National College of Ireland

Post Graduate Diploma in Cloud Computing

2011/2012

|  |
| --- |
| Motorist Offences  Penalty Points  Administration System |

Enterprise Frameworks Project

Claire Collins, x11106425

Anthony O’Toole, x11106697

Ronan Leonard, x11106611



I hereby certify that this material, which I now submit for assessment of the programme of study leading to the award of Master of Science in Web Technologies is entirely my own work and has not been taken from the work of others save and to the extent that such work has been citied and acknowledged within the text of my work.

Signed:………………………………

Date:

Student Number:

Signed:………………………………

Date:

Student Number:

Signed:………………………………

Date:

Student Number:

Table of Contents

[Introduction 5](#_Toc331237093)

[Project Plan 9](#_Toc331237094)

[Requirements Analysis 10](#_Toc331237095)

[Software Development Methodology 14](#_Toc331237098)

[Use Cases 15](#_Toc331237099)

[Architecture / Design Approach 23](#_Toc331237100)

[Models 25](#_Toc331237101)

[Implementation 30](#_Toc331237105)

[Design patterns and architectural patterns 33](#_Toc331237106)

[Cross-cutting Concerns 35](#_Toc331237107)

[Security 35](#_Toc331237108)

[Configuration 36](#_Toc331237109)

[Scalability 37](#_Toc331237110)

[Testing Approach 37](#_Toc331237111)

[Other Relevant Features 38](#_Toc331237112)

[Discussion 41](#_Toc331237113)

[Appendices 42](#_Toc331237114)

# Introduction         Motivation

We agreed with the development of a Motor Penalty Points System for three reasons. Firstly, we were inheriting a relational database structure that was built for Oracle, and re-engineering its structure to fit the object oriented world using C#. The big challenge was to let go of a database first development approach where tables relate using a common key and instead use a code first development approach, writing .Net classes to define the conceptual model for our application. Secondly, by inheriting an existing tables and relational database structure we were employing a model that was proven to work. Finally, modelling a complex structure meant overcoming challenges which gave exposure to code first development, object oriented programming, test driven development, service oriented architecture, patterns and implementation of a 5 tier application architecture using Visual Studio 2010, C# and MVC 3.5 in .Net Framework 4.0. The benefit of completing the project was the learning curve that was overcome to get to the finish line.

The Penalty Point system was introduced in Ireland in 2002 to encourage safe driving and reduce casualties on Irish roads. The number of driver offences attracting penalty points has grown to 42 since the system was introduced. A penalty point is a formal reprimand from the Gardaí given to a motorist who has committed a penalty point offence. Penalty points are endorsed on the offending motorist’s driving licence record. Penalty points show a motorist is guilty of committing a specific driving offence. A motorist has the option of paying a fixed fine on a penalty point offence rather than having to refer the matter to court. Some penalty point offences attract a mandatory court summons where the court decides on the fine to be paid by the motorist. A motorist accumulating 12 penalty points in a three year period attracts an automatically disqualification from driving for six months. Penalty points are removed from a motorist’s driving licence after three years. When an offence is recorded a fixed charge notice is issued to the motorist. The motorist has 28 days from the date of issue to pay the fine. A motorist paying within the subsequent 28 days will pay the fixed charge fine plus 50%. Where a fine has remained unpaid after 56 days a court summons is initiated.

**Project Scope**

The purpose of the project is to develop an online application to track and process penalty point offences committed by motorists on Irish roads in the Republic of Ireland. The application will store the existing 42 penalty point offences, allowing for additional penalty points to be added as they are introduced in the future. The list of penalty point’s offences is offered as a service. The application allows the entry of driver offences committed by motorist and stores the driver offence information including information on the offending motorist, driving licence status (i.e. 'full licence', 'provisional', 'disqualified' and 'no valid Irish licence') and the vehicle driven at the time of the offence. Penalty points, fixed fine charges and court summons are applied to driver offences based on the existing 42 penalty point offences stored by the application. The application allows the driver offence to complete its natural lifecycle by passing through a series of stages defined by the offending motorist actions. The stages that a driver offence can pass through are 'new offence', 'penalty point notification issue', '28 day notification', '56 day notification', 'fine paid', 'court summons' and 'deleted'. The application has a 'daily tasks' feature consisting of five tasks to supporting the driver offence lifecycle. Each task, when run, automatically passes the corresponding driver offences onto the next stage in their evolution. The application stores an index of all driver offences where offending motorists have been issued with a court summons notification. The application has an inbuilt search feature on driver offences. Driver offences can be searched by driver licence (or PPSN), by vehicle registration, by offence type or a combination of these. The application has an inbuilt statistics feature on driver offences providing statistics on ' the number of drivers by license type', ' the number of penalty points per driver per county' and ' the number of offences per offence type'. The application implements a user account management feature that defines three levels of access (i.e. garda role, admin role and super role) to application functionality. An administrator creates an account for a new user, assigning a role, username and password to that account. The administrator then provides the user with the corresponding username and password credentials which he uses to log into the application. Once logged in the user accesses only application functionality defined by the role assigned to his user account.

The application is useful because, not only is it designed to allow driver offence information to be entered, stored, searched for and interpreted, most importantly the application does not interfere with the natural lifecycle of a driving offence. The application enforces it by allowing the actions of the motorist to dictate what stage in the lifecycle to pass a driver offence too.

The goal of the project was to develop an application that would meet the requirements of a National Motorist Penalty Point System based on the penalty point system for driver offences introduced in Ireland in 2002.

To meet the project goal we employed the strategy below to ensure that each stage in a software application lifecycle was completed in a predictable, efficiently and repeatable manner included the follow steps:

1. Functional Design: researching the existing penalty point system used in Ireland (customer focused design) and inheriting a existing database model proven to work in Oracle (developer focused design) helped determine the need to develop this application, what it will do and who will use it.
2. Technical Design: how functional design goals could be achieve was predefined by the Enterprise Frameworks project outline document. Visual Studio 2010, ASP.NET, MVC, C#, SQL Server 2008, Code-First Development, Unit Testing, and a 5-Tier Architecture implementation had to be the development environment.
3. Programming & Implementation: The DAL layer and Models and SOA were constructed by Anthony. The BLL layer, Model Views, Controllers, Views and, User Logins and Roles were implemented by Claire and Ronan.
4. Application Testing: we followed a 'design-test-design' approach when developing the application. Unit Tests were implemented Anthony on the 'List of Offences' feature. System and integration testing was implemented by Claire and Ronan as frontend functionality was implement as new functionality and features were built into the application or the application was modified were a fault was detected or functional and technical goals had not be met. Quality Assurance was based on fault detection.
5. System Documentation: each team member maintained a personal diary recording progress’s, issues, challenges, solutions, tests, completed tasks, outstanding tasks etc. At the end of the project a technical report was produced.

Key characteristics of the project in terms of functionality and deliverables on completion include:

1. Functionality:
   1. CRUD functions for Listed Offences.
   2. CRUD functions for Driver Offences to include Driver, Vehicle and Offence information.
   3. Search functions on Driver Offences.
   4. Statistics functions on Driver Offences.
   5. Automated Daily Tasks functions for notifications and court summons on Driver Offences.
   6. An index view of Court Summons.
   7. An index view of List Offences is provided as SOA.
   8. Enforce the natural lifecycle of a driver offence dictated by motorist actions.
   9. Object oriented programming is used to enforce the ideal of 'store once, use many' for Driver Offence, Driver, Vehicle and Offence models.
2. Deliverables:
   1. The applications initial release on the 28th July 2012 will provide baseline functionality for all functions highlighted under 'Functionality'.

Some of the traits and qualities the application exhibits that are considered as common attributes of an enterprise application (based on Martin Fowlers Patterns of Enterprise Architecture) includes persistent data, concurrent data access, lots of user interface screens, layered architecture and provision for services.

**Background Research & Investigation**

With no prior experience of Enterprise Frameworks, little knowledge of Object Oriented Programming and ten weeks to release date, inheriting a proven Oracle database model allowed us to implement the data models from the start of the project using code first development. Access to documentation and direct access to the database engineer responsible for the inherited model's design, implementation and documentation allowed us to gain a thorough understanding of its structure at a very early stage in the project development lifecycle. This provided a baseline enabling the database developer to design and implement data models defining database tables, and enabling the business logic and UI developers to research Ireland's existing Penalty Point System and investigate the business requirements for the new online application. Use cases were then produced defining the key business functionality and data.

Project constraints stemmed from an acquired development environment and architectural model. The development environment operating within the enterprise was Visual Studio 2010, ASP.Net 4.0, MVC 3.5, C#, DB Context and SQL Server 2008. Object Oriented Programming and Code First Development were stated as preferred methodologies, with n-Tier Architecture Design stated as the preferred model dividing the entire application into tiers or layers i.e. database layer, data access layer, business logic layer, presentation layer and UIs. A additional request for Service Oriented Architecture to provide a public facing service was fated.

Using 5-Tier Architecture Design allowed the database developer to focus on designing and implementing data models and implementing the Data Access Layer to store application data to, and retrieve application data from, a SQL Server database. It also allowed the business logic and UI developers to focus on designing and implementing the Business Logic Layer and Presentation Layer (including Model-Views, Controllers and Views) to validate and process data entered by the end-user and to deal with the applications user interface. The division allowed Business Logic and UI developers to drive application design by answering customer needs and maximising value to the business, while allowing the database developer to drive database design by looking at the technological possibilities first and shaping them to what the wants.

Implementing an Agile approach for application development allowed developers to use a simple iterative process with emphasis on collaboration between Business Logic and UI developers and the database developer, and creativity in design and development, and the use of technologies, to produce a functioning online Motorist Penalty Point application by the 10 week deadline.

Implementing a design-test-design approach allowed developers to build-up application functionality feature by feature. Each feature was tested independently of the working application. It then was integrated into the working application and the entire application was retested. Additional tests were performed against business and functions goals and modifications made were goals were not met. The end product produced from each iteration of the design-test-design approach was a working application with increased functionality.

# Project Plan

Claire Collins

* Installation of Visual Studio 2010, SQL Server 2008 and MVC3.5.
* Investigated Code First Development to assist Anthony in resolving issues surrounding the use of POCOs to generate a database schema.
* Production of use cases describing key functionality and report template.
* Implemented the Business Logic Layer and Presentation Layer for 'Listed Offences'.
* Implemented the Business Logic Layer and Presentation Layer for 'Driver Offences'.
* Implemented Search feature on driver offences and application interface.
* Implemented Statistics feature on 'Driver Offences'.
* Implemented Daily Task interface and key functionality for '12 penalty disqualification 11 notification' and '3 year penalty deletion notification'.
* Administered and implemented user accounts, permissions and roles. Building of master application. Technical Report write up. Presentation.
* Ongoing collaboration with team members, and documentation of personal progress.

Anthony O’Toole.

* Generation of initial Project Idea
* Research on SQL server installation, configuration & connection to Visual Studio.
* Documentation & diagrams as an initial aid to communicate how salient aspects of the application & how it might map to the n-tier Software Architecture.
* Initial ‘Entity-Relationship Model’ research & design.
* MVC-DAL (coding & shakedown testing).
* Suggested table scanning algorithms for later use & amendment in BLL Layer.
* Coding of TDD (Unit Test) for Listed Offences.
* Technical Report write up. Presentation.

