a software design pattern that separates an application's data (model), user interface (view), and control logic (controller) into distinct modules. This separation provides many benefits, including improved modularity, maintainability, and reusability of code.

In Web geospatial system breaks down to client-side and server-side operations. A client is obviously a web browser. The server side comprises of a Web Server, WebGIS Software and Geospatial database. In this research project, we considered MVC as an appropriate design pattern for developing integrated information management systems due to its ability to effectively separate concerns and enhance the overall structure and maintainability though web based systems. This research Application is developed with Cloud-based Web Application with Microsoft SQL Server 2016 as back-end database system for recording data. The server side logic is developed using an open-source framework called ASP.Net Core from Microsoft. The front-end is developed using Razor pages, JavaScript, jQuery, HTML and CSS. Hence, this research intends to develop compatible Computer software System to integrate with Web-GIS services and real time tracking and communication system through web technologies in a single integrated software development to launch over the Internet. The APIs of GIS software applications are integrated with other communication software APIs to make the system more robust in terms of usage and management, especially disseminating reliable and Real time information and processing of location data in an amalgamated system.

2.1 Integration of Different Technologies in RIIMS

The design of RIIMS was based on the context of the research scale. The idea was to have an integrated system to manage data from a selected government organization tagged with geospatial information management into a single integrated system. The creation of the RIIMS system utilized a combination of technologies, as illustrated in table 1. RIIMS utilizes standard tools like ASP.NET Core, ArcGIS API, and SignalR to enable the customized integration of these technologies to create a scalable, real-time system tailored for administrative functions. The design of the software application customized to the contextual requirements of the organization under study.

Table 1: Design technologies used in RIIMSs

Development Technologies	Version
Programming Language	C# version 10.0
Development Framework	ASP.NET Core 6
Integrated Development Environment (IDE)	Visual Studio 2022
Target Platform	Windows/Linux
Hosting Web Service	IIS 10.0
MailKit (Auto-Email Notification Service)	2.0.3
SignalR (InApp message service)	6.0.0.0
TwilioClient (SMS)	5.33.1
Database Management System	SQL Server 2014
Bootstrap	4
JQuery	Version 3.5.1
Syncfusion (Document Editor)	18.4.0.32

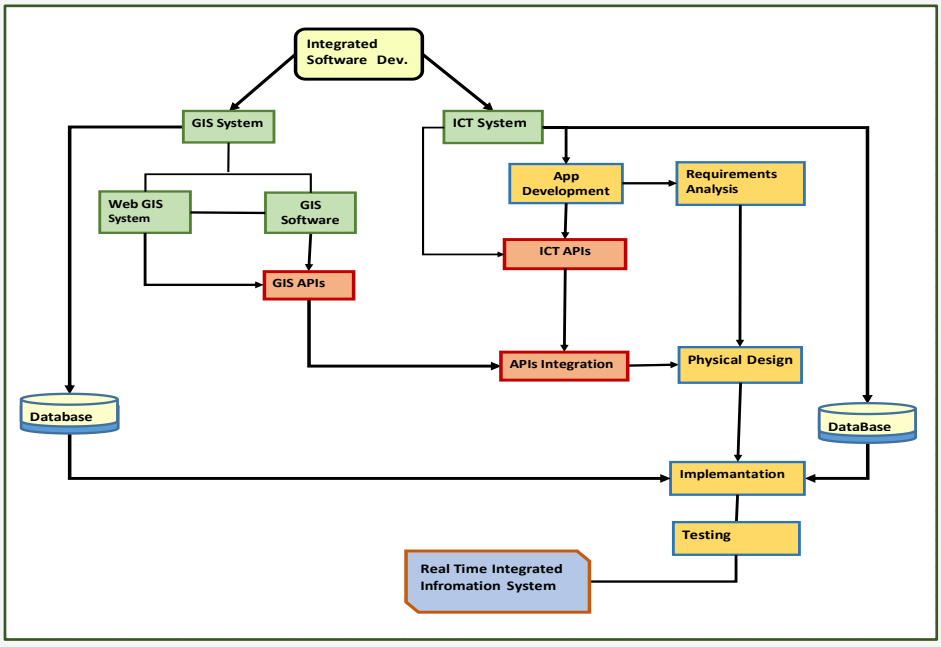


Figure 1: Methodology design layout

In the research, several workflows were identified. In each workflow, there are officers/actors involved who are given certain roles. The roles are basically the job titles for the respective officers. Furthermore, the identified officers with certain roles have access to certain set of information or resources in the system as authenticated. To meet the contextual requirements of the proposed system, we chose ASP.NET Core as our software development stack. It is an open-source web development framework from Microsoft that is used for building web applications, APIs and micro services. It is a cross-platform framework that can be used on Windows, Linux and macOS. ASP.NET Core is a versatile and powerful framework that provides developers with the tools they need to build high-quality web applications and APIs. The initial development of the system was based on ASP.NET Core 3.1 and later we upgraded the framework to ASP.NET Core 6. The later version comes with several newer features and improvements over the previous versions.

The database integration in this system is primarily emphasized on the ability to access any source of data within or outside of this information in almost any application. Data in this application are created in SQL Server in relational database formats. This database is organized into rows and columns, which collectively form a table with attribute information. That Data is typically structured across multiple tables, which are joined together via a primary key recorded with a unique ID called the key. The system is now able to store, update and process unique information needs in various formats via Internet Application Services (IAS) between user communities.

The components of information in the integrated database system that contain locational data will relate to coordinate system. Hence geospatial information and related location-based services will provide the digital connection between a place, its people, and their activities, and will be used to illustrate what is happening, where, how and why it is happening.

3. Geospatial Integration Programming

A WebGIS program is developed with multiple Application Programming Interface (APIs) integration which obviously provides a bonding pathway connecting geospatial system and RIIMS system application through web services. Furthermore, the technologies of Web Service, which is supported by the protocols of Extensible markup language (XML), Geography Markup Language (GML) or Scalable Vector Graphic (SVG) are best suited options to link in the RIIMS software system development for data sharing and exchange among different users.

Furthermore, number of WebGIS software choices offer application programming interfaces (APIs) that bring a means by which developers can influence the published data and processing services of other Web Application services to build and customize applications through standardized interfaces with external web GIS software, data, and services. A list of commonly used open source and proprietary software APIs used in web GIS programming are provided by Sack (2017) on projects focused on most popular among others are Google Maps API, Open Layers, Mapbox GL JS, ArcGIS REST API, Esri ArcGIS API for JavaScript, ArcGIS Python API, Leaflet and ArcGIS Runtime software development kits (SDKs). For instance, the Esri Geo Services REST specification makes available a standard way for clients to connect with servers through the REST architecture engaging Restful web services and URLs [24]. In web GIS programming, a hosted of Web GIS software libraries and other scripting language libraries such as JQuery has provided globally available and reliable collections of modular code that can be used by developers to simplify writing programs. Open Layers and Leaflet are examples of open-source libraries used to create web map mas-hups [25]. Web GIS programming in this project is a type of software development that provides a means of handling internet, browser-based software application development tasks which require unique solutions to web GIS processing options and web mapping problems.

Web-GIS integration breaks down to client-side and server-side operations (Figure 2). Web GIS programming is applicable to both mobile as well as desktop software application development.

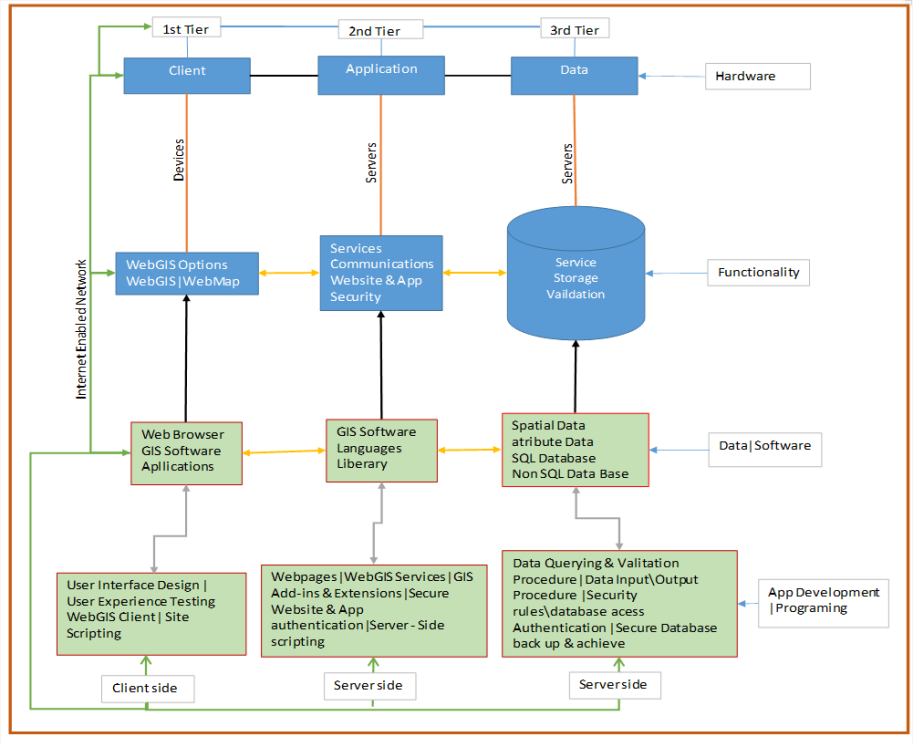


Figure 2: WebGIS interface programming design

A client is obviously a web browser. The server side comprises of a Web Server, WebGIS Software and Geospatial database. In the programming process a browser is usually running software applications by submitting HTTP and /or HTTPS requests to a server hosting WebGIS resources through a Uniform Resource Locator (URL). The server in return replies by providing resources and or performing functions that are requested by the user through the programming rules. This process reviews the fundamentals basics of web GIS programming, accompanying the Web Mapping and other entries in the Programming and Development section, the Web GIS entry in the Computing Platforms, and the User Interface and User Experience (UI/UX) Design entry in the Cartography and Visualization section [22] and [23].

