

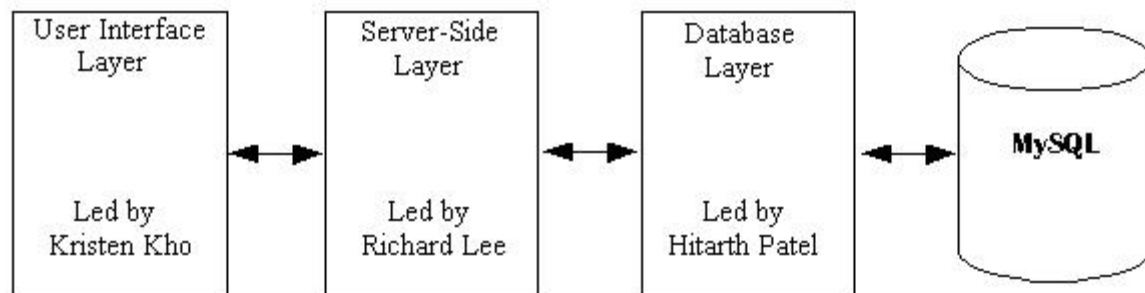
Software Design Spec Shumoku 種目

Hitarth Patel, Richard Lee, Kristen Kho

May 16, 2008
CSE 210

Overview

Shumoku is constructed in a layered architecture that is more commonly known as the three-tier architecture. It consists of a User Interface (Presentation Tier), Server-Side Layer (Logic Tier) and Database Layer (Data Tier). Each team member leads one of the tiers: Kristen - User Interface, Richard - Server-Side, Hitarth - Database. Each lead does not necessarily implement the whole layer, but is responsible for its delivery and asking for additional support when required.



Risk Assessment

- Modular design and data-hiding will be a challenge for us. Ideally, in most cases, a change in one layer should not affect the other two layers.
- Interfacing between layers will also be a challenge for us. Requirements that go across multiple tiers will be difficult to integrate since each component may complete the feature at a different time. Thorough unit testing will be required. Communication between members will be very important as labor/components will be divided up.
- Requirements may rapidly change. Implementation of new or modified requirements may require design changes that affect multiple layers.
- SVN has been a challenge to set up. Right now, we are storing the code in one account, but we would like multiple accounts to be able to access it at the same time.

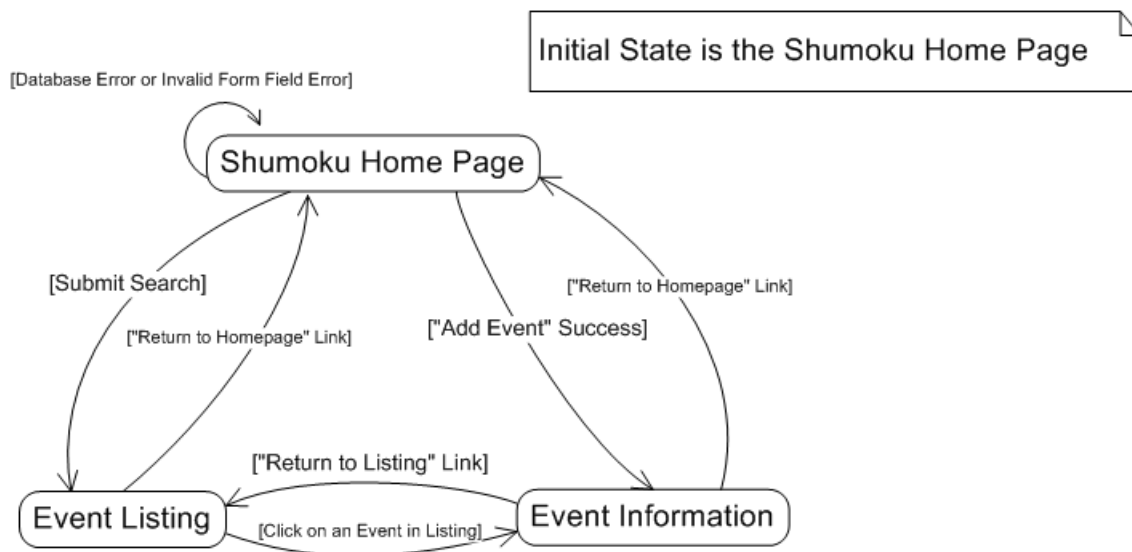
User Interface

The Shumoku website provides an interface between the user and the Server-Side Layer. It collects input from the user through the use of form fields and calls functions provided by server-side PHP scripts. The results are then presented back to the user. Currently, the user has two options: "Find Events" and "Add Event". The user is only required to fill out a minimum amount of information for each option but may enter additional information if desired.

