**System Overview**

The system overview is structured into a three tiers hierarchy.

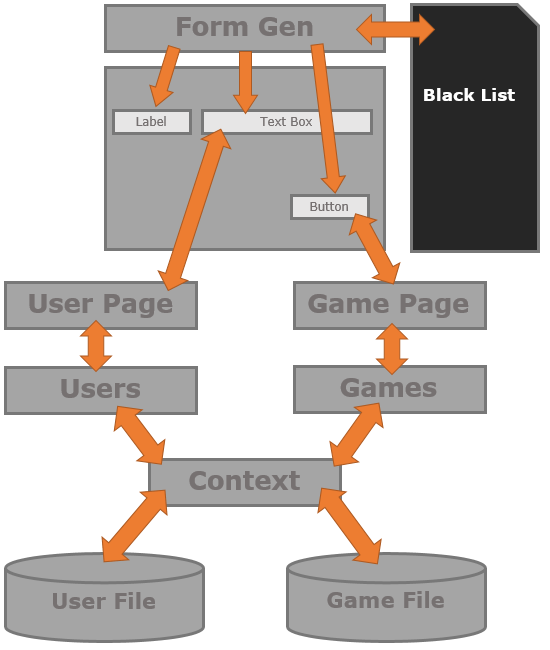
At the Front-End lives the Form. The Form is initially blank. The Controls that appear within the Form are placed there by a module called Form-Gen.

At the Back-End live the files. All User, Game and Order data are saved to, and read from, files. The module that is responsible for this is called the Context.

In between the Front-End and the Back-End are the Users, Games and Orders, collectively called ‘Entities’. Entities interact directly with the Context.

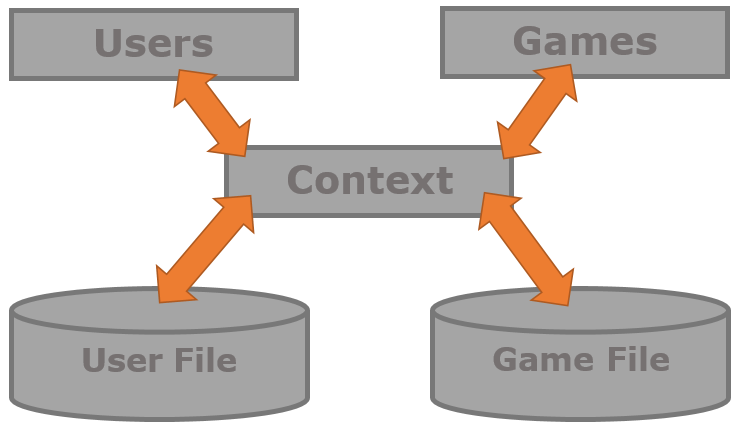
Each class of Entity has a number of Form Pages. Pages act as an intermediary between the Form Controls and the Entities. Pages direct the Form-Gen to place the appropriate Controls on the Form. Event handlers are hooked into these Controls. The Pages will populate the Controls with values obtained from the Entities. Data may also flow in the opposite direction, from the Form Controls, through the Pages Event Handlers towards the Entities.

The Black-List defines user access to the system. This is simply a list of Form Controls and Pages which should not be displayed to the currently logged in User.



**The Context**

The Context is responsible for managing the data. Each class of Entity is allocated its own Dictionary. Items in the Dictionaries are accessed by a primary key field.



The Dictionaries are serialized to and from files during the Start-Up and Clean-Up phases. Each Dictionary is associated with its own file.

If the files are not found at Start-Up, then a Hard-Coded default pre-sets are generated in memory. This ensures that there always exists at least one Manager Entity.

