National College of Ireland

PGDCLOUD

2011/2012

BlueTeam	
Des Harrold	11106506
James Larmon	11106590
Paul Kelly	11106565

PlanApp

Semester 3 Project for

Enterprise Frameworks

The technical report documents the following areas:

1. Background research and investigations

The project goal was to develop a bridge enterprise application that would be readily acceptable to an IT department operating as part of a risk-averse enterprise which was putting pressure on for exponential capacity growth with a significant reduction in cost of operation. An early approach investigated was the use of publicly available API's such as the Dublin City Council Maps API. Locally relevant API's are few and of little added value at present. Consequently it was decided to make use of an easily understood Cloud facility and to combine it with a locally hosted processing application. Large storage capacity was an obvious candidate for exploitation and there was already an organisation using such an approach in the Irish market i.e. IMRO.

The specific project aims developed into an application which would

- Collect planning application details and store them on a locally hosted database
- Associated drawings and documents would be uploaded to the Microsoft Azure Cloud infrastructure using Azure's Binary Large Object facility
- The links to the drawings could be emailed to all relevant parties within the planning authority and would be available for public inspection as well.
- The ability to manage the application details and associated drawings would also be available.

2. Project Plan

The Project roles/streams were assigned as follows:

Role	Assigned to:
Requirements Analysis	James Larmon
Application Logic	Des Harrold
UI design and test	Paul Kelly

The three streams would operate in parallel from week2 to week7 meeting as a project group once a week. Week8 and week9 were targeted to combine the outputs from each stream into a working application and week10 was for documentation and presentation preparation.

Familiarity with the Visual Studio development environment took most of the first 3 weeks of the project's timescale but it has enhanced development productivity in the later stages. The original project was commenced with ASP.Net MVC 3.5 and then upgraded to MVC 4. One unwelcome discovery is that Visual Studio's Designer View is no longer supported for the Razor View Engine which meant having to edit CSS files to achieve a basic UI. The detailed work breakdown was as follows:

Task	Responsible	Date Assigned	Date Due
Requirements definition & refinement	JL	Jun 2	ongoing
Model(s) description	DH, JL	Jun 9	Jun 16
Technology profile decision	DH, JL, PK	Jun 9	Jun 16
Prototype production and coding task split	DH	Jun 9	Jun 23
UI development	PK	Jun 16	Jul 7
Local DB development	JL	Jun 9	Jun 23
Azure Blob handling and test	DH	Jun 9	Jun 23
Code integration	DH	Jul 7	Jul 14
Unit Test development (Repository pattern)	JL, DH	Jun 30	Jul 7
Project report	PK, JL, DH	Jul 7	Jul 20
GitHub upload	DH	Jun 2	ongoing
GitHub final upload	DH		Jul 27

3. Software development methodology employed

AGILE METHODOLOGY:

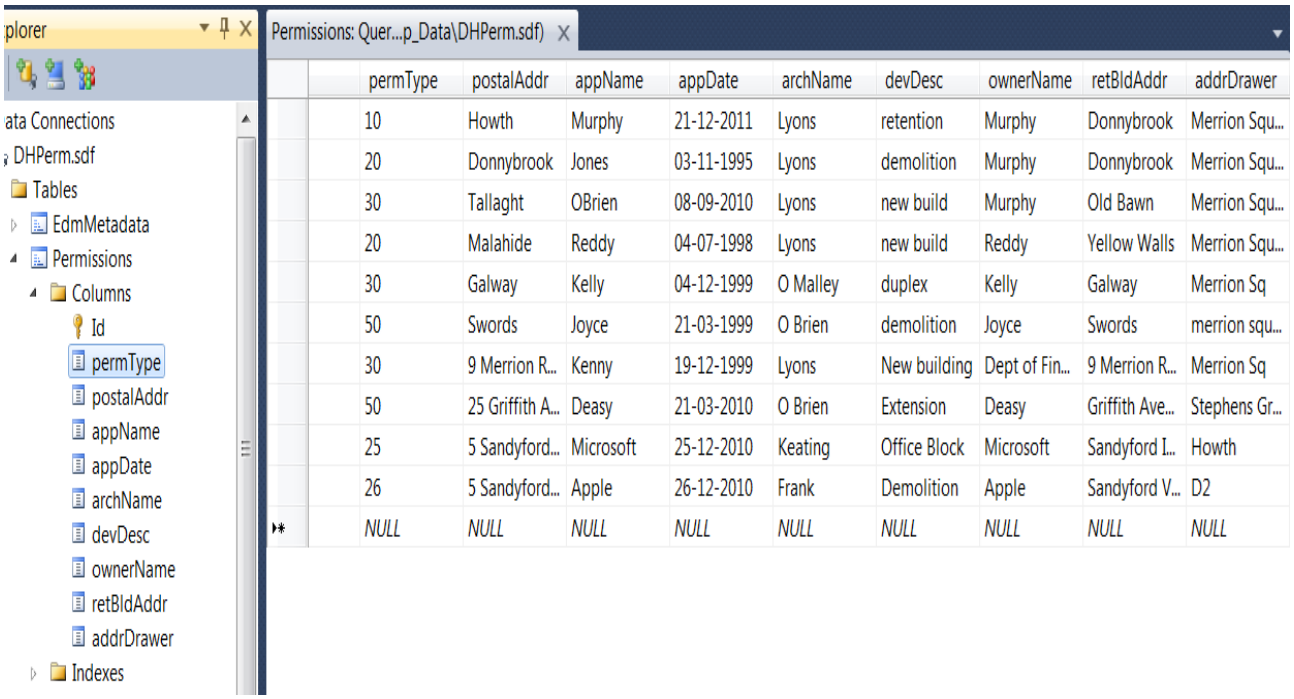
- MODEL DRIVEN APPLICATION
- ACTIVE RECORD ARCHITECTURE
- AZURE STORAGE FOR FILES/MAPS

MODELLING THE DATABASE:

- ADDED THE VALIDATION TO THE MODEL
- PERSISTED OUR MODEL ONTO THE DATABASE
- ENTITY FRAMEWORK GENERATES CALLS TO THE DATABASE (See Appendix 1)

SEARCHING THE DATABASE:

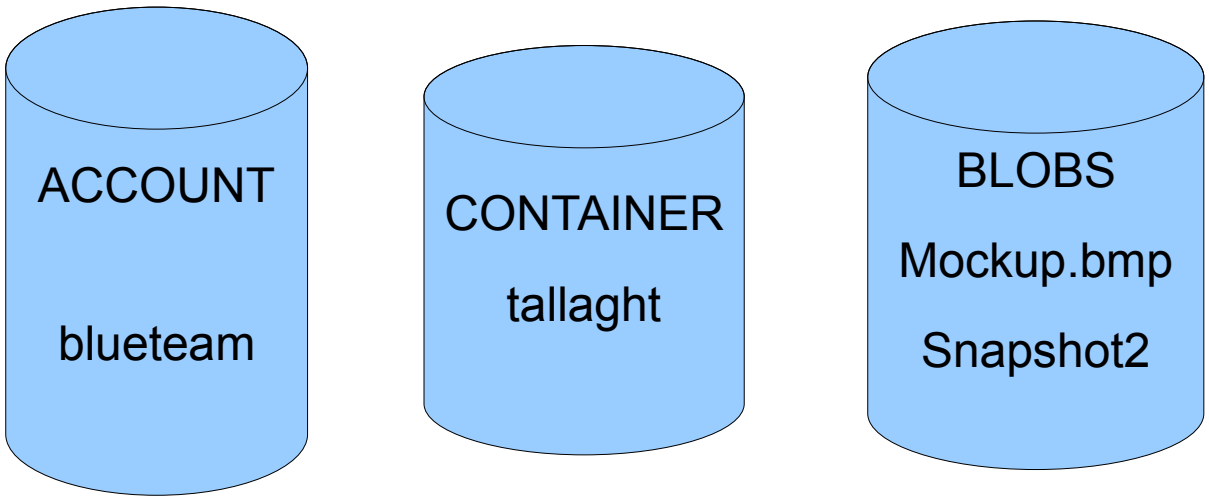
The database can be searched by the applicant's postal address for which there is a dropdown box, or by applicant name. The PermissionController sets the search parameters for the SearchIndex View. (See Appendix 2)