3.1 Web-Based Locational Database Design

This research employs Microsoft SQL Server 2014 in relational database formats which provide an application programmer's interface (API) applicable for very large databases. It is one of the most popular and compatible SQL database normally used for web application development programming. This makes it easy to set in SQL into programming applications and also to set in custom cored into relational databases system. This flexibility is often bring into play by creating custom types of data base and custom procedures that represent new sorts of relationships between the databases in various sources. The components of integrated database system that contain locational data will relate to coordinate system. In the SQL database integration, it is potentially fitting to create a two-dimensional data type that represents a point of a geographic location with the capability to have a new logical operator describing the Distance. The Open Geospatial Consortium (2003) has defined a set of spatial extensions to the SQL standard. The SQL extension for simple Geographical features like Points, Lines and Polygons with number of new spatial operators that deal basically with any sort of two-dimensional relationship in the database system.

Hence in RIIMS geospatial system and related location-based services is obviously providing the digital connection between a place, its people, and their activities and it is used to illustrate what is happening, where, how and why it is happening at the particular location of interest. The relational database in SQL is the simple relational model but yet powerful in handling large data in any forms and sizes for a broad variety of information needs. It uses a structured database that simplifies user requirements by identifying and accessing data in relation to another piece of data in the database that are essentially interchangeable. Often, database is organized into tables with data points relate to each other and manage in a secure, rules-based, standard operation that maintaining data consistency across the applications. In the RIIMS integrated database system, SQL server type data and C# Data type in the Entity Framework (EF) core application perhaps be a linkable structured data which is compatible in the system. Hence, the mapping between C# data types and database data types may vary depending on the database provider one using with EF Core, as different database systems have their own data type systems. Table 2 is a summary of a general mapping table between some common C# data types and their corresponding SQL Server data types compatible and used in the RIIMS development.

Table 2: Datatype mapping from entity framework core to SQL server

C# Data Type	SQL Server Data Type
int	int
string	nvarchar(max)
decimal	decimal
DateTime	datetime2
bool	bit
byte	tinyint
byte[]	varbinary(max)
Guid	uniqueidentifier
double	float
float	real
short	Smallint
long	bigint
char	nchar
enum	int (as integer value)

The expansions of information technologies have gained importance in the worldwide organizations due to their efficiency, reliability with low costs and fast access to information. However, advance in information management systems, especially data management, directly affects GIS system. GIS is obviously a database system but encountered serious problems at the beginning phase of it through its structures and exchange of different data types. Even though organizations and individuals involved with GIS have developed many different solutions, [26] boldly pointed out that data exchange still remains as one of the main problems. However, advanced development in internet related technologies extend GIS in a new horizon. Therefore, it is an integrated information system that applies internet technologies to boost up GIS system in data collection, data dissemination, manipulation, retrieval and updates become easier.

The recent web technologies provide lots of tools in GIS functions to be analyzed and consequently related spatial data can be used to solve more complex business and organizational problems. It supports its real-time maintenance and distributions of up to date and reliable data in a timely manner and improves the overall efficiency of information users.

4. System Architectural Design Pattern

The Software design patterns are essential templates that offer proven solutions to recurring design problems in software development. There are different design patterns that are used in software design and development. Below are few of the commonly used design patterns in software engineering that are applicable to integrated information system development.

- The Singleton pattern ensures a class has only one instance and provides a global point of access.
- The Factory pattern abstracts object creation, allowing subclasses to decide which class to instantiate.
- In the Observer pattern, an object maintains a list of dependents (observers) that are notified of any state changes.
- The Strategy pattern defines a family of algorithms and makes them interchangeable without altering client code.
- The MVC (Model-View-Controller) pattern separates an application into three interconnected components for managing data, user interfaces, and control logic.
- Dependency Injection is a technique that involves providing the dependencies required by a class rather than allowing the class to create those dependencies itself.
- The Decorator pattern adds behavior to objects dynamically, avoiding the need for excessive sub classing.
- The Adapter pattern lets incompatible interfaces work together by providing a wrapper with a common interface.

These design patterns, among many others, contribute to more modular, maintainable, and flexible software architectures.

However, in this research project, we considered MVC as an appropriate design pattern for developing integrated information management systems due to its ability to effectively separate concerns and enhance the overall structure and maintainability of such systems. We also utilized the ASP.NET Core's built-in Dependency Injection Containers (IoC) to manage and resolve dependencies automatically.

4.1 Model View Controller

In integrated information management systems, there are multiple layers of data processing, manipulation, and presentation. For that reason, MVC is a software design pattern that separates an application's data (model), user interface (view), and control logic (controller) into distinct modules. This separation provides many benefits, including improved modularity, maintainability, and reusability of code. The choices of ASP.NET MVC framework also provides options to the ASP.NET Web Forms pattern which helps to create MVC -based Web Applications. The design structure is shown in Figure 3 and expanded functionalities architectural competent in subsequent sections.

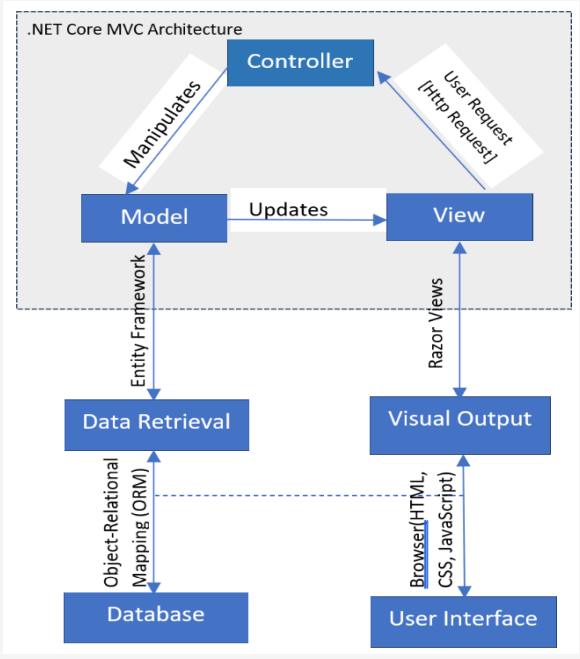


Figure 3: .NET core MVC architecture

The use of this pattern gives us the flexibility to maintenance and upgrading of system application a lot in integrated software design. It plays critical roles in the efficient design and management of software application. In the sub-section bellow will provide it roles in the development, it impacts and efficiency of its functionalities.

4.2 MVC Architecture Components

The MVC Architecture is a three components software application pattern. The interaction of it components and the flow of its communication assembly is described below in Figure 4 with description step by step in sequential interaction in the system structure from the client side to sever side and return massage transmitted back to client side.

- **Model**: This is responsible for managing the data and business logic of the application. It represents the structure and behavior of the data. In a .NET Core application, the model can consist of classes, structures, enums, and other data-related components.
- **View**: The view is responsible for displaying the data to the user and capturing user input. It represents the user interface elements. In a .NET Core application, the view is often implemented using Razor views, which blend HTML markup with C# code.
- **Controller**: The controller handles user requests and acts as an intermediary between the view and the model. It processes user input, interacts with the model to retrieve or update data, and determines which view should be displayed to the user.

In .NET Core, controllers are typically implemented as classes that respond to HTTP requests. In a .NET Core MVC application, the communication flow typically goes as explained in Figure 5 from Client side to middleware to server side and return messages to client as platforms specified bellow in an internet enable network environment.

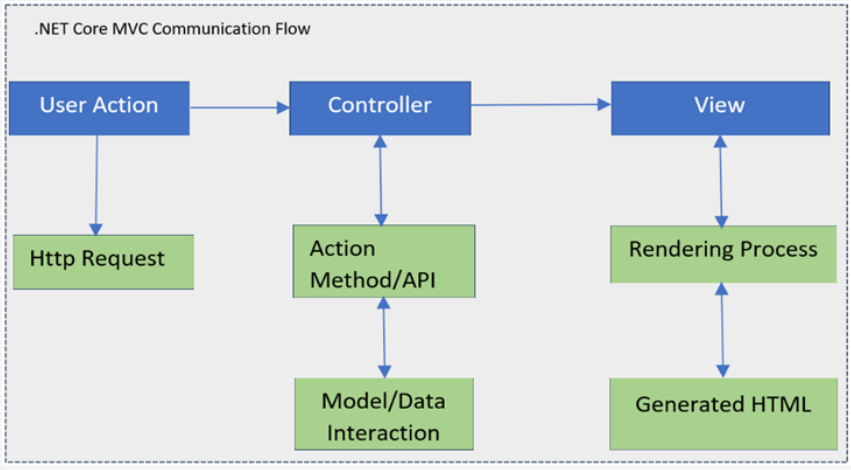


Figure 4: The steps describe the communication flow of the MVC design pattern

Step 1: User Action: The process starts with a user action, such as clicking a link or submitting a form in the user interface.
Step 2: **HTTP Request**: The user action triggers an HTTP request, which is sent to the appropriate controller based on the URL and route configuration.
Step 3: **Controller**: The controller receives the HTTP request and identifies the appropriate action method to handle it.
Step 4: **Action Method**: The action method in the controller processes the user's request. It can interact with the model to retrieve or update data from the database.
Step 5: **Model/Data Interaction**: The action method interacts with the model to retrieve data from the database or perform other business logic operations.
Step 6: **View**: Once the necessary data is retrieved, the controller selects the appropriate view to render. View uses the data provided by the controller to generate HTML content.
Step 7: **Rendering Process**: The view template is processed, and any necessary data is inserted into the template to generate the final HTML output.
Step 8: **Generated HTML**: The generated HTML output, which includes the user interface elements and data, is sent back to the user's browser as an HTTP response.
Step 9: **User Interface**: The user's browser receives the HTML response and renders the user interface, displaying the requested content to the user.

Figure 5: Steps describing the communication flow of MVC design pattern

5. Geolocation Tagging in RIIMS

Geolocation tagging, or the process of associating geographical coordinates of location with digital content such as photos, videos, or social media posts, offers several benefits. One of the benefits is that it forms the basis for numerous location-based services such as navigation, mapping, weather updates, tracking location, local business recommendations and so forth. These services rely on accurate geolocation information to deliver personalized and relevant content up to date and reliable information to users.

5.1 Integration of Third-Party Locational API in RIIMS

Location API services, like those provided by ArcGIS and Google Maps, are pivotal for applications requiring location-based functionalities. Both ArcGIS and Google Maps offer APIs that enable developers to access and incorporate location data seamlessly. This research focused on integrating ArcGIS APIs to explore locational data and services. The ArcGIS platform is commendably working well with a variety of APIs, tools and standards to create a robust Geospatial solution. ArcGIS perhaps provides a far-reaching advanced system for developing mobile, desktop and Web applications as well as interfacing with Web-GIS services. ArcGIS also has met the specific requirements of this project in terms of data analysis, custom application development and other functionalities. Likewise, ArcGIS Web API provides a comprehensive set of tools, libraries, and services provided by Esri, a renowned provider of GIS technology. This powerful API enables RIIMS development to create captivating and dynamic web maps and services by seamlessly integrating geospatial data and advanced functionalities into the applications.