**The User Classes**

TODO: Add a write up about the User Class Hierarchy

**The Game Class**

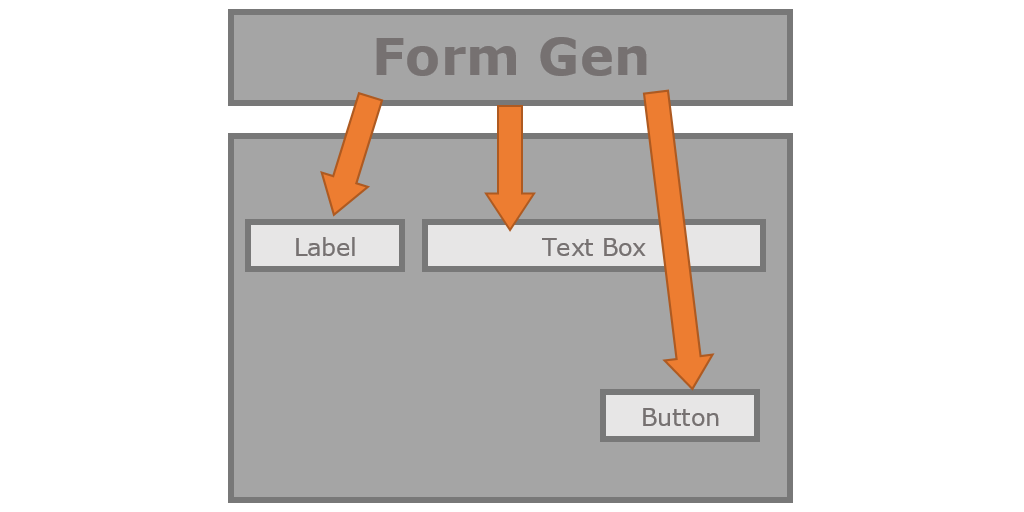
TODO: Add a write up about the Game Class

**The Order Classes**

TODO: Add a write up about the Order Class

**The Form-Gen**

The Form-Gen is responsible for managing the Form. The Form is entirely dynamic, and is created programmatically at Run-Time.



The Logical Units that the Form-Gen deals with are called Pages. Pages are a group of Form Controls that always appear together. Each Page is given its own unique name. The convention used internally is ‘entity.purpose’

The Page named ‘user.form’ refers to the set of Controls that represent the members of a User. The ‘user.edit’ Page defines additional elements like a submit Button.

Pages are typically overlaid, one on top of the other. Each page contributes towards defining a particular Form.

**The Header Page**

The Header Page is a set of Controls that are always present. The Header Page is automatically affixed to the top of any Form.



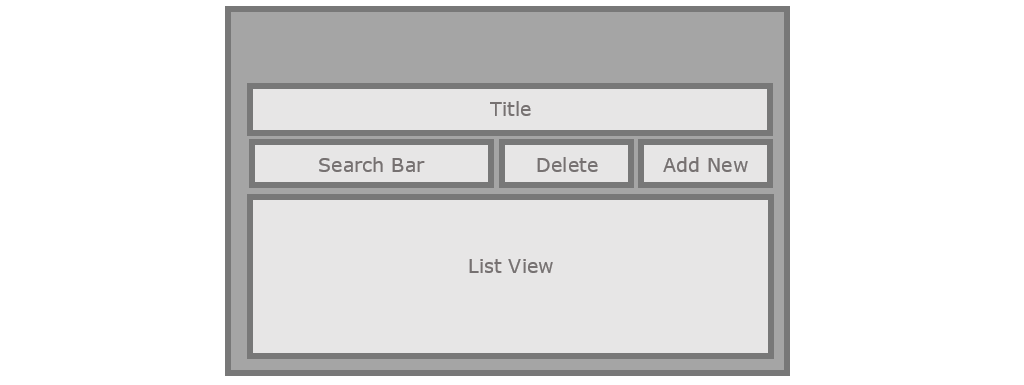
The Header Page defines Controls used for Logging Out, and selecting a class of Entity which the User needs to process.

The Header Page is given the name ‘header.form’ and is defined by the Log-On class at start up.

The Entity Selection Tabs allow the System to move seamlessly between manipulating Users, Games, and Reports. All the while maintaining a consistent layout and interface.

**The List Page**

The List Page exposes the contents of the Dictionary associated with the currently selected Entity class. Each class of Entity must define a List Page. These pages are named ‘user.list’, ‘game.list’, and ‘report.list’.