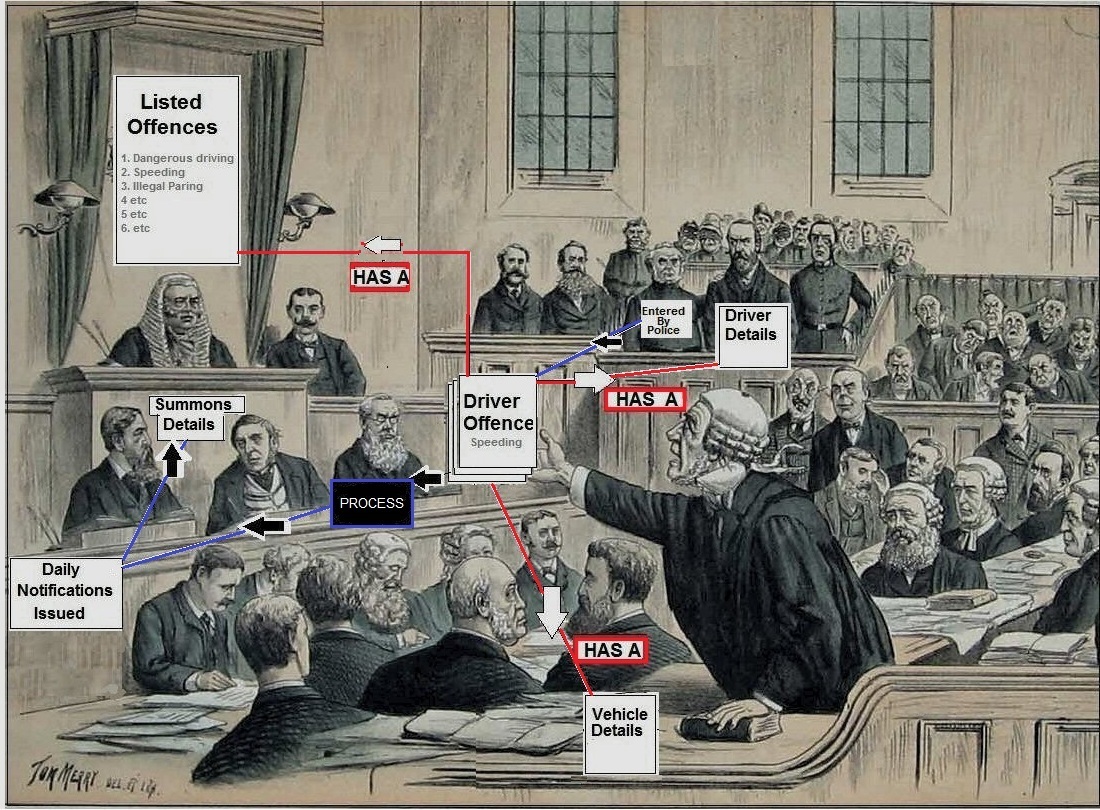
The dialectic between the’ Customer facing’ BLL centric view of the world & the DAL/ Database centric view of the world also proved interesting and productive in synthesizing a workable application.

Ronan Leonard.

* Pushed Files to Git.
* Daily Task Coding.
* SummonsDetail.DAL shakedown testing.
* Technical Report write up. Presentation.
* Data Entry.

# Requirements Analysis

**Modelling the Real World**



**A system to administer a Department of Justice Motoring Offences Penalty Points.**

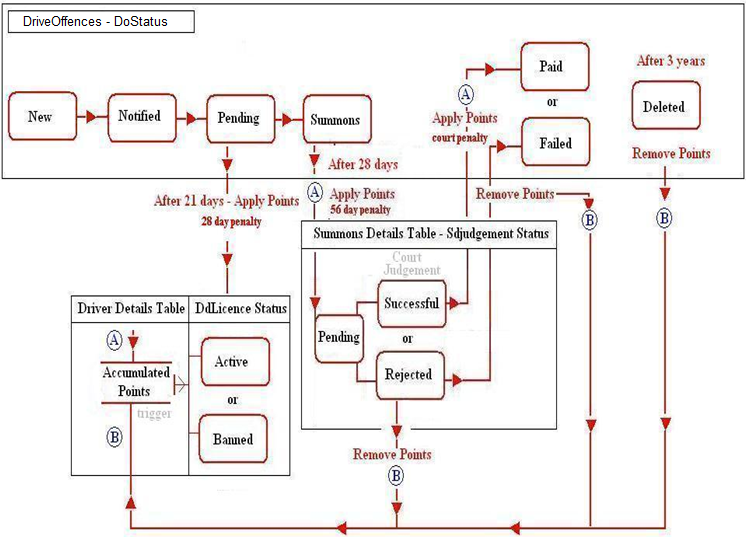
* Methods for the Police to record Statutory Driving Offence against a driver
* Methods to allow Legal System to administer said offences

# 

**The project is implemented as a n-tier ‘Domain Model’ architecture**

**We also need a daily task procedure to process the Offence through its legal lifecycle**

# Daily Task



Daily batch process runs daily or can be triggered on demand

*These rules are an example of typical processing & may change as the project evolves*

* This process notifies users of new offences. It will utilise the 28 day penalty element & apply points after 21 days accordingly
* This process is also a method whereby a daily check of all on-responders outside of the 28 day payment limit is made. If found it will utilise the 56 day additional penalty element and issue a summons & apply points accordingly.
* This process is also a method whereby a daily check of all non-responders with judgements issued against drivers. If found it will apply the court points penalty element
* This process is also a method whereby a daily check of all non-responders with judgements issued in favour of drivers. If found it will decrement the 56 day additional penalty element
* This process is also a method whereby a check of all points over 3 years. If found it will decrement those points via the DRIVER\_BANNED method
* This process is also a method whereby daily notifications issued table is populated for use by the daily tasks report.

**Functional Requirements**

Standard functions:

Legal Offence Interface

* Ability to add new offences introduced at a future date.
* Ability to edit and delete existing offences while maintaining the integrity of existing driver offences.
* Ability to view existing offence details.

Driver Offences Interface

* Ability to add new driver offences including vehicle, driver and offence details.
* Ability to edit and delete existing driver offences.
* Ability to view existing driver offence details.

Search features

* Ability to search existing driver offences by driver license (PPSN) number, vehicle registration number, offence type or any combination of these search keys.

Court Summons

* Ability to generate a list of court summons and drill down to view further details.

Service Oriented Architecture

* Provision of a service to view the listed offences.

Reporting requirements:

Statistics features

* Ability to generate statistics based on 'penalty points by offence type', 'penalty points incurred per driver by county' and 'number of drivers by driver license type'.

Administration functions:

Daily Tasks features

* Ability to run automated daily tasks to generate notifications and court summons on driver offences. Notifications are required for new driver offences, fixed-fine charges unpaid after 28 days, court summons for fixed-fine charges unpaid after 56 days, license disqualification for drivers accumulating 12 or more penalty points and driver offences deleted after 3 years.
* If a daily task is not run then it must roll over to the next day's run.

Authentication and Authorisation functions:

User Login

* Access to application functionality must be implemented through user login.
* Three levels of access exist. The 'Super' role has full-access all areas of the application. The 'Admin' role has full-access to Daily Tasks, Summons, Search and Statistics features, limited-access to Legal Offences, and no-access to Driver Offences. The 'Worker' role has full-access to Driver Offences, Search, Statistics and Summons, limited-access to Legal Offences, and no access to Daily Tasks.

**Non-Functional Requirements**

Security:

* User accounts are managed by an administrator.
* Roles and permissions are set by an administrator.
* Users must log in to access application functionality.
* Key focus is on keeping unauthorised people out of the application and keeping personal data private, protected and confidential.

Performance:

* 3 second response time on calculations, queries and statistics
* 2 second response time on saves.

Availability:

* Online access via URL.

Integrity:

* Application data must match the original data entered by the end-user.
* Processing must ensure that application data accurate and correct.
* Statistical calculations and results must be accurate and correct.
* Searches results must be accurate and correct.
* Daily tasks must ensure that the right records are updated, that updates are accurate and correct, and that updates do not interfere with or modify other application data.

Usability:

* Interactive, user friendly, easy to use and easy to learn application.
* A useful and usable application.
* Simplistic data entry, retrieval and views.
* No deep rooting for data or functionality.
* Least amount of time spent entering data, searching for records, generating statistics or running daily tasks.
* Application data and text must be readable and understandable.

# Software Development Methodology

We utilized an ‘Agile’, lightweight, incremental prototyping approach to the software development.

* Code First approach
* Database generated from a coded OOModel description
* Dbcontext (DTO ) wrapped the Parent/Child relationships of the OOModel into a Data Transefer Context object

The agile development methodology allowed us to build the application progressively by building upon small working functional features during its 10 weeks development lifespan.

Initial development took an MVC approach & the removed Database I/O calls from the controller to a DAL Layer (MVC-Dal). The efficacy of the DAL layer was the tested from the views generated from the dbcontext model.

Once the Model & Dal layer was established, the Business Logic Layer was implemented. It is presumed that on a commercial site the separation of concerns imposed by a Repository pattern would allow for independent, parallel development of the BLL & Dal layers to take place by the individual team members (i.e. architectural de-coupling ensures decoupling of the development life cycle) and the BLL could use mockups for their testing. However, in our project, there was still some linear interdependence between the two areas.

The light-weight Agile Software Development methodology was used to structure, plan and control the development process for the application. The methodology allowed us to incorporate a level of flexibility into the development process to overcome the constraints that stemmed from our lack of knowledge of the development environment. The Agile methodology allowed us to apply a level of practicality, uncomplicatedness, simplicity and common-sense into the delivery of the finished application. The methodology allowed us to focus on keeping code simple, and testing application features often during the development process. Functional application features were integrated into the application once they were ready. The agile development methodology allow us to build the application progressively by building upon small working functional features during its 10 weeks development lifespan.

The process developed to build the application was as follows:

1. Initial planning: The entire application was divided into functional requirements.
2. Planning: Deciding upon what action was to be done next. The outcome triggered the iterative process (i.e. step 2 to step 7).
3. Requirements: A requirement was selected for further development.
4. Analysis and Design: Decisions were made on how the function would operate, how it would be implemented and who would implement it.
5. Implementation: The function was implemented.
6. Testing: The function was tested independently of the application. Errors or faults detected were resolved.
7. Evaluation: The function was then evaluate against the functional and business requirements and modification to met them, if required.
8. Delivery: Once the function was ready, it was integrated into the master application. This activity also triggered the iterative process.

# Use Cases

|  |  |
| --- | --- |
| **Use Case Name** | Create Driving Offence |
| **Use Case ID** | PPUC001 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Create Driving Offence use case allows a User to record an offence committed by a motorist to the system and includes the offence details, vehicle details, Driver details, location, date of offence, penalty points and date applied, and status of offence. |
| **Goal / Scenario** | User wants to record an offence committed by a motorist. |
| **Actor(s)** | Garda |
| **Preconditions** | User has access to internet  User can log in to the system as a worker  User has information on the offence, driver and vehicle details. |
| **Assumptions** | * The offence, driver and vehicle information entered is authentic and valid. * All motorists have a driving licence and are identified by their driving licence number |
| **Triggering Event** | The User wants to record an offence committed by a motorist. |
| **Main Path (Happy Path)** | Use case begins when system display the Driver Offence index screen.  1. User clicks 'Create New' button.  2. System displays "Create New Driver Offence" screen.  3. User enters driver offence information into screen (offence description, location, offence date, status of offence, penalty points and date applied, vehicle registration, type, make and engine capacity, and driver name, address, licence number, licence status and accumulated points.).  4. User clicks 'Create' button.  5. System saves the driver offence information into the database.  Use case ends when the system returns the Driver Offence index screen. |
| **Alternative Path** | The User cancels the creation of a driving offence. The process is stopped and Driving Offences remains unchanged. The user is returned to the Driver Offence screen. |
| **Exception Path** | Invalid driving offence information:   * information is missing * driving offence information entered does not comply with its definition.   If the system identifies during the creation of the driving offence that invalid information is added the system will:   1. describe to the User which information was invalid and suggest valid information. 2. prompt User to re-enter valid driving offence information. 3. validate re-entered driving offence information. 4. if driving offence information is valid save the driving offence to the database. 5. if offence information is invalid execute step 1 above. |
| **Business Rules** | 1. A motorist can have a full Irish licence, a provisional Irish licence or no driving licence. 2. A motorist without a driving licence cannot be identified by a driving licence number. 3. A driver of a vehicle is not assumed to be the registered owner of a vehicle involved in the driving offence. 4. Where the driving offence is caught on camera, the registered owner is assumed to be the driver. If the registered owner is not the driver the driver details must be updated accordingly with the details of the actual driver. The offence applies to the driver and not the registered owner. 5. The information entered must comply with the driving offence definition. |
| **Post conditions** | DrivingOffences stores the new driving offence. The driving offence is available for use across the system. |

|  |  |
| --- | --- |
| **Use Case Name** | Search |
| **Use Case Identifier** | PPUC002 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Search use case allows a User to perform a search (by driver licence number, vehicle registration or offence type) on all stored driving offences committed by motorists and return all corresponding records, if any exist. |
| **Goal / Scenario** | The user wants to view a specific set of driving offences that correspond to a driver licences number, vehicle registration, offence type or any combination of these search keys. |
| **Actor(s)** | Garda |
| **Preconditions** | User has access to internet  User can log in to the system as a worker  User has the driving offence information to perform a search. |
| **Assumptions** | All motorists have a driving licence and are identified by their driving licence number. All vehicles are identifiable by a registration number. |
| **Triggering Event** | The User defines a search criteria on all driving offence records. |
| **Main Path (Happy Path)** | Use case begins when system display the Search screen.  1. User selects the search key(s) from drop down lists representing vehicle registration numbers, driver license number and offence type.  2. User clicks 'Search' button.  3. System displays all driving offence records that match the search criteria on the "Driving Offence" screen.  Use case ends when the system returns and displays all driving offence records that match the search criteria. |
| **Alternative Path** | 1. The User cancels the search. The process is stopped and the Driving Offences screen remains unchanged. 2. The search returns no records that match the search criteria. The message 'No records found' is displayed on the DrivingOffences screen. |
| **Exception Path** | Invalid driving offence search information:   * driving offence search information entered does not comply with its definition.   If the system identifies during the search that invalid information is entered the system will:   1. describe to the User which information was invalid and suggest valid information. 2. prompt User to re-enter valid search information. 3. validate re-entered search information. 4. if search information is valid then search all stored driving offences returning only those records that match the search criteria. 5. if search information is invalid execute step 1 above. |
| **Business Rules** | 1. A motorist without a driving licence cannot be identified by a driving licence number. 2. A driver of a vehicle is not assumed to be the registered owner of a vehicle involved in the driving offence. 3. The search information entered must comply with the driving offence definition. 4. A search can be performed by vehicle registration, driving licence number, date of offence and location of offence. |

