1.0 Design Overview

The customer required that the software solution be a Web-based application that uses standard technology in widespread use. This requirement heavily influenced our design decisions.

Web-based applications are typically Java Applets, or dynamically generated HTML documents. The latter was chosen for its simplicity and lightweight components. Several platforms exist to ease the development and maintenance of Web-based applications based on dynamic HTML. Java Server Pages (JSP) was chosen as a base technology for numerous reasons:

- It is a proven technology that is often used for this class of applications.
- Uses the Java language which offers strong typing and an assortment of pre-made libraries.
- Is in use with numerous other service providers such as DoIT.

A JSP page is an HTML document with Java code embedded inside. A JSP Container will compile the page into a servlet automatically. The servlet is executed whenever a web browser requests the corresponding JSP page, and that servlet processes input from the web browser and dynamically generates an HTML page returned to the browser.

There are numerous commercial packages that offer support for JSP. Apache Tomcat was chosen because it is freely available and in widespread use. However JSP pages are usually container independent thus a different choice of server/container platform could be used with little effect on the design and implementation.

Finally data persistence is achieved by using an SQL database. For development purposes, the PostgreSQL database system will be used however the design nor implementation will not require this choice of database platform. Any RDBMS system that has a suitable Java Database Connector (JDBC) driver can replace PostgreSQL with very little effort.

2.0 Components

High-level components are shown in Figure 1. Choice of components was influenced by the following goals:

- Off-the-shelf components are used wherever possible. This reduces development effort, cost, and maintenance.
• A web designer should be able to update the user interface without detailed knowledge of the system design and implementation.

• The server should maintain very little state about each transaction. This reduces the complexity with regards to race conditions and makes the system more scalable.

• Basic object-oriented design should be followed.

The client will use a standard web browser, such as Internet Explorer or Mozilla, to access the system. Apache Tomcat will receive requests from the browser and invoke the corresponding servlet. The servlet will carry out the necessary actions and build a page that is returned to the user.

JSP pages only contain code related to the user interface. Application logic is performed by “handler” classes. This is a conscious effort to abstract important parts of the system away from the design of the HTML and user interface. In essence, it is desired to prevent any future user interface developer from accidently misusing, or compromising the system specifications.

A “permissions oracle” is associated with every user logged into the system. Before a handler carries out an action, it consults the permissions oracle to determine the validity of the action.

Database wrapper classes also exist such that records can be accessed and used in a uniform way by the handlers. A freely available tool, called SQL2JAVA, is a relational-object entity mapper. It will machine-generate classes to represent every table in the database. These classes use the standard JDBC database driver to access the PostgreSQL database. The data wrapper classes are also responsible for validating any updated data entered by the user.

Finally, separate components must exist to authorize and handle credit cards and university IDs. These are assumed to exist outside of the system and would be provided by the service provider. For example, DoIT has both services available for web-based applications. For the development of this system, these services will be simulated.
3.0 Security

This section outlines the security policy that the system will follow. The policy is broken into four parts: preventing tampering with the system software, limiting access to the database, secure client communication, and validating requests. Each part is covered in a subsection. See Figure 2 for an overview.
3.1 Preventing system tampering

No system can be secure if malicious entities are allowed to tamper with trusted system components. Similarly, security is difficult if access is allowed to backend components. The simplest and most common solution to this problem is to require that the computers hosting the trusted components are secure. This system makes the same requirement. All of the components in the system, except for the client (web browser), are trusted and need to be protected by the administrators of those systems. The only access to the trusted part of the system by untrusted parties is through HTTP requests. All computers hosting server components must be secure to access by untrusted parties.

3.2 Limiting access to the database

The database may be hosted on a different computer from the web-server program and other server components. The database server must be secure as per section 3.1. Additionally, the database server must be on a trusted network and no database connections are allowed from other networks. Since the network is trusted, the system can access the database without encryption. Some commercial database systems provide secure authentication and connection mechanisms, and that is an acceptable alternative for untrusted networks provided that the JDBC driver for that DBMS uses those mechanisms.

3.3 Secure client communication

Communication with the client is dangerous since the communication goes over untrusted networks. This opens the system up to replay attacks, packet sniffing, and an assortment of
other attacks. To secure communication with the client, Secure Socket Layers (SSL) must be used to encrypt all communication between the client and the web server. Encrypting the communication provides privacy and also ensures that the cookie, used to identify the user’s session, is secure while in transit.