Additionally, this research incorporated Leaflet, a prominent open-source JavaScript library known for its mobile-friendly interactive maps, further enhancing the capabilities of RIIMS. It is integrated only as prototype and later all other WebGIS APIs including Google maps will be integrated in the system to perform more effective and convenient services with regards to locational services. Perhaps, RIIMs Application is an ascendable system with extendable functionalities which is capable of performing various GIS requests such as proximity Analysis, availability analysis, maps and graphs and charts to be generated within the system to make it more automatons services to system users.

5.2 ArcGIS API Integration

ArcGIS supports various geolocation data types that can be used to represent different types of geographic information such as Point, Line, Polygon, Multipoint, Polyline, Multipolygons and so forth. This research utilized Point for capturing location-specific data to describe such as school data and the Line and Polyline to track the road networks etc. The non-spatial attribute values are fetched directly from the RIIMS database and can be easily used to update locational data in ArcGIS Web APIs. The interchangeable data format enabled users to update locational data in both ways from RIIMS to ArcGIS geospatial database and vice versa. The ArcGIS software is a versatile and comprehensive geospatial platform that empowers users to work with spatial data effectively, analyze geographic patterns, and create informative maps and applications across various industries and sectors.

The Third-party integration (Figure 6) of ArcGIS in RIIMS empowers organizations to enhance their geospatial capabilities by seamlessly collaborating with external tools, data sources, and web-based GIS services. This integration enriches geographic data through data enrichment, enables the creation of custom applications that combine ArcGIS mapping with other functionalities, ensures interoperability with various data formats and protocols, leverages cloud services for scalability, facilitates internet -of -thing (IoT) integration for real-time sensor data analysis, supports the development of location-aware mobile apps, integrates with analytics and business intelligence tools for comprehensive data analysis, provides web services for web-based applications, optimizes performance with spatial databases, and enables real-time service by incorporating live data streams.

Such integration has extended the utility of ArcGIS in integrated system across diverse industries and applications, fostering more informed decision-making and innovative geospatial solutions in any organizations. Digitally connected location-based services will provide the connection between the RIIMS database and GIS Database to carry out interconnect-functions with a place, its people, and their activities, and will be used to illustrate what is happening, where, how and why it is happening in GIS queries functions. RIIMS is a scalable system with the capability of expendable software application where the system can be modified at will, add new third-party APIs, add new modules, remove and or change existing functionalities to accommodate additional organizational requirements.

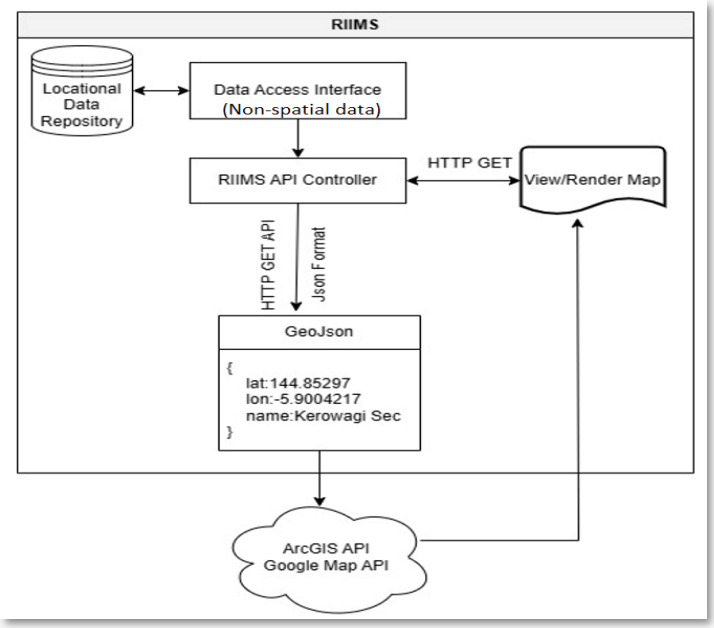


Figure 6: High level illustration of integration of third-party API in RIIMS

6. Results and Discussions

In these discussions are basically looking at three major fundamentals of RIIMS; The system design and MVC application components, the Web-based data modeling and Integration of ArcGIS Services in the RIIMS System application. The designed stages of the RIIMS are primarily involving integration of data management from different data sources. To make the system easier to maintain and function effectively, it has introduced different controller classes in the project for purpose of maintainability and productivity of its operational functions. In controller class has put specific functionality/action methods to capture the server-side logics for processing data and data management tasks.

6.1 Application Design of RIIMS

In the RIIMS development, the MVC software is utilized in the design pattern that separates an application's data, user interface and control logic into distinct modules. This separation provides many benefits, including improved modularity, maintainability, and reusability of code. The MVC models structure in the RIIMS design is illustrated in the Figure 7 with MVC design structure indicated with red arrows. The Model represents the structure and behavior of the data. In a .NET Core application, the model is consisting of classes, structures, enums, and other data-related components. The View in the RIIMS design is basically in command of putting on show the data in the system to user and capturing user's in-put. It is user functional-user interface elements. In a .NET Core application, the view is often implemented using Razor views, which blend HTML markup with C# code. The Controllers in .NET Core, application are typically implemented as classes that respond to HTTP requests. It processes user input, interacts with the model to retrieve and or update data and determines which view should be displayed to the user. It handles user requests and acts as an intermediary between the view and the model. The RIIMS application contains all definitions for entities and database access, so it basically requires adding the packages of MVC in the structure. Visual Studio 2022 is used in this research as it comes with a built-in MVC project template to make it well-suited and easy to manage.

6.2 Data Modeling

RIIMS utilized the object-relational data modeling framework in ASP.NET Core using Entity Framework (EF) core. This enabled RIIMS to define the model classes, configure the database context and use Entity Framework's APIs to interact with the database. In EF core, Object-Relational Mapping (ORM) fits in well with development of integrated system as it is a technique used to bridge the gap between the object-oriented programming world and the relational database world. In the context of ASP.NET Core, Entity Framework is a popular ORM tool that enables developer to work with databases using object-oriented code.

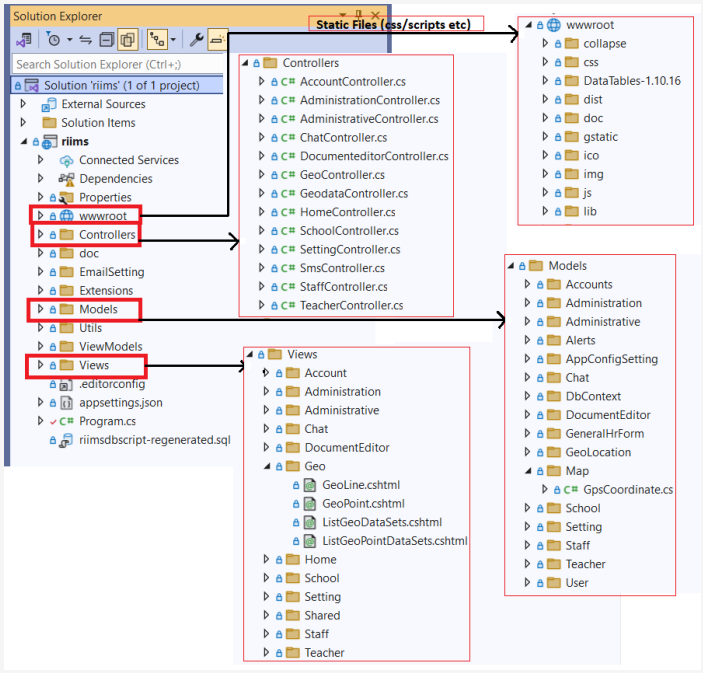


Figure 7: RIIMS project structure in visual studio 2022

```
Package-Manager: Install-Package Microsoft.EntityFrameworkCore
dotnet cli: dotnet add package Microsoft.EntityFrameworkCore
```

Figure 8: EF core used in RIIMS application

```
namespace riims.Models.School
{
    public interface IAspNetSchool
    {
        Task<AspNetSchool> AddAsync(AspNetSchool aspNetSchool);
        Task<AspNetSchool> UpdateAsync(AspNetSchool aspNetSchoolChanges);
        Task<AspNetSchool> DeleteAsync(string SchoolCode);
        Task<IEnumerable<AspNetSchool>> GetSchoolsAsync();
        Task<AspNetSchool> GetSchoolAsync(string? SchoolCode);
    }
}
```

Figure 9: Define service contract interface in RIIMS

In order to use the Entity Framework Core, we installed the package to RIIMS project using the following command (Figure 8) from Command Line Interface (CLI) in *.nuget* gallery. In the EF core comes with built-in conventions for the design of the model class, setting the database context and its interaction with the database through interface methods in RIIMS application.

6.2.1 Data models service contract binding on web services

In developing the RIIMS application, RIIMS established the service contract bindings by implementing interfaces for each data model entity, corresponding to the respective database tables. Leveraging the ASP .NET Core framework's integration with SQL Server, it has successfully constructed a web service that communicates with the RIIMS database, which is hosted on SQL Server. The construction of the web service is done by creating API endpoints in the respective controller classes which allows clients to interact with the SQL Server database through HTTP requests which further exemplified in controller logic section of the subsequent section. The step-by-step approach in service contract binding is further discussed in subsequent pages: Defining Service Contract interface in the coding has shown in Figure 9. It has outlined the contractual obligations of each data model entity, encompassing the methods accessible to clients. To keep things concise, it has also omitted the implementation details for all these interfaces.