|  |  |
| --- | --- |
| **Use Case Name** | Fine Paid |
| **Use Case Identifier** | PPUC003 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Fine Paid use case allows a User to update the status of a driving offence to 'Fine Paid'. |
| **Goal / Scenario** | The user wants to update the status of a specific driving offence to 'Paid'. |
| **Actor(s)** | Garda |
| **Preconditions** | User has access to internet  User can log in to the system as a worker  User has the required driving offence information. |
| **Assumptions** | The offence exists in the database. |
| **Triggering Event** | The User performs a search for a specific offence. |
| **Main Path (Happy Path)** | Use case begins when system display the driver offence record on the DriverOffences index screen.  1. User selects 'Edit' link to right of driver offence record.  2. System displays driver offence record for editing.  3. User selects 'Fine Paid' from the Status drop-down list.  2. User clicks 'Save' button.  3. System saves updated record to the database.  Use case ends when the system returns an 'Updated Offence Status Saved' message. |
| **Alternative Path** | The User cancels the update. The process is stopped and the driving offence remains unchanged. |
| **Exception Path** | The offence does not exist. The process does not begin. |
| **Business Rules** | The status of an offence can be New Offence, Penalty Notification, 28 day Notification, Summons notification, Disqualification notification, Deleted, or Fine Paid. |
| **Post conditions** | The status of the offence is 'Paid'. The corresponding record in DrivingOffences is updated. |
| **Included Use Cases** | UC Lookup Driving Offences |

|  |  |
| --- | --- |
| **Use Case Name** | Extend Offences List |
| **Use Case Identifier** | PPUC004 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Administration, Garda, Sergeant |
| **Description** | Extend Offences List use case allows the User to add a new penalty point offence for motorists to the existing list of offences and includes the offence description, the penalty points and fixed charges or court summons it attracts) |
| **Goal / Scenario** | To update the existing list of offence to include new penalty point offences for motorist introduced by the Minister for Transport. |
| **Actor(s)** | Administrator |
| **Preconditions** | User has access to Internet  User can log in to the system as an administrator  User has information on the new penalty point offence |
| **Assumptions** | The new penalty point offence for motorists has been introduced by the Minister for Transport. The offence information entered is valid. |
| **Triggering Event** | User wants to add a new penalty point offence to the existing list of offences. |
| **Main Path (Happy Path)** | Use case begins when the system displays the ListedOffences index screen.  1. User clicks 'Create New' button.  2. System displays "Create Offence" screen.  3. User enters penalty point offence information into screen (offence description, penalty points attracted, fixed charge or court summons attracted, charge up to 56 days).  4. User clicks 'Save' button.  5. System saves the offence information into the database.  Use case ends when the system returns an 'Offence Saved' message. |
| **Alternative Path** | The User cancels the creation of a new penalty point offence. The process is stopped and the ListedOffences remains unchanged. The user is returned to the ListedOffences screen. |
| **Exception Path** | Invalid penalty point offence information:   * information is missing * penalty point offence already exists * penalty point offence information entered does not comply with its definition.   If the system identifies during the creation of the offence that invalid information is added the system will:   1. describe to the User which information was invalid and suggest valid information. 2. prompt User to re-enter valid offence information. 3. validate re-entered offence information. 4. if offence information is valid save offence to the database. 5. if offence information is invalid execute step 1 above. |
| **Business Rules** | * Offence description must be a maximum of x characters long and is mandatory. * Penalty points are whole number and is mandatory. * Fixed charge are monetary amounts and is mandatory. * Court summons are not mandatory. * Charge up to 56 days are monetary amounts and is mandatory. |
| **Post conditions** | ListedOffences is extended. The new offence is available for use from the DriverOffences screen and reporting/statistics screens. |

|  |  |
| --- | --- |
| **Use Case Name** | Generate Offence Notifications. |
| **Use Case Identifier** | PPUC005 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Generate Offence Notification List use case allows a User to generate and view a list of all new driving offences that require an offence notification to be posted to the motorist who committed the driving offence. |
| **Goal / Scenario** | The user wants to identify the motorists of a new driving offences that have not been posted an offence notification . |
| **Actor(s)** | Garda |
| **Preconditions** | User has access to internet  User can log in to the system as a worker |
| **Assumptions** | All motorists have a driving licence and are identified by their driving licence number. |
| **Triggering Event** | The User wants to generate a list of all new driving offences that require an offence notification to be posted to the motorist. |
| **Main Path (Happy Path)** | Use case begins when system display the Daily tasks screen.  1. User selects each notification type in turn from the dropdown box.  2. User clicks the 'Run Task' button.  3. System updates status of any driver issued a notification.  4. System displays a message showing all drivers issued a notification.  Use case ends when the system returns and displays all driver sent a notification. |
| **Alternative Path** | 1. The User cancels the search. The process is stopped and the Driving Offences screen remains unchanged. 2. The search returns no records that match the search criteria. The message 'No records found' is displayed on the DrivingOffences screen. |
| **Exception Path** | 1. None |
| **Business Rules** | 1. A motorist without a driving licence cannot be identified by a driving licence number. 2. A driver of a vehicle is not assumed to be the registered owner of a vehicle involved in the driving offence. 3. All new offences recorded have a status of 'New'. 4. When an offence notification is sent to the motorist who committed the offence the status is updated to 'Notified'. |
| **Post conditions** | System remains unchanged. |
| **Outstanding Issues** | Motorists who commit an offence without holding a driving licence cannot be recorded in the system.  An Account Management System needs to be set up so the 'Search Driving Offence' use case is restricted to Users with a worker account. |

|  |  |
| --- | --- |
| **Use Case Name** | Update Driving Offence |
| **Use Case Identifier** | PPUC008 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Update Driving Offence use case allows a User to update a driving offence record that already exists. |
| **Goal / Scenario** | The user wants to update a specific driving offence e.g. changing the status of the offence or updating driver details etc. |
| **Actor(s)** | Sergeant |
| **Preconditions** | User has access to internet  User can log in to the system as a worker |
| **Assumptions** | All motorists have a driving licence and are identified by their driving licence number. |
| **Triggering Event** | The User wants to update the details of a specific driving offence. |
| **Main Path (Happy Path)** | Use case begins when system display the driver offence index screen.  1. User selects 'Edit' link.  2. System displays Driving Offence Edit screen.  2. User updates offence information.  3. User clicks 'Save' button.  4. System saves the updated driver offence information into the database.  Use case ends when the system displays the Driver Offence Index screen. |
| **Alternative Path** | The User cancels the update. The process is stopped and the driving offence record remains unchanged. The Driving Offences screen is displayed. |
| **Exception Path** | Invalid driving offence search information:   * driving offence search information entered does not comply with its definition.   If the system identifies during the search that invalid information is entered the system will:   1. describe to the User which information was invalid and suggest valid information. 2. prompt User to re-enter valid driving offence information. 3. validate re-entered driving offence information. 4. if driving offence information is valid save the driving offence to the database. 5. if offence information is invalid execute step 1 above. |
| **Business Rules** | 1. A motorist without a driving licence cannot be identified by a driving licence number. 2. A driver of a vehicle is not assumed to be the registered owner of a vehicle involved in the driving offence. 3. The driving offence information entered must comply with the driving offence definition. |
| **Post conditions** | Driving offence record has been updated in the database. |

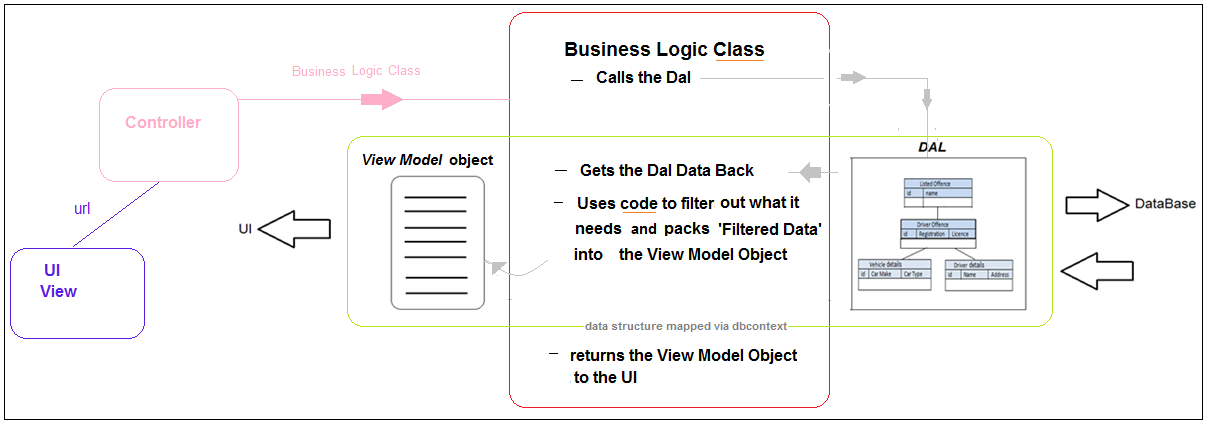
|  |  |
| --- | --- |
| **Use Case Name** | Statistics |
| **Use Case Identifier** | PPUC011 |
| **Version No** | 1.0 |
| **Version Date** | 10/07/2012 |
| **Author** | Claire Collins |
| **Stakeholders** | Garda |
| **Description** | The Statistics use case allows a User to retrieve predefined statistical views of driver offence data. |
| **Goal / Scenario** | The user wants to view specific statistical data on driver offences. |
| **Actor(s)** | Garda |
| **Preconditions** | User has access to internet  User can log in to the system as a worker  User has the driving offence information to perform a search. |
| **Assumptions** | All motorists have a driving licence and are identified by their driving licence number. All vehicles are identifiable by a registration number. |
| **Triggering Event** | The User selects a specific statistical view of driver offence data. |
| **Main Path (Happy Path)** | Use case begins when system display the Statistic screen.  1. User selects the statistics view from drop down lists.  2. User clicks 'Generate Statistics' button.  3. System displays the requested statistical view of driver offence data.  Use case ends when the system displays a statistical view of driver data. |
| **Alternative Path** | 1. 'Generate Statistics' returns no records that match the statistics requested. |
| **Exception Path** | 1. None |
| **Business Rules** | 1. A motorist without a driving licence cannot be identified by a driving licence number. 2. A driver of a vehicle is not assumed to be the registered owner of a vehicle involved in the driving offence. 3. The search information entered must comply with the driving offence definition. 4. A search can be performed by vehicle registration, driving licence number, date of offence and location of offence. |
| **Post conditions** | System remains unchanged. |

# Architecture / Design Approach

**N tier System Architecture Pattern**

|  |
| --- |
|  |

**N tier paradigm description**



1. UI (View) goes to Controller then goes to Business Logic Class.

2. The Business Logic Class calls the DAL

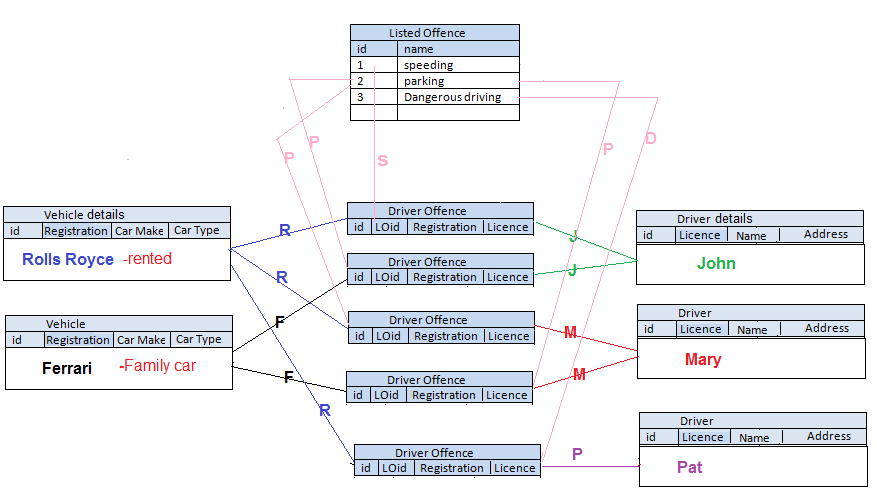
3. The DAL returns the Dal Model data to the Business Logic Class

