Spirit & Place Festival

Butler University EPICS
Fall 2013

Team Leader	*Jason Rowe*
Database Management	*David Harting*
Application Design	*Stephanie Boylan*
Client	*Robert Ferrell*
Professor	*Panos Linos*

Table of Contents:

1 Executive Summary

Spirit and Place Festival is a community project managed by The Polis Center, part of the IU School of Liberal Arts at IUPUI. First started in 1996, each year The Polis Center hosts the Spirit and Place Festival, which reaches around 20,000 people annually through "never-seen-before" programs. "Artists and authors, entrepreneurs and neighborhood organizers, storytellers and scholars come together with singles and couples, families and friends in a true community conversation. Think: TED lecture, but interactive and on steroids." The festival's mission is to provide civic engagement through creative collaborations among the arts, humanities, and religion. "Spirit & Place is an example of Polis's commitment to creating collaborative, practical, and effective solutions for the communities in which we live."

Currently, The Sprit and Place Festival is utilizing an outdated web application for clients to fill out event information to apply to be in the next festival. The current application form is not user-friendly in terms of appearance, usability, and length. Overall, our team felt that users are overwhelmed by the amount of information on each page of the application, which created a tedious process. In addition, this form has to be filled out multiple times if the user is hosting more than one event in the festival. We decided to create a more user-friendly process that reduces the amount of work needed to submit an event application. By implementing a design that is centered around each specific event, we are able to increase the efficiency of submitting an event by decrease the amount of time spent on filling out tedious information that is identical for each event.

This document is to be used as a tool for understanding how to complete EPICS Project Spirit and Place Festival Application. The following topics will be discussed to facilitate the understanding: (1) Goal of Project, (2) Current State of Project, (3) Work that Remains Unfinished, (4) Tools to Complete Project.

2 Semester Goals and Accomplishments

This Engineering Projects in Community Service (EPICS) group goal for this project is to overhaul the web application that was last updated in 1995, to a newer and modern style. The client desires the application to facilitate better navigation and have a cleaner user interface.

The previous semester's team that worked on the Sprit and Place Festival Application developed a Balsamiq mock-up of the new application appearance. With no code easily accessible, we effectively started from scratch and redesigned the web application in Visual Studio 2012. The goals for the semester were as follows:

1. Improve the flow of the registration form
   a. Increase usability for applicants with a user friend interface
   b. Reduce amount of text per page
   c. Combine tabs to increase efficiency
   d. Update the layout of the application for better navigation
2. Upgrade the application from Visual Studios 2005 to Visual Studios 2012
3. Install and recreate a local version of their SQL database to our server
4. Focus on implementing key functionality features
   a. Redesign all application pages
   b. Implement new application design
   c. Attach to their SQL database

Over the course of the semester, we met all of our goals concerning redesign of the application. The interface is much cleaner and easier to navigate, allowing the user to have a much better experience while filling out the application.

We did a lot of work with the SQL database, setting up a framework for a Web API that the next semester's group can build upon. There is further documentation of this later on in this dossier.

3 Individual Contributors

4.1 Jason Rowe

Jason is a senior Computer Engineering, Computer Science, and Mathematics major at Butler University. His role as team leader required him to facilitate communication between all parties, manage work assignments, provide updates to both the client and the professor, and oversee the final presentation and report.

4.2 David Harting

David is a junior Computer Science and Psychology major, with a minor in Mathematics. David took control of the data access and database management aspects of the project, setting up a Web API and developing technical documentation to help the next semester's team hit the ground running.

4.3 Stephanie Boylan

Stephanie is a junior Computer Engineering and Computer Science major. Stephanie was in charge of redesigning the entire application using Visual Studio 2012, implementing many brilliant design features using CSS.

4 Data Access and Database Information

David was tasked with creating a data access layer for a web application that will be used to gather information about events and speakers at the Spirit and Place Festival. After researching, he decided that a Web API would be the most modern, flexible, and extensible approach. He created an ASP.NET MVC 4 Web API. This section contains information on his research, how he created the API, and how the next team can quickly pick up where he left off.

Tools:

SQL Server Management Studios 2012

- For querying the Spirit database
- For accessing stored procedure details in the Spirit database
- For creating new stored procedures in the Spirit database
- Available for free from Butler students through Dreamspark (contact Amy, the departmental secretary if unsure how to get to Dreamspark)

Visual Studios 2012

- For creating ASP.NET MVC 4 Web API projects
- Available free from Butler students through Dreamspark (contact Amy, the departmental secretary if unsure how to get to Dreamspark)

Remote Desktop Connection

- For accessing the spirit.butler.edu server, where the Spirit database is running on SQL Server
- Comes standard on Windows machines. Other free remote desktop clients exist, including VMWare.
- Please see “Accessing the Spirit Server and Database” to learn how to connect

Telerik Fiddler

- Allows you to compose and inspect http requests
- Used it to test SpiritApi; after writing each controller function, compose an HTTP request in Fiddler to make sure the function works correctly and returns the correct data
- Available free from telerik (http://www.telerik.com/download/fiddler) (.NET 4)

Overview:

People often discuss CRUD operations when discussing data access layers. CRUD is an acronym for create, read, update, delete. A data access layer that supports CRUD operations allows users to create new objects and insert them into the database, retrieve records from the database, update information about records in the database, and delete records from the database.

Data access layers are described as RESTful if they adhere to certain HTTP protocols. HTTP, or hypertext transfer protocol, is a way of transferring information across the internet. RESTful web services use HTTP requests consistently. There are different kinds of HTTP requests, and RESTful web service always responds to the same kind of request in the same way. One way to think of RESTful web services is that certain HTTP requests always map to certain CRUD operations. In SpiritApi, GET requests trigger read operations, PUT requests trigger update operations, POST requests trigger create operations, and DELETE requests trigger delete operations.

The basic architecture of SpiritApi is divided into Controllers, Repositories, and Entities. Entities are objects that represent the “real world” things that we want to store information about. For instance, Events and Users are entities that are represented in the database. The web form will request information about Events and Users, as well as request to store new Events and Users in the database. Controllers are the part of SpiritApi that the web form can talk to. When the web form makes an HTTP request, it “hits” a SpiritApi controller and finds the appropriate function on the controller. The controller function then performs some operations and may return some data back to the web form. Controllers depend on repositories to actually talk to the database. SpiritApi repositories use SqlCommands (an ADO.NET object) to execute stored procedures in the Spirit database. The stored procedures then return information to the repository class, which returns the information to the controller. This structure allows for the parts that talks to client applications (controllers) to be separate from the parts that talk to the database (repositories).

Resources:

Implementing CRUD Operations in ASP.NET WEB API

- http://www.asp.net/web-api/overview/creating-web-apis/creating-a-web-api-that-supports-crud-operations

- This is a text, step-by-step tutorial for actually creating a project in Visual Studio and writing code! I used this tutorial to build FishAPI.