Database view in visual studio showing user records

The database is stored in the App_Data folder and is called DHPerm.sdf

WINDOWS AZURE BLOB STORAGE CONCEPT FOR A SAMPLE URL:



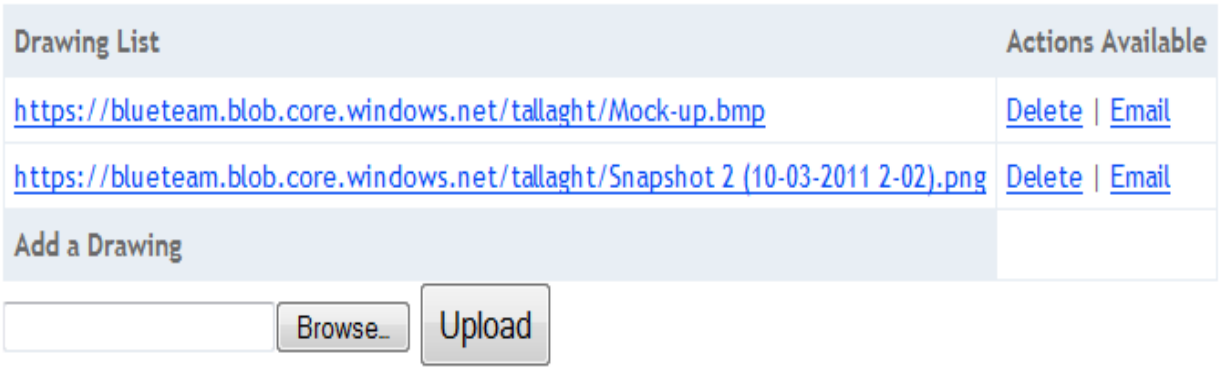
The code for accessing the Storage Account is detailed in Appendix 3.

Blob Storage:
In order to store images of maps and plans associated with the project, a Storage Account was created in Windows Azure. The Storage Account for this application is called 'blueteam'. The containers within this account store the files as blobs and can be configured to expose data publicly to the world or privately for internal inspection.

The images can be uploaded and retrieved from storage using HTTPS. For the purpose of this project the containers have been set up as public access. Individual blobs are stored inside a container. The container is created inside the DHBlobController.cs and has the following format:

DHBlob/Index/tallaght

Storage account - blueteam
Container - tallaght
Blobs - Mock-up.bmp
- Snapshot 2(10-03-2011 2-02).png



4. Requirements analysis

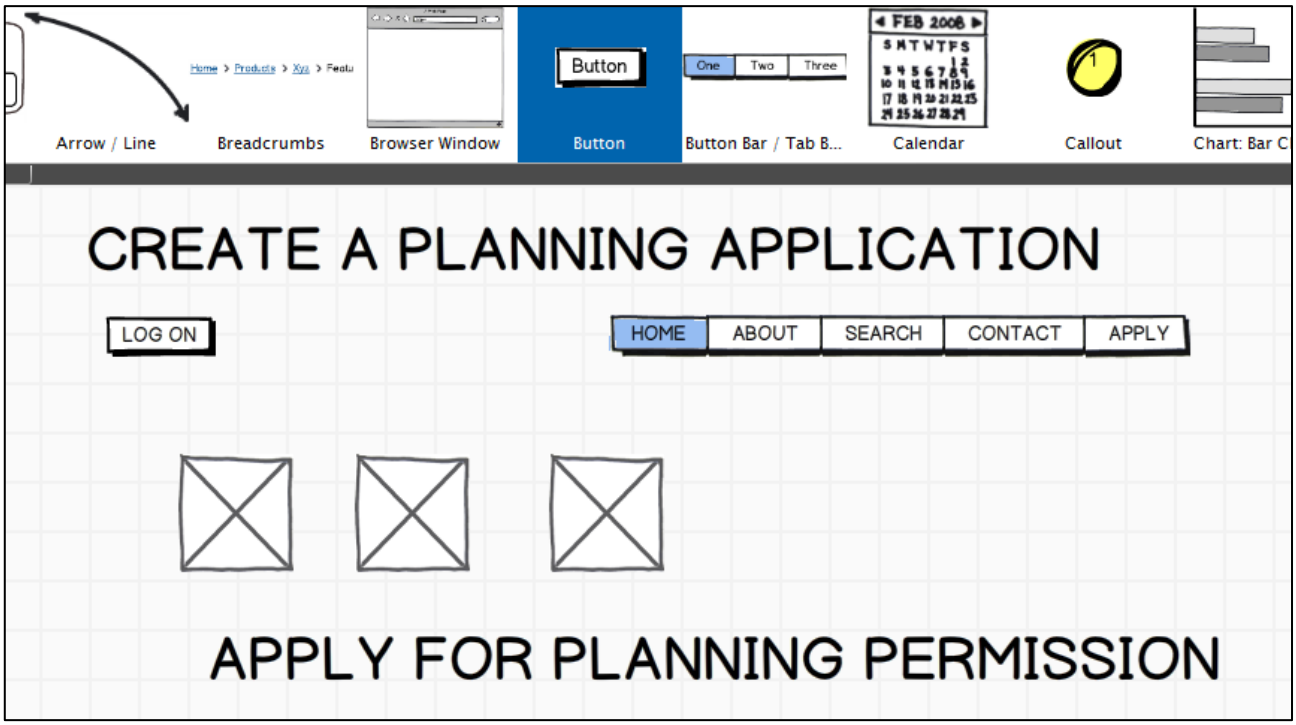
BUSINESS GOAL:

Provide a Planning Application system that will be used by a City Council to manage the requests by clients for various forms of planning permission.

The Application will hold the applicant's data and any maps and drawings pertaining to the property for which planning permission is being sought.

BUSINESS REQUIREMENTS:

– Council employees managing the application need to log on before processing any client transactions.
– Log on will identify which council employee handled the specific application.
– Create page enters new client details.
– Details page displays all details of a particular client on the database.
– SearchIndex page allows the database to be searched by Postal Address and Applicant Name.
– Records can be created, edited, deleted, and displayed.
– Link to maps and drawings held in storage.