4. The Business Logic Class filters the returned (DAL model) data into a View Data

Model object (which was defined as a UI View Model) and then returns this to the

View (UI)

**Proposed Penalty Points Entity Relationship model**



1 Listed Offence to many Driver Offences.

1 Vehicle Details to many Driver Offences.

1 Driver Detail to many Driver Offences.

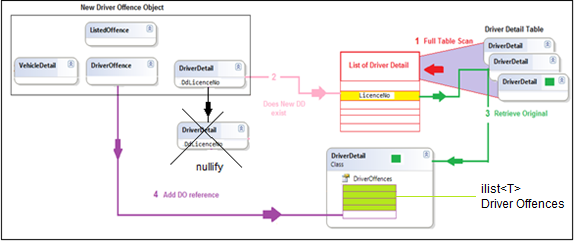
Many Vehicles to Many Drivers **(**Driver Offences acts as intermediate xref table**)**

# Models

**Data Model - Approach 1:** uses Ilists<T> for M :->1

|  |
| --- |
| **Driver Offence Object Model Relationship**    Listed Offence(**LO)** **1:m** Driver Offence. Vehicle Details(**VD)** **1:m** Driver Offences.( **DO)**  Driver Detail **1:m** Driver Offences. Vehicles **m:m** Drivers  *Driver Offences effectively acts as intermediate xref table for DriverDetails\_VehicleDetails*  *Note:* ***No Keys on* DriverDetail:DdLicenceNo , VehicleDetail:VdRegistrationNo , DriverOffence:doListedOff** see "Using Has Required to Help When there is no Foreign Key Property"[http://msdn.microsoft.com/en-us/data/hh134698.aspx](https://db3prd0206.outlook.com/owa/redir.aspx?C=-zWzJzfKa0mAQwIwj-9MSK-KlEmDN88IDQlKOA2QmwCSWtmIZqVDPee8xMMDL9dSqNmZrjvFGR4.&URL=http%3a%2f%2fmsdn.microsoft.com%2fen-us%2fdata%2fhh134698.aspx) |

**OO Driver Detail & Vehicle Detail table scanning algorithm**

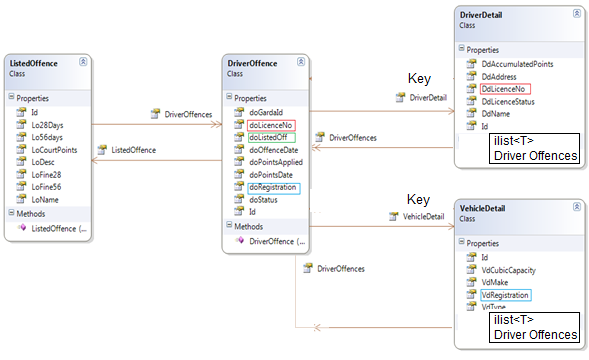
****

**1.** The **OO Driver Detail & Vehicle Detail table scanning algorithm** for the "Driver Offence and Driver Detail processing", is dependent on **no**t allowing a driver offence deletion. If a deletion did occur the "extracted" list's index+1 would now not map to a Driver detail ID on a 1:1 basis.( i.e. if we delete a DriverOffence DriverDetail tuple(pair)  with an id (say of 3) then the **List of Driver Details** (index+1) correspondence to DriverDetail ID would break down)  
  
 List Of Drivers    Driver Detail  
  index                    id  
   0                         1  
   1                         2  
   **2**                        **4** <--- 3 has been deleted  
   3                         6

Also cascaded deletes themselves become very complex

**Now, here's the onion** (i.e. one of those things that will make you cry)  
This algorithmic approach was later found to be very inefficient as it requires full table scans.

**Data Model - Approach 2 -** Using Keys for faster reads



We explored annotating **Unique Keys** onto DriverDetail : DdLicenceNo , VehicleDetail:VdRegistrationNo DriverOffence:doListedOff

**\*** effectively acting as unique keys for DO->DD, DO->VD and LO->DO searches

But in OO we use an ID (which is an int) as our primary key.

- This comes from (a) **ORM (Object Relational Mapping)** implemented using

LINQ TO SQL (System.Data.Linq.Mapping)

We need primary keys which are alphanumeric (Registration & Licence No).

- This can be done with (b) System.ComponentModel.DataAnnotations

**But (a)** & **(b)** don't like each other & won't work together.

So it seems that we would have to use **System.Data.Linq.Mapping.AssociationAttribute** and then use some **data mapping** techniques in the **OOPenaltyPointsContext.cs** .:-  
e.g.

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<DriverDetail>()

.HasKey(dd => dd.DdLicenceNo);

modelBuilder.Entity<VehicleDetail>()

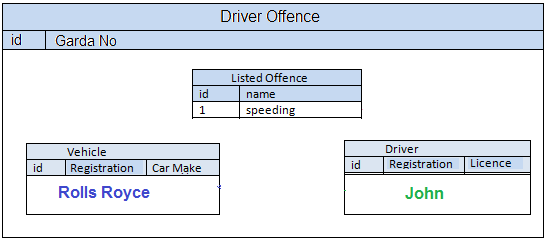
.HasKey(vd => vd.VdRegistration);

}

     This creates a clustered index (and is visible in SQL Server management studio under the indexes folder of the specific table). **BUT** this causes the ID’s on the Driver Detail and the Vehicle Detail to always be ‘zeroed’, so OO integrity is compromised. Also entity tracking is a nightmare.

**Data Model - Approach 3 –** Removed Ilists & Keys

We then utilized this OO model structure

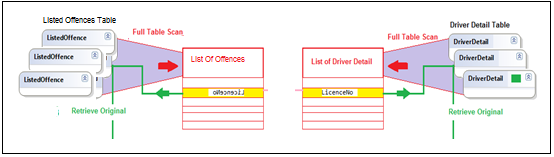


This model **successfully persisted** to the database and is a purer OO solution

as the Driver & Vehicle children are stored ‘physically’ within in the Offence so we could more easily dot notate down to the child level.

However due to the inability to apply keys (as indicated in approach 2) we still had to use full table scans

Cascade of updates & Deletes(Upserts/Downserts) have not been implemented but we can ‘mark’ a Driver Offence object as having been deleted & user can re-enter a new one.



So we encountered a combination of **OO RDB impedance mismatch , DAL layer** issues and alack in our skillset to be able to resolve them.

*Acceptable Performance –*

*DAL must operate in accordance with expected service levels*

*Data Mapping*

*- Our Linq ModelBuilder skills could not resolve key mapping issues (approach 2)*

*So we had to use full table scans where indexed-key reads would be much more efficient.*

*- with more time we might have resolved these issues.(Next Version perhaps)*

Also some BLL code might be more appropriately migrated to the DAL

# Model Views

DriverOffenceUI and getDriverOffenceUI model views: The driver offence class combines a driver object, vehicle object and offence object together to make a driver offence object. To reduce complexity and transfer of unnecessary data, a simple model view was defined to pass driver offence data from the data access layer via the business logic layer to the presentation layer, and vice versa. Formatting of a driver offence object to a simple model view is implemented at the business logic layer. There are two model views, the first passes data by reference (it is a class) and the second passes data by value (it is a struct). The properties of these model views are: id, gardaId, location,status, offenceDate, offenceId, description, driverId, LicenceNo, FName, SName, Address1, Address2, Address3, LicenceStatus, vehicleId, Type, Make, and Capacity.

DailyTasksUI model view: This model view contains a single 'Name' property and was implemented to hold the value of the daily task selected by the end-user which was used by controller to determine a method call to the business logic layer.

DriverLicenceUI model view: This model view was implemented for use in calculations for 'Number of Drivers by Licence Type' statistics. The model view contains a 'licenceNo' property to hold the value of the licence number for a unique driver. A list of declared to hold each unique driver pulled from the driver offences list.

LicenceTypeUI model view: This model view was implemented for use in calculations for 'Number of Drivers by Licence Type' statistics to hold the total number of licence types for each unique driver from the driver offences list. It contains a property for each licence type. The properties are: FullLicence, Provisional, Disqualified, NoLicence and Other.

StatisticsUI model view: This model view contains a single 'Name' property and was implemented to hold the value of the statistics calculation selected by the end-user which was used by controller to determine a method call to the business logic layer.

DriverPointsUI model view: This model view was implemented for use in calculations for 'Penalty Points Incurred Per Driver Per County' statistics to hold the total number of penalty points accumulated by a unique driver and the driver’s county of residence. The properties include County and AccumulatedPoints.

DriverPointsByCounty model view: This model view was implemented for use in calculations for 'Penalty Points Incurred Per Driver Per County' statistics to hold the total number of unique drivers for each penalty point amount from 1 point to 12 or more points and the total number of drivers accumulating penalty points, by county. The properties include name, TotalP1, TotalP2...Total12, and TotalDrivers.

PointsByOffenceType model view: This model view was implemented for use in calculations for 'Penalty Points by Offence Type' statistics to hold the total number of penalty points accumulated by offence type, the total number of drivers committing each offence type and the offence type itself. The properties include name, totalDrivers, and totalPoints.

**Code First Migrations**

Currently the application does not utilise **Code First Migrations.** These are used to implement changes to a production database when model changes occur, and at the same time, preserving the historic data residing within the database. Where the model changes are such that it is not possible to retain the data, the data can be dumped/downloaded to an external file. The table can be changed via the code migration and the historic data can then be uploaded.

Our project is currently using a development technique to drop and recreate the database (from the home controller) when model changes occur; by un-commenting/re-commenting the lines (in green) below:-

OOPenaltyPointsContext db = new OOPenaltyPointsContext();

databaseExists = db.Database.Exists();

if (databaseExists)

{

// for development purposes only

//db.Database.Delete();

//db.Database.Create();

}

else

{

db.Database.Create();

}

**We would expect to use Code First Migrations in a commercial environment.**

# Implementation

Whilst doing this project we had to decide how Implementation of particular OO concepts would occur. These included database, tables and relationships, unit tests, crud, search, reports, UI, navigation, data transfer, validation, security, authorisation, authentication.  
  
As mentioned in the introduction we were inheriting a relational database structure that was built for Oracle, and re-engineering its structure to fit the object oriented world using C#. As we were also inheriting an existing database structure we would be employing a model that was proven to work. By using 5-Tier Architecture Design, which enabled us to focus on designing and implementing data models and implementing the Data Access Layer to store application data to, and retrieve application data from, a SQL Server database.  
  
We did 2 unit tests on the ListedOffenceDAL and also the SummonsDetailDAL. These passed with flying colours.  
  
When developing the application CRUD which refers to all of the major functions that are implemented in relational database applications was very important to us, as it is very relevant at the user interface level of our application. We ended up having it in 2 places Listed Offences and also Driver Offences (which included Driver, Vehicle and Offence information).

Validation is also very important as we want to make sure the end user enters the correct data type and does not leave fields empty.

For our security we had to make sure that certain features were restricted to users who had specific clearance. For example anyone is able to view listed offences but to access the CRUD functionality you had to have super level clearance.

Authorisation and authentication is very important as we needed to make sure that data belonging to the general public could not be compromised. By having a secure login on our application we could restrict data usage to users with specific roles e.g. Admin or Super Users.  
**Standard functions**

Legal Offence Interface:

CRUD functionality was build for listed offences. The process flows from view to controller to business logic layer and finally to data access layer, and vice versa. The Create and Read actions are standard for MVC design, however, the Edit and Delete actions had to be implemented to prevent the actual updating or deletion of existing data-store records. This is because an offence type is an object of a driver offence. Deleting the record permanently means it is no longer available to driver offence record. This cannot be allowed. Regarding the Edit action, if the existing offence type record is updated then all existing driver offences referencing that offence type use the updated record which no longer reflects the penalty points and fixed charge fines applied in the past. Instead, for both Edit and Delete actions a status property is set to 'Deleted' allowing existing driver offences to use the original offence type record, and preventing new offences from using it. The Edit action includes taking a copy of the original offence type record, applying the changes and creating a new offence type, that then becomes available to new driver offences. The Index action, at the business logic layer, implements a foreach loop construct inside which a conditional statement is used to check the status of each offence, ignoring all records marked 'Deleted' and returning all others to the presentation layer.