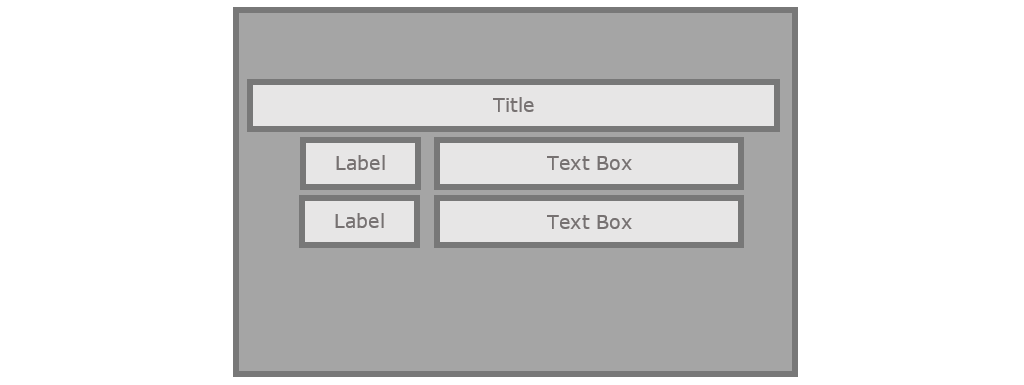


List Pages are typically made up of a List View, a Title, a Search Bar, and depending on the Access Rights of the Logged In User, may have an Add New and Delete Button.

The List View was designed such that Sorting by Columns and Item Selection are all handled automatically, with Entities remaining oblivious to the entire process.

**The Form Page**

The Form Page represents the member fields of the associated Entity. Whilst the other Pages that make up a Form may be carbon copies of each other, the Form Pages are distinct from each other. These pages are named ‘user.form’, ‘game.form’, and ‘report.form’.



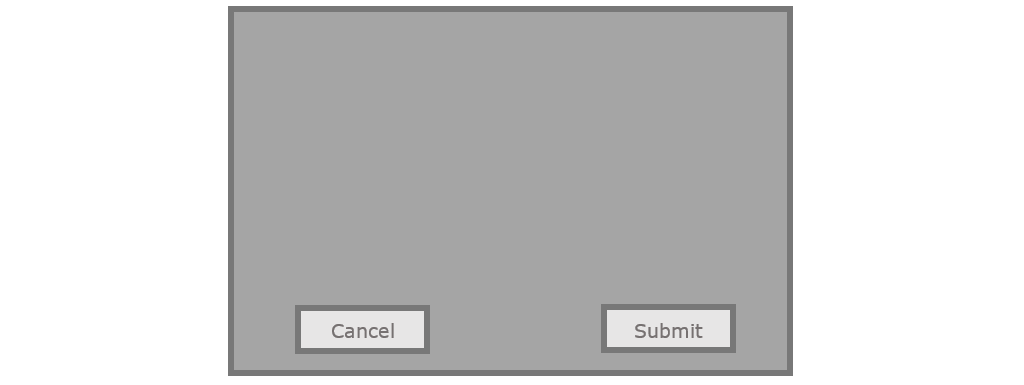
Form pages contain a group of Control, such as a Title, Labels, Text Boxes and Date-Time Pickers.

There is a one to one correspondence between the Controls that appear in the Form Page and the member fields of the Entity class.

The Form Pages are selected automatically by the System depending upon which class of Entity the User has actively selected. They are chosen by ‘Context’.

**The Extra Pages**

Extra Pages are used to add the finishing touches to the final Forms. These pages are given names such as ‘user.view’, ‘game.edit’, ‘report.make’, ‘user.drop’. Because these forms go hand and hand with the ‘entity.form’ Pages. You only need to specify the Extra Page when requesting a new Form to be generated.



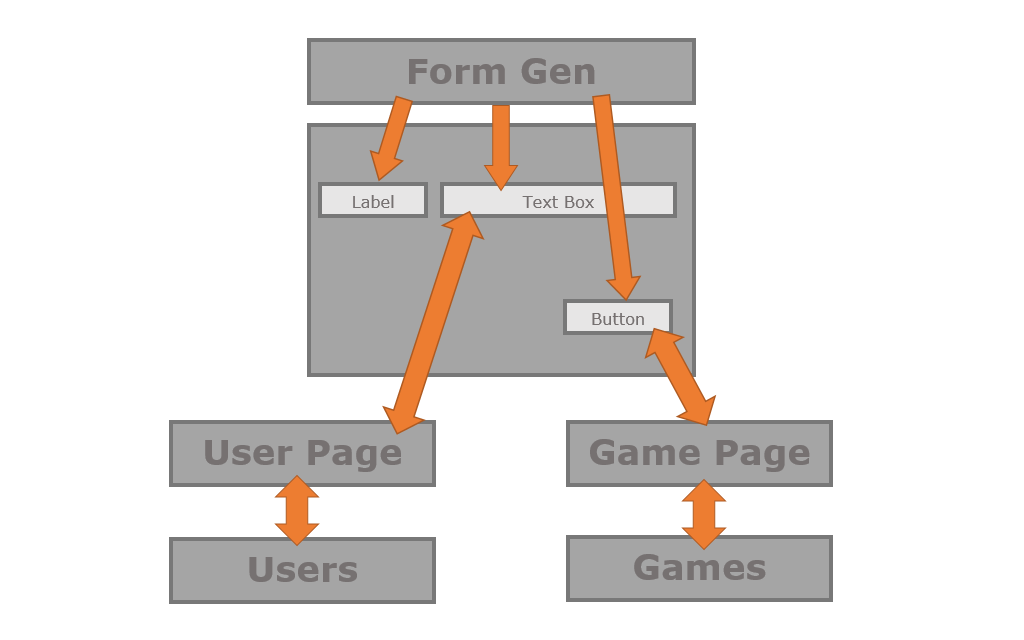
The buttons on these forms are monitored by an associated Page Managers. The Page Managers will respond to these actions in the appropriate manner.