- *Read the first few paragraphs (until Create a Web API project) first*. This will give you an overview of CRUD operations and what we are trying to do with a data access layer. Then, come back to this when you are ready to actually start working on SpiritApi, or when trying to build a toy project.

Short video tutorial of ASP.NET Web API projects

- http://www.asp.net/web-api/videos/getting-started/aspnet-web-api
- Scott Hanselman shows you (really quickly) how to build a controller and demonstrates a bit of the functionality
- This will give you a decent idea of what Web API projects are all about, but it will not be helpful in a step-by-step sort of way as you build your own
- *Watch this before you actually try to build anything*.
- Take away: ASP.NET web API projects deliver data in the form request by the client – so you (the data access enginer) don't have to specify how to return the objects. If you want the GetAll function to return a list of events, just set the return type to IList<Event>. The project will figure out how to format it and send it to the client.

A Guide to Designing and Building RESTful Web Services with WCF 3.5

- http://msdn.microsoft.com/en-us/library/dd203052.aspx
- SpiritApi is a Web API project, not a WCF project. However, the beginning of the guide is about HTTP requests and RESTful design patterns. It's the most detailed of the materials listed. *Read this after the preceding material*. Or take a glance at it first, click through some of the other material, then come back to this again and try to make more sense of the general information. Don't worry about the stuff related explicitly to WFC.

A Beginner's Tutorial for Understanding ADO.NET

- http://www.codeproject.com/Articles/361579/A-Beginners-Tutorial-for-Understanding-ADO-NET
- This was the main resource I used when using ADO.NET classes (SqlConnection, SqlCommand, etc.) in the SpiritApi repository classes.
- *Read this when you are trying to understand how to actually talk to the database from a Web API*.

Consuming a Web API

- http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api
- This is the next part of the first tutorial listed. It demonstrates how to build another API, *and* how to build an application that sends requests to that web API to get data.
- *Reading this will be helpful in understanding how SpiritApi will fit into the whole project. It will also be helpful to client side programmer's attempting to send requests to SpiritApi (e.g., using ajax calls)*.

You can look at a Stack Overflow discussion that steered me toward Web API rather than other Web service projects here: http://stackoverflow.com/questions/9348639/wcf-vs-asp-net-web-api.

StackOverflow and the Microsoft Developer Network were my most frequently used resources throughout development and will often be the top hits for Google queries.

Progress:

SpiritApi

Created an ASP.NET MVC 4 Web API project in Visual Studio 2012.

Added the following classes:

- Entities
  - Event (Figure 3)
  - User (Figure 6)
  - QueryResult (see note in “Next Steps”) (Figure 8)
  - UserSecretInformation (see note in “Next Steps” under Route Mapping) (Figure 7)
- Controllers
  - EventsController (Figure 1)
  - UsersController (Figure 4)
- Repositories
  - EventRepository (Figure 2)
  - UserRepository (Figure 5Figure 4)

I added a connection string tag inside the configuration tag of the Web.config:

```
<connectionStrings>
<add name="DefaultConnection" providerName="System.Data.SqlClient"
connectionString="Data Source=DAVID;Initial
Catalog=TPCPrograms2013;Integrated Security=True" />
 </connectionStrings>
```

This allows “DefaultConnection” to be used in the code, in place of a hard-coded connection string. This is safer and more extensible. For instance, this is the instantiation of the SqlConnection inside of the EventRepository constructor.

```
con = new
SqlConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].C
onnectionString);
```

Stored Procedures

The create procedure or alter procedure queries for each stored procedure referenced can be found in the Appendix. Each query lists the parameters for the query at the beginning, and lists the output parameters after the output keyword near the end of the query.

Created the following stored procedures:

- GetMiniEventsById (Figure 11)
    - returns a partial record of an event that has the same ID as the argument
- InsertMiniEvent (Figure 12)
    - inserts a new record into the event table, with the values of the attributes determined by the parameters
    - returns a partial record of the inserted event
    - *Note: you could simply return the ID (assigned by the database). However, it may be more useful to return the entire record back to the client, as it may represent data slightly differently than the parameter values.*
- UpdateMiniEvent (Figure 14)
    - finds the record of an Event with the same eventID as the ID parameter and sets the attributes of the record equal to the values of the parameters
    - returns a partial record of the updated event
    - *Note: this does not need to return the event. The Web API does not need to give the updated object back to the client, but it could be useful.*
- InsertUser (Figure 13)
    - inserts a record of a User into the Users table, with the attributes of the record determined by the values of the parameters
    - returns the inserted record
    - *Note: this could return just the user's ID instead*
- UpdateUserPasswordAndSecretAnswer (Figure 15)

I modified the UpdateUser (Figure 14) stored procedure to return the modified record.

- action: looks up a User by Id, then alters their attributes to match the records
- returns: the altered records
- *Note: this does not have to return anything. You could set nocount off, then simply return the number of records affected. If 1 record was affected, then the update was a success.*

You can run these queries on a copy of the Spirit database to add these stored procedures. Please see "Handle full events" under "Next Steps" for an explanation of MiniEvents. To alter an existing stored procedure in SQL Server Management Studios, right click on an existing stored procedure and choose to alter the procedure. A query will be generated. Make your desired modifications and then run the query to update the database.

Using ADO.NET in SpiritApi

ADO.NET is basically a library of classes that make it easy to talk to databases from Visual Studio projects. Here, I will step through an example of how ADO.NET is used in SpiritApi.

Each repository has a private SqlConnection variable.

```
private SqlConnection con = null;
```

This variable is instantiated in the repository constructor.

```
        public EventRepository()
        {
                con = new
                SqlConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"]
                        .ConnectionString);
        }
```

Inside of repository functions, SqlCommands are created from the SQL connection. The CommandType field is set to specify that the command is going to use a StoredProcedure in the database. If the stored procedure requires parameters, these need to be added to the SqlCommand. The SqlParameter class is used to accomplish this.

```
        public Event Get(int id)
        {
                SqlCommand cmd = con.CreateCommand();
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.CommandText = "GetMiniEventById";
                SqlParameter paramId = new SqlParameter("@eventId", id);
                cmd.Parameters.Add(paramId);

                //more stuff happens in here
                // an event is returned at the end

        }
```

You can use a SqlDataReader object to collect the results from a executing a command. The result will often be the records affected by the query. Each execution of Read() advances to the next record.

```
        con.Open();
        rdr = cmd.ExecuteReader();
        if (rdr.Read())
        {
                result = rdr.CreateEvent();
        }
```

Note the CreateEvent() command, which is an extension I built. See Figure 9 for a description of all the Reader extensions used in this project.