Driver Offences Interface:

The driver offence is made up of objects from four tables i.e. driver offence, driver details, vehicle details and offence type tables. The driver offence object holds a reference to the offence type, driver detail and vehicle detail objects. A DriverOffenceUI model has been implemented at the business logic layer to forward the combined data from all 4 objects, as a simple format, to the presentation layer. Driver offence data passed from the presentation layer is reformatted at the business logic layer into a driver offence object before being passed to the data access layer.

CRUD functionality required implementation beyond that of MVC design. At the business logic layer the Create action had to create an record for each of the driver details and vehicle details if they did not exist already in each of the corresponding tables, as well as creating a record for the driver offence in the driver offence table. If the vehicle details or driver details exists (the offence type will always exist because it is pulled from listed offences data store which is legal data) then the Create action updates the driver offence with a reference to the existing objects. The Create action also allows the updating of vehicle details or driver details but does not allow changes to the drivers license number or the vehicles registration number. The Edit action updates the details of a driver offence including driver details and vehicle details but it does not allow the update of the offence description, driver license number or vehicle registration number. This lock-in resulted from using the code:

db.Entry(driveroffence.DriverDetail).State = EntityState.Unchanged;

db.Entry(driveroffence.VehicleDetails).State = EntityState.Unchanged;

db.Entry(driveroffence.ListedOffence).State = EntityState.Unchanged;

in the CreateDriverOffence action in the data access layer. The code was implemented as a solution to stop the duplication of records in the driver detail, vehicle detail and offence type tables when records are created or edited. Dropdown lists for driver details properties status and county, and for driver offence property status are hardcoded.

The Read action displays the data formatted using the DriverOffenceUI model view. Similar to the listed offences a status field is used to flag a driver offence record as deleted instead of deleting the driver offence permanently from the data store. The reason for this is that statistics are calculated based on driver offence data over a prolonged period. Permanently deleting records would result in inaccurate and incorrect calculations. The Index action, at the business logic layer, implements a foreach construct inside which a conditional statement is used to check the status of each driver offence, ignoring all records marked 'Deleted' and returning all others to the presentation layer.

Search features:

Searching is perform on the driver offence records. There are three search keys which include the driver license number (for looking up driver details), vehicle registration (for looking up vehicle details) and offence description (for looking up offence type). Each of these search keys is provided as a drop down list contain the corresponding data field from the data store (this is for demo purposes generally). Each search key also contains a '(None)' value. The search works by selecting 1, 2 or all 3 search keys and clicking on 'Search' button. The search feature is implemented at the business logic layer and searches all driver offences records held in the data store. Inside a foreach loop construct a conditional statement is used to check if the status of each driver offence ignoring all records marked 'Deleted'.

Court Summons:

The Court Summons has Index and Read actions only. These records are generated automatically by running Daily Tasks.

Service Oriented Architecture:

Listed offences has been implement as a service.

**Reporting requirements**

Statistics features:

There are 3 statistical views all of which are calculated as a result of the user selecting a statistical view from the dropdown list and clicking 'Generate Statistics' button. Model views are used for all statistics either to pass objects by value (struct) or pass objects by reference (class). The foreach construct and the if..else and switch conditional statements are used to manipulate the data contained in the driver offences to return valuable statistics. All records from the driver offences table are iterated through.

**Administration functions:**

Daily Tasks features:

There are 5 daily tasks to be run each updating the status of driver offences based on business rules e.g. a new offence status is updated to penalty notification, a penalty notification is updated to 28 day notification after 28 days of unpaid fixed-fine charges, a 28 day notification is updated to a court summons after 56 days of unpaid fixed-fine charges, driver offences with paid fixed-fine charges are updated to deleted after 3 years, and drivers accumulating 12 or more penalty points are updated to disqualified. Again all driver offence records are iterated. The foreach construct and the if..else and switch conditional statements are used to verify which driver offences statuses to update. The business logic layer takes care of all the processing and forwards updated records to data access layer to saving to the data store. Viewbag is used to return a message to the presentation layer to display in a view. The message contains the details of all updated driver offences and drivers.

**Authentication and Authorisation functions**

ASP.Net website administration tool was used to create three roles (i.e. Super, Admin and Worker) and set up three user accounts (for example purposes) with each account assigned a specific role. Inside the presentation layer, that is the DriverOffenceController and ListedOffenceController the following piece of code [Authorize(Roles = "Super, Worker")] was placed above selected functions to keep unauthorized users out and allow access for authorized users. Although anyone can register with the site access is granted to certain functions and views on the basis of the role the user account is assigned through the administration tool.

# Design patterns and architectural patterns

[ASP.NET](http://asp.net/) MVC provided a pattern-based approach to building our application. We had full control over markup and were able to create a separation of concerns in a very clean, orderly way. With the MVC architectural pattern (Model -View - Controller) the entire application is divided into three main components, the model, view and controller. We added a business logic layer and combined the views and controller into a presentation layer. The pattern proved to be very suited to the Agile Software Development approach we were taking. [ASP.Net](http://asp.net/) MVC provided a development environment enabling us to build the application much quicker than we expected.  The framework provided a lightweight, highly testable framework which we used to test the data access layer methods for Listed Offences and Court Summons. Another beneficial feature of the framework was that it integrated [ASP.Net](http://asp.net/) features like membership based authentication. We used ASP.NETs administration tool to set up user accounts, roles and permissions. The MVC model allowed us to separate input logic, business logic and UI logic and provided a loose coupling between them. The pattern specified where each type of logic should be contained in the application, for example, the UI logic belongs to view, business logic belongs to model and the input logic belongs to the controller. The pattern helped us manage complexity. We were able to focus on a single aspect of the application at a time e.g. views or business logic for a particular piece of functionality. Because there is a separation of concerns we each were able to focus on developing different parts of the application without interfering with each other. That said, collaboration between team members was vital.

Using N-Tier Architecture approach for application development was a requirement fated to us as part of the project scope. The .Net framework provides great support for building applications using N-Tier Architecture. With N-Tier Architecture our entire application was divided into separate parts or layers. Each layer performed a specific task e.g. data access, business logic and user interfaces. Our application had 5distint layers i.e. database, data access layer, business logic layer, presentation layer and views.

We coded views as UI templates to define how application data would be displayed to, and entered by end users. They gave us a way to provide an interface to end users to access the application's functionality e.g. for CRUD functionality for 'listed offences', 'driver offences' and 'court summons', and to provide an interface for end-users to run 'daily task', 'statistics' and 'searches' functions.

We coded the presentation layer (PL) to deal with the application UIs. For 'listed offences' and 'driver offences' and 'summons' controller files make the decision about which view to call to display application data to end-users or to get data from end-users or provide an interface to application functions such as 'search', 'daily tasks' and 'statistics', in response to user requests.

We coded the business logic layer (BLL) to take responsibility for the processing of application data associated with 'driver offences' and 'listed offences' e.g. CRUD functionality on driver offences, statistics on driver offences, searches on driver offences, notification updates generated from daily tasks, processing of vehicles for search function, and processing of drivers for search function. The BLL ensures correct and accurate application data is passed to the presentation layer, and correct and accurate end-user data is passed to the data access layer.

We coded the data access layer (DAL) to take responsible for storing end-user data to, and retrieving application data from, the database for 'listed offences' and 'driver offences'.

We used SQL Server 2008 as the data store for all application data.

The process through our 5-Tier Architecture works as follows:

* The end-user wants to edit a driver offence and requests the data via a view.
* The DAL retrieves the data and passes a driver offence object to the BLL.
* The BLL receives the driver offence object, formats it as a simple model view and forwards it to the PL.
* The PL receives the driver offence model view and decides to call a view to display the driver offence data to the end-user.
* The end-user changes the data and initiates an Edit action.
* The PL validates the data. If valid the PL formats the data as a view model and passes it to the BLL.
* The BLL reformats the data as a driver offence object and hands it to the DAL.
* The DAL updates the database.

When we first started coded it seem unnecessary to divide responsibility across some many layers. However, using a 5-tier architecture provides the following benefits:

* tight coupling between UIs, business processes and the database is removed.
* changes to the database and data access methods will not affect the presentation layer, business logic layer or views.
* table and column names are removed from client side code.
* The presentation layer and business layer do not know where data comes from as the location is transparent.
* Although more time consuming and resource consuming at the start, it is easier to modify or extend the application without interfering with client side.

 DB context is itself a Data Transfer Object (or DTO) which is a design pattern used to transfer data between application layers or subsystems. This design pattern is used used with data access objects to retrieve data from the database. DTO behaviour is limited to getter and setter methods for its data.

The n-tier Software Architecture is a based on an MVC design Pattern

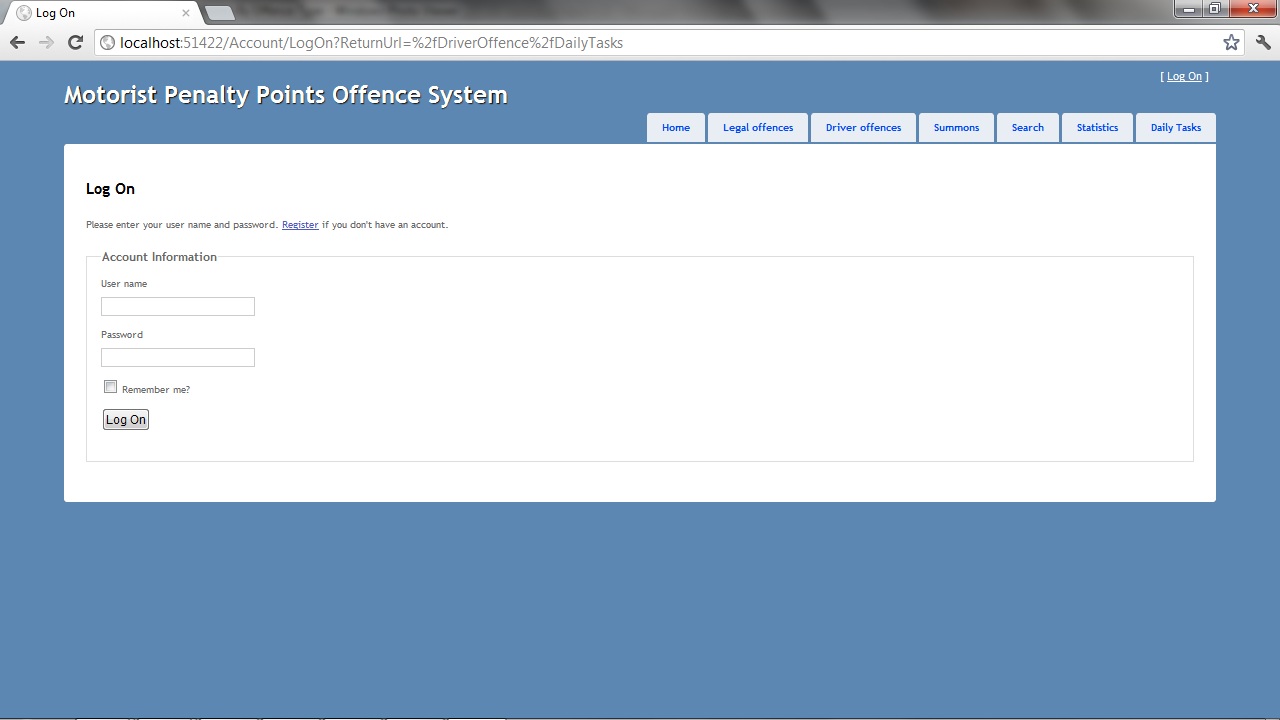
# Cross-cutting Concerns

Security is a feature of our application that spans across a number of layers e.g. listed offence controller, driver offence controller and daily tasks. The Windows Logon security uses ‘Cross-Cutting’ IOC annotation to secure individual component functionality. Authorization and authentication of users is vital because the application stores personal data belonging to the public. To protect this data users are assigned roles and must use their assigned username and password credentials to log into the application.  Depending on the role a user is assigned, the user may have full, limited or no access to certain parts of application.

In release version 2.0 of the Motorist Penalty Points Application we will be implementing exception handling strategy, a communication strategy to allow interaction between components across layers e.g. messaging system, and a logging strategy to prevent users from denying their actions and protect against repudiation threats.

# Security

The application has 3 different types of users on the system:  
1 Super role -access to everything (claire/password9)  
2 Admin role - access to daily tasks/listed offences/stats/search (ronan/password9 )  
3 Worker role - access to driver offences/stats/search (anthony/password9)  
  
If you log in using the usernames and passwords listed above you can see what each of the 3 users is able to do. The 3 users mimic typical users of the application. You will see screenshots of the application showing how it works and how much a user can access without logging on in the appendices.