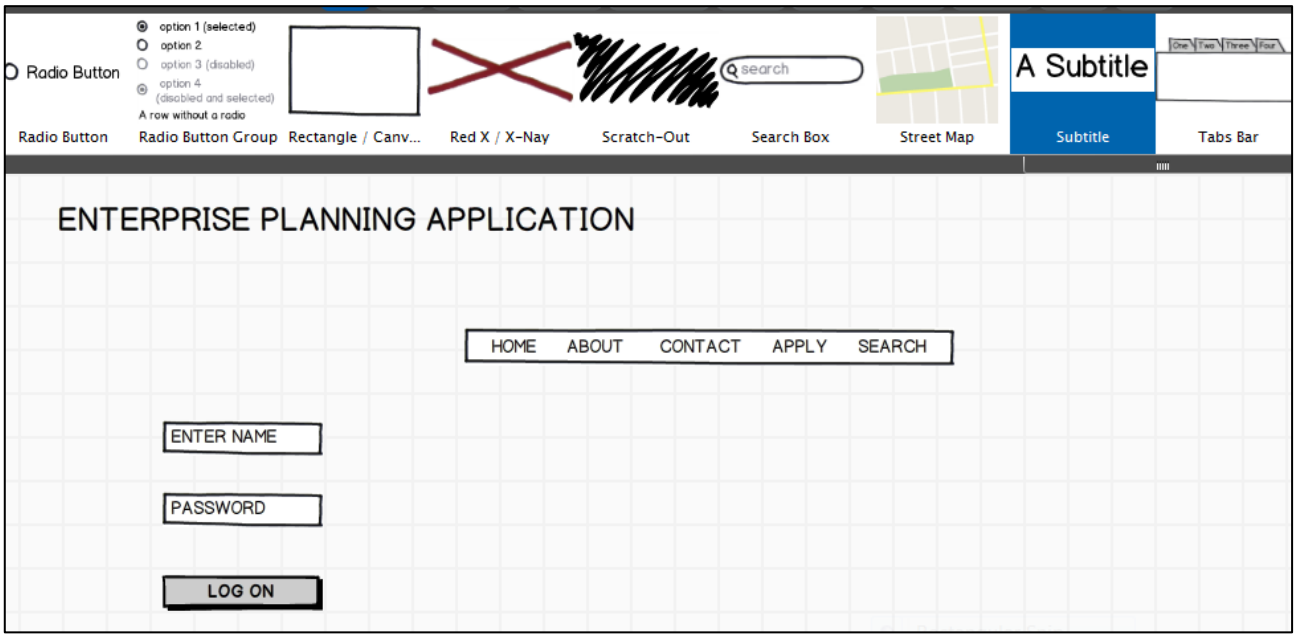
MOCKUP OF HOME PAGE OF APPLICATION

LOG ON BUTTON FOR ADMINISTATION STAFF IN COUNCIL

VARIOUS RELEVANT IMAGES

Business Requirement for staff to log on to system:

LOG ON PAGE FOR ADMINISTATION STAFF

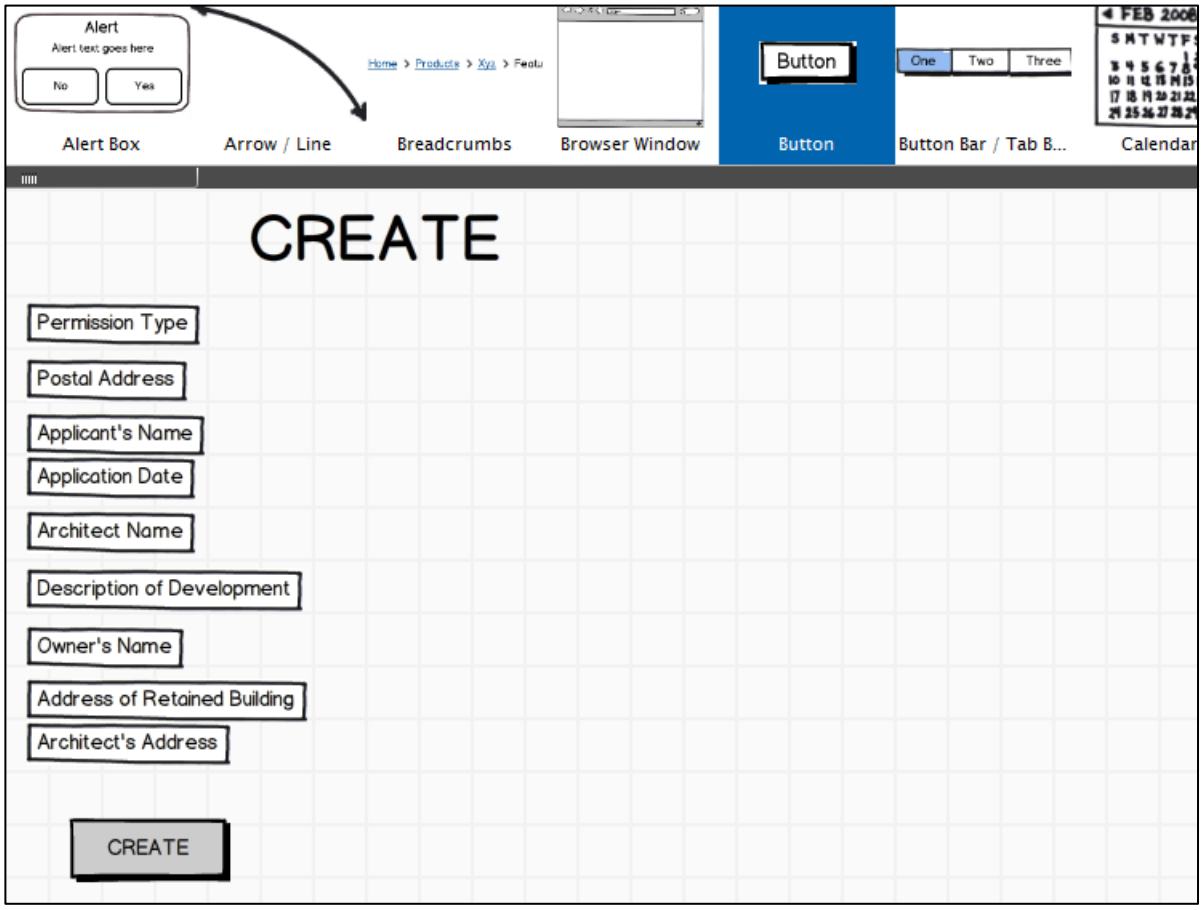


MOCKUP OF LOG-ON PAGE FOR ADMINISTATION STAFF

Business Requirement that applicant's details can be recorded:

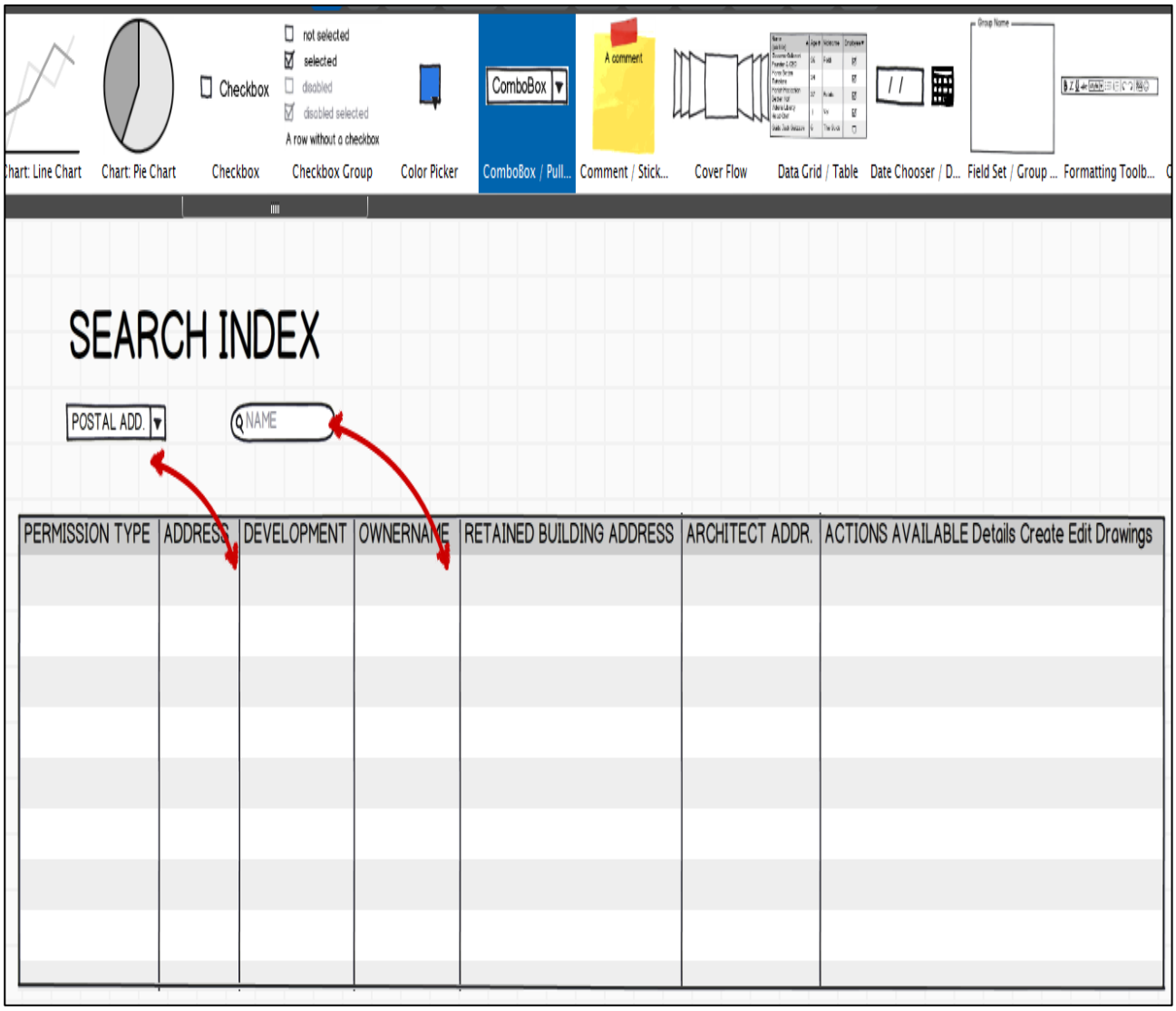
LOG ON PAGE LEADS TO PAGE WHERE THE APPLICANT'S DETAILS ARE RECORDED.

