Lunar Logic Services

System Overview and Design

1. Introduction
2. System Overview
3. System Architecture
4. Data Design
5. Human Interface Design

Authored by: Jesse Wood, Casey DeDore, Logan Gosson

Introduction

The purpose of this web application is for users to easily find out more information about what services Lunar Logic provides. The web app is supposed to be interactive and fun, and allow a means for a user to “explore” what services are available.

The website will allow the user to select what services they are interested in and develop their own personal plan. Once they are satisfied with what they have selected, they will fill out a contact form that will send an email to Lunar Logic. Their contact information and the services they have selected will be sent in the email so that Lunar Logic can contact them with the appropriate information about the potential services that they may purchase.

System Overview

The system allows functionality for a user to explore what services are available. They are able to interact with a node system to reveal and obfuscate parent and child services as they go. They are able to select, or deselect, what services they desire at any time to add to their “Personal Plan.”

The system allows interaction with the graph by not only means of clicking or hovering on nodes, but also by interacting with the List of Services in the Personal Plan.

The system is web-based using ASP.NET with an MVC architecture. The system will utilize a JavaScript library known as Arbor.JS to display the node structure. The system will use Entity Framework for object relational mapping. The service data will be stored in the database and then transferred to the node structure by means of AJAX/JSON.

The system must allow for dynamic data entry, node population, and service list creation. The services that are provided could change at any time, so the system must be able to generate itself to allow for any services that are stored in the database.

The system will allow an Admin user to add new services, or change and delete them at any time.

System Architecture

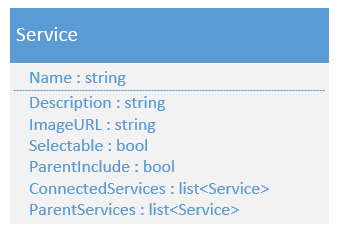
**Services – From the Database To The Screen**

The display of services and their data is the linchpin for our whole website. They are used to populate the graph of nodes that is the focus of the site and primary point of intrigue for users. Services contain information that is used to drive the behavior of the graph, to be displayed to the user, and to be adjusted by the site administrator whenever they see fit.

**Services on the Service Side**

As far as a user is concerned, a service consists of a name, description, and an icon. Under the hood, a service also includes a list of its connected services, an option to determine its select-ability, and others. A class, called the service model, is used to contain this information. This service model is translated using Entity Framework into a relational model to save data to the database.

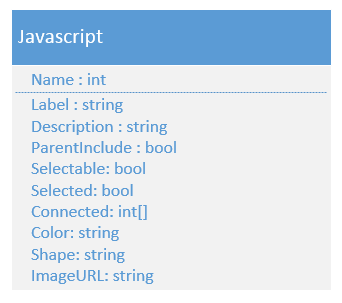
The C# service model is shown below.

ParentInclude and ParentServices are not currently used. ParentInclude exists for scenarios where when in node form and a service is selected by user, it optionally would select its parent nodes automatically. ParentServices would contain all of the service's parents. This is different than ConnectedServices in that it only lists the nodes that are higher in the node hierarchy. ConnectedServices lists any and all services connected.

**Services on the Client Side**

When passed from the server to the client, services take on a slightly different form. A few extra properties are added to each service, such as selected, color, and shape and label. The id of the service, which is an auto-generated key created by the database, is passed as the name. The original human readable name is passed as the label. Selected determines whether or not the user has selected that service. The label (human readable name) is what is displayed on the node and wherever the name is needed. Color and shape are used to style the nodes on-screen.

The Javascript service object is shown below.



A collection of these objects is stored for use in bot1h arbor to populate the nodes, and the javascript that drives the sidebars. These nodes are not persistent. Based on various conditions, the node graph is algorithmically constructed and services from the list are inserted as nodes in the graph. These nodes are also removed when they are not needed. This is why the list of services is needed.

The service nodes contain the same information as the services stored in the list, aside from special data that is needed to drive the behavior of the graph. Their structure is a little different to accommodate the graph's functionality, so a slight conversion is needed. Aside from the name, the graph stores the rest of the properties in a data object within the node. Accessing node x's selected property would be done by x.data.name, whereas service y in the list would have the selected property accessed by y.name.

The graph system also stores other data in nodes, but they are specific to arbor and will not be described here.

Data Design

Server Side

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Column Name | Data Type |
| Service | Object used to store service information and attributes of node functionality | Name  Description  ParentInclude  Selectable  ImageURL  ConnectedServices  ParentServices | STRING  STRING  BOOL  BOOL  STRING  COLLECTION  COLLECTION |

Client Side

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Column Name | Data Type |
| service | Object that server-side data model is translated into, and then eventually translated into object that arbor.js uses. | Name  Label  Desc  Selectable  Selected  Connected  Parent  Color  Shape  Imageurl | STRING  STRING  STRING  BOOL  BOOL  ARRAY  STRING  STRING  STRING  STRING |

Human Interface Design