If you click on the daily tasks tab you will see as above that you are asked to log in.

When you log in you also have an option to register as a user. If you do this you will have limited functionality. To get proper access to the system you will have to ask admin to set this up for you. To set up proper access to the system please follow the following steps:  
  
1. Run application in browser,  
2. While running go back into visual studio and select project menu-asp.net configuration.  
3. Go into controller and above the methods you want to allow or disallow access to place the following line:

[Authorize(Roles = "Super, Admin, Worker")]

Each property defined in the models contains validation that is used by the view to make sure the end user enters the correct data type and does not leave fields empty.

# Configuration

Code First development did not demand the creation of a database. Instead models were created and the system used these models to create the database for us. One of the issues with this is that when the database schema changes the whole database has to be created meaning the data is lost. Best practises used when creating models were the field 'Name' to describe table columns and letting database generate auto generate the tables primary key. In releaseV2 we will use the field 'IsActive' to avoid physical deletion of records from the database so we do not lose any vital information if records get deleted accidently or on purpose.

The project solution created was a [ASP.NET](http://asp.net/) MVC3 web application, with the directory structure automatically created. Library Package manager was used to install the 'Entity Framework DLL' . A reference to the dll was then added to the application solution. Nuget Package Manager was used to download and install third party libraries into the .NET framework.

The OOPenaltyPointsContext model was created by inheriting the DBContext class. Model classes were created contain properties to represent the field of a database table. Each table has a separate model class defined. Each property contains a validation logic which is implemented at the view ensuring the end user enters the correct data. To represent relationships between driving offence and vehicle details, driver offence and driver details, and driver offence and offence type, iCollections<entity> will be used to define these relationships in release V2.

The application was compiled to add model references to the solution. From this point forward the data access layer, business logic layer, controllers and views were created with their respective folders. Language-Integrated Query (LINQ) is used to bridge the gap between objects and data by writing queries against collections of objects.

# Scalability

The ability of our application to function well when changed in size or volume to meet user needs is limited. Initially M:1 relationships were set between driver offence and driver details, and driver offence and vehicle details. When it came to building the business logic layer an iTracker error prevented the application from running. The error related to an initial object storing a reference to a second object which stored a reference to the initial object. The relationships were set up using a public virtual iList so it appeared that the driver offence store a reference to a driver detail which store a reference to the original driver offence resulting in the iTracker error. At the time we resolved the issue by removing the relationships and allowing the driver offence to contain a driver object and a vehicle object. Since we have learned that this issue could be resolved using an iCollection which will be implemented in release V2. As a direct result of the initial fix, all functions in the business logic layer that relating to searches, statistics, daily tasks etc retrieve all records from the driver offence and then use conditional statements to eliminate records. The more records held in the data-store the slower the performance of the application due to the retrieval of large record-sets, and iteration of these sets for key functionality.

Once the issue above is resolved in release V2, using [ASP.NET](http://asp.net/) MVC3.5 will allow the application to function well in a rescaled environment e.g. moving the application from a smaller to a larger operating system application performance will improve and the application will be able to handle more users.

The use of a n-tier architectural model allows logic to be restructured or changed, or the underlying data-store and data access logic to be changed without interfering with the rest of the application, as long as method definitions are not changed.

# Testing Approach

**DAL Unit Testing**

**The creation of a formal Unit Test package**

A professional implementation of a Test Driven Development ‘testing rig’, would probably implement a combination

* **Repository pattern** (which would separate the DAL layer from the physical data source (using ‘fakes’ and’ mock’s to represent the physical data layer) .
* A **factory pattern** would allow a configurable db context (DTO) to allow switching between two physical databases.(Production & Test)
* A Unit of Work design pattern might also be of benefit to ensure transactional integrity of complex OO objects when persisting them to the database.

When the DAL layers were created, an MVC-Dal approach to **shakedown** test via auto-generated & amended views was utilized. This ensured that basic database CRUD was operable. Also, we tested basic algorithms for scanning tables, some of which was incorporated (in part & at a later date) within the BLL layer.

As a proof of concept, to facilitate test case development and testing, we utilized **Microsoft. VisualStudio. TestTools.UnitTesting** and built and ran a set of test cases to test the DAL database access methods of the Listed Offences class and it’s database CRUD operations

*Note: Tried a local Unit Test OOPenaltyPointsContextTest to create a test version of the main project database. This did create a 2nd test,but the test dbcontext could not be cross cut into main project (as this uses its own version dbcontext ,i.e. a factory pattern is required).So used the* ***main project*** *database as a target for the* ***testing project.***

These rudimentary tests reinforced the notion that:

* TDD can achieve a greater degree of ‘code coverage’ testing.
* Re-orientates the development ‘mind-set’ to

- re-think alternatives

- Work to a higher level of quality coding practices.

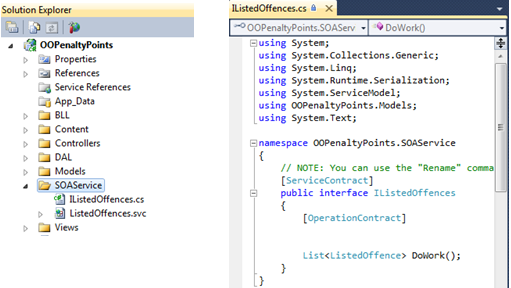
* Provides a repository of test data that can be re-used into the future

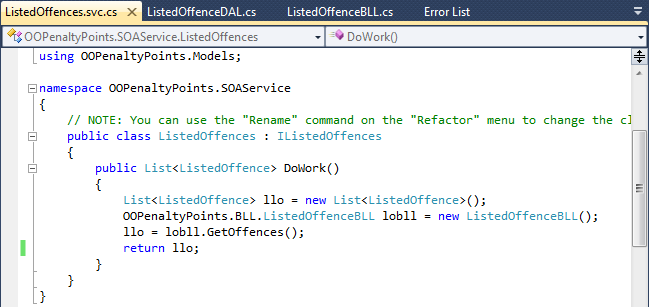
**Testing of business logic layer, presentation layer and views**

Implementing a design-test-design approach allowed developers to build-up application functionality feature by feature. Each feature was tested independently of the working application. It then was integrated into the working application and the entire application was retested. Additional tests were performed against business and functions goals and modifications made were goals were not met. The end product produced from each iteration of the design-test-design approach was a working application with increased functionality. The login feature was tested by logging into the application under all 3 user accounts and testing that each user could only access application functionality and data assigned to their roles.

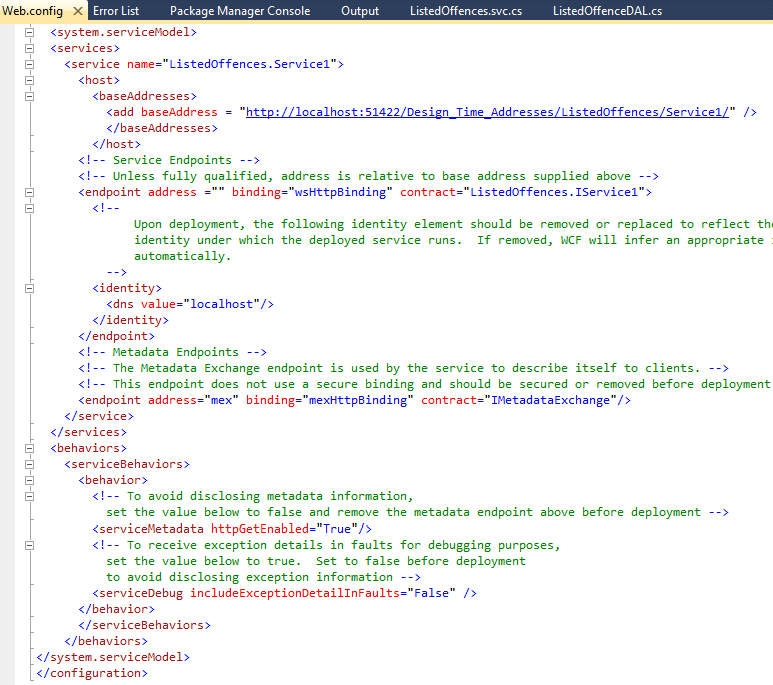
# Other Relevant Features

**Exposing the Listed Offence Table as an external SOA service**

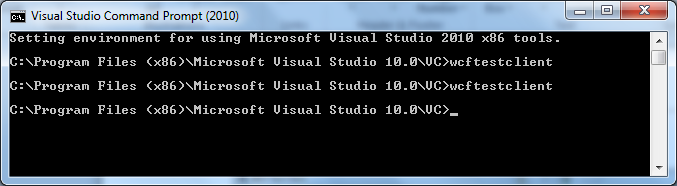


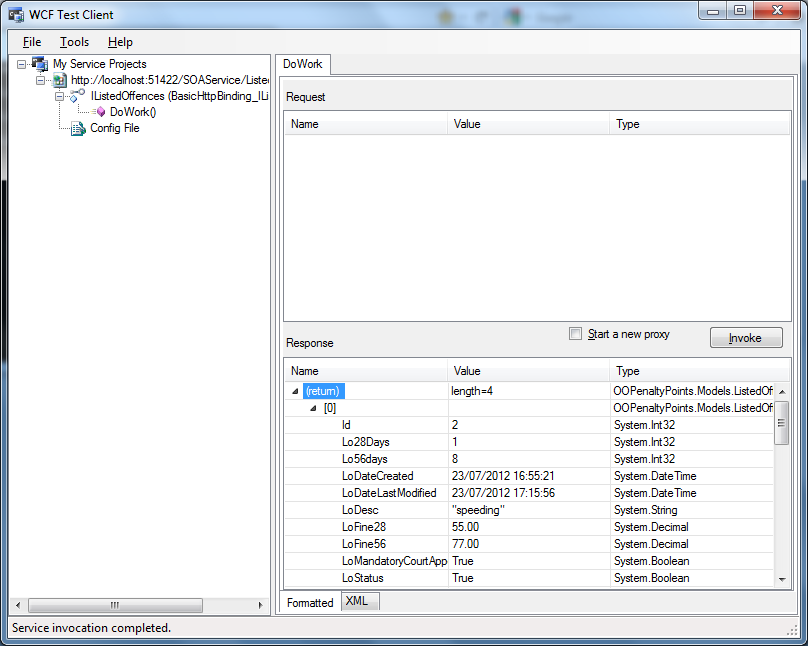


Web.config. Note :local host ,also, start & end points



**Test SOA**





# Discussion

Could not understand the need for N-Tier Architecture until we started to implement the business logic layer, presentation layer and views. Being able to focus on certain aspects of the application and the ability to change the logic inside the methods without effecting other layers were beneficial and time saving. Although it is very cumbersome and time consuming at the start of the project the N-Tier Architecture did make implementation easier as the project progressed.

Using an Agile software development methodology and a design-test-design approach helped to progress the application development. These approaches forced look at the application as simple pieces of functionality rather than a entire application. Implementing a piece of functionality, testing it and building upon it when the next piece of functionality allowed building of the entire application swiftly and speedily. The use of a iterative process helped build the application in a orderly, practically fashion allowing flexibility when required.

Initially the division of workload was very difficult. Not having a good understanding of the architecture, development environment, code first development, test driven development etc meant that the focus was on building a database to house all application data and defining the relationships between tables. This is a relational database approach which is, of course, not the focus in code first development. In code first development models are created first, then a context model is created by inheriting from DBContext class which creates the database from the models. The idea of the database not being of primary importance was hard to grasp at the start. It makes sense now that the database is there to house database but that's all we need to know about it.

Eventually, we came to understand what the tier architecture model was and with assistance divided the workload into two parts i.e. business logic and presentation layers, versus database and data access layer.