The user interacts with the website by filling out form fields. We currently use several different form components: buttons, drop-down fields (multiple and single selection), radio buttons, check boxes, text fields, and text areas. More advanced components include a calendar date selector and an accordion, which are written in JavaScript. JavaScript is also used to validate form input data before sending off the content to the server.

Our design goal for the User Interface was to make it as simple and intuitive as possible. Our choice of components allows us to present only the information that is necessary to the user. Our form design allows the user to fill out only the minimal amount of data required to complete a task but gives the option to fill out more when required. For example, the user can perform an event search simply by entering her location. However, she can refine her search by specifying a category, date range, etc. Similarly, when adding an event the user only has to specify an event name, time and location to submit it, but may include additional information such as a description, contact information, etc. The optional information is hidden until the user clicks on a "More Options" button.

Since the forms for searching and adding events are on the same page, there are only 3 possible pages or states. The second page is the event search results page. It contains a map and listing of all events that meet the user's criteria from the event search form. The third page is the event information page which the user is directed to after successfully creating an event or clicking on the event from the search results page. The user may also view this page if they wish to preview their event before adding it.



Languages

PHP, HTML, JavaScript, CSS

Risk Assessment

- Implementation of our User Interface design is fairly straightforward so on-time delivery is not an issue. Superficial changes and additional form fields may be added relatively quickly. However, we may find that our idea of how the user will interact

with the interface may be different from how users act in real situations. Therefore, usability testing is required in each iteration.

- Currently, the User Interface calls stubs to functions in the Server Side Layer. This allows us to make sure that these components are communicating properly. When the Server Side Layer is complete, we must make sure that the information communicated is correct for the actual integration.

Screenshots

The homepage screen - by default the 'Find Events' accordion is open:



Find Events

Enter your city (required):

[More Options](#)

Add Event

What is Shumoku?

Copyright © 2008 Team Shumoku. All rights reserved.

The homepage screen - after the user has clicked on "More Options":

Find Events

Enter your city (required):

[Less Options](#)

Optional search fields:

Enter the zipcode:

Enter the event name:

Specify the date range:
Start:
May 15 2008

End:
May 15 2008

Enter the name of the venue:

Select one or more categories (CTRL+Click to select multiple categories):

- Party
- Cause
- Education
- Meeting
- Music/Art
- Sports
- Trips
- Other

Select a cost range:
anything

Other flags:
Catering/free food Raffle/free stuff

Add Event

What is Shumoku?

Copyright © 2008 Team Shumoku. All rights reserved.

Server-Side Layer

The system's server-side layer contains most of the domain logic. The UI and Database communicate through this layer. This layer also performs actions such as sending email and address translation using Google Map geocode service.

This layer's current implementation contains two classes, Event and Event Processor. The Event class represents a single event. An Event instance can be constructed and passed to the EventProcessor for further action. EventProcessor currently performs four functions: adding an event, getting events, sending email and address translation.

