# Spirit Quest Design Documentation

**Forrest Wilcox**
**Gareth Bosch**

## Overview

Spirit Quest is a web application built with MySQL, PHP, HTML, CSS, and Javascript, and a cross-platform mobile native application built with Xamarin Forms. Both applications deliver the Spirit Quest game. The purpose of the game is for users to complete trivia and scavenger hunt style quests around UAA campus to learn about the school and its resources. Players also compete with fellow students by earning Spirit Points and Badges.

Our project consists of improving Spirit Quest in two ways: building web forms to allow game writers to easily add content to the game database, and building a prototype for the mobile application as a proof-of-concept.

### Web Form for Game Writers

Spirit Quest is a joint effort between UAA's Computer Science Department and English Department. Computer Science faculty and staff write the web application and database while the English faculty write the game story content and design the quests. The writers have designed and written the game's stories in a hierarchical manner with two levels. The broadest category is the Questline; there are only a few Questlines and each one covers an important resource or part of UAA campus--like the Consortium Library. Each Questline consists of many smaller Quests; these are the individual tasks a game player must complete by going to a destination or answering a question.

In the game's MySQL database, Questlines are contained in a table which is in a one-to-many relationship with the Quests table. The web form designed and built for this project allows game writers to add and delete Quests from a given Questline. To do this safely, the form maintains database referential integrity by having the writer first select which Questline they are going to delete or add a new Quest to. When writing a new Quest, all fields in the database table are accounted for in the form: the number of tasks this Quest will consist of, how many times it can be repeated for points, Quest title, task type, completion required, question or task, answer or goal, and points awarded. Once all these fields are filled in the writer may submit the form (or clear it) and the new data is inserted into the Quests table. When the web game is refreshed it will show the new content.

### Mobile Application

With the mobile app, a user launches the app and is taken to a “Start” screen to start their UAA Spirit Quest. After starting, the user must complete the quests in order to be able to progress to the next step in the quest. Like the web app, Spirit Points are awarded on quest completion and badges are awarded on Questline completion. The mobile app also makes use of the smartphone’s GPS device, therefore GPS checking can be used to verify if a user is actually located at the quest’s objective, for instance determining if the user is in the UAA Library, and checking the GPS of the user to check if those GPS coordinates match.

The design is a straightforward quest completion approach, similar to a “To Do” like application. A user can complete a quest only if the user has completed the previous required quests. Upon quest submittal, the application checks the user’s input (for example a multiple choice answer, free form text answer, or GPS location) and shows a dialog to the user whether the answer was correct or not. If the answer is incorrect, an incorrect message is shown and the user can change their answer and submit again. If the answer is correct, the user is alerted that their answer was correct and the user is taken back to the questline page.

## Description

Basic overall application log flow is that Spirit Quest is a Create, Read, Update, Delete app (CRUD) that lets users complete Questlines that have individual Quests to earn Spirit Points, then eventually earn Badges.

### Web Form for Game Writers

The form for adding Quests from the Spirit Quest database is written in a PHP file called addquests.php. The larger Spirit Quest code base, written by Professor Kenrick Mock, supplies database connection handling in the file includes.inc. Each field in the Quests table is represented in the form as an input field. See the Add Quest form for game writers in figure 1 below.