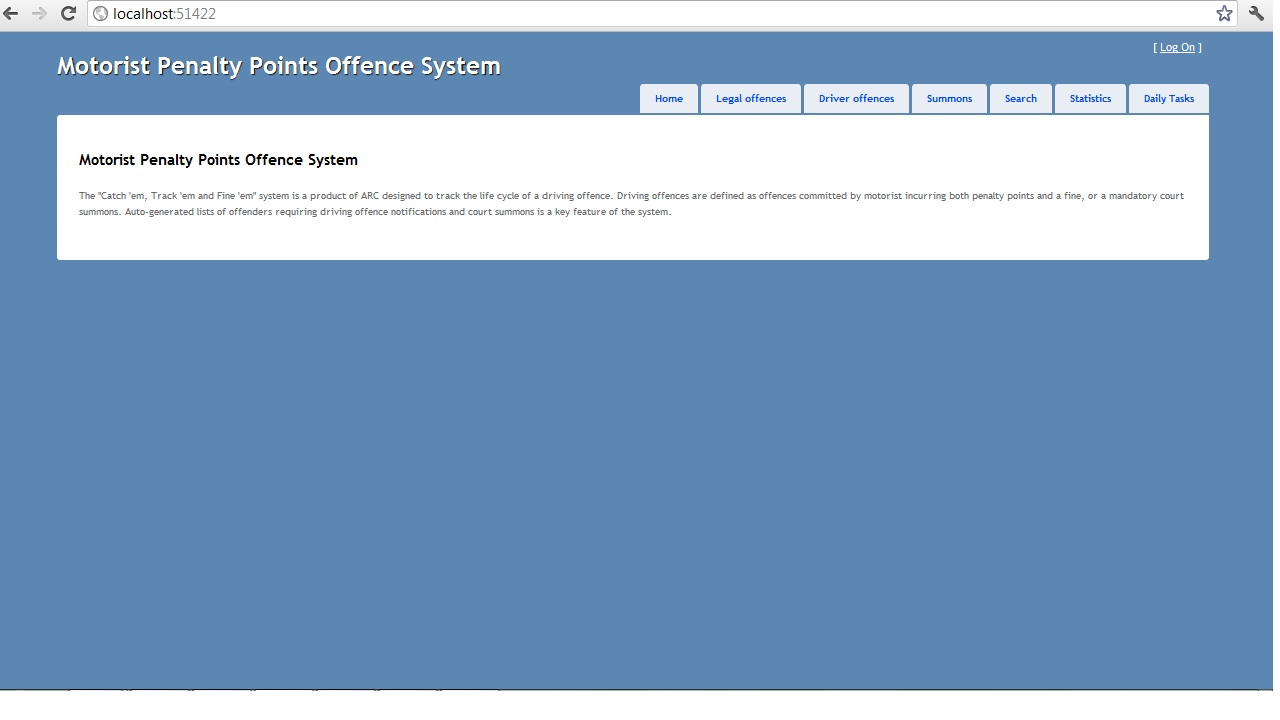
The use of objects was not as difficult to grasp as we expected. It made sense when implementing the business logic layer, presentation layers and views that objects were a very neat way of working with data as they represented real world objects. Model views were a nice way to forward data from the business logic layer to the presentation layer especially when data was being pulled from several different tables. It was a little bit cumbersome having to format the data at the business logic layer but the benefits in time savings, smaller data packages to transmit and less waste of data bits is worth the effort.

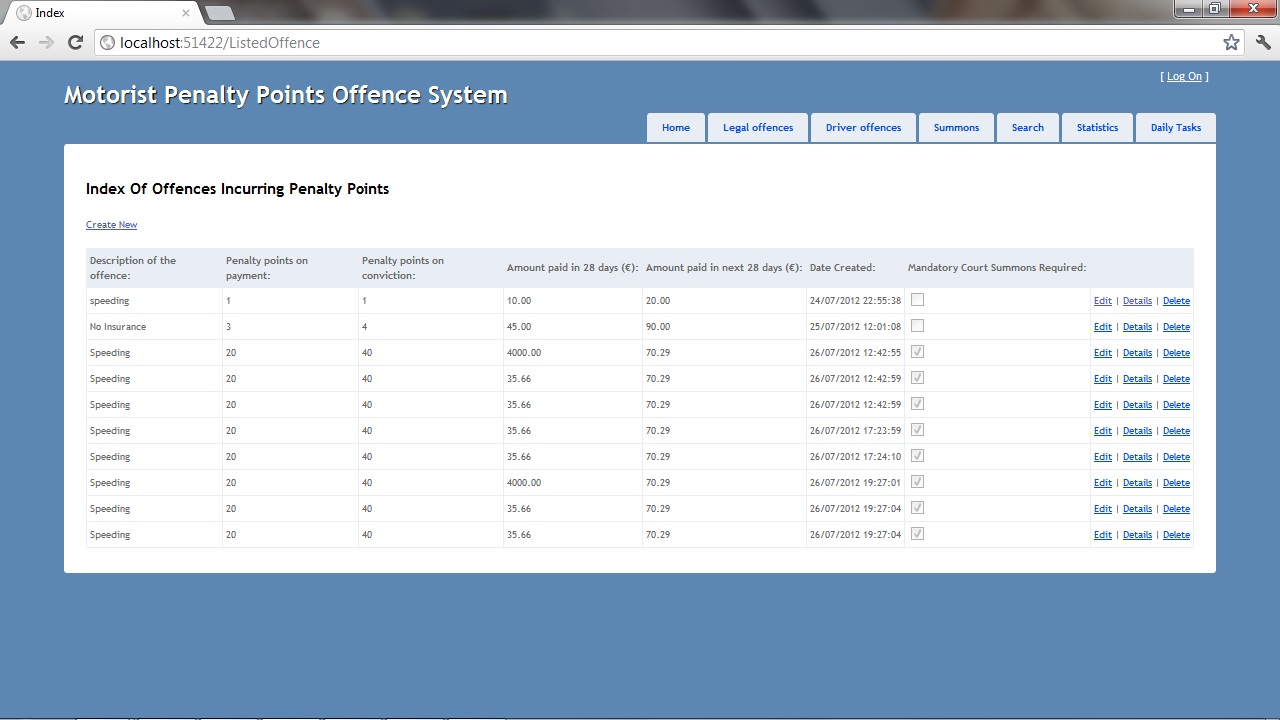
The solution achieved baseline functionality for all functional requirements. Because of having to iterate through all records in the driver offence database for the majority of methods in the business logic layer the application is not scalable at present because when the number of records stores in the database grows the performance of the application will immediately decrease and possible cause the system to crash out. This will all be fixed in the release V2.0. The use of full table iterations was a direct result of our lack of experience with the development environment. We have found the solution i.e. using iCollection to define relationships between tables. At the moment we have implemented conditional statements to ignore records tagged as 'deleted' etc as a means of improving performance. It is a short term solution only.

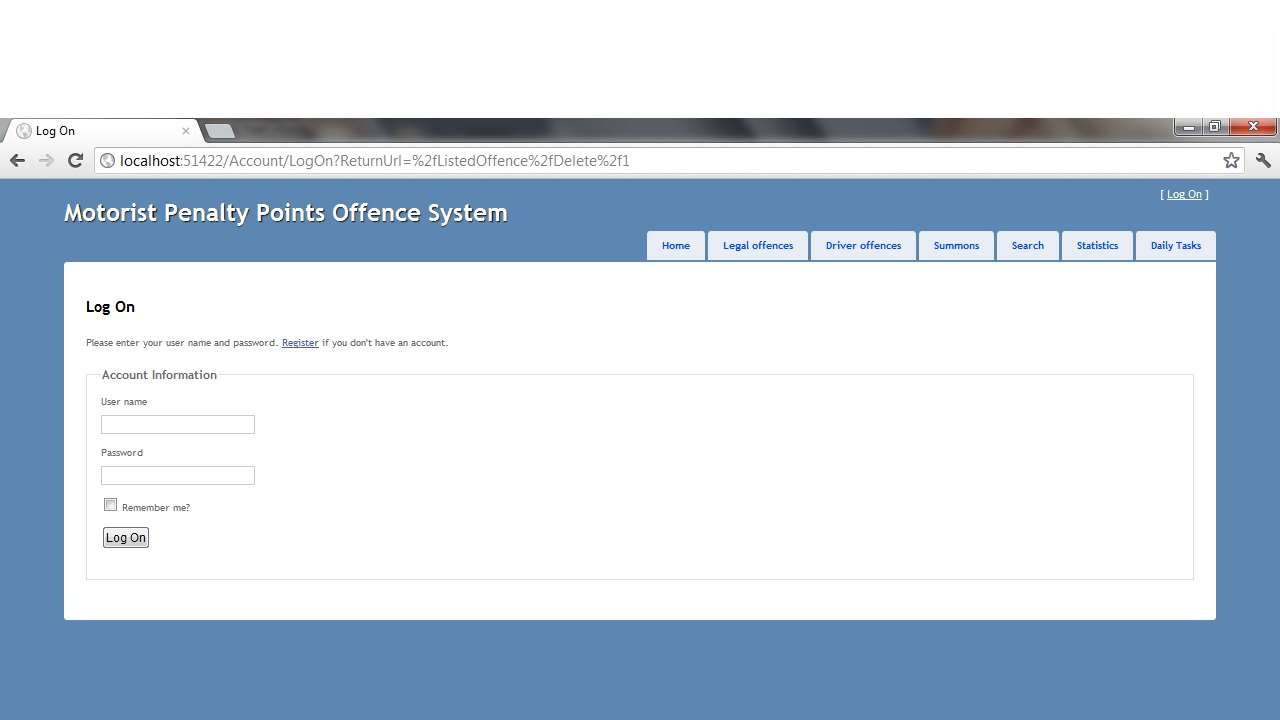
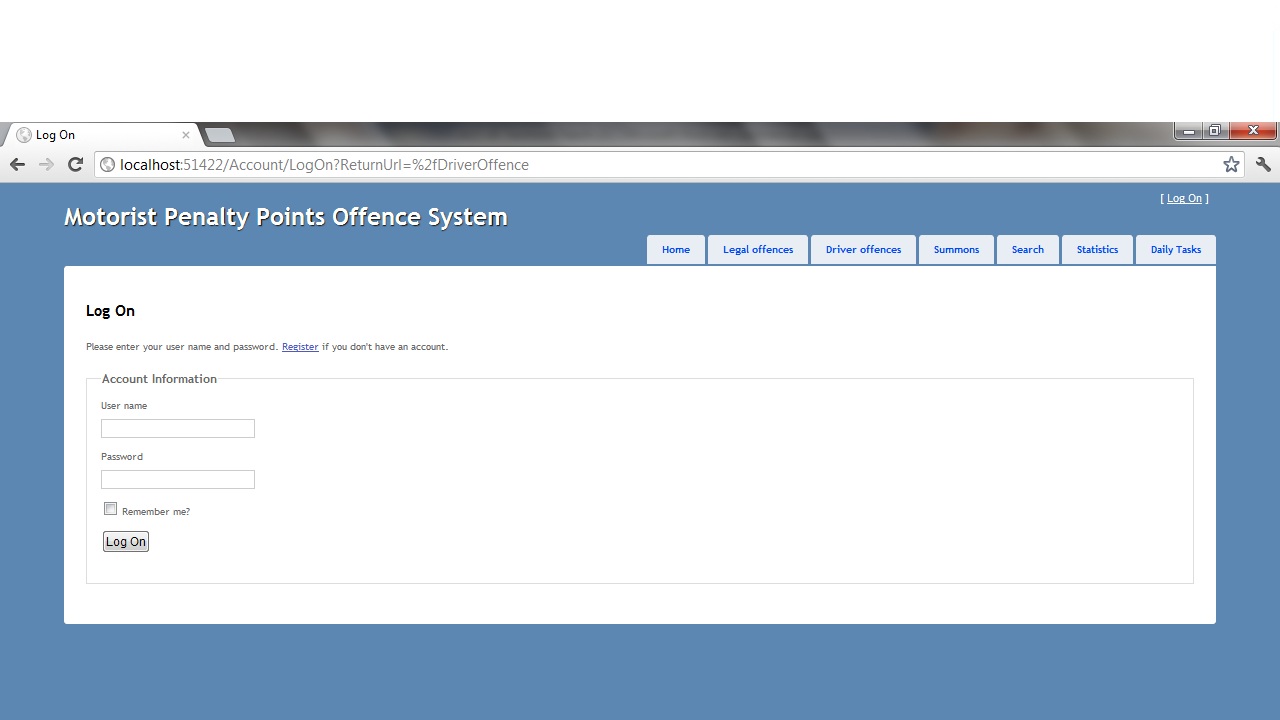
Additional functionality that needs to be added is the ability to archive records after an extended period of time, using time periods to search records, for statistical analysis, daily tasks and displaying of records sets. This should greatly improve performance and records available would be more accurate to the time period.  
  
The only real issue we had is that we could not implement Keys on OOModel.