Recommended Actions:

What to Learn

Although I spent more time in Visual Studio than in SQL Server, I recommend that a group member who has taken databases take on this part of the project. If you have not taken databases, I recommend familiarizing yourself with some basic relational database and SQL topics, including:

1. how a relational database represents information
2. what a table, record/row/tuple is
3. what an attribute is
4. how to write select, insert, delete, and update statements in T-SQL (T-SQL is the version of the SQL created by Microsoft for SQL Server)

The queries required for this project are relatively simple and straightforward. Only a basic understanding of SQL will be needed.

Next, I recommend reading through and watching the content I listed in the Resources section. This will give you a good starting point for understanding how data access layers fit into design patterns, what a Web API should do, and how to implement a Web API on the .NET framework.

Next Steps

1. QueryResult class (Figure 8)
   Controller functions which should return integers return a QueryResult object, which is essentially a wrapper for a primitive int. I attempted to return Int32 objects as well as primitive int variables, but I could not get these to be received as Json from HTTP requests composed in Fiddler. Investigate to see if there is a standard way to return primitive data types from Web APIs.

2. Handle full events
   I created stored procedures for events that only return a limited number of event attributes. The Event entity should have attributes added for all columns in the Event table in the database, and the stored procedures should be able to accommodate these attributes as well. The existing procedures distribute managing event attributes across several different stored procedures. This approach may be appropriate as well, depending on the flow of the form.

3. Dependency injection
   The SpiritApi controllers instantiate a static repository class. This is bad design. Follow the link for a tutorial that discusses design patterns and how to use the Dependency Resolver to fix this problem. http://www.asp.net/web-api/overview/extensibility/using-the-web-api-dependency-resolver

4. Route mapping
   For security, the User entity class does not contain a SecretAnswer or Password. My thinking was that client applications should not be able to request a User and get this private information. So the PUT request for updating this secret information is separated. However, the controller behaves erratically when it has multiple PUT requests. Routes can be added or modified in the RouteConfig.cs file to allow for multiple PUT requests. Alternatively, SecretAnswer and Password information could be put inside the User entity (stored procedures would then need to return this information), and these could simply be nullified (e.g., set the value to null or to the empty string) before returning the information to the client application.

5. Connect the SpiritApi project to the Spirit database on a Butler server, and run SpiritApi on the Butler server. Then, see if the web form can make requests from the SpiritApi database.
   As stated previously, SpiritApi is currently connected to local copy of the TPCPrograms2013 database. I could not get SpiritApi to connect to the copy running on the spirit.butler.edu server. SpiritApi should run on this server, connected to this database, so that it is accessible to client applications (i.e., the windows form other teammates are building).

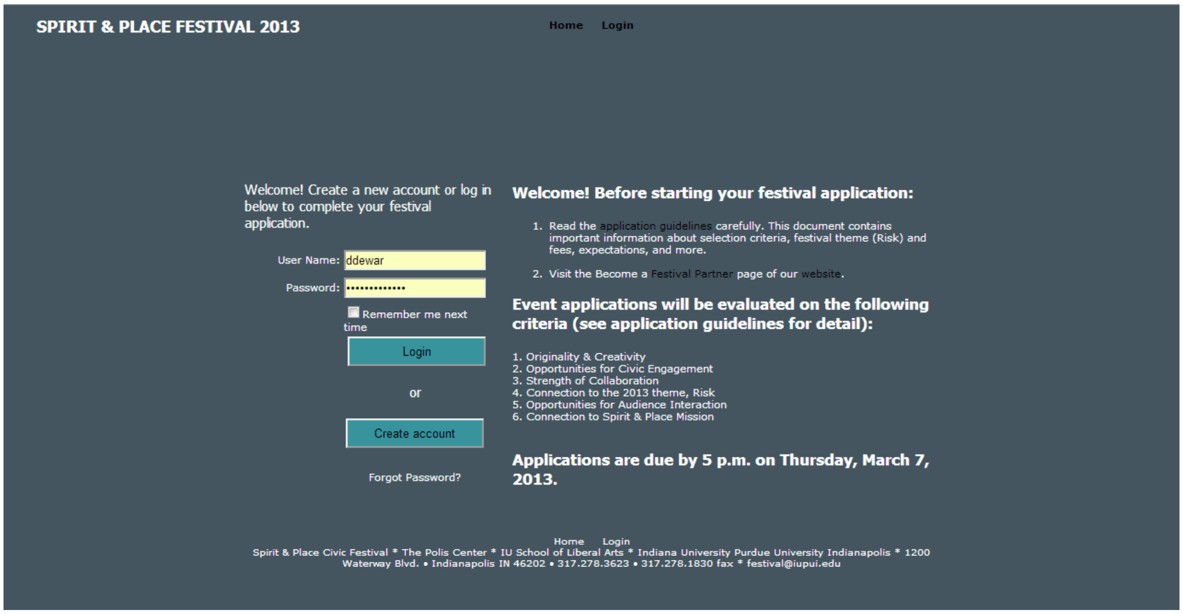
6. Identify needed functionality and create new controllers and functions
   SpiritApi at this point only allows for CRUD operations on Events and Users. Further operations will be needed for the project. Further discussions with the client will be needed to identify further functionality needs. Each entity used will need its own controller in SpiritApi.

Accessing the Spirit Database and Server

1. Request Nate Partenheimer to allow all group members to access spirit.butler.edu
2. Use a remote desktop client while connected to Butler wifi
3. Login with Butler credentials (assuming part 1 went well)
4. On the Connect to Server Window, set Server Type to Database Engine, Server name to spirit.butler.edu, and Authentication to Windows Authentication. Click Connect.
5. TPCPrograms2013 is the database for this project.

5 Current Application

At first glance, the current Spirit and Place application login page appears extremely out of date. The majority of the page is ‘white space’, meaning the content is not using the full parameters of the screen, and the color scheme is not inviting. The title for the application doesn’t stand out very well, the user would only guess what it is by the fact that they clicked on the website link. The current website is designed with simple html, CSS tactics as well as ASP.NET; therefore it is missing some scripting that makes websites these days more noticeable and popular. After analyzing the issues of the current application, our team focused importance on implementing a better layout to improve the navigation and give the website a more modern appearance. We felt that user’s were skipping over important information due to the large quantity of text per tab throughout the application. Recent studies show that user’s do not read most information on websites other than what they consider important or necessary and scan over the majority of the data.



SPIRIT & PLACE FESTIVAL 2013

Welcome! Create a new account or log in below to complete your festival application.