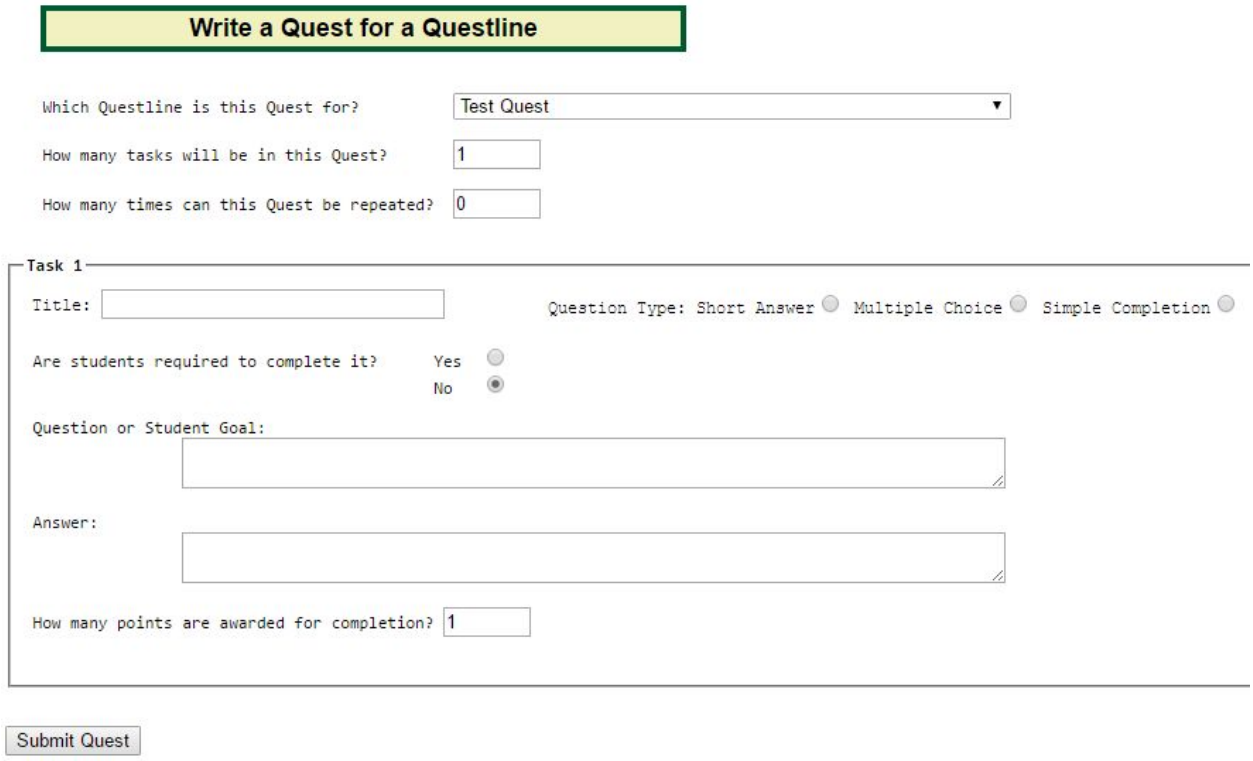


Figure 1: Add Quest form

Flow of user inputted data after user submits form:

```
populate form with Quest data
click Submit Quest
foreach field1-i
    Superglobal_array ← data in fieldi
if Superglobal_array has values set for our variables
    scrub HTML entities from each
    display on webpage
else wait for form post
if Superglobal_array has values set for our variables
    scrub MySQL escape strings from each
    insert values into corresponding database table fields
else wait for form post
```

**Mobile app**

Logic flow for checking the answer for the mobile app:

```
User clicks submit quest
User's input is checked against a hard coded value in the app
If user.input is correct:
     Display a dialog with Success! Quest completed.
     User's progress is saved to the database so the user can
     Start the next quest in the Questline
Else:
     Display an alert informing the user that their answer was
     incorrect.
```

Several GPS locations are used for the mobile app, and the tolerance for checking whether the user is located near these GPS locations is around 200 meters around the target.

a CSE Engineering Building (NAD83 GPS Coords: 61.19100 -149.82390)
b UAA Commons Square/Ground (NAD83 GPS Coords: 61.18928 -149.83205)
c UAA Registrar Office (NAD83 GPS Coords: 61.18512 -149.86519)

## Mobile App Flow


Figure 2: General flowchart of the mobile app

Description of the flow pictured in Figure 2:

Step 1: User Start Screen
Step 2: Questline Screen
Step 3: Questline's quests Screen
Step 4: Quest details containing text input or other verification checks for user to complete the quest and progress further in the questline.

# Details

As noted before, there are some major differences between the two applications since one is web based and the other is mobile based.

The mobile app uses a local SQLite database to track user progress and does not interface at all with the MySQL database the web app uses. The mobile app could have used the MySQL database by implementing an API from the mobile app to the MySQL database, however that was out of scope.

### Web Form for Game Writers

To organize the form fields on the webpage, a table is used with a fieldset inside it. Fieldset borders are used to separate each task within a Quest from one another (since there can by multiple tasks inputted into the form. The Submit Quest button appears just below the last fieldset. Under that is the display of the record that was just entered.

The form:

- Method attribute: `post`
- Action attribute: current server - `$_SERVER['PHP_SELF']`
- Input type attribute: `submit`

The fields:

- Questline
    - The Questline to which the currently entered quest belongs to
    - Selection list
    - List of Questlines populated from function `getQuestlineTitles()`
    - Must be selected to submit a form to protect referencial integrity
- Number of tasks
    - A Quest may have multiple subparts, called tasks
    - Number field

    - Min 1 - max 99
- Number of times the Quest is repeatable
    - Quests may be completed multiple times for points
    - Number field
    - Min 0 - max 99
- Task title
    - Text field
- Task question type
    - Multiple choice, short answer, or completion (any answer will do)
    - Set of three radio buttons
- Task completion required
    - Some tasks must be completed by the player to finish the Quest
    - Yes or no
    - Set of two radio buttons
- Task question or goal
    - Text area
- Task answer
    - Text area
- Task points awarded
    - Number field
    - Min 1 - max 99

The functions:

- `cleanHTML($var)` & `cleanMySQL($var)`
    - Protects against malicious users
    - Strips field input of HTML entities and MySQL escape strings
- `getOneQuestlineID($q)`
    - `$q` is a Questline title
    - Function queries the Questlines table for that primary key
    - This key is used as a foreign key for the new record created in `insertIntoTable()`
- `getQuestlineTitles()`
    - Queries the Questlines table for all titles
    - Returns an array of strings
- `checkForInput()`
    - Checks if the form has submitted data to the server
    - If the `$_POST` superglobal array has data for all fields, displays data
    - Scrubs data with `cleanHTML($var)`
- `insertIntoTable()`
    - Insterts the submited data into Quests table
    - Scrubs data with `cleanMySQL($var)`

**Mobile app**

Xamarin Forms was used to create a portable shared mobile application that will work on iOS, Android, and Windows Phone UWP (Universal Windows Platform). The local SQLite3 database schema/design also differs greatly from the web app's database because portability is in mind, and interfacing with a remote database like what the web app uses was not implemented.

The SQLite database that stores information about the user's progress is detailed below:

**User**

`UserName (string, primary key)`
`LastQuestStepCompletedID (int)`

- Keeps track of the user's last quest step completed to track the users progress for completing quests.

`LibraryBadgedEarned (bool)`

- badges are earned when the last Quest (step) is completed for a Quest line and the user earns the badge.

`CampusTourBadgeEarned (bool)`

For accessing GPS locations, the GeoLocator plugin for Xamarin was used.

The mobile app logic itself usually contains a few methods per page:

`Init()` - sets up the form and logic for the form (`InitializeComponent()`)
`OnQuestSubmit()` - checks the user's input to verify if the answer was correct.
`GoToPreviousPage()` - goes to the user's previous page

For Android, API level 23 was targeted, iOS v10, and UWP was the latest at Windows Phone UWP v8.1.