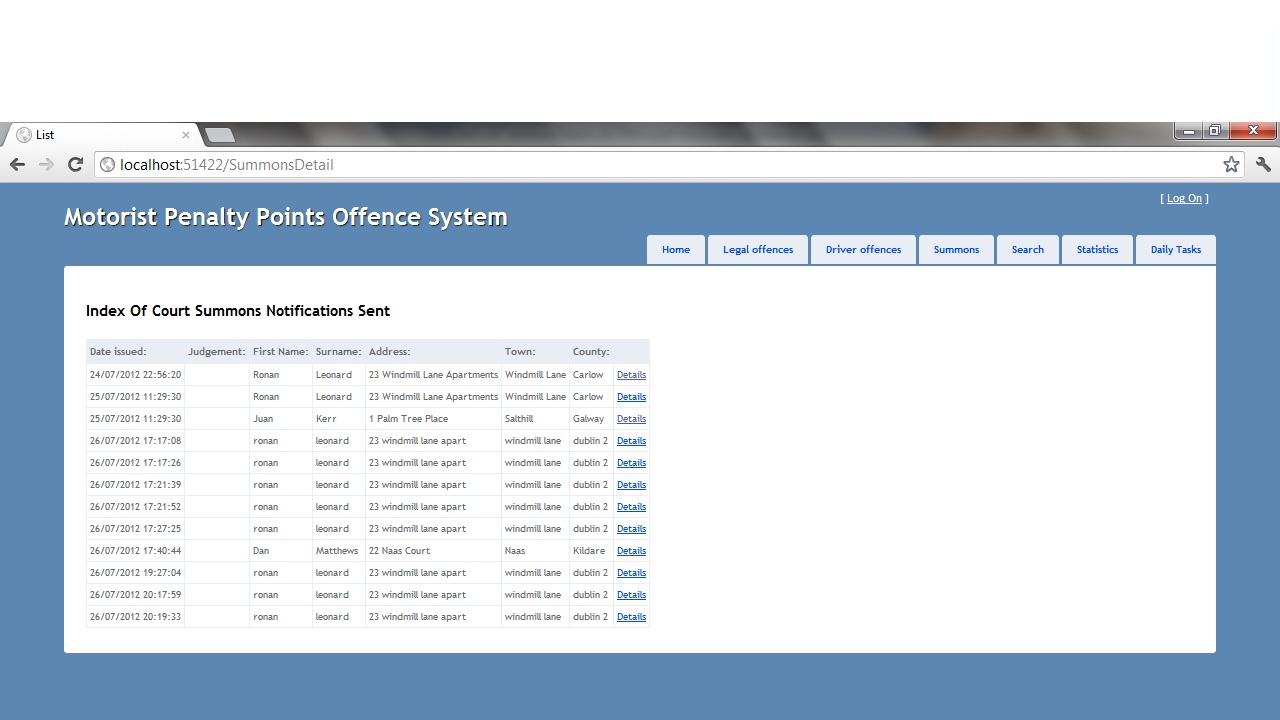
Overall the project was very enjoyable and a huge learning experience. There was two different points of view on the team i.e. an object focus member and a data focused member. This made a very interesting team dynamic which actually fitted natural into the structure of the application i.e. object focus is very much the business logic and presentation layer, whereas data focus is very much the database and data access layer.

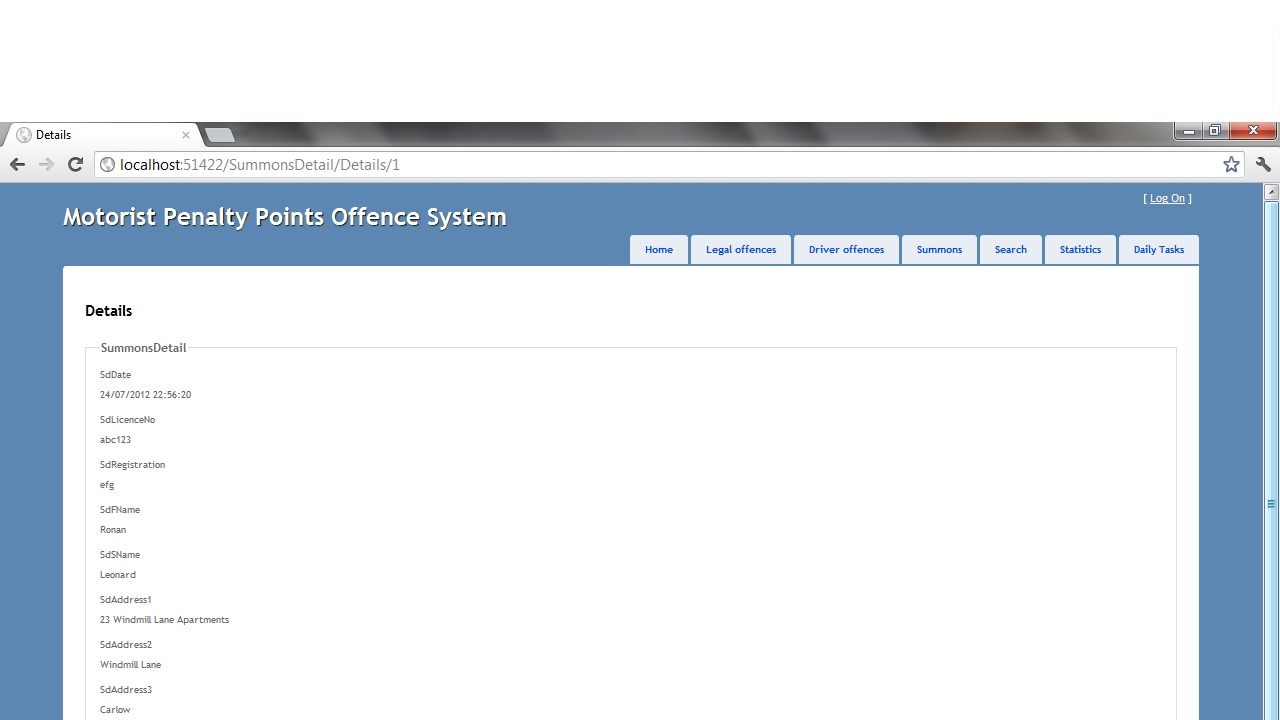
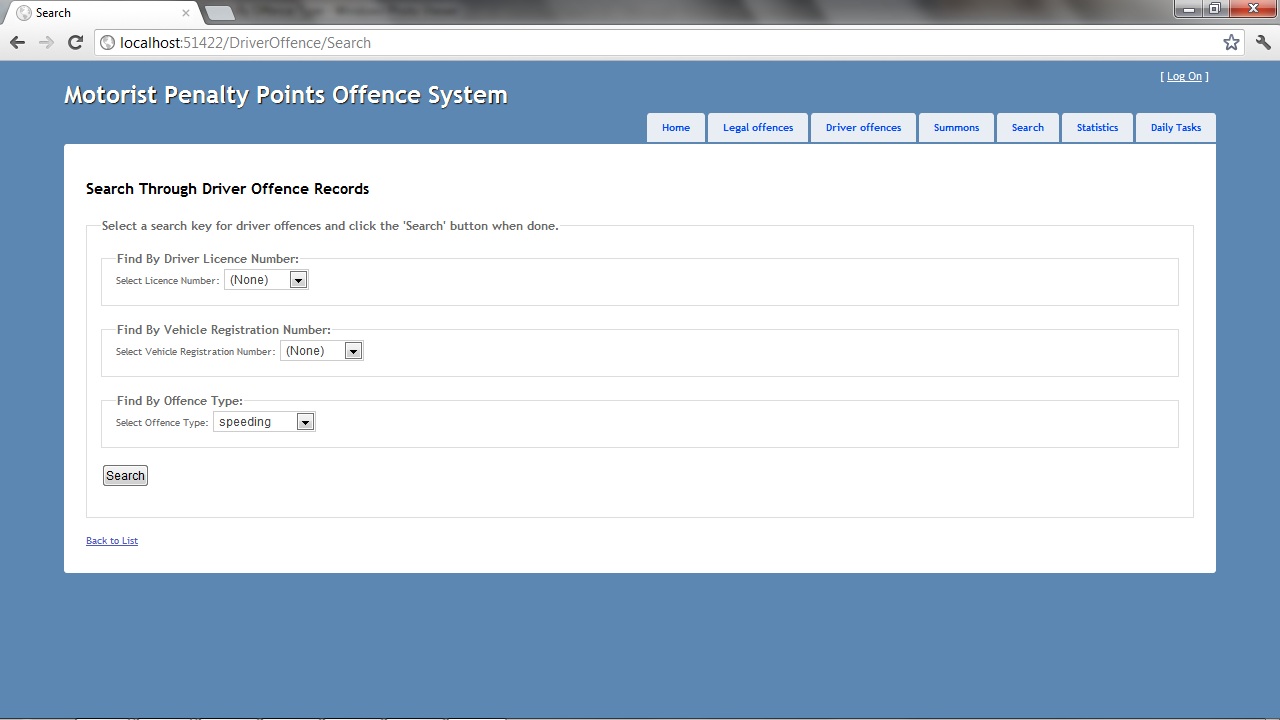
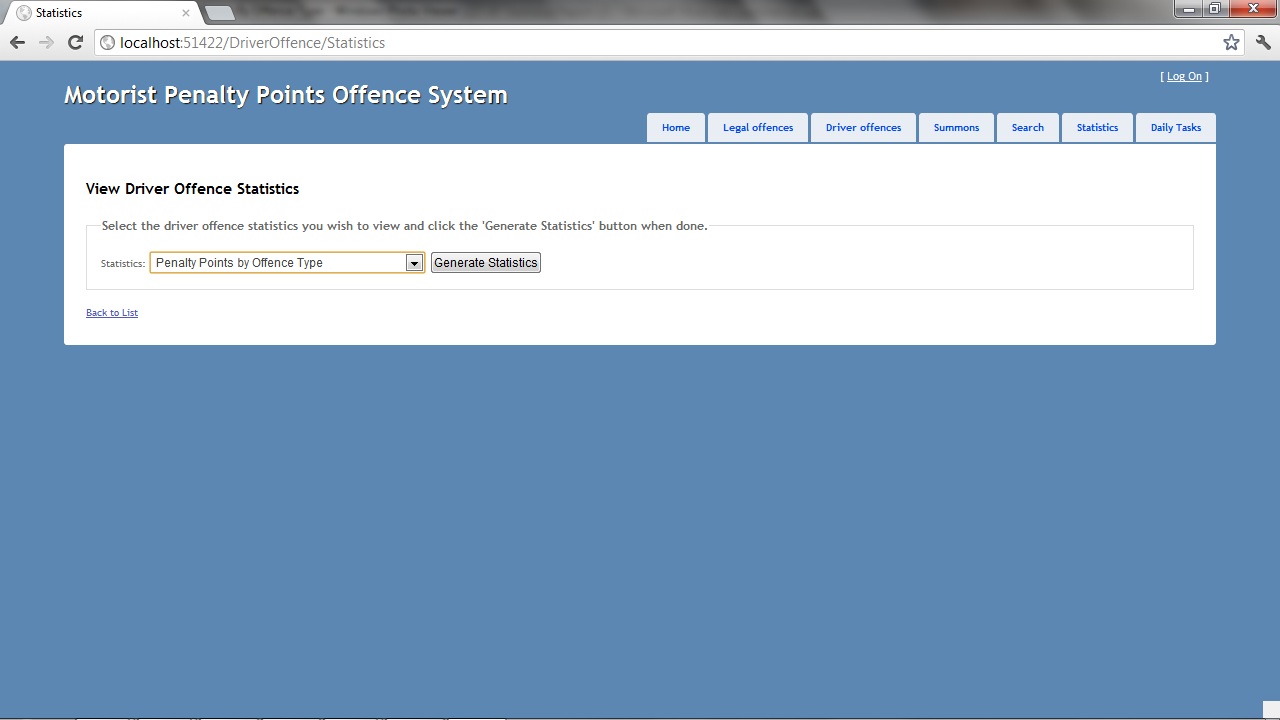
# Appendices



This is the home page that appears when the application is first loaded and also when you are not logged in.  
  
 When you click on the Legal Offences Tab you get a brief overview of all the Legal Offences in the database. You can also get the same information for one specific offence.

 If you want to create, edit or delete an offence you will be asked for your logon details as you can see by the screen shot above.  
  


If you click on the Driver Offences tab you will be asked for your login details. If you can’t provide this then you can’t view all the Driver Offences records.  
  


If you click on the summons tab you will get a list like the one above. You can see names, dates, addresses and also judgments details.   
  
 If you click on the details link beside each record, you can also see Licence No and Car Registration details.  
  
  
If you click on the search tab you can search by Driving Licence Number, Vehicle Registration Number, and it brings back offence description, offence date, location, status, licence no, first name, surname, licence status and vehicle registration. If you click on the edit, details or delete link by each record you will be asked to log in.  
  
 If you click on the search tab you will see as above that you can generate statistics for penalty points by offence type, penalty points incurred per driver per county, numbers of drivers by licence type.