User Name: ddewar

Password: •••••••••••••

□ Remember me next time

Welcome! Before starting your festival application:

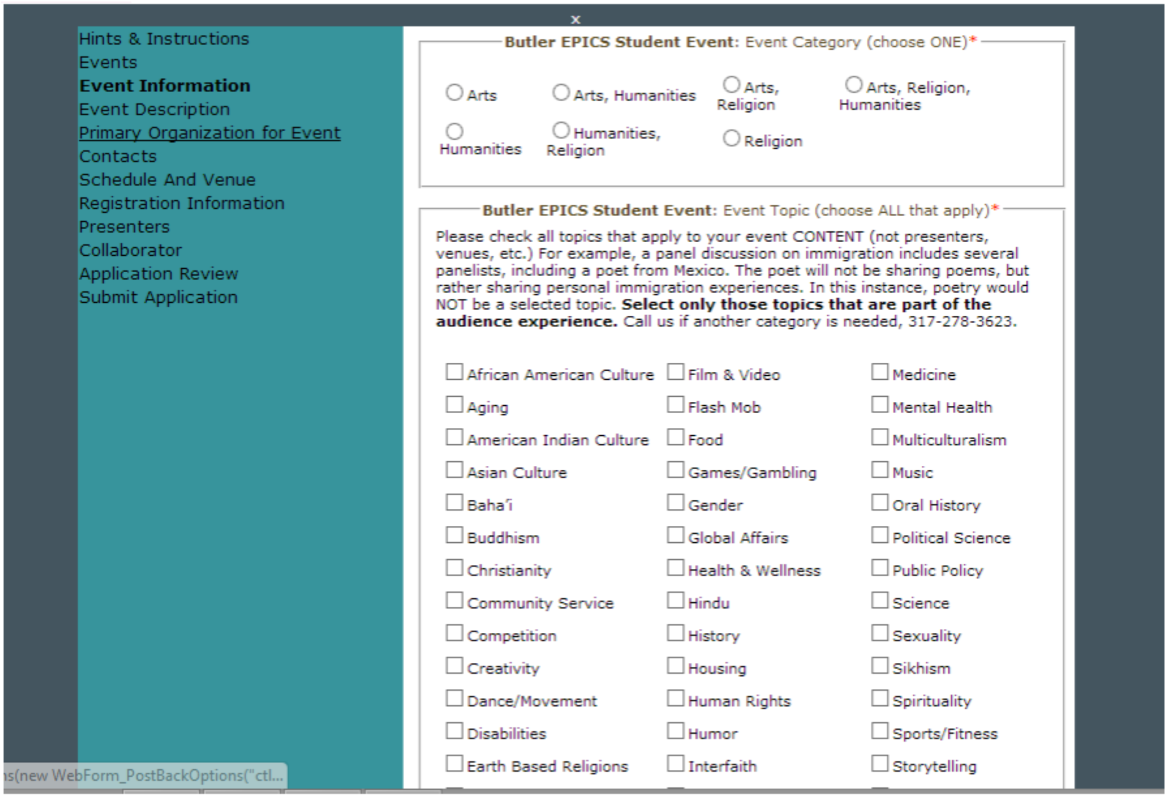
1. Read the application guidelines carefully. This document contains important information about selection criteria, festival theme (Risk) and fees, expectations, and more.
2. Visit the Become a Festival Partner page of our website.

Event applications will be evaluated on the following criteria (see application guidelines for detail):

1. Originality & Creativity
2. Opportunities for Civic Engagement
3. Strength of Collaboration
4. Connection to the 2013 theme, Risk
5. Opportunities for Audience Interaction
6. Connection to Spirit & Place Mission

Applications are due by 5 p.m. on Thursday, March 7, 2013.

Spirit & Place Civic Festival * The Polis Center * IU School of Liberal Arts * Indiana University Purdue University Indianapolis * 1200 Waterway Blvd. • Indianapolis IN 46202 • 317.278.3623 • 317.278.1830 fax * festival@iupui.edu



Butler EPICS Student Event: Event Category (choose ONE)*

○ Arts ○ Arts, Humanities ○ Arts, Religion ○ Arts, Religion, Humanities
○ Humanities ○ Humanities, Religion ○ Religion

Butler EPICS Student Event: Event Topic (choose ALL that apply)*

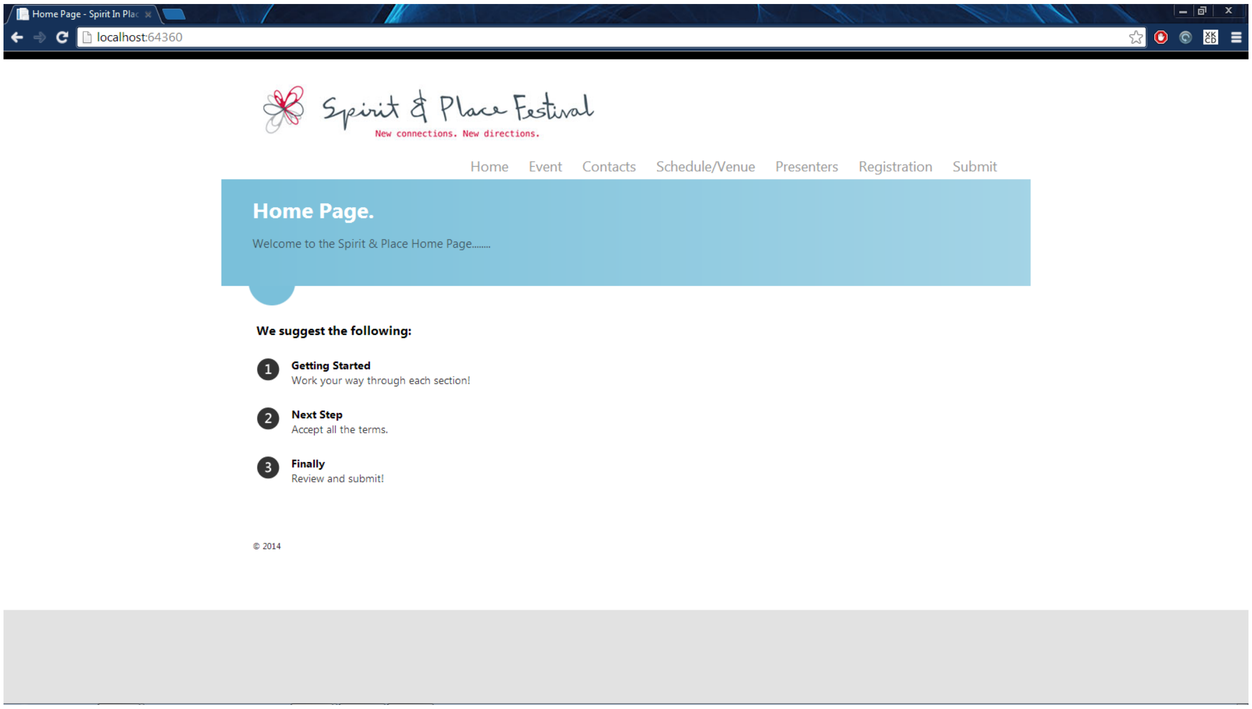
Please check all topics that apply to your event CONTENT (not presenters, venues, etc.) For example, a panel discussion on immigration includes several panelists, including a poet from Mexico. The poet will not be sharing poems, but rather sharing personal immigration experiences. In this instance, poetry would NOT be a selected topic. **Select only those topics that are part of the audience experience.** Call us if another category is needed, 317-278-3623.