6.2.2 Implementation of service contract on web services

The Implementation of Service Contract Interface in coding is shown in Figure 10. In RIIMS Development it has created a class that implements the service contract interface. These classes contain the actual implementation of the service methods in the RIIMS development. Register the Service Implementation in Program.cs Class is shown in Figure 11. In order to use the interface methods implemented with the ASP.Net Core dependency container, the service implemented is registered in program.cs class. The program.cs file in ASP.NET Core of MVC application is an entry point of an application. It holds logic to start the server and attend to the request and also configure the application at the server side. This file mainly serves as the entry point for the application and is responsible for configuring the web host and starting the application.

```
namespace riims.Models.School
{
    public class SQLAspNetSchoolRepository : IAspNetSchool
    {
        private readonly AppDbContext context;
        public SQLAspNetSchoolRepository(AppDbContext context)
        {
            this.context = context;
        }
        public async Task<AspNetSchool> AddAsync(AspNetSchool aspNetSchool)
        {
            await context.AspNetSchool.AddAsync(aspNetSchool);
            await context.SaveChangesAsync();
            return aspNetSchool;
        }

        public async Task<AspNetSchool> DeleteAsync(string SchoolCode)
        {
            AspNetSchool record = await context.AspNetSchool
                                        .FirstOrDefaultAsync(x => x.AspNetSchoolId == SchoolCode);
            if (record != null)
            {
                var r = context.AspNetSchool.Remove(record);
                r.State = EntityState.Deleted;
                await context.SaveChangesAsync();
            }
            return record;
        }
        public async Task<AspNetSchool> GetSchoolAsync(string SchoolCode)
        {
            return await context.AspNetSchool.FirstOrDefaultAsync(x => x.AspNetSchoolId == SchoolCode);
        }
        public async Task<IEnumerable<AspNetSchool>> GetSchoolsAsync()
        {
            return await context.AspNetSchool.ToListAsync();
        }
        public async Task<AspNetSchool> UpdateAsync(AspNetSchool aspNetSchoolChanges)
{
            var r = context.AspNetSchool.Attach(aspNetSchoolChanges);
            r.State = Microsoft.EntityFrameworkCore.EntityState.Modified;
            await context.SaveChangesAsync();
            return aspNetSchoolChanges;
        }
    }
}
```

Figure 10: Implementation of the Service Contract Interface in RIIMS

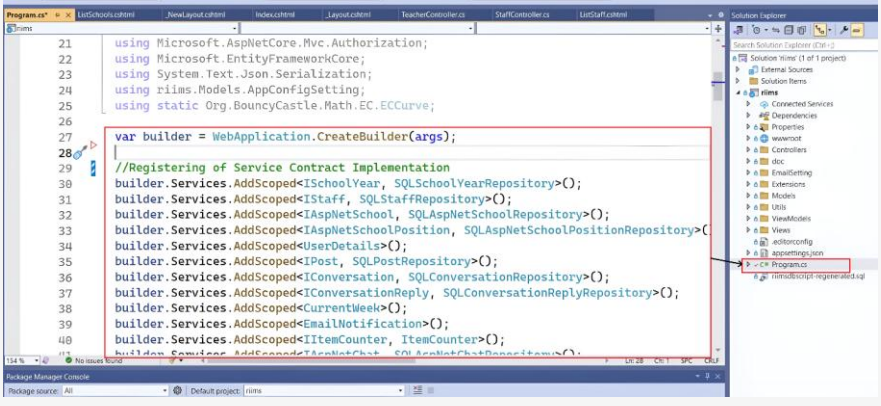


Figure 11: Service Implementation Registered in Program.cs class

6.2.3 Database-connection registration and configuration

Database connection in ASP.NET Core is mainly through connection string to connect to the database which is stored in the appsettings.json. It is the bridge that serves to link ASP.NET Core Applications and the database. The Database connection string configured in *appsettings.js* file is shown Figure 12(a) and registered is program.cs file is shown in Figure 12(b) respectively. To enable interaction with the database through the service implementation, RIIMS established API end-points within the controller classes specified in succeeding Sections. Furthermore, it has employed Entity Framework Core for handling interactions with the SQL Server. Basically, web development, an "endpoint" is a specific URL or URI (Uniform Resource Identifier) that a client sends HTTP requests to in order to interact with a web service or application. These endpoints correspond to specific actions or operations that the service can perform. The URL pattern can be mapped to various request handlers, including action methods. In Figure 13 illustrates an API endpoint for listing school records in RIIMS.

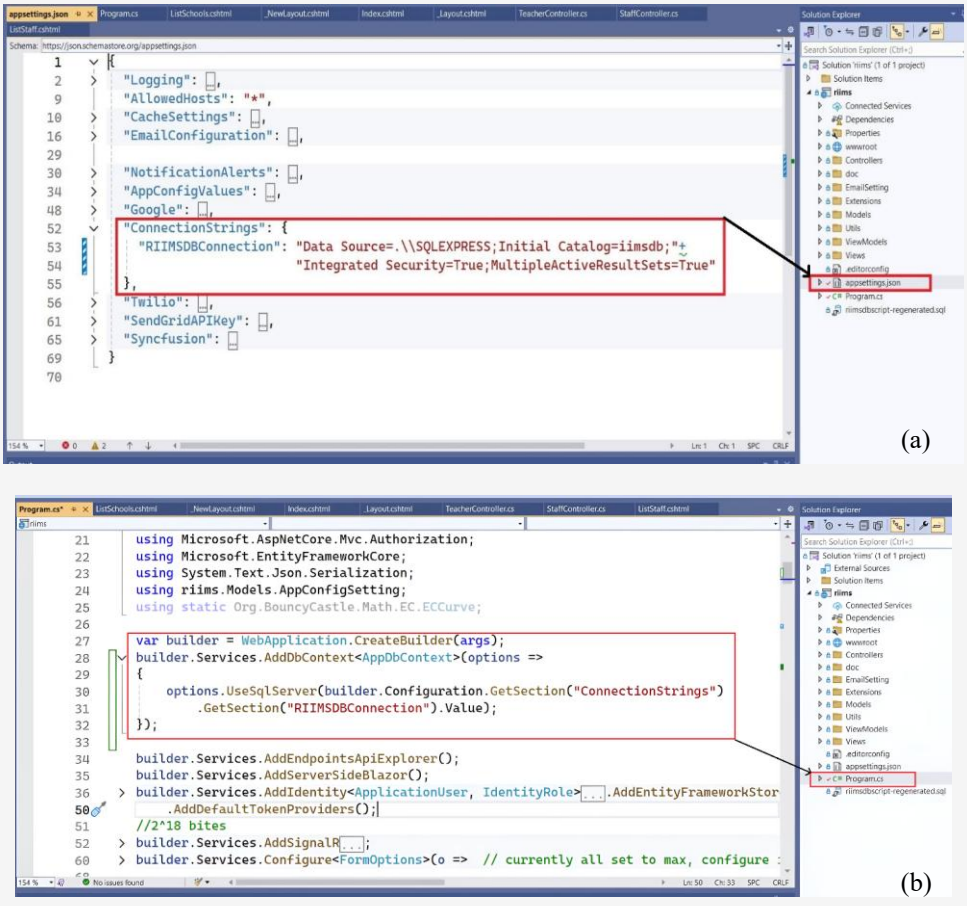


Figure 12: (a) Database connection string configured in appsettings.js file
(b) Database connection strings registered in *program.cs* file

```
[HttpGet]
  [Route("api/[controller]/listschool")]
   public async Task<IActionResult> ListSchool()
   {
          var schools = await schoolRepository.GetSchoolsAsync();
          var model = new CreateSchoolViewModel
          {
               Schools = schools.ToList()
          };
          return PartialView("_ListSchools", model);
   }
```

Figure 13: Create API endpoints in RIMS

Most times, the basic CRUD operations (Create, Read, Update, and Delete) are performed on the entity data models and the changes are applied to the database records through the Entity Framework Core.

6.3 Controller Logics

The controller classes (Figure 14) in the RIIMS project are where the workflows or business processes are executed through action methods. These controllers are tasked with managing incoming HTTP requests and providing appropriate responses to the client side. Multiple controller classes were established within the project, each containing its own set of action methods. To achieve scalability in RIIMS, several controller classes were introduced**.** In each controller action methods were defined to capture the business logics. This made the application more modular so that the requirement changes can be easily captured and integrated in the application. Furthermore, ASP.NET Core MVC controllers explicitly specify their dependencies through constructor injection. It includes native support for Dependency Injection (DI), which enhances the testability and maintainability of applications in the RIIMS.

6.3.1 Action Methods

The action methods are procedures within a controller class (Figure 15) program that handle incoming HTTP requests and return HTTP responses. Action methods are the core building blocks of MVC (Model-View-Controller) and Web API applications. They are responsible for performing specific tasks, processing data, and returning the appropriate HTTP status codes and content to the client. Each action method is typically associated with a specific HTTP verb (GET, POST, PUT, DELETE, etc.). This association is achieved through attributes such as [HttpGet], [HttpPost], [HttpPut], and [HttpDelete] in the application. Furthermore, they are often decorated with route attributes like *[Route]* to define the URL endpoint at which they can be accessed. The route template determines how the action method is mapped to a URL. Action methods can accept parameters, which are populated with values from the HTTP request (query string, route data, request body, etc.). These parameters allow data to be accessed from the client. Moreover, the action methods return an Action Result or a specific derived type like View Result, Json Result, File Result etc. The Action Result represents the result of action and can be used to return different HTTP status codes and content types. The following code retrieves the list of schools through the ListSchools method defined in the school controller class. The method is modified to include business logics of authentication and resource authorization. When, this action method is invoked, the code checks whether the user invoking method is logged in.

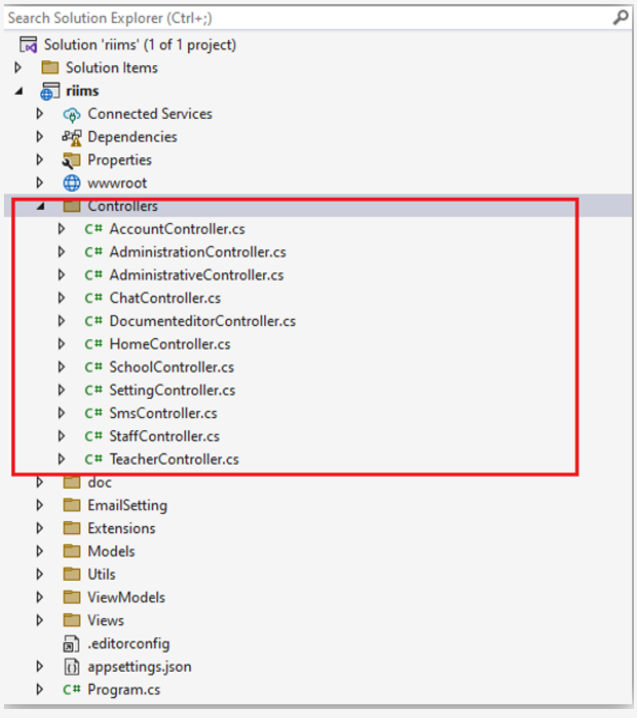


Figure 14: Controller classes in RIIMS

```
  [HttpGet]
   [Route("api/[controller]/listschools")]
   public async Task<IActionResult> ListSchools()
   {
      ApplicationUser loginUser = await
signInManager.UserManager.FindByEmailAsync(User.Identity.Name);
      if (loginUser != null && (await userManager.IsInRoleAsync(loginUser,
UserRoles.SystemAdminRole)
         || await userManager.IsInRoleAsync(loginUser, UserRoles.ARORole)
         || await userManager.IsInRoleAsync(loginUser, UserRoles.PEARole)) &&
signInManager.IsSignedIn(User))
       {
           var schools = await schoolRepository.GetSchoolsAsync();
           var model = new CreateSchoolViewModel
           {
             Schools = schools.ToList()
           };
          return View(model);
       }
       else
       {
         return RedirectToAction("AccessDenied", "Administration");
       }
    `
```

Figure 15: Action method in RIIMS application

If so then the user has to be in one of the roles to retrieve the list of schools through the dependency injection service. The method returns a View as an Action Result. Otherwise, the method uses the RedirectToAction to send a response to the client's browser with a redirect instruction to the AccessDenied method which is defined in the Administration controller class. As a result, the user's browser will make a new request to the URL associated with AccessDenied.

