Contents

[Index **Error! Bookmark not defined.**](#_Toc207616226)

[Technical Summary Document - Technical Summary 2](#_Toc207616227)

[Problem Statement 2](#_Toc207616228)

[Design Considerations 2](#_Toc207616229)

[Architecture or Patterns Used 2](#_Toc207616230)

[Key Decisions and Trade-offs 2](#_Toc207616231)

[Technical Features Implemented 3](#_Toc207616232)

[Dependency Injection 3](#_Toc207616233)

[Unit Testing Approach and Tools Used 3](#_Toc207616234)

[Error Handling and Logging 3](#_Toc207616235)

[External Libraries/Packages Used 3](#_Toc207616236)

[Async or Parallel Operations 3](#_Toc207616237)

[Known Limitations and Future Scope 4](#_Toc207616238)

[Limitations 4](#_Toc207616239)

[Future Scope 4](#_Toc207616240)

# Technical Summary Document - Technical Summary

## Problem Statement

The **Insurance Fraud Detection System** aims to tackle the problem of fraudulent insurance claims by detecting suspicious activities during the claims submission process.

Claims can be created with a user ID or even anonymously. Once claims are submitted into the system, they can later be checked with an **AI classification model** (when trained) to decide whether a claim is genuine or fraudulent.

Following main features are added

* **SignalR** – Real-time claim status updates.
* **Custom Logging** – System logs saved to the file system.
* **CI/CD with GitHub Actions** – Automatic deployment to the server.
* **Environment-based Settings** – Dynamic configs for Dev, Staging, and Production.
* **Unit Testing** – Basic xUnit tests added for validation

## Design Considerations

### Architecture or Patterns Used

* **Clean Architecture**: Ensures separation of concerns, testability, maintainability, and scalability.
* **Domain-Driven Design (DDD)**: Models the business domain accurately to keep business logic at the core.
* **CQRS (Command Query Responsibility Segregation)**: Separates write operations (commands) from read operations (queries) for performance and scalability.

### Key Decisions and Trade-offs

* **SQLite** chosen for development simplicity, though not ideal for production-scale workloads. The SQLite database is stored inside a dedicated **Database** folder and kept up to date with **Entity Framework migrations**.
* **SignalR** used for real-time updates on fraud detection.
* **xUnit** selected for unit testing due to good CI/CD integration.
* **AI/ML Model Not Yet Added**: No AI/ML model is currently implemented because model training requires significant data and time. At present, fraud detection is rule-based.

## Technical Features Implemented

### Dependency Injection

* The system uses the built-in **Microsoft.Extensions.DependencyInjection** for IoC.
* A **separate static class** is responsible for configuring and registering all services, repositories, scoped services, and hosted services. This centralizes dependency management and ensures consistent service lifetimes across the application.

### Unit Testing Approach and Tools Used

* **xUnit** is used for unit testing.
* Current coverage includes services, repositories, controllers, and value objects.

### Error Handling and Logging

* **ILogger Interface with Custom Logging Service**:  
  The system extends the built-in **ILogger<T>** to create advanced log files. All unhandled exceptions and error messages are written to structured log files.
* **Global Exception Handling**: Not added as middleware. Instead, exception capture is handled via **ILogger** to simplify the design.
* **Trade-off**: Writing logs to files introduces processing overhead. In high-load scenarios, this can lead to slower performance or potential deadlocks.

### External Libraries/Packages Used

* **Entity Framework Core** – ORM for database operations.
* **SignalR** – Real-time communication between client and server.
* **Swagger/OpenAPI** – API documentation.
* **xUnit** – Unit testing framework.

### Async or Parallel Operations

* **Async/Await**: All claim request APIs are implemented using async methods, ensuring non-blocking execution.
* This allows requests to run without waiting for others to complete and provides non-preemptive concurrency for better responsiveness.
* **SignalR** operations are also asynchronous, ensuring efficient communication between multiple clients.

## Known Limitations and Future Scope

### Limitations

* **Database**: SQLite, while suitable for development, lacks scalability for high-volume production workloads.
* **Fraud Detection**: Currently rule-based, without AI/ML models.
* **User Authentication**: No authentication/authorization system in place.
* **Reporting**: Limited reporting and analytics features.
* **Logging Trade-off**: Current custom logger writes synchronously to files, which can increase CPU and I/O overhead.

### Future Scope

* **AI/ML Integration**: Train and integrate machine learning models to improve fraud detection accuracy and performance.
* **Database Upgrade**: Use scalable databases like SQL Server or PostgreSQL.
* **Authentication & Authorization**: Add OAuth/JWT with role-based access.
* **Notifications**: Implement email/SMS alerts for fraud and claim updates.
* **Pipeline-based Logging**: Introduce batched/pipeline-based logging to reduce CPU overhead by grouping log writes.
* **Caching**: Integrate Redis for faster access to frequent queries.
* **Microservices**: Transition towards microservices for scalability and modular development