Some actions require that the contents of the Form are to be copied to a specific Entity (Submit), other actions require that a new Form should be generated (Cancel)

These Extra Pages are the glue that hold the entire Application together.

**The Page Managers**

Each class of Entity has a number of Pages. Pages act as an intermediary between the Form Controls and the Entities. It is the responsibility of the Pages to define the Pages which are to be associated with a given class of Entity.



By taking this responsibility out of the Entity classes we can keep our Entities more simple. Users are allowed to just be Users. This separation of concerns simplifies the overall system.

**The Log-On Module**

The Log-On Module is responsible for ensuring that only authorized Users are allowed to access the System. The Log-On class behaves much like a Page Managers. It defines its own Page and responds to its Events.

Each User account is password protected. In order to log in to the System, a user must first provide a correct username and password combination.

Passwords are not stored or remembered by the System, rather the password is used as a key to encrypt the username. This produces a cipher text version of the username. This cipher text is stored as a hidden member of the User class.

When logging in the username and password provided must produce the same cipher text, or the log in attempt fails.

We chose not to store the passwords as our system serializes all Entities. This would leave the passwords vulnerable to malicious Users with access to a Hex Editor.

Further by encrypting the username in this manner we can ensure that if one User account becomes compromised it will not affect other Users that happen to be using the same password.`

**The Black List**

The Black List is the part of the system that micro manages user access to the system by maintaining a list of Form Controls or even entire Pages which should not be displayed to the currently logged in User.

No Form Controls can be displayed in the Form without first getting a thumbs up from the Black List.

Each class of User is associated with its own blacklist. The Type of the User is used as a key into a Dictionary, ensuring that the appropriate List of strings are used at runtime.

If the named control is contained within the blacklist associated with that User, then the Control will not be added to the Form.

The Black List operates invisibly to the rest of the system with the exception of the Form-Gen class.

**The Context API**

public bool AddUser(string key, User user);

AddUser is used to add a defined user to the user list.

Once a game has been added it can later be referred to by name.

public string GetSelectedUser();

Returns the primary key of the currently selected user.

public User GetUser(string username);

GetUser allows internal access to the contents of the user list. To access a user you must provide the username.

public void SetLogged(string key);

Caches a local copy of the logged in users primary key.

public Entity GetLogged(string type);

Gets the logged in user.

public bool AddGame(string key, Game game);

AddGame is used to add a defined game to the game list. Once a user has been added it can later be referred to by name.

public string GetSelectedGame();

Returns the primary key of the currently selected game.

public Game GetGame(string title);

GetGame allows internal access to the contents of the game list. To access a game you must provide the title.

public Entity GetSelected(string type);

Gets the selected entity based on the given type name.

public void SetSelected(string type, string key);

Sets the selected entity based on the given type.

**The Form-Gen API**

public Control GetControl(string pagename, string key);

GetControl allows access to the contents of the controls created at runtime. To access a control you must provide the name of the containing page, and the name of the control on that page.

public Page GetPage(string pagename);

GetPage retrieves the Page manager by name.

public bool AddPage(string key, Dictionary widgets);

AddPage is used to add a defined form page to the dictionary. Once a page has been added it can later be referred to by name.

public bool BuildHeader();

BuildHeader is responsible for building the common header form controls that appear at the top of every page.

public bool BuildPage(string pagename);

BuildPage is responsible for dynamic runtime form generation. Form widgets\_dict are identified by name. This name is used as a key to an entry in a dictionary. The controls are retrieved and added to the form.