6.3.2 Client views and performance optimization

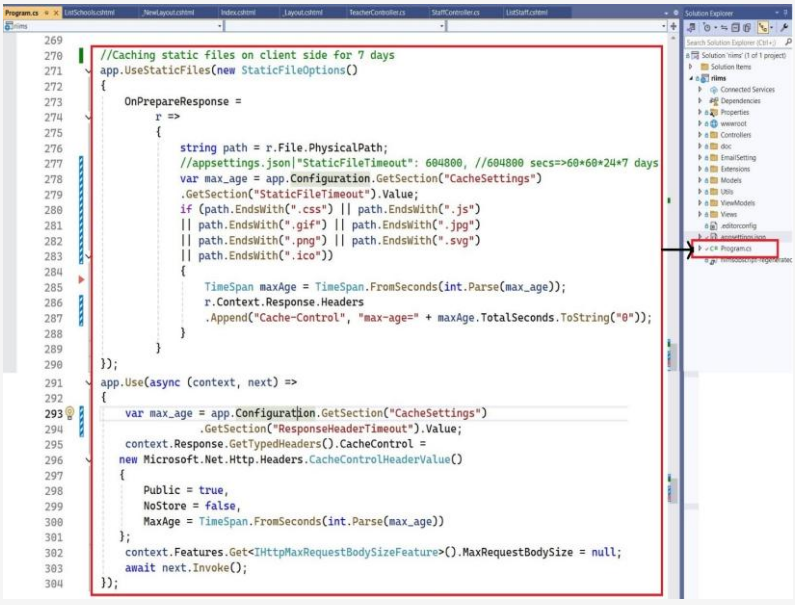
In ASP.NET MVC, views and partial views are typically associated with action methods through a combination of naming conventions and explicit view return statements within the controller action methods. The mapping between action methods and views is crucial for rendering HTML content to the browser when a user accesses a specific URL. Performance optimization is essential for providing a positive user experience, achieving business goals, improving search engine visibility, reducing operational costs, and remaining competitive in the digital landscape. It has a direct impact on user satisfaction, conversion rates, and the overall success of web applications. There are various techniques and strategies involved to improve loading speed, responsiveness, and resource efficiency. In the design of RIIMS application, we employed the following techniques as described in the subsections.

6.3.2.1 Browser coaching

This project implemented caching strategies in RIIMS to store static assets like images, stylesheets and scripts on the user's device. This reduces server requests for subsequent visits and improving load times. Further shown in Figure 16, is the configuration of the caching static the files set in the *Program.cs* file. The timeout values for the cache settings are stored in the *appsettings.json.* The values are read from the file through the Configuration extension manager in the ASP.NET Core. The file will be reloaded after every seven days as set in the configuration code as shown in Figure 16 configuration cache. In the Figure 17(a) Cache configuration timeout values defined in appsettings.json file. Figure 17(b) Header details in CustomScript.js file downloaded on the client side browser.

6.3.2.2 Asynchronous programming

In ASP.NET Core, the *Task<T>* type is used to represent an asynchronous operation that returns a result of type *T*. It plays a fundamental role in asynchronous programming, allowing one to perform non-blocking operations, which is crucial for building responsive and scalable applications. Asynchronous programming (Figure 18) enables system to perform operations that may take some time to complete (e.g., I/O operations, network requests, or database queries) without blocking the calling thread. This is essential for maintaining the responsiveness of the application, especially in scenarios like web services or user interfaces in an integrated information system like RIIMS. It define a method that performs an asynchronous operation and needs to return a result, it can use Task <T> as the return type. In the following code, the async keyword, Task return value, await keyword and ToListAsync method makes the code execute asynchronously.

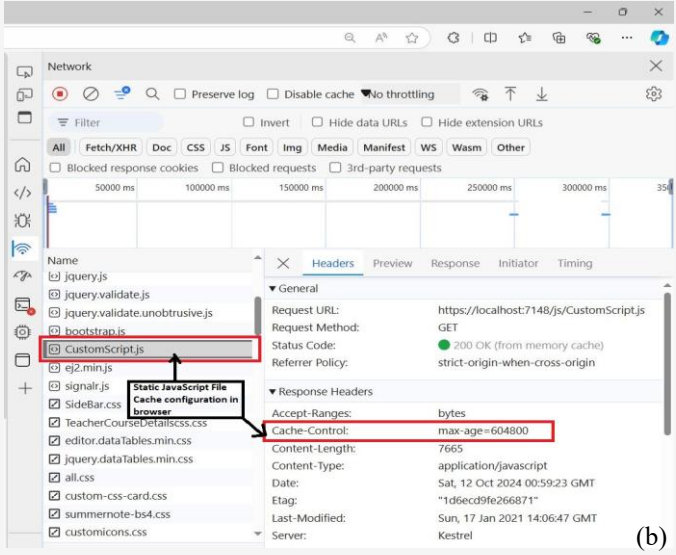


```
//Caching static files on client side for 7 days
app.UseStaticFiles(new StaticFileOptions()
{
    OnPrepareResponse =
        r =>
        {
            string path = r.File.PhysicalPath;
            //appsettings.json|"StaticFileTimeout": 604800, //604800 secs=>60*60*24*7 days
            var max_age = app.Configuration.GetSection("CacheSettings")
            .GetSection("StaticFileTimeout").Value;
            if (path.EndsWith(".css") || path.EndsWith(".js")
            || path.EndsWith(".gif") || path.EndsWith(".jpg")
            || path.EndsWith(".png") || path.EndsWith(".svg")
            || path.EndsWith(".ico"))
            {
                TimeSpan maxAge = TimeSpan.FromSeconds(int.Parse(max_age));
                r.Context.Response.Headers
                .Append("Cache-Control", "max-age=" + maxAge.TotalSeconds.ToString("0"));
            }
        }
});
app.Use(async (context, next) =>
{
    var max_age = app.Configuration.GetSection("CacheSettings")
                .GetSection("ResponseHeaderTimeout").Value;
    context.Response.GetTypedHeaders().CacheControl =
    new Microsoft.Net.Http.Headers.CacheControlHeaderValue()
    {
        Public = true,
        NoStore = false,
        MaxAge = TimeSpan.FromSeconds(int.Parse(max_age))
    };
    context.Features.Get<IHttpMaxRequestBodySizeFeature>().MaxRequestBodySize = null;
    await next.Invoke();
});
```

Figure 16: Cache configuration registered in the Program.cs file

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "CacheSettings": {
    "CacheName": "RIIMS-Cache",
    "EnableLogs": "True",
    "StaticFileTimeout": 604800, //604800
secs=>60*60*24*7 days
    "ResponseHeaderTimeout": 10 // secs
  }
}
```

(a)



(b)

Figure 17: (a) Cache configuration timeout in appsettings.json file
(b)Header details in client side browser