□ African American Culture
□ Aging
□ American Indian Culture
□ Asian Culture
□ Baha'i
□ Buddhism
□ Christianity
□ Community Service
□ Competition
□ Creativity
□ Dance/Movement
□ Disabilities
□ Earth Based Religions
□ Film & Video
□ Flash Mob
□ Food
□ Games/Gambling
□ Gender
□ Global Affairs
□ Health & Wellness
□ Hindu
□ History
□ Housing
□ Human Rights
□ Humor
□ Interfaith
□ Medicine
□ Mental Health
□ Multiculturalism
□ Music
□ Oral History
□ Political Science
□ Public Policy
□ Science
□ Sexuality
□ Sikhism
□ Spirituality
□ Sports/Fitness
□ Storytelling

Our team wanted to improve the overall user experience for the Spirit and Place Festival application. By implementing time efficient ideas into our design concepts, users will be able to easily identify the process of filing out the application without explicit instructions written on the form. Not only is the current application lengthy, the overall appearance can create an undesirable view of the festival since it is so outdated.

What is the reason to have an updated web application? It represents the business as a whole, as a product of The Polis Center, customers could base their opinions of The Polis Center by their web application's appearance. Our team's overall goal is to reflect a positive image of the Polis Center through the revival of the Spirit and Place Festival Application.

6 New Application



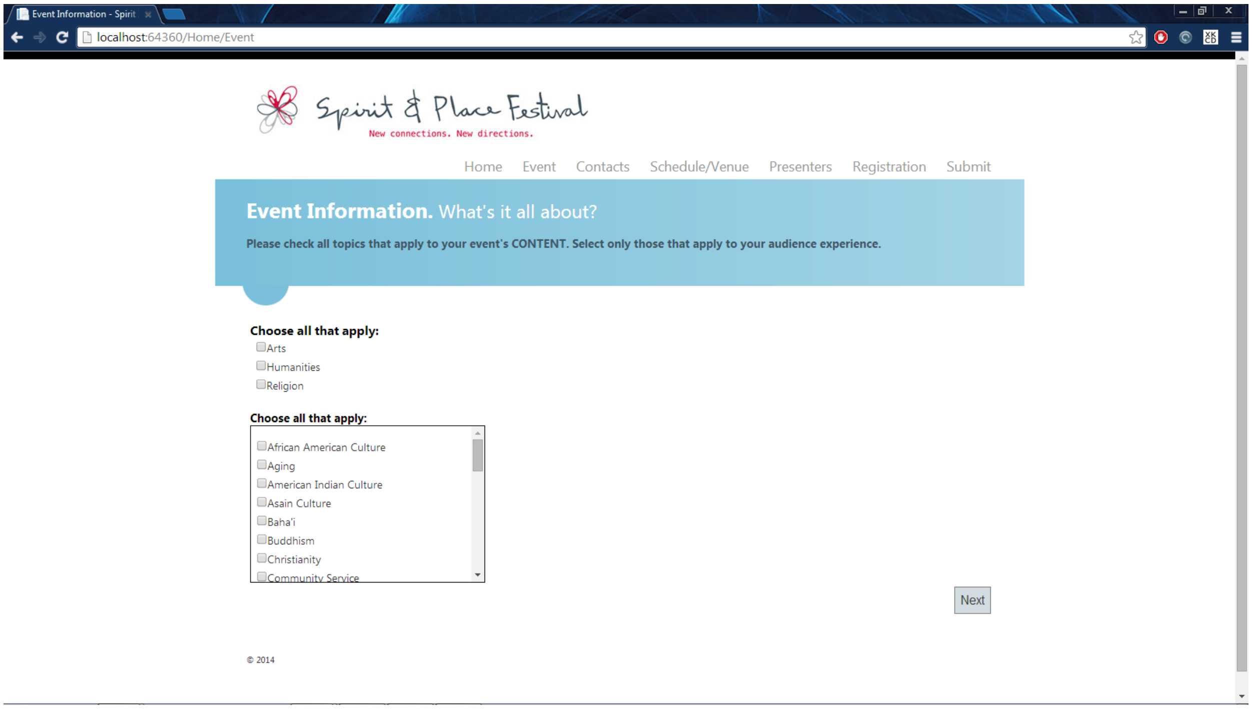
Home Page.

Welcome to the Spirit & Place Home Page.......

We suggest the following:

1. **Getting Started**
   Work your way through each section!
2. **Next Step**
   Accept all the terms.
3. **Finally**
   Review and submit!

© 2014



Event Information. What's it all about?

Please check all topics that apply to your event's CONTENT. Select only those that apply to your audience experience.

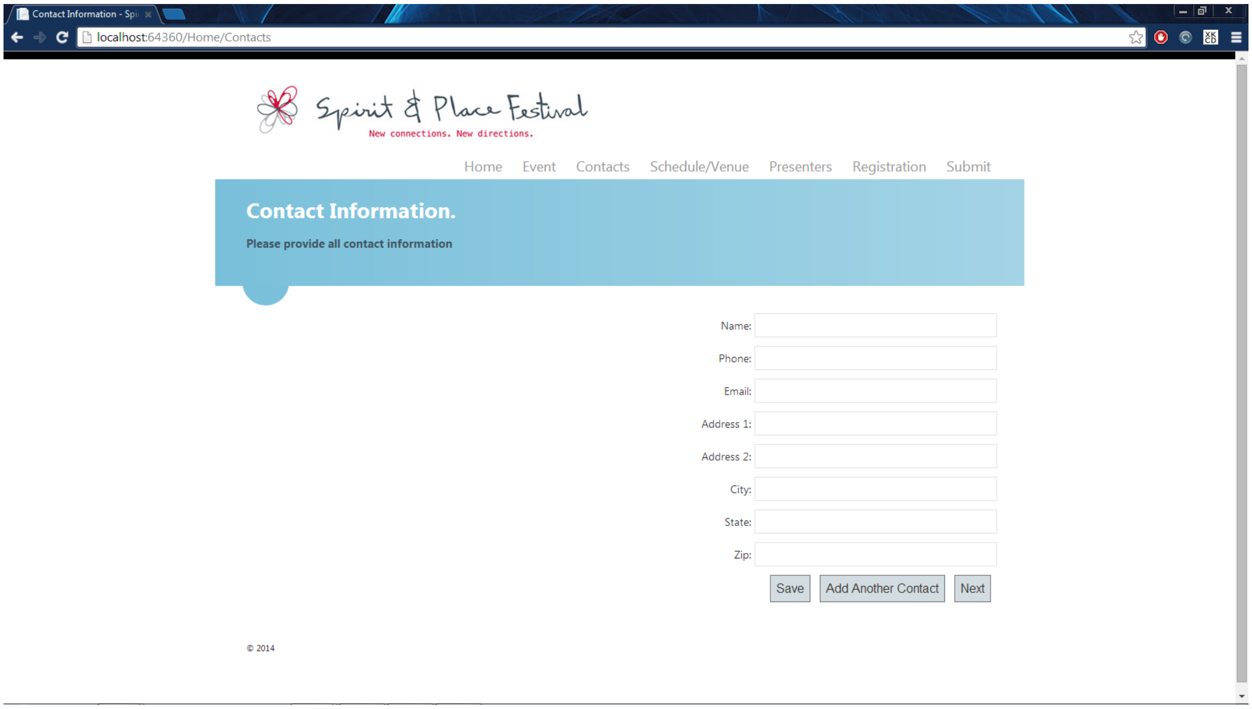
Choose all that apply:

□ Arts
□ Humanities
□ Religion

Choose all that apply:

□ African American Culture
□ Aging
□ American Indian Culture
□ Asain Culture
□ Baha'i
□ Buddhism
□ Christianity
□ Community Service

© 2014



Contact Information.