All relevant details are entered and Create button uploads them to the database.



MOCKUP OF CREATE VIEW TO ENTER DETAILS OF NEW CLIENT

Business requirement that database can be searched:

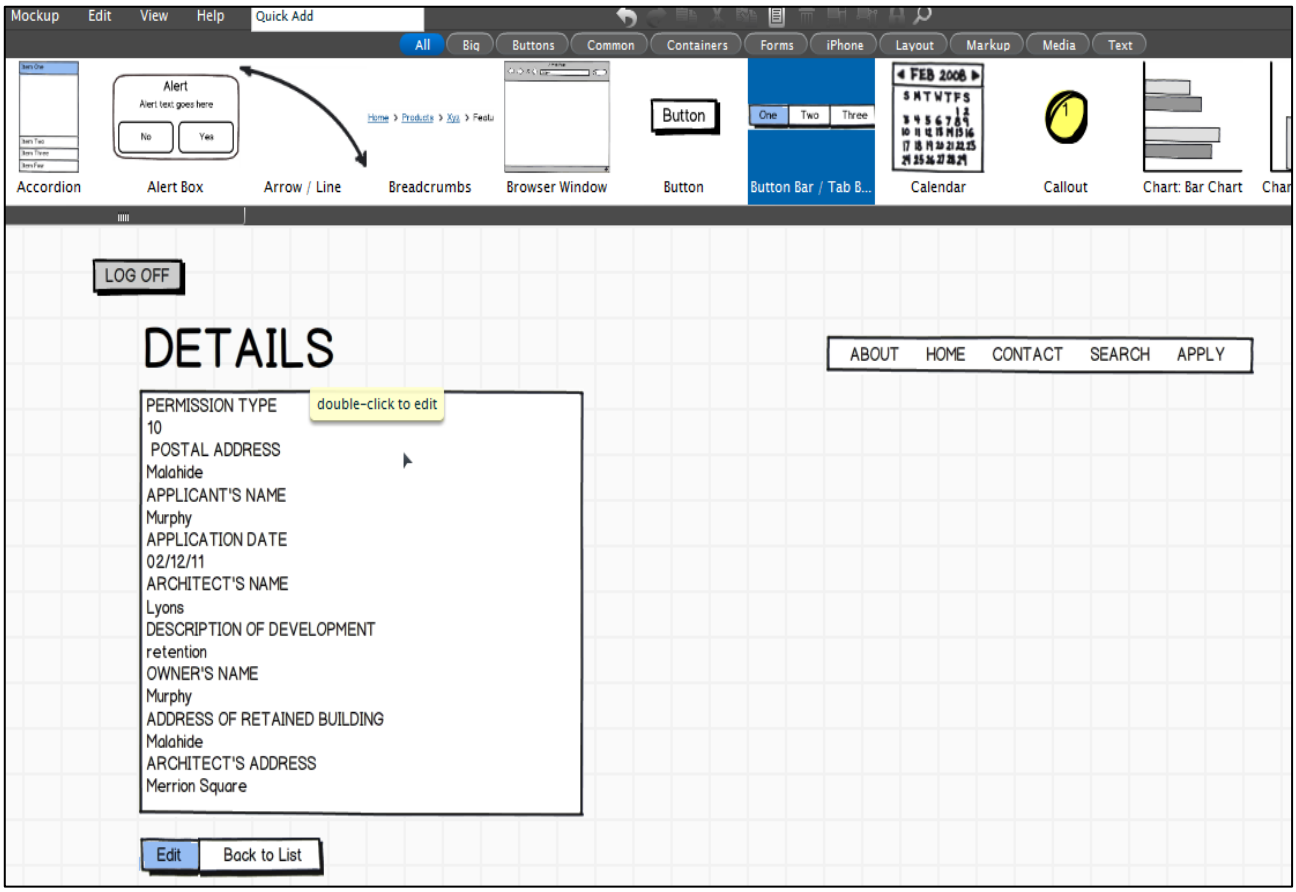


Mockup design of the SearchIndex View.

Dropdown menu to search database by postal address
Second box to search by owner's name
Links to allow administrator to:

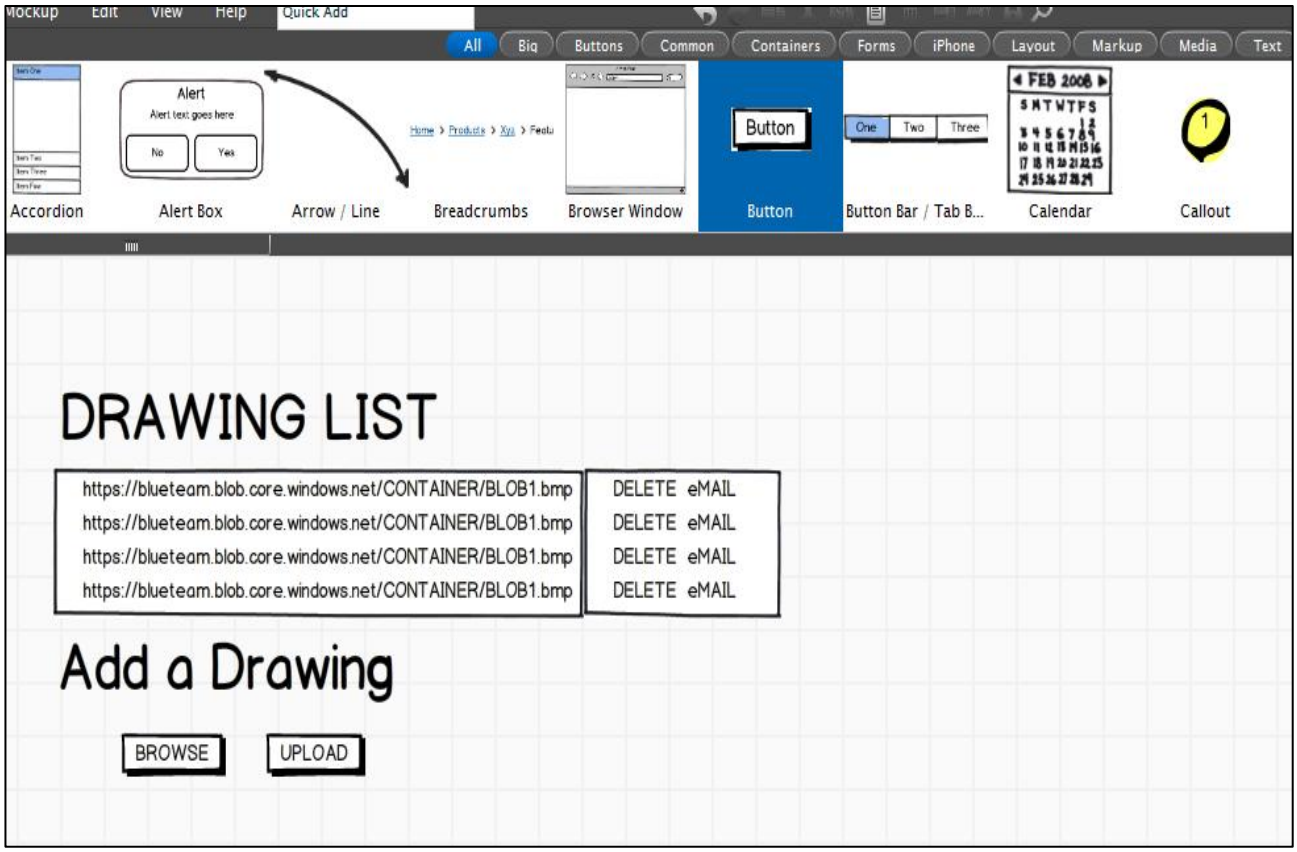
- edit
- details
- delete
- manage drawings

Business Requirement a client's details can be displayed:



MOCKUP OF DETAILS PAGE SHOWING ALL DETAILS OF A PARTICULAR CLIENT

Business Requirement to Manage Drawings:



MOCKUP OF PAGE DISPLAYING LIST OF MAPS AND DRAWINGS

Each drawing or map is stored in a container as a blob.

They are downloaded by clicking on relevant link.

A drawing can be uploaded to storage by browsing for it in the local Drawings folder and clicking the upload button.

5. Use cases

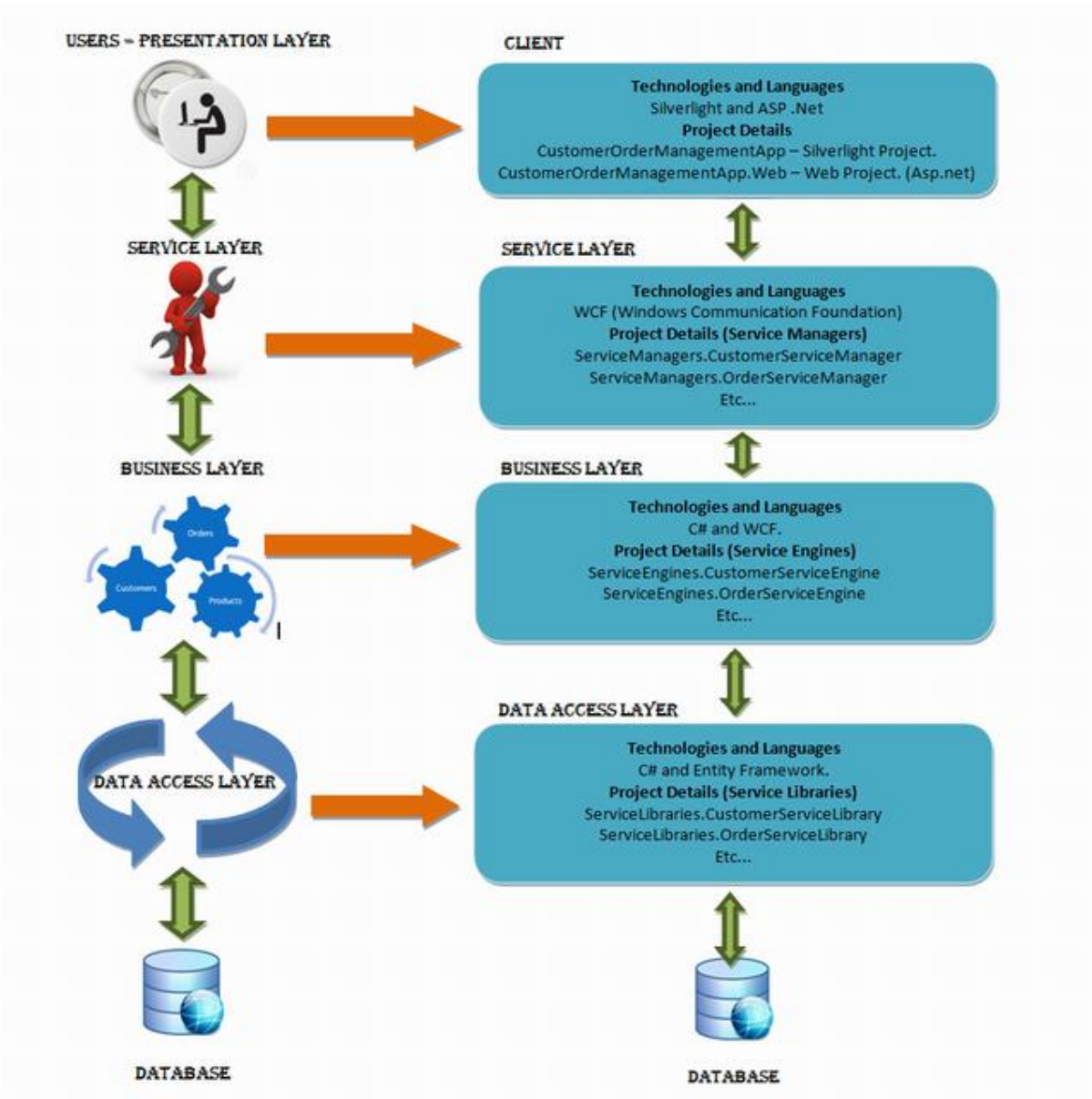
Title	**Creating a Planning application**
Description	User accesses the system and clicks the apply tab where they will fill out the planning application form and click create to complete their application.
Primary Actor	User requiring planning permission.
Preconditions	User must be registered and logged into system.
Postconditions	User's planning permission application is created and pending approval.
Main Success Scenario	1. User fills in required fields and clicks "create". 2. System displays confirmation message that application has been successfully created. 3. User can search for and view details of their application.
Extensions	1a. User leaves a field blank or inserts an illegal character - 1a1. System displays error message stating that all fields are required or certain characters not allowed here. - 1a2. User fills in incomplete field or deletes illegal characters and completes application process.

Title	**Uploading a drawing or map**
Description	User accesses the system and under the “search” tab they locate their own information and select manage drawings. They will use the “browse” button to locate the required drawings (in a drawings directory on the c: drive) on their computers and click the “upload” button to send these drawings.
Primary Actor	User in the application process.
Preconditions	User must be registered and logged into system. Drawings or Images must be stored in a Drawings directory on the user’s c: drive.
Postconditions	User’s drawings are uploaded successfully to Azure and can be viewed from system panel.
Main Success Scenario	1. User clicks “manage drawings” next to their application information and then clicks “browse“. 2. Required files or drawings are selected from Drawings folder and uploaded. 3. System stores images using Azure BLOB storage and user can view drawings or images by clicking on them in the drawing list.
Extensions	2a. User selects drawing from a location other than Drawings folder. - 2a1. System will not upload file or image.