Technologies

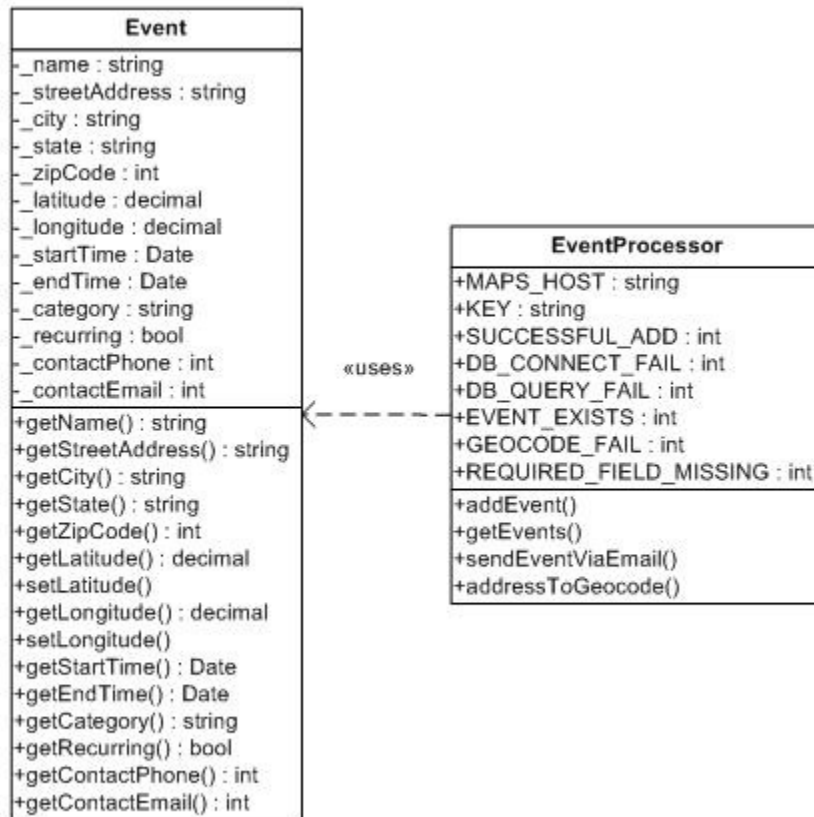
PHP, Google Maps Geocoding, Other third-party services (text messaging, Outlook?, Facebook Developer?, MySpace?)

Risk Assessment

- The server-side layer will tend to change a lot in the future as additional requirements are implemented. This layer must be flexible to the addition of new classes (like Location and EventInfo) and functionality (like Outlook reminders and text messaging).
- As the system grows, unit testing will become more important as integration may lead to breaking of previous functionality. Test drivers are already being written for each procedure in this layer, so our code can be easily tested and maintained. Testing can take place with the absence of a GUI. Test classes will be based on the PHP SimpleTest framework, which is similar to JUnit.
- Since the server-side layer is also an interface between the GUI and Database, it is important that we hide details about each of the other layers. For example, the GUI should not be able to see or modify the query being done to add or get events. In our prototype, we did not separate this, which we found made it harder to change and maintain.
- While developing communication between the GUI and server-side layer, we found that the GUI would need specific information regarding errors that may occur while the server or database performed actions. This led to the addition of error codes to our Error Page requirement so that the UI could display meaningful messages to the user. These error codes included Database Connection Failure, Database Query Failure, and Event Already Exists in the System, Geocode Translation Failure and Required Fields Unfilled.
- Our Text Messaging may be at risk of being moved down as an additional requirement since we cannot find proper modules to implement this functionality. One possible implementation is to do a third-party hack, further risks analysis and research must be done before making a decision on this feature.

Diagram

The EventProcessor takes in Event to add or send via email. It also returns events to the GUI layer.



Database layer

For iteration 1, the Database will only be used for storing and retrieving event information on the Shumoku_Event table. In future iterations, more tables will be added to support additional features.

Tables

- Shumoku_Event - stores event details such as (EventName, Category, Location, DATE, TIME etc. (Ready for iteration 1)
- Sumoku_User - stores user information and events created by the user.
- Shumoku_friends - stores user's friends' information, so Shumoku can displays which friends are going to same event.
- Shumoku_user_events - User can create private events and invite friends (Private events will only be displayed to User's Friends).
- Shumoku_Rideshare - Friends can share ride to attend an event.
- Shumoku_LOGs - To log user activity (sending emails or text messages) and number of views per event.

Languages

PHP, MySQL

Requirements

- Database driver will be implemented in PHP and it must hide database specific information from server.
- It will only provide methods for storing and retrieving information about events.
- Testing database drivers.
- Return valid error codes to the caller and document the schema scripts.
- Aging- Create DB script that will remove past events from Shumoku every day/week/month.

Risk Assessment

- On-time delivery: - Shumoku schema and drivers must be delivered on time for integration testing.
- Slow data retrieval: - Data retrieval needs to be efficient for event search.
- Schema is hosted on 3rdParty server: - If the sever is down, Shumoku will not be available to users.
- Limited functionality- Given limited time, Shumoku may not include some features in first release.
- Design change: To support additional functionality, the schema design will be changed.