Please provide all contact information

Name:

Phone:

Email:

Address 1:

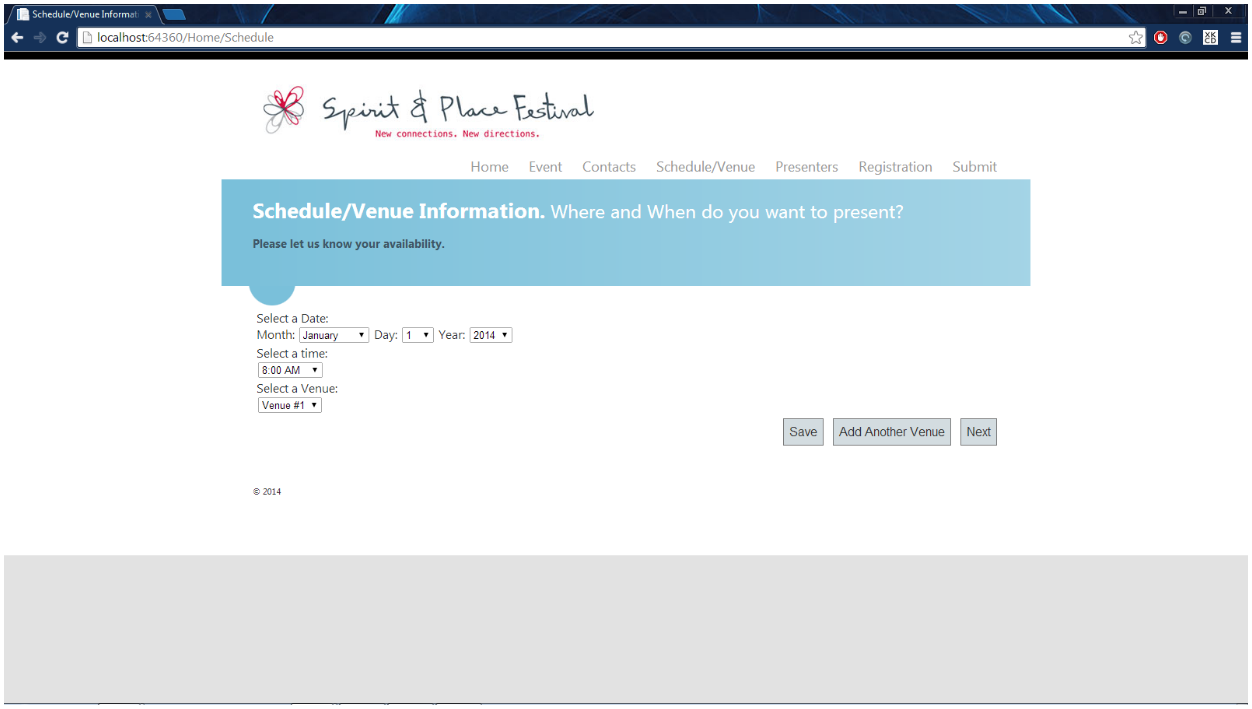
Address 2:

City:

State:

Zip:

© 2014

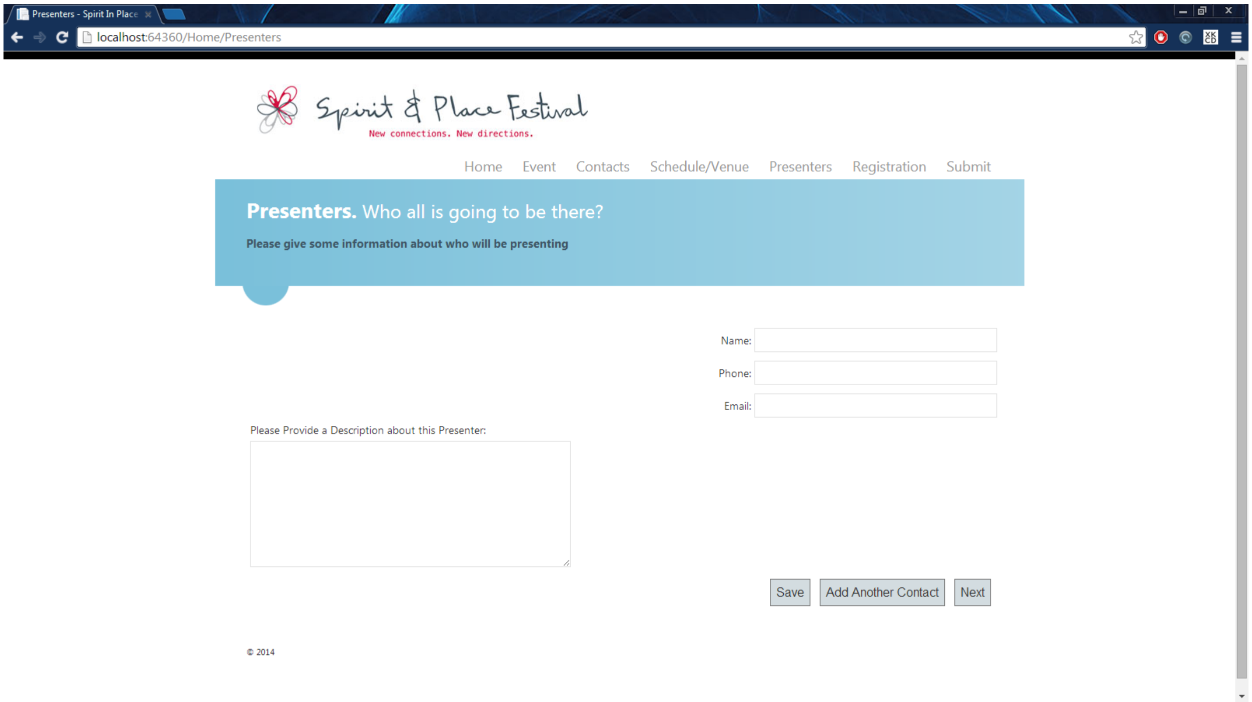


Schedule/Venue Information. Where and When do you want to present?