### 3.4 Validating requests

With our chosen JSP container and web server, Tomcat and Apache, sessions are tracked using a small cookie. The cookie is secure in transit and the cookie to session mapping is handled by Apache and Tomcat in a secure manner. This section describes how, given the session for a request, the system ensures that the request is permitted.

When the user initiates a session, they must log in by providing their username and password. The system then will hash that password and query the database to find the id of any matching user. If such an identifier is found, a new permissions oracle is created for that user id and a mapping is made from the session id to that permissions oracle object.

To validate a request then, the system must check that the request has a session, the session has a permissions oracle object, and finally that the permissions oracle object grants permission to handle the request. This is important since, with a dynamic HTML web application, it is not sufficient to rely on the user interface (UI) to prevent access to restricted parts of the system. A malicious user could fabricate a request that accesses part of the UI that they would otherwise not have access to.

The particulars of capabilities granted based on different types of permissions is not specified here, but must be consistent with the permission roles and functional requirements specified in the requirements document for this system.

### 4.0 Data Model

The persistent data tracked by the system is shown in the entity-relationship diagram, Figure 3. This section will provide a brief description of each entity and relationship, as well as their fields.
4.1 club

Club entities represent the different Hoofers clubs. This should be fairly static data, and currently is just a mapping from a unique id to descriptive club name.

id, integer: This is a unique id used to identify the club internally.

name, varchar: This is the display name of the club.

4.2 uwid_type

University id numbers are one of four different types: student, graduate student, faculty/staff, or union. This table maps a unique id to each of the four types. This is static data.

id, integer: unique id used to identify the type internally.

name, char(15): display name of the type.

4.3 puddles_user (user)

This entity is used to store general user account information. There will be one row for each registered user.

id, integer: unique id used to identify the user internally.

user_name, char(50): unique name used to log into the system. (shown as “Login Name” in diagram)

password, char(28): user’s hashed password used when logging in.

uwid, char(10): user’s University id number (5 or 10 digits).

uwid_type, integer: id of the uwid_type entity that describes this user’s University id.

union_expr_date, date: If the user’s University id is a union type id, then this is the date that their union membership expires.

last_name, varchar: This is the user’s last name.

rest_of_name, varchar: first part of the user’s name.

address1, varchar: first line of the user’s street address.

address2, varchar: second line of the user’s street address.

city, varchar: city part of the user’s address.

state, char(2): two letter code for the state part of the user’s address.

zip_code, char(10): zip code part of the user’s address.

phone, char(10): user’s phone number.

email, varchar: user’s email address.
email_opt_out, boolean: This is true if the user only wishes to receive emails about updates to their account.

contact_name, varchar: name of the user’s emergency contact.

contact_number, char(10): This is the phone number of the user’s emergency contact.

suspended, boolean: true if the user has been suspended from the entire system.

4.4 membership_option

Membership options are used to define the different cost/duration options available to users when buying club memberships. Each club has its own set of these options.

id, integer: unique id used to identify the option internally.

club, integer: identifies the club entity that provides this option.

name, varchar: descriptive name used when displaying the option.

start_date, date: Any user buying a membership using this option will become a member starting on this date.

end_date, date: Any user buying a membership using this option will have their membership expire after this date.

available, boolean: If this is false, users will not be allowed to use this option to buy membership even if they otherwise could.

uwid_type, integer: Only users that have a University id of this type are allowed to buy membership using this option.

cost, float: This is the price, in dollars, that the user must pay to buy membership using this option.

4.5 permission_option

Permission options are the different permission roles allowed by the system, such as Club-level Super User or Clerk. This is static data.

id, integer: unique id used to identify the option internally.

name, varchar: the option’s display name.

per_club, boolean: true if the role provides privileges in the context of a club and false otherwise.

4.6 rating_option

Rating options are the different types of skills or certifications that users can be rated on. Each club has its own set of these options to rate its members on.
id, integer: unique id used to identify the rating option internally.
name, varchar: descriptive name used when displaying the option.
club, integer: identifies the club entity that the option belongs to.
levels, integer: indicates what levels of skill are allowed for this option. A level of zero (unrated) is always allowed, so this must be a positive number. The possible skill levels are then 0 <= skill level <= levels.

4.7 activity_type

There are currently three different types of activities: trips, classes, and equipment. This table provides identifiers for those types.

id, integer: unique id that identifies the type internally.
name, char(20): descriptive name used when displaying the type.