```
namespace riims.Models.School
{
    public class SQLAspNetSchoolRepository : IAspNetSchool
    {
        private readonly AppDbContext context;
        public SQLAspNetSchoolRepository(AppDbContext context)
        {
            this.context = context;
        }
        // Asynchronous database query
        public async Task<IEnumerable<AspNetSchool>>
GetSchoolsAsync()
        {
            return await context.AspNetSchool.ToListAsync();
        }
       //...
    }
```

Figure 18: Coding of asynchronous ASP.NET Csore framework operations in RIIMS

In this code fragment, Task<*IEnumerable<AspNet School*>> indicates that the method will return a task that will eventually yield a collection of school records when the asynchronous operation completes. When working with *Task<T>,* often async and *await* keywords are used. The *async* keyword is used to mark a method as asynchronous, and the await keyword is used to await the completion of a *Task*. This allows the calling thread to continue executing other tasks while waiting for the asynchronous operation to finish. Asynchronous code execution in ASP.NET Core framework provides multitude advantages, including improved application responsiveness during time-consuming tasks, efficient resource utilization by freeing up threads, scalability for handling concurrent requests, faster I/O operations, simplified parallelism without explicit thread management, reduced thread blocking, smoother user experiences, more efficient CPU usage, lower network latency through concurrent requests, robust cancellation and timeout capabilities, and streamlined error handling, making it a fundamental technique for building high-performance, responsive, and resource-efficient applications. This makes the integrated information system through web services more effective and efficient in its functionalities. Furthermore, pushing content/updates to the client side without reloading the entire webpage can help achieve better performance and responsiveness in web applications, particularly when dealing with data changes. It minimizes server load, reduces response times, and enhances the user experience by delivering updates quickly and efficiently.

6.4 Locational API Services in RIIMS

In order to integrate and share geospatial data with third-party APIs, RIIMs development introduced a module to create API using minimal API feature in ASP.NET Core 6 that allows creating lightweight and simplified web APIs with a minimal amount of configuration. It is purposely to allow RIIMS API to be used by third-party. In the RIIMS Application it is integrated with ArcGIS APIs and further explained in sub sequent sections. Other Web GIS APIs is also integrated but, in this study, mainly focuses on the integration of ArcGIS API and processing capabilities in RIIMS.

6.4.1 Locational API design and integration in RIIMS

RIIMS created a *GeodataController* class (Figure 19) and decorated the class with the *[ApiController]* attribute which provides a convenient way to setup the API controller with a range of default behaviors and conventions that are typically useful when building web APIs. Additionally, it has reduced the amount of boilerplate code that needs to be written for common API-related tasks and promotes best practices for building RESTful APIs. The controller class basically exposes the endpoint API so that the service can be accessed without authentication. Hence, RIIMS application further decorated the API methods with the [AllowAnonymous] attribute.

6.4.2 Defining locational API in RIIMS

In the API controller class, we defined the geolocation APIs and further formatted the data in Json format with each entry comprising of latitude, longitude and locational attributes. We created two API endpoints; one for Point geolocation data (Point API) and other for LineString data (LineString API) particularly for road network for this research. The custom implementation of RIIMS geolocation APIs for producing a Point (Figure 20(a)) and LineString geolocation data (Figure 20(b)) are the codes programmed in the system. If there are GeoJSON file records found in RIIMS, then the data is further processed into a JSON format as stated in lines 12 to 26 of the codes highlighted in green.

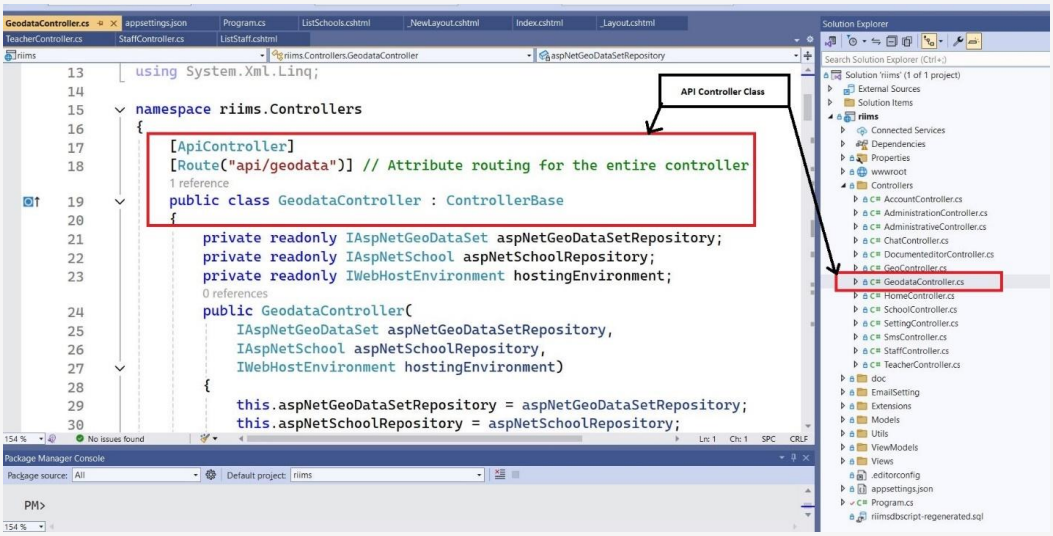


Figure 19: API controller class in RIIMS

(a)

```
[HttpGet]
[Route("getgeolocations")]
[AllowAnonymous]
public async Task<IActionResult> GetGeoLocations()
{
    var geoLocation = await aspNetGeoDataSetRepository.GetAllAsync();
     var schoolRecords = await aspNetSchoolRepository.GetSchoolsAsync();
    var geoPointLocationQuery = geoLocation.Where(x => x.GeometryType == "Point").ToList();
    List<GeoLocation> list = new List<GeoLocation>();
    List<GeoPoint> geoPoints = new List<GeoPoint>();
    //Read the Json files
   if (geoPointLocationQuery.Any())
    {
        foreach (var record in geoPointLocationQuery)
        {
            var filePath = Path.Combine(hostingEnvironment.WebRootPath, record.Url);
            var geoJson = System.IO.File.ReadAllText(filePath);
            var converter = new GeoJsonConverter();
            var featureCollection = JsonConvert.DeserializeObject<GeoJSON.Net.Feature.
                FeatureCollection>(geoJson, converter);
             var jsonObject = JObject.Parse(geoJson);
             // Access the "features" array and convert each object to GeoLocation
             var featuresArray = jsonObject["features"].ToObject<List<GeoLocation>>();
             list.AddRange(featuresArray);
         }
     }

      var data = list.Select(x => new { Name = x.properties.Name,
                                        Latitude =
double.Parse(x.geometry.coordinates[0].ToString()),
                                        Longitude =
double.Parse(x.geometry.coordinates[1].ToString())
                                      }).ToList();
     if (data.Count > 0)
      {
          foreach (var d in data)
          {
              var school = schoolRecords.FirstOrDefault(x=>x.SchoolName==d.Name);
              GeoPoint p = new GeoPoint
              {
                  Latitude = d.Latitude,
                  Longitude = d.Longitude,
                 Name = d.Name,
                 SchoolCode= school==null?"Unavailable":school.AspNetSchoolId,
                 Level = school == null ? "Unavailable" : school.SchoolLevel.ToString(),
                 SchoolType = school == null ? "Unavailable" : school.SchoolType,
                 Address = school == null ? "Unavailable" : school.SchoolAddress,
                 District=  school == null ? "Unavailable" : school.District,
                 LLG=school == null ? "Unavailable" : school.LLG,
                 Agency = school == null ? "Unavailable" : school.Agency,
             Ward = school == null ? "Unavailable" : school.Ward,
             YearEstablished = school == null ? "Unavailable" : school.YearEstablished.ToString(),
         };
```

Figure 20: (a) Point Geolocation API implementation in RIIMS (continue next page)
(b) LineString Geolocation API implementation in RIIMS

(a)

```
                geoPoints.Add(p);
            }
        }
            // Return the list of GeoLocation objects
            var dataset = from a in geoPoints
                          select new
                          {
                              lat = a.Latitude,
                              lon = a.Longitude,
                              name = a.Name,
                              SchoolCode = a.SchoolCode,
                              Level = a.Level,
                              SchoolType = a.SchoolType,
                              Address = a.Address,
                              District = a.District,
                              LLG = a.LLG,
                              Agency = a.Agency,
                              Ward = a.Ward,
                              YearEstablished = a.YearEstablished
                          };
            return Ok(dataset);
        }
```

Figure 20: (a) Point Geolocation API implementation in RIIMS (continue from previous page)
(b) LineString Geolocation API implementation in RIIMS

(b)

```
[HttpGet]
[Route("geolinestringdata")]
[AllowAnonymous]
public async Task<IActionResult> GetGeoLineStringData(string FeatureName)
{
    var geoLocation = await aspNetGeoDataSetRepository.GetAllAsync();
    var geoPointLocationQuery = geoLocation.Where(x => x.GeometryType == "LineString" && x.FeatureName ==
    FeatureName).ToList();
    List<Position> listpositions = new List<Position>();
    string lineStringName = "Unknown";
     //Read the Json files
     if (geoPointLocationQuery.Any())
     {
         foreach (var record in geoPointLocationQuery)
         {
             var filePath = Path.Combine(hostingEnvironment.WebRootPath, record.Url);
             var geoJson = System.IO.File.ReadAllText(filePath);
             var converter = new GeoJsonConverter();
             var featureCollection =
             JsonConvert.DeserializeObject<GeoJSON.Net.Feature.FeatureCollection>(geoJson, converter);
             var jsonObject = JObject.Parse(geoJson);
              //Access the Geo LineString name
              lineStringName = jsonObject.SelectToken("features[0].properties.Name").Value<String>();
             //Extract the Geo LineString coordinates from the JSON object
             var coordinatePoints = jsonObject.SelectTokens("features[0].geometry.coordinates");
             if (coordinatePoints.Any())
             {
                 for(int i=0;i< coordinatePoints.Children().Count(); i++)
                 {
                    var token= jsonObject.SelectTokens("features[0].geometry.coordinates["+i+"]");
                     if (token != null)
                     {
                         var lat = jsonObject.SelectToken("features[0].geometry.coordinates[" + i +
                                   "][0]").Value<String>();
                         var lon = jsonObject.SelectToken("features[0].geometry.coordinates[" + i +
                                   "][1]").Value<String>();

                         var nex = new Position(lat, lon);
                         listpositions.Add(nex);
                     }
                 }
             }
         }
    }
    var geoLineString = new GeoLineString
    {
        Name = lineStringName,
        LineString = new LineString(listpositions)
    };
    return Ok(geoLineString);
}
```

Figure 20: (a) Point Geolocation API implementation in RIIMS
(b) LineString Geolocation API implementation in RIIMS

Each time when a feature is extracted, it is converted to a list of geolocation format in RIIMS as shown in lines 23 and 24 of the codes within. Likewise, for the LineString geolocational data, it defined the method with the route geo-LineString data basically reads some geolocation data from a GeoJSON file stored in RIIMS as stated in lines 6 and 7. The data is then further extracted based property name of the geolocation in the attribute dataset as shown in lines 11 to 40. Furthermore, the extracted coordinate values and the respective point names are modelled into a list of positions as shown in lines 27 to 38 within. Finally, it is then passed the formatted LineString list of data points into the custom data format class (GeoLineString) as shown in line 41 to 45 to generate connected segments of line data of the geolocation

6.4.3 Usage and output of the locational API

The geolocations API can be used (Figure 21) from third-party applications or can also be used within the RIIMS application as illustrated in coding in Figure 21. The JavaScript-jQuery software library has been used to integrate with the ArcGIS mapping API as also shown in the subsections. The key focused in this discussion are two geolocation APIs; the Point Geolocation API output in JSON format Figure 22(a) and Line String Geolocation API output in JSON format Figure 22(b) respectively. The testing of API was done using a third-party tool, Postman (https://www.postman.com/) which is a lightweight API client. Postman is a standalone software testing API (Application Programming Interface) platform to build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses in system applications. In the Figure 22(a) RIIMS Point Geolocation API output in JSON format: (b) RIIMS Line String Geolocation API output in JSON format.