Please let us know your availability.

Select a Date:
Month: January Day: 1 Year: 2014
Select a time:
8:00 AM
Select a Venue:
Venue #1

© 2014



Presenters. Who all is going to be there?

Please give some information about who will be presenting

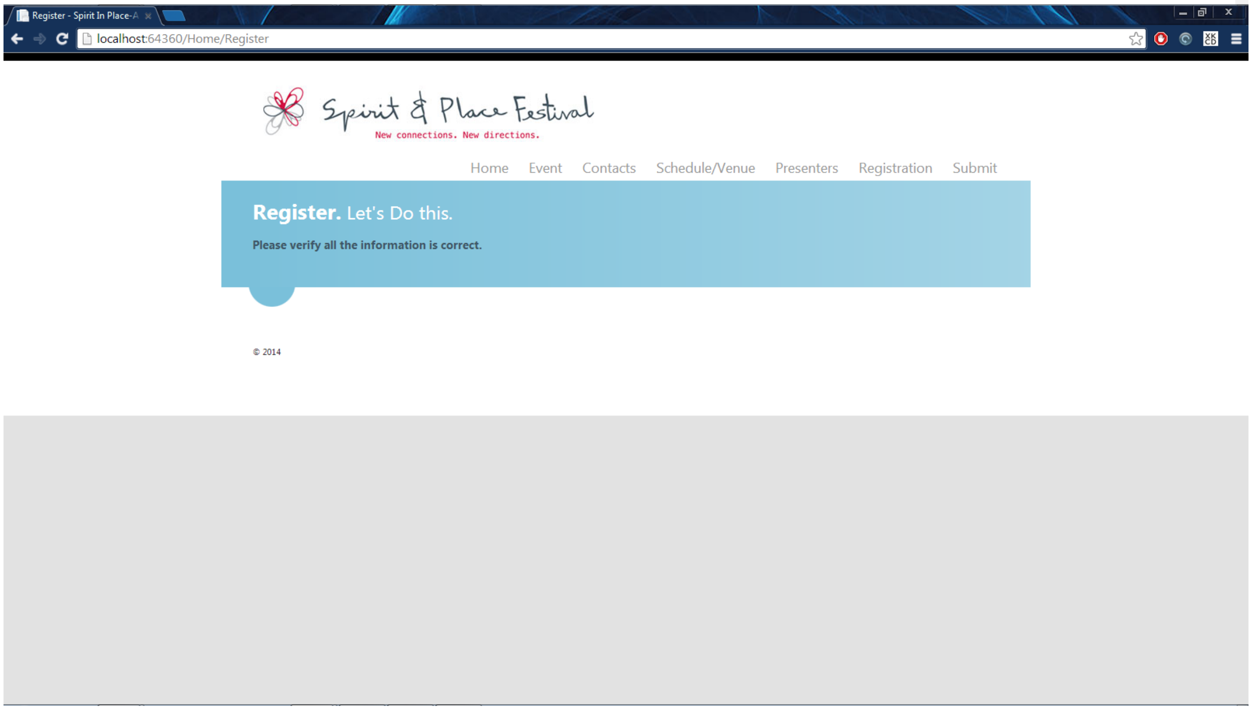
Name:

Phone:

Email:

Please Provide a Description about this Presenter:

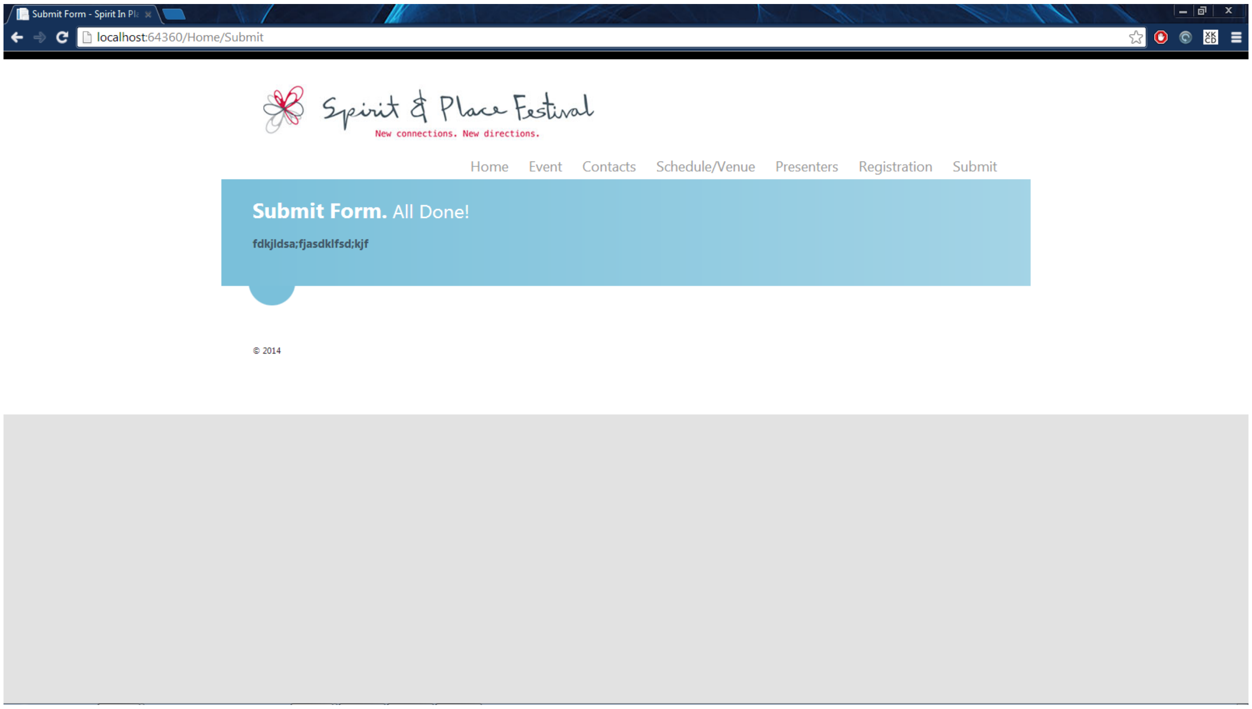
© 2014


Register - Spirit In Place-A
localhost:64360/Home/Register
Spirit & Place Festival
New connections. New directions.
Home
Event
Contacts
Schedule/Venue
Presenters
Registration
Submit

Register. Let's Do this.

Please verify all the information is correct.