Title	**System Maintenance**
Description	User needs to edit and get details of specific applications. Also, user needs to delete older planning applications and delete certain image files and maps.
Primary Actor	Administrators
Preconditions	User must be registered and logged in with administrator privileges.
Postconditions	User can perform all of the additional actions granted by additional privileges such as edit and delete.
Main Success Scenario	• Administrator logs in. • Accesses the additional links such as edit, details, and delete in the “Search” tab. • Performs the appropriate action by clicking on the specified link and clicks the button to confirm. • Changes have been made to system records.

6. Architecture/Design approach

Enterprise Application Architecture Model reference:

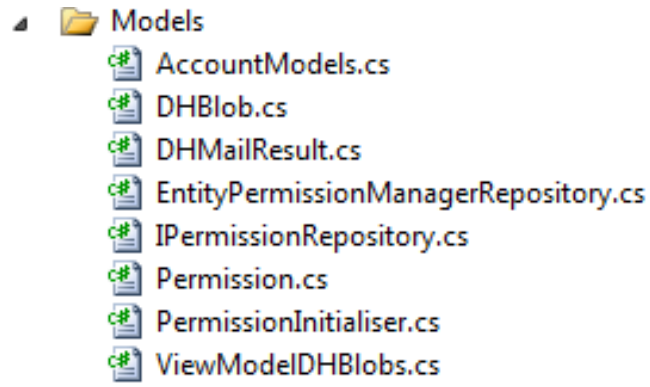


Program layers

Layers:	Planning application
User – presentation	Data input including scanned maps and building plans
Service layer	Storage using the address as the blob-container access key
Business layer	Submission process
Data access layer	Storage for public access

7. Models (Class Models / Data Models)

MODEL STRUCTURE FOR PLANNING PROJECT:



MODELS:

Permission.cs model

The Permission.cs class defines the model of a basic planning application which the database implements. A get and set method is defined for each attribute of the permission class. Each required field is also validated and will return a meaningful ErrorMessage in the event of an incorrect user entry. A full description is detailed in Appendix 1.

PermissionInitialiser.cs model

The PermissionInitialiser class is used to reinitialise the database with sample data in the event of a change in the table schema.

```
  public class PermissionInitialiser :
DropCreateDatabaseIfModelChanges<PermissionDBContext>
    {
        protected override void Seed(PermissionDBContext context)
        {
            var permissions = new List<Permission>
            {
                new Permission
                {
                    permType = 010,
                    postalAddr = "Tallaght",
                    appName = "Murphy",
                    appDate = "21-12-2011",
                    archName = "Lyons",
                    devDesc = "retention",
                    ownerName = "Murphy",
                    retBldAddr = "Donnybrook",
                    addrDrawer = "Merrion Square"
                },
               };
            permissions.ForEach(d => context.Permissions.Add(d));
        }
    }
}
```

DHBlob.cs model

DHBlob class models the get and set methods for the blob storage account.

```
public class DHBlob
    {
        public string blobURI { get; set; }

        public string blobContainer { get; set; }

        public string blobRef { get; set; }
    }
```

ViewModelDHBlobs.cs model

ViewModelDHBlobs class is iterated over to produce the listing of blobs in storage for the specific container.

```
public class ViewModelDHBlobs
    {
        public string id { get; set; }

        public List<DHBlob> Blobs = new List<DHBlob>();

    }
```

8. Implementation of particular OOP constructs

The following OOP constructs are used in the application code:

- Dynamic dispatch – when a method is invoked on an object, the object itself determines what code gets executed by looking up the method at run time in a table associated with the object.
- Object inheritance (or delegation)
- Open recursion – the late-bound variable this is used to allow a method defined in one class to invoke another method that is defined later.

Other OOP concepts used in the program code include:

- Object Classes
- Instances of classes
- Methods which act on the attached objects.
- Message passing
- Abstraction

Decoupling

Decoupling refers to interfaces that separate code modules from particular use cases, which increases code re-usability. The use of the Repository pattern is an example of this.

9. Design patterns and architectural patterns implemented in the application

The application is a classic Separated Presentation MVP Pattern which bridges locally held planning permission data with associated graphical data held in the Azure Cloud to take advantage of the Cloud's cost effective large storage capacity. The Cloud storage has other benefits of easy shared access with built-in backup features.

The application uses

1. The Active Record architectural pattern for the local Permission database as the domain entities map closely to tables in the data model and domain logic is mainly validation (e.g. Appendix 1 : Permission Model).

2. A ViewModel subset of the Domain Model to hold the Azure BLOB Container lists which is iterated over to produce the list of drawings held e.g.

```
public class ViewModelDHBlobs
  {
     public string id { get; set; }
     public List<DHBlob> Blobs = new List<DHBlob>();
  }
```

3. A Data Access Layer is used by the Repository pattern for the following reasons:

   - To facilitate business logic unit tests without a dependency on the external database.
   - To provide greater separation of concerns to aid maintainability.
   - To assist application evolution by having a more loosely coupled architecture (tests can be run after every change).
   - To facilitate a domain model approach.

   As part of the repository pattern the program creates interface for the data-access class. The repository interface is used to populate a mock repository for the unit tests of the controller methods.

10. How cross-cutting concerns have been handled

The key areas considered are listed below and were handled in the most pragmatic method consistent with the available development time:

- Authentication & Authorisation
  - ASP.Net MVC4 framework support is utilised to provide 2 higher levels of security
- Caching
  - Not an issue for the main code but an in-memory database copy is utilised for the unit tests
- Communication
  - The email service was upgraded from Webmail to System.Net mail during development
  - There are network file-size limitations that apply to the blob creation as it is done in a single write. For the current development only files < 4Mb (one block) are used.
- Configuration Management
  - There are two configuration details that are hard coded into the application:
    - The folder where the drawings to be uploaded are held (C:\Drawings)
    - The storage account id (in the PlanApp\Web.config file)
  - These details could in the next release be handled by a Dependency Injection
- Exception Management
  - All program generated error messages are self-explanatory
  - All other exceptions are recoverable
- Logging
  - Provided by the ASP.Net framework
- State Management
  - Provided by the ASP.Net framework
- Validation
  - Strict input validation is carried out to preserve the data integrity
  - Back end column level validation can be run periodically to ensure nothing has been inserted into the local database
  - Uploaded files are scanned by the sites virus checking software

11. Security of the application

Authentication and Authorization

The application makes use of the functionality provided by the .Net framework and uses the pre-built pages that allow a user to register on the system, log in and change their password. Forms authentication is used to provide a login page for the user. This is the most common form of authentication. The login form used has the logic to validate a user and a cookie will be created by the .Net framework on successful validation. This cookie will be checked for each time a user attempts to log in.

Our application has leveraged the framework and taken advantage of the Account Controller, Account Models, and Account Views provided.

Anonymous users can only access basic features of the system such as Home, About and Contact. Should they attempt to access other tabs in the system they will be asked to log in or register if they don't already have an account.

Authenticated or registered users can perform additional actions such as applying for planning permission through the provided online form, searching for details, manage drawings, uploading files and emailing files.

There is a need in this system to hide some of the CRUD action links from both anonymous and authenticated users as we do not want users being able to delete other user's application details, for example. To configure this we have used the Asp.net web site administration tool to create different roles for users. An administrator role has been created for a select number of users and only these users are authorized access to certain features such as edit, details and delete.

The system is set up with two separate views for authenticated and administrative users with these extra links (edit, details, delete) being hidden from authenticated users.