```
Usage:  GET /api/geodata/getgeolocations
            GET /api/geodata/ geolinestringdata?featurename= Gumini%20Road
```

Figure 21: Usage of geolocation APIs in RIIMS

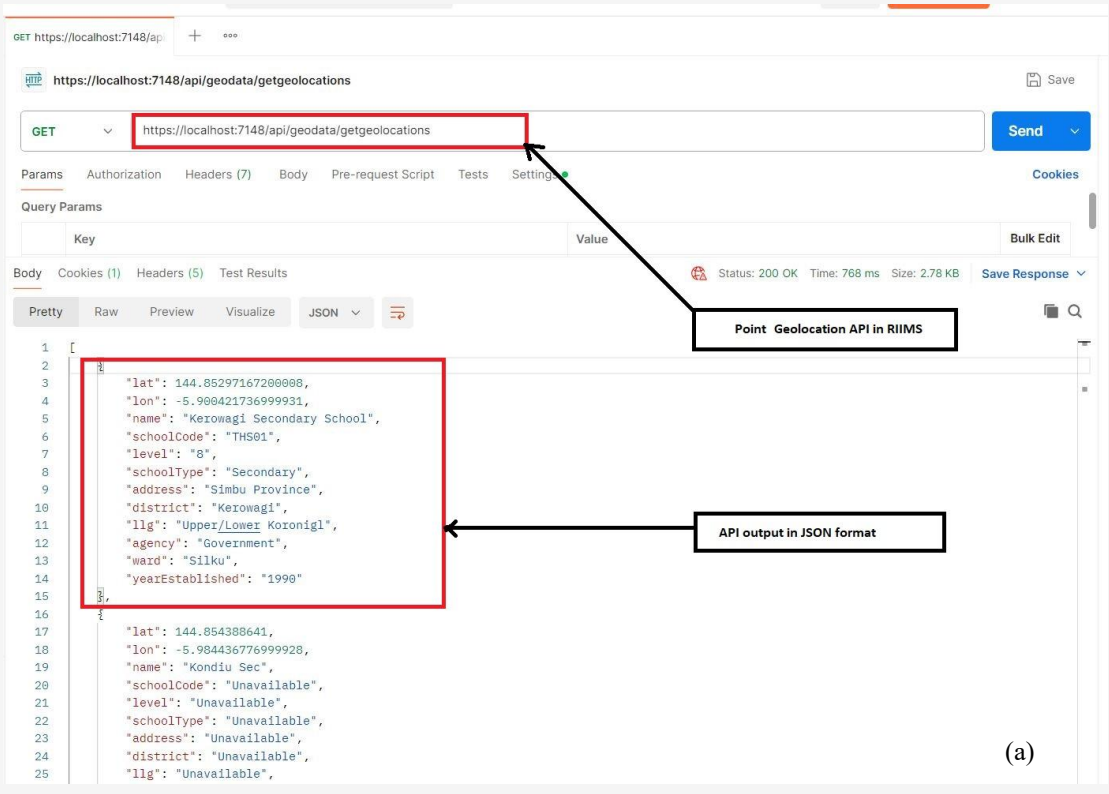


Figure 22: Geolocation API in RIIMS (a) Postman API response
(b) Postman API response of LineString

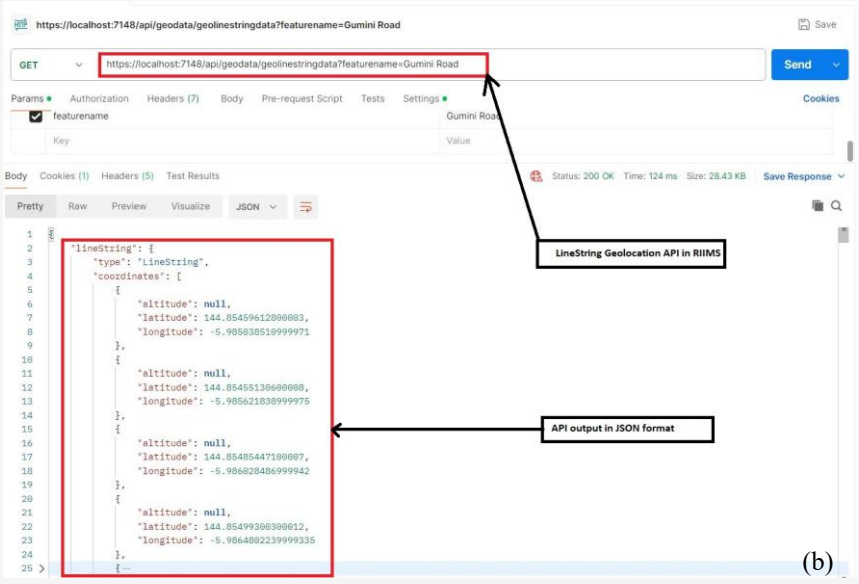


Figure 22: Geolocation API in RIIMS (a) Postman API response
(b) Postman API response of LineString

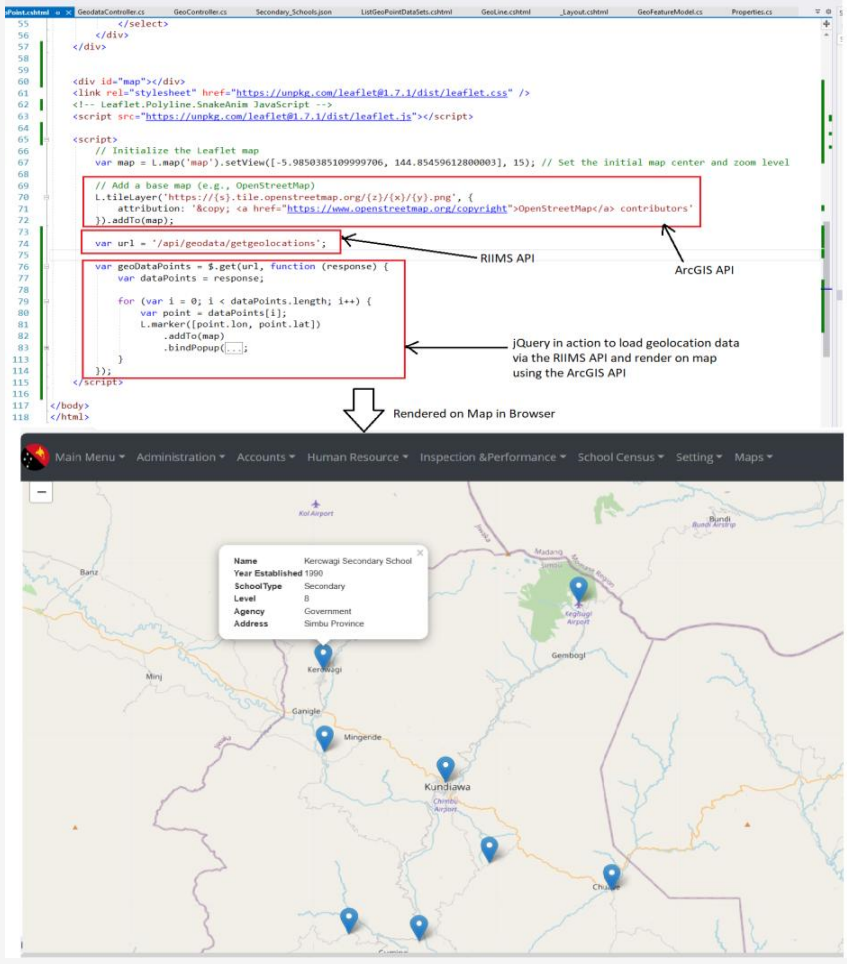


Figure 23: Calling RIIMS Point API using jQuery and integrating with ArcGIS API to render point geolocation data on ArcGIS map

6.5 Integration of RIIMS locational API with ArcGIS API

Using the geolocational APIs created in RIIMS, are then able to utilize them to associate with the locational information captured in the RIIMS system with the ArcGIS mapping API and rendered the information on the ArcGIS location services over the internet. ArcGIS APIs has been the first choice in this research as it provides advance GIS features, mapping capabilities and analytical functions to access ArcGIS Services over the web services.

6.5.1 Point geolocation data API

For the point geolocational data in RIIMS, the collection of specific information such schools, bridges, health centers, clinics and other locational - related information under the study region are used to develop SQL database. All this information was captured and stored in RIIMS.

It is then utilized the RIIMS in-app API together jQuery loading geolocation with the ArcGIS mapping API to render the specific locational information as shown in Figure 23 on the map with its attribute descriptions.

6.5.2 LineString geolocational data API

To demonstrate the mapping capabilities of *LineString* geolocation data in RIIMS, it has tagged all locational information using georeferenced data extracted from ArcGIS software (desktop version) in GeoJSON format which is an open standard format designed for representing simple geographical features, along with their non-spatial attributes. It customized the data format by developing its own custom formats in RIIMS. LineString API using jQuery and integrated with ArcGIS API to render line geolocation data on ArcGIS map in browser as shown in Figure 24.

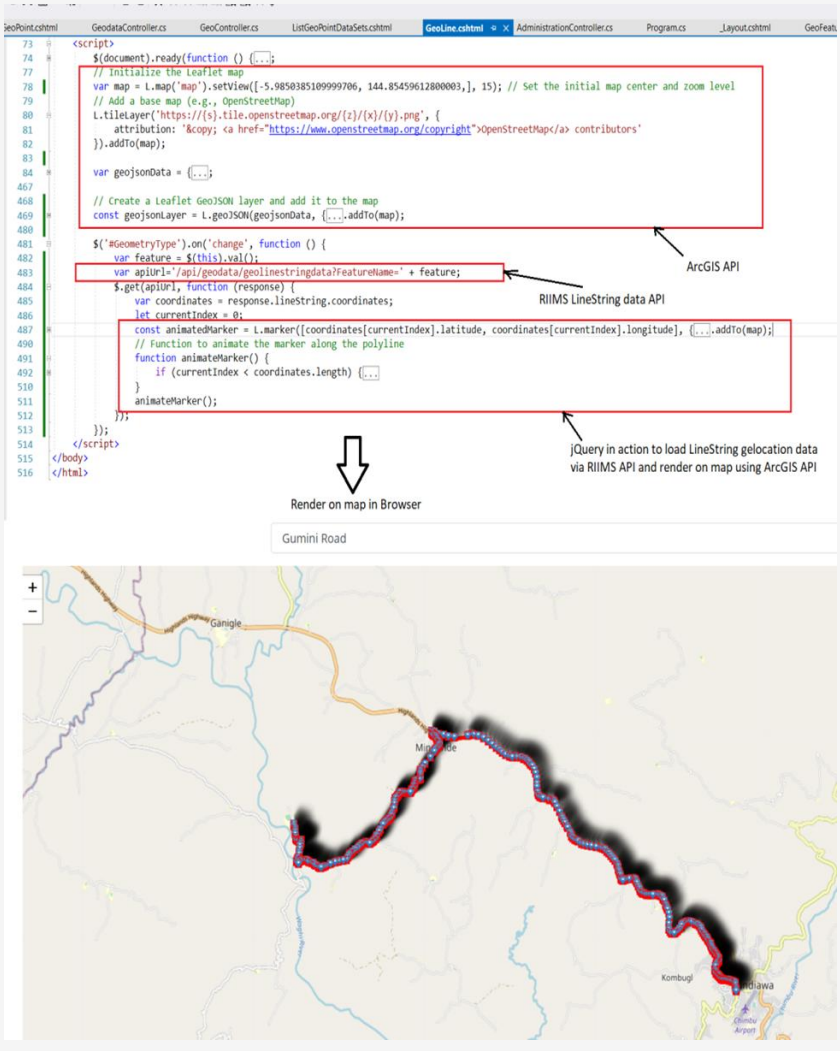


Figure 24: Calling RIIMS LineString API using jQuery and integrating with ArcGIS API to render point geolocation data on ArcGIS map

6.6 Performance Target of the System

The RIIMS System is developed to enable any professionals without GIS knowledge to operate under GIS environment for data sharing, analysis and processing. The system is highly user-friendly with secure authentication to ensure effective usage and real time data management. RIIMS target high-speed and reliable geospatial processing within the internet enable environment. For example, it delivers fast system response times for geospatial queries, such as an average of 200 milliseconds for point geolocation API requests. It can process large data volumes efficiently, handling up to 10,000 geospatial records in under 2 seconds. The application is developed as a scalable platform; RIIMS remains in its prototype development phase but demonstrates strong potential for expansion to accommodate additional requirements.

7. Conclusion

RIIMS software application integration with real-time WebGIS functions to enhance the capacity of administrative functions to manage operational function both spatial and non-spatial data in a more clustered and centralized user friendly platform. Hence problems regarding data collection and management, planning and coordinating administrative function will be solved through the use of RIIMS online platform. It is an online service which provides access to geospatial information and seamlessly deals with geospatial information stored in a database to execute simple and complex geospatial analysis over the internet and return results that contain locational information to user community.

The RIIMS is programed to capture all operational requirements with different user authentication to enable effectiveness of its functions through hierarchy of user connectivity. This Integrated database system will fit in with WebGIS APIs through the database that contains locational information to relate coordinate system. Hence related location-based services provide the digital connection between a place, its people, and their activities, and will be used to illustrate what is happening, where, how and why it is happenings in a real time setting. This research is particularly aiming to bridge the gap between traditional approach of administrative functions and data handling functions and modern digital functions. It aims to foster enhanced planning, better coordination and effective service delivery functions in any organisation. The RIIMS application is designed to handle huge dataset from various sources to be linked in the system. All the locational data are georeferenced to be digitally connected with their attributes information to generate automatic locational result over the internet for user communities. This system is connected with WebGIS services over the internet and can be able to process geolocational services from single request to complex and generate complex current up to date and reliable results for user communities.

References

[1] Zhao, Y., Li, K. K., Ng, K. P., Ng, C. H. and Lee, K. A., (2012). The RNA Pol II Sub-Complex hsRpb4/7 is Required for Viability of Multiple Human Cell Lines. *Protein Cell,* Vol. 3(11), 846-854 https://doi.org/10.1007/s13238-012-2085-7.

[2] Giuliani, G., Dubois, A. and Lacroix, P., (2013). Testing OGC Web Feature and Coverage Service Performance: Towards Efficient Delivery of Geospatial Data. *Journal of Spatial Information Science*, Vol. 7, 1-23. https://doi.org/10.5311/JOSIS.2013.7.112.