© 2014


Submit Form - Spirit In Pl
localhost:64360/Home/Submit
Spirit & Place Festival
New connections. New directions.
Home
Event
Contacts
Schedule/Venue
Presenters
Registration
Submit

Submit Form. All Done!

fdkjldsa;fjasdklfsd;kjf

© 2014

7 Future Goals

While our team made substantial progress on the Sprit and Place Application project, due to time constraints, learning curves and scrapping the existing code, not all of the goals set were achieved. Listed below are the next steps for the next phase(s) of the Spirit and Place Application project:

1. Serving up primitives
2. Handle full events
3. Dependency injection
4. Route mapping
5. Run SpiritApi on another server
6. Work closely with front end developers and client to identify and complete needed functionality
7. Form designs can be more robust to account for various sized displays, etc.
8. Slight design modifications as necessary to ensure the application process is foolproof
9. Add code to allow user to smoothly move through pages ("Next" functionality)
10. Data validation

Our work will be uploaded to a shared folder accessible within the new EPICS website, including all code for the application forms.

Appendix

EventsController.cs

Figure 1 Handles HTTP requests to api/events

EventRepository.cs

Figure 2 Talks to database on behalf of EventsController

Event.cs

Figure 3 Carries information about an Event. Used by EventsController and EventRepository.

UsersController.cs

Figure 4 Handles HTTP requests to api/users

UserRepository.cs

Figure 5 Talks to database on behalf of UsersController

User.cs

Figure 6 Carries information about a User. Used by UsersController and UserRepository.

UserSecretInformation.cs

Figure 7 Carries Password and SecretAnswer members, associated with a user. Separated from User for security reasons, but there may be better solutions to this problem.

QueryResult.cs

Figure 8 Used for serving up integers to a client application. Used in UsersController and EventsController. There may be better and more standard solutions to this problem.

DataReaderExtensions.cs

Figure 9 Contains functions I created to facilitate using the DataReader object, including preventing exceptions thrown when attempting to access a null attribute as well as creating Users and Events from a DataReader containing attributes for these objects.

alter UpdateUser.sql

Figure 10 Changed the UpdateUser stored procedure to return the attributes of the updated record

create GetMiniEventsById.sql

Figure 11 Takes an EventID and returns a partial record of an event that has the same ID as the argument

create InsertMiniEvent.sql

Figure 12 Takes an event object (represented as SQL parameters), inserts that event into the Events table, and returns the record for the event, including the new, automatically generated EventID.

create InsertUser.sql

Figure 13 Takes a User (represented as SQL parameters), inserts that user into the Users table, and returns the record for the user, including the new, automatically generated UserID.

create UpdateMiniEvent.sql

Figure 14 Takes an Eventobject (represented as SQL parameters), finds the Event record with the same EventID, and updates that record’s attributes to match those of the SQL parameters. It returns the updated record.

create UpdateUserPasswordAndSecretAnswer.sql

Figure 15 Takes a UserID, a Password, and a SecretAnswer. Updates the User record with the same UserID as the argument by setting its Password and SecretAnswer attributes to match the arguments.

WEEKLY STATUS REPORT (WSR)
2/17/2014

TO:	David Harting, Stephanie Boylan, Panos Linos
FROM:	Jason Rowe
SUBJECT:	Status report for week 2.9.14 – 2.15.14

I. RED FLAGS:

II. ISSUES:

- Have no contact information for Client
- Files are not linked properly on Wiki

III. ACCOMPLISHMENTS:

2.10.14

- Wiki page has been updated with team picture

IV. ACTION ITEMS FOR FOLLOWING WEEK (2.16.14 – 2.22.14)

- Contact client to begin discussing goals. (Jason)
- Continue exploring Gitbash for collaboration. (David)
- Review previous team dossier and gain solid understanding of what has been done, and what was left undone from initial expectations. (David, Stephanie, Jason)

WEEKLY STATUS REPORT (WSR)
2/24/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for week 2.16.14 – 2.22.14

I. RED FLAGS:

II. ISSUES:

- Small issues ongoing with wiki

III. ACCOMPLISHMENTS:

- Client has been contacted and a meeting is scheduled for 2.26 at 4pm

IV. ACTION ITEMS FOR FOLLOWING WEEK (2.23.14 – 2.29.14)

- Meet on 2.26 with client. (All)
- Continue exploring Gitbash for collaboration. (David)
- Review previous team dossier and gain solid understanding of what has been done, and what was left undone from initial expectations. (All)
- Begin work based on outcome of client meeting. (All)

WEEKLY STATUS REPORT (WSR)

3/3/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for week 2.23.14 – 3.1.14

I. RED FLAGS:

II. ISSUES:

III. ACCOMPLISHMENTS:

- Meeting with client went well.
- Team assignments have been made for ongoing work.

IV. ACTION ITEMS FOR FOLLOWING WEEK (3.2.14 – 3.8.14)

- Begin researching asp .net and other methods we may use for database work. (David)
- Create prototype forms based on previous team's mockups. (Stephanie)
- Contact Nate Partenheimer concerning database used last semester, how to access, etc. (Jason)

WEEKLY STATUS REPORT (WSR)
3/17/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for week 3.10.14 – 3.16.14

I. RED FLAGS:

II. ISSUES:

III. ACCOMPLISHMENTS:

- Updated wiki.
- David has made good process with an application to work on database access.
- Steph has begun work on application design.

IV. ACTION ITEMS FOR FOLLOWING WEEK (3.17.14 – 3.23.14)

- Teach me and Steph what you've learned about database work that we'll need to know. (David)
- Continue working on application design. (Stephanie)
- Contact Nate Partenheimer concerning database used last semester, how to access, etc. (Jason)

WEEKLY STATUS REPORT (WSR)

3/24/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for week 3.17.14 – 3.22.14

I. RED FLAGS:

II. ISSUES:

III. ACCOMPLISHMENTS:

- Updated wiki.
- More progress on application design.
- Talked to Nate to make the three of us local and SQL admins on the server spirit.butler.edu to access copy of database.
- Planning team meeting for Wednesday in order to work together on what we've accomplished and plan the next steps.

IV. ACTION ITEMS FOR FOLLOWING WEEK (3.24.14 – 3.30.14)

- Meet Wednesday.
- Develop next action items during meeting.

WEEKLY STATUS REPORT (WSR)

4/14/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for weeks 3.24.14 – 4.13.14

I. RED FLAGS: None

II. ISSUES: NONE

III. ACCOMPLISHMENTS:

- Updated wiki.
- Application forms nearing completion.
- Team Meeting planned for this Wednesday.

IV. ACTION ITEMS FOR FOLLOWING WEEK (4.14.14 – 4.20.14)

- Meeting Wednesday to put together what we’ve done so far and begin coding prototype.
- Contact client to set up presentation date and time (4.21 or 4.23).

WEEKLY STATUS REPORT (WSR)
5/2/2014

TO: David Harting, Stephanie Boylan, Panos Linos
FROM: Jason Rowe
SUBJECT: Status report for weeks 4.14.14 – 5.2.14

I. RED FLAGS: None

II. ISSUES: NONE

III. ACCOMPLISHMENTS:

- Met with client and gave final presentation
- Assembled dossier and uploaded to team Wiki

IV. ACTION ITEMS FOR FOLLOWING WEEK (5.2.14 – 5.8.14)

- Complete poster.