Validation

The project has followed the Open Web Application Security Project (OWASP, May 2009) recommendations on input data where feasible. This application uses a mix of both a white list approach and a blacklist approach to data validation.

All fields are validated and required to create a successful planning application. Certain fields require numbers, letters and correct date formats. Also, certain invalid characters (<, >, ", ', &, /) are not allowed. These special characters are rejected to prevent switching into any execution context, such as script, style, or event handlers. As part of the next release the use of hex entities as a substitute may be included. In addition to the 5 characters significant in XML (&, <, >, ", '), the forward slash is included as it helps to end an HTML entity.

Guarding against Threats

Our application guards against malicious users by using customized views to protect sensitive features of the application from users without the correct level of authorization. This application is Roles-based, leveraging the .Net framework and taking advantage of the site administration tool already provided.

We are aware of the threats caused by script injection, which is why all of our input fields are heavily validated in our Permission class. For instance, a user trying to enter a <script> tag into any of our input fields will be greeted with an error message.

12. Configuration of the application

The application is built around 3 main screens

1. Creation/Edit of a planning permission application
2. Management of the associated drawing data held remotely in the Cloud
3. Database search facilities (by address/container and by applicant name)

There are 3 security levels: open, staff and administrator each of which governs the functions presented with the screens above i.e.

> Open: – information only (may be opened up at a later stage to allow registered public users create records)
>
> Staff: – create/edit permission records and upload associated drawing files to the Cloud
>
> Administrator: – staff functions plus delete capability of all data

The test functions are held in a separate project which runs a suite of 8 tests using the repository pattern and an in-memory copy of the database. A further data-driven test can be run to read the live permission database and to carry out a column level check for the presence of any dangerous characters (<, >, &, “, ‘, /).

Basic Setup

This application is configured using the .Net 4 framework and is an ASP.NET MVC 4 web application. The default Internet Application template is used with the Razor View Engine. The programming language is C#.

Login/Membership

Logon or membership is configured using the default settings and configuration provided by the ASP.NET MVC framework. Most of the ASP.NET web application templates are preconfigured to use forms authentication and the Internet Application template we have used to create this application provides a default implementation straight out of the box. The AccountController Controller class has been automatically created for us along with the Views for login and registration. Standard Membership and Roles providers are used by the controller, which are configured in the Web.config file of the application. Membership or Login information is stored in a SQL Express Database which is named ASPNETDB.MDF, found under the App_Data folder.

Database

With this application we have used a Model first approach. A class has been created in the Models folder called “PermissionInitialiser” which automatically recreates the database should the model change at any time. If the model should change at any time the database is dropped or deleted and a new database is created from the revised model.

Our database (DHPerm.sdf), which collects details from our online planning application form, is found in the App_Data folder. This is a SQL Express database.

Database connection strings are added in the Web.config file of the project in the following format:

```
<configuration>

  <connectionStrings>

    <add name="PermissionDBContext" connectionString="data
source=|DataDirectory|DHPerm.sdf" providerName="System.Data.SqlServerCE.4.0" />

    <add name="ApplicationServices" connectionString="data
source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
providerName="System.Data.SqlClient" />

  </connectionStrings>
```

Views

This application again takes advantage of the features provided by the ASP.NET framework. The application uses the website administration tool to create roles. There are different views for authenticated users and administrative users. This is configured using the "IsInRole" method with various functionality completely hidden from authenticated users, who have less privileges than administrative users.

Layout

All of the styling and CSS are contained in the Content folder in a file called Site.css. This is where things like font size, colours, backgrounds and heading styles are set. All of the pages in the application maintain a consistent look and feel. This is configured through a Shared folder in the Views directory in a file called _Layout.cshtml. Links to the Site.css file, various jquery files used throughout and various tabs to appear on all pages are included in this file.

13. Scalability of the application

The application is scalable as regards both the database used to hold user information and the blob storage used to hold images/maps.

Blob (Binary Large Object) storage

Blobs are the simplest way to store large amounts of unstructured text or binary data such as video, audio and images. Blobs are an ISO 27001 certified managed service, can auto-scale to meet massive volume of up to 100 terabytes and are accessible from virtually anywhere via REST and managed API's.

The application currently uses the default block size of 4Mb. Files larger than this would need reconfiguration. Microsoft recommends the use of a third-party upload utility such as

(Table updated on 1/5/2011)

Windows Azure Storage Explorer	Block Blob	Page Blob	Tables	Queues	Free
Azure Blob Studio 2011	X	X	·	·	Y
Azure Blob Compressor Enables compressing blobs for upload and download	X	·	·	·	Y
Azure Blob Explorer	X	·	·	·	Y
Azure Storage Explorer	X	X	X	X	Y
Azure Storage Explorer for Eclipse	X	X	X	X	Y
Azure Storage Simple Viewer	X	·	X	X	Y
Azure Web Storage Explorer A portal to access blobs, tables and queues	X	X	X	X	Y
Cerebrata Cloud Storage Studio	X	X	X	X	Y/N
Cloud Berry Explorer	X	X	·	·	Y
Clumsy Leaf Azure Explorer Visual studio plug-in	X	X	X	X	Y/N
Factonomy Azure Utility	X	·	·	·	Y
Gladinet Cloud Desktop	X	·	·	·	Y
MyAzureStorage.com A portal to access blobs, tables and queues	X	X	X	X	Y
Space Block	X	·	·	·	Y
Windows Azure Management Tool	X	X	X	X	Y
Windows Azure Storage Explorer for Visual Studio 2010	X	X	X	·	Y

The file upload method needs to be able to handle network errors as the standard library methods are not sufficient and are not very tolerant of block transmission failures.

The following are the scalability targets for a single storage account:

"(Windows Azure Storage Abstractions and their Scalability Targets, Brad Calder, http://blogs.msdn.com/b/windowsazurestorage/archive/2010/05/10/windows-azure-storage-abstractions-and-their-scalability-targets.aspx)"

- **Capacity** – Up to 100 TBs
- **Transactions** – Up to 5,000 entities/messages/blobs per second
- **Bandwidth** – Up to 3 gigabits per second

The 100TB is a strict limit for a storage account, whereas the transactions and bandwidth are the current targets we've built the system to for a single storage account.

Within a storage account, all of the objects are grouped into partitions. Therefore, it is important to understand the performance targets of a single partition for our storage abstractions, which are:

- **Single Blob** – the partition key for blobs is the "container name + blob name", so we can partition blobs down to a single blob to spread out blob access across our servers. The target throughput of a single blob is:
    - Up to 60 MBytes/sec

The above throughputs are the high end targets for the current system. What can be achieved by the application very much depends upon the size of the objects being accessed, the operations (workload) and the access patterns. All services need to be tested for the performance at the partition level for their workload.

When the application reaches the limit to what a partition can handle for the workload, it will start to get back "503 server busy" responses. When this occurs, the application should use exponential backoff for retries. The exponential backoff allows the load on the partition to decrease, and to ease out spikes in traffic to the partition. If this is a regular occurrence, then the application should try to improve its data partitioning and throughput as follows for the different storage abstractions:

- **Blobs** – consider using Windows Azure Content Delivery Network for delivering anonymous access to hot blobs to achieve higher throughput for commonly accessed blobs.
- The latency is typically around 100ms when accessing small objects (less than 10s of KB) from Windows Azure hosted services that are located in the same location (sub-region) as the storage account.
- Once in awhile the latency can increase to a few seconds during heavy spikes while automatic load balancing kicks in.

Windows Azure Drive allows applications to mount a BLOB formatted as a single volume NTFS VM. You can move your VMs between private and public clouds using Windows Azure Drive.