[3] López-Pellicer, F. J., Renteria-Agualimpia, W., Béjar, R., Valiño, J., Zarazaga-Soria, F. J. and Muro-Medrano, P. R., (2011). Implantation of OGC Geoprocessing Services for Geoscience. *Computer and Geoscience,* Vol. 47, 13-19. https://doi.org/10.1016/j.cageo.2011.10.023.

[4] Baumann, P., (2010). The OGC Web Coverage Processing Service (WCPS) Standard. *Geoinformatica*, Vol. 14, 447-479. https://doi.org/10.1007/s10707-009-0087.

[5] Kralidis, A. T., (2007). Geospatial Web Services: The Evolution of Geospatial Data Infrastructure. In: Scharl, A., Tochtermann, K. (eds) The Geospatial Web. Advanced Information and Knowledge Processing. Springer, London. https://doi.org/10.1007/978-1-84628-827-2_22.

[6] Peng, Z., Zhao, T. and Zhang, C., (2011). Geospatial Semantic Web Services: A Case for Transit Trip Planning Systems. Geospatial Web Services: Advances in Information Interoperability 2011, 169-188. https://doi.org/10.4018/978-1-60960-192-8.ch008.

[7] Giuliani, G. Dubois, A. and Lacroix, P., (2013). Testing OGC Web Feature and Coverage Service Performance: Towards Efficient Delivery of Geospatial Data. *Journal of Spatial Information Science*, Vol. 7, 1-23. http://dx.doi.org/10.5311/JOSIS.2013.7.112.

[8] Vinueza-Martinez, J., Correa-Peralta, M., Ramirez-Anormaliza, R., Franco Arias, O. and Vera Paredes, D., (2024). Geographic Information Systems (GISs) Based on WebGIS Architecture: Bibliometric Analysis of the Current Status and Research Trends.

Sustainability, Vol. 16(15). https://doi.org/10.3390/su16156439.

[9] Dhurandhar, P., Tamrakar, A. and Patra, J. P., (2022). A Review on GIS-based Online Information System for Rural Development in Chhattisgarh. *International Journal of Health Sciences*, Vol. 6(S2), 8226-8231. https://doi.org/10.53730/ijhs.v6n.

[10] Meng, Y. and Chen, W., (2020). Dynamic Information Management System of Agricultural Economy Based on WebGIS. In: Huang, C., Chan, YW., Yen, N. (eds) Data Processing Techniques and Applications for Cyber-Physical Systems (DPTA 2019). *Advances in Intelligent Systems and Computing*, Vol. 1088. Springer, Singapore. https://doi.org/10.1007/978-981-15-1468-5204.

[11] Krishna, G., Ratnam, K., Ravi, P. and Sridhar, P., (2015) A Holistic Approach for the Development and Implementation of Robust and Cost-Effective Enterprise WebGIS Business Solutions. *Journal of Geographic Information System*, Vol. 7, 478-485. https://doi.org/10.4236/jgis.2015.75038.

[12] Choi, Y., (2023). GeoAI: Integration of Artificial Intelligence, Machine Learning, and Deep Learning with GIS. *Applied Sciences*, Vol. 13(6). https://doi.org/10.3390/app13063895.

[13] Hardwicke, T. E., Mathur, M. B., MacDonald, K., Nilsonne, G., Banks, G. C., Kidwell, M. C., Hofelich Mohr, A., Clayton, E., Yoon, E. J., Henry Tessler, M., Lenne, R. L., Altman, S., Long, B. and Frank, M. C., (2018). Supplementary Material from "Data Availability, Reusability and Analytic Reproducibility: Evaluating the Impact of a Mandatory Open Data Policy at the Journal Cognition. The Royal Society. *Collection,* Vol. 5(8). https://doi.org/10.6084/m9.figshare.c.4175039.v1

[14] Tedersoo, L., Küngas, R., Oras, E., Köster, K., Eenmaa, H., Leijen, Ä., Pedaste, M., Raju, M., Astapova, A., Lukner, H., Kogermann, K. and Sepp, T., (2021). Data Sharing Practices and Data Availability Upon Request Differ Across Scientific Disciplines. *Sci Data,* Vol. 8. https://doi.org/10.1038/s41597-021-00981-0.

[15] Thanos, C., (2017). Research Data Reusability: Conceptual Foundations, Barriers and Enabling Technologies. *Publications*, Vol. 5(1). https://doi.org/10.3390/publications5010002.

[16] Yoon, A. and Copeland, A., (2020). Toward Community-Inclusive Data Ecosystems: Challenges and Opportunities of Open Data for Community-Based Organizations. *Journal of the Association for Information Science and Technology,* Vol. 71(12). https://doi.org/10.1002/asi.24346.

[17] Yoon, A., Jeng, W., Curty, R. and Murillo, A., (2017). In between Data Sharing and Reuse: Shareability, Availability and Reusability in Diverse Contexts. *Proceedings of the Association for Information Science and Technology,* Vol. 54(1), 606-609. https://doi.org/10.1002/pra2.2017.14505401085.

[18] Overpeck, J. T., Meehl, G. A., Bony, S. and Easterling, D. R., (2011). Climate Data Challenges in the 21st Century. *Science,* Vol. 331(6018), 700-702. https://doi.org/10.1126/Science.1197869.

[19] Wulder, M. and Coops, N., (2014). Satellites: Make Earth Observations Open Access. *Nature,* Vol. 513, 30-31. https://doi.org/10.1038/513030a.

[20] Wagemann, J., Clements, O., Marco Figuera, R., Rossi, A. P. and Mantovani, S., (2017). (2018). Geospatial Web Services Pave New Ways for Server-Based On-Demand Access and Processing of Big Earth Data. *International Journal of Digital Earth*, Vol. 11(1), 7-25. https://doi.org/10.1080/17538947.2017.1351583.

[21] Amirian, P. and Alesheikh, A. A., (2008). Publishing Geospatial Data through Geospatial Web Service and XML Database System. *American Journal of Applied Sciences*, Vol. 5(10), 1358-1368. https://doi.org/10.3844/ajassp.2008.1358.1368.

[22] Sack, C., (2017). Web Mapping. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2017 Edition), John P. Wilson (ed.). https://doi.org/10.22224/gistbok/2017.4.11.

[23] Quinn, S., (2018). Web GIS. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2018 Edition), John P. Wilson (ed). https://doi.org/10.22224/gistbok/2018.1.11.

[24] Esri, (2010). Esri GeoServices REST Specification Version 1.0. Retrieved November 1, 2018, from https://www.esri.com/library/whitepapers/pdfs/geoservices-rest-spec.pdf.

[25] Swift, J. and Goldberg, D., (2019). *Web GIS Programming. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2019 Edition), John P. Wilson (ed)*. https://doi.org/10.22224/gistbok/2019.1.5.

[26] Ozdilek, O. and Seker, D., (2008). *A Web-Based Application for Real-Time GIS.* Microsoft Word - 934.doc.