The local Permission Database has many up-scaling options both on a single local server or distributed on several local servers. It could also be moved to the Azure Cloud if necessary in the future to avail of the clouds horizontal and vertical expansion possibilities.

SQL Database is a multitenant service which maintains scalability by automatically moving databases from heavily accessed machines to other machines.

14. Testing Approach (in terms of both functional and non-functional requirements)

Functional Testing Phases during development

1. Local Database (CRUD functions)
2. Blob storage and retrieval
3. Email Integration

Non-functional setup & testing

1. Azure storage account setup
2. Logon validation
3. Blob management
4. Database integrity check

Nine Unit tests have been included in the solution is a separate project directory. Eight of the tests use the Repository pattern to create an in-memory copy of the database and the following tests are run:

T1) Verifies that the Index method returns a view named Index

T2) Adds the first two Permission records to the mock repository and then verifies that they are in the ViewData Model object in Index view

T3) Tests the routing on a successful get method

T4) Test for creating a Permission record

T5) Verifies that an HTTP POST to the Create method adds a valid Permission to the repository

T6) Verifies that methods correctly handle exceptions

T7) Reads the Permission records into the mock repository and then verifies that there is no illegal character present

T8) Data-driven test which carries out a column level check on the database to ensure the validation rules have not been bypassed

T9) Adds a Permission record containing an illegal character to the mock repository and then verifies that it is picked up

15. Other relevant features of the application

a. Use of client-side processing

- Strict input validation to ensure no dangerous characters can be input and that the address is usable as the blob container name.

b. Use of web services

- System.Net mail services are used to email the blob url to other departments for inspection

c. IoC containers

- The Blob controller (DHBlobController) is a form of IoC container and is a candidate for migration to a published Web Service in the next major release. This will enhance the site's ability to adopt an SOA approach to its Cloud development.

16. Appendices

Appendix 1: Permission Model

```
public class Permission
    {
        [Key] public int Id { get; set; }

        [Required(ErrorMessage = "Permission Type is required")]

        [Range(10, 100, ErrorMessage = "Permission Type must be between 10 and
100")]
        public int permType { get; set; }

        [Required(ErrorMessage = "Postal Address is required")]
        [RegularExpression(@"^[a-zA-Z0-9-\u0020]{1,50}$", ErrorMessage =
            "Only letters and numbers are allowed here")]
        public string postalAddr { get; set; }

        [Required(ErrorMessage = "Applicant Name is required")]
        [RegularExpression(@"^[^\u003c\u003e\u0026\u0022\u0027\u002f]{1,50}$",
ErrorMessage =
            "Characters < > & \u0022 ' / are not allowed here")]
        public string appName { get; set; }

        [Required(ErrorMessage = "Application Date is required")]
        [RegularExpression(@"^((((0[1-9])|([1-2][0-9])|(3[0-1]))|([1-
9]))\x2d(((0[1-9])|(1[0-2]))|([1-9]))\x2d(([0-9]{2})|(((19)|([2]([0]{1}))))([0-
9]{2})))))$", ErrorMessage =
            "Check the Date")]
        public string appDate { get; set; }

        [StringLength(20)]
        [RegularExpression(@"^[^\u003c\u003e\u0026\u0022\u0027\u002f]{1,50}$",
ErrorMessage =
            "Characters < > & \u0022 ' / are not allowed here")]
        public string archName { get; set; }

        [Required(ErrorMessage = "Development Description is required")]
        [RegularExpression(@"^[^\u003c\u003e\u0026\u0022\u0027\u002f]{1,50}$",
ErrorMessage =
             "Characters < > & \u0022 ' / are not allowed here")]
        public string devDesc { get; set; }

        [StringLength(20)]
        [RegularExpression(@"^[^\u003c\u003e\u0026\u0022\u0027\u002f]{1,50}$",
ErrorMessage =
            "Characters < > & \u0022 ' / are not allowed here")]
        public string ownerName { get; set; }

        [Required(ErrorMessage = "Retained Building Address is required")]
        [RegularExpression(@"^[a-zA-Z0-9-\u002e\u0020]{1,50}$", ErrorMessage =
            "Only letters and numbers are allowed here")]
        public string retBldAddr { get; set; }

    [Required(ErrorMessage = "Address of Architect is required")]
    [RegularExpression(@"^[a-zA-Z0-9-\u002e\u0020]{1,50}$", ErrorMessage =
        "Only letters and numbers are allowed here")]
    public string addrDrawer { get; set; }
```

Appendix 2: Search Parameters

PermissionController.cs code to set search parameters:-

```
 public ActionResult SearchIndex( string planAddr, string searchString)
        {
            var AddrLst = new List<string>();

            var AddrQry = from d in db.Permissions
                   orderby d.postalAddr
                   select d.postalAddr;
            AddrLst.AddRange(AddrQry.Distinct());
            ViewBag.planAddr = new SelectList(AddrLst);

            var permissions = from m in db.Permissions
                         select m;

            if (!String.IsNullOrEmpty(searchString))
            {
                permissions = permissions.Where(s =>
s.appName.Contains(searchString)).Take(3);
            }
```

Code for dropdown box in the SearchIndex.cshtm View:-

```
 @using (Html.BeginForm()){
         <p> Postal Address: @Html.DropDownList("planAddr", "All")
         Applicant: @Html.TextBox("SearchString")
         <input type="submit" value="Search" /></p>
        }
```

Appendix 3: Cloud Container Access Code

```
// Retrieve storage account from connection-string
            CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

All the blobs held in the storage account reside inside a container. A CloudBlobClient object is used to reference a container (`"tallaght"`) and a container can be created if it does not exist.

```
   // Create the blob client
            CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

            // Retrieve a reference to a container
            //CloudBlobContainer container = blobClient.GetContainerReference(id value
= postal address);
            CloudBlobContainer container =
blobClient.GetContainerReference("tallaght");

            // Create the container if it doesn't already exist
            container.CreateIfNotExist();
```

Uploading blob into a container :

To upload a file to a blob a container reference (`"tallaght"`) is used to get a blob reference (`"Mockup.bmp"`). Once you have a blob reference, you can upload any stream of data to it by calling the UploadFromStream method on the blob reference.

```
 // Retrieve reference to a blob named "Mockup.bmp"
            CloudBlob blob = container.GetBlobReference("Mockup.bmp");

            // Create or overwrite the "dhblob" blob with contents from a local file
            using (var fileStream = System.IO.File.OpenRead(@id))
            {
                blob.UploadFromStream(fileStream);
            }

            return RedirectToAction("Index", new { id = id });
        }
```

Uploading drawing to Blob storage:

For a drawing to be uploaded to the azure blob storage account it must firstly be contained in a local folder(`"C:\Drawings"`) which is referenced in the **DHBlobController.cs** (This approach is used to overcome the inability of the browser to directly access the local hard disk.)

```
 // This action handles the form POST and the upload
        [HttpPost]
        public ActionResult GetFname(HttpPostedFileBase file, string
conatinerId)
        {
            // Verify that the user selected a file
            if (file != null && file.ContentLength > 0)
            {
                // string filePath =
Path.GetFullPath(file.FileName);


                // extract only the fielname
                string fileName = Path.GetFileName(file.FileName);
                // store the file inside ~/App_Data/uploads folder
                //var path =
Path.Combine(Server.MapPath("~/App_Data/uploads"), fileName);
                string filePath = Path.Combine(@"C:\Drawings",
file.FileName);
                //save the file to our local path
```

The drawing is uploaded to a container in the azure storage account which is private by default. To make the drawings accessible to everyone for the purpose of this project the container permissions are set to public.

```
  container.SetPermissions(
                    new BlobContainerPermissions { PublicAccess =
BlobContainerPublicAccessType.Blob });
```