4.8 activity

Activity entities represent general schedulable activities that the clubs can provide and users can enroll in, such as classes or trips. Each activity is sponsored by one club and has one user that leads the activity.

id, integer: unique id used to identify the activity internally.
club, integer: identifier of the club entity that is sponsoring the activity.
creator (leader), integer: identifier of the user that will lead the activity and act as the contact person for that activity.
name, varchar: display name of the activity.
summary, varchar: one-line summary of the activity.
description, varchar: multiple-line description of the activity which can either be plain-text or contain HTML tags.
activity_date, DATE: date that the activity is scheduled to begin on. This is mainly intended for display purposes.
join_by_date, date: last day that users are allowed to register for the activity.
cost, float: price, in dollars, that users must pay to enroll for this activity.
max_capacity, integer: number of users that are allowed to enroll in the activity.
ip_restriction, varchar: Some future uses of the system will require the ability to restrict activity enrollment requests to particular client computers. This lists the one IP address that is allowed.
to submit enrollment requests, or it is null to allow requests from any client computer.

mem_required, boolean: if true, then enrollment is restricted for this activity to users that are members of the club that sponsored this activity.

activity_type, integer: the identifier of the activity_type entity that typifies this activity.

4.9 membership

Membership relates users to clubs to indicate that the user is or was a member of that club.

id, integer: unique id to identify the membership internally.

userid, integer: identifier of the user that owns this membership.

club, integer: identifier of the club that this is a membership in.

join_date, date: the user first became a member of this club on this date.

expiration_date, date: the user’s current membership in this club expires on this date.

suspended, boolean: true if the user is currently suspended from this club.

volunteer_hours, integer: number of volunteer hours that the user has accumulated in this club.

4.10 permission

Permission relates users to permission options in clubs to say that a user has been granted a level of permission within a club.

id, integer: unique id used to identify the permission internally.

userid, integer: identifier of the user that has been granted this permission.

permission_option, integer: identifier of the permission_option that says what type of permission has been granted.

club, integer: identifier of the club that the permission has been granted in. Club should be ignored if this permission’s permission option is not a per-club option.

4.11 rating

Ratings are relations between users and rating options such that the user has some level of skill in that rating option.

id, integer: unique id used to identify the rating internally.
ration_option: identifies the rating_option entity that says what type of skill that the user is being rated on.

level, integer: user’s current level of skill for the rating option. A level of 0 (or a missing row) indicates that the user is unrated (has no skill). Higher levels indicate greater aptitude, so a level of 3 is better than a level of 2. Possible values are >= 0 and <= to the rating option’s levels field.

4.12 activity_requirement

Activity requirements are relations between activities and rating options. Each activity can have a list of activity requirements such that a user wishing to enroll in the activity must meet or exceed each requirement. To meet or exceed an activity requirement, the user must have a rating for the requirement’s rating option, and that rating’s level must be >= to the requirement’s level.

id, integer: unique id used to identify the requirement internally.

activity, integer: identifies the activity entity that requires this level of skill at this rating option.

ration_option, integer: identifies the ration option that says what type of skill/certification is being required.

level, integer: the minimum skill level required in this rating option.

4.13 activity_registration

An activity registration relates a user to an activity such that the user has enrolled in that activity. The number of activity_registrations for an activity should never exceed that activity’s max_capacity unless explicitly overridden by a user with sufficient privileges (clerk/superuser).

id, integer: unique id that identifies the registration internally.

activity, integer: identifies the activity that the user registered for.

userid, integer: identifies the user that the registration belongs to.

joind, datetime: the time stamp that says when the user enrolled for this activity.

4.14 activity_waitinglist

An activity waiting list relates a user to an activity such that the user wishes to enroll in that activity but the activity is full. An activity will have one of these for each user waiting to enroll in that activity.

id, integer: unique id that identifies the waiting list entry internally.

activity, integer: identifies the activity that the user wishes to enroll in.
userid, integer: identifies the user that is on the list for this activity.
joind, datetime: the time stamp that says when the user joined this waiting list. This allows the users on the waiting list to be prioritized.

5.0 Sequence Diagrams

5.1 Login and Edit Account Info

Two sequence diagrams are presented in Figure 4 and Figure 5 to show two scenarios that occur in the system. Descriptions of each sequence will follow with the corresponding steps in the figure indicated with parenthesis.

To start the login process depicted in Figure 4, the user will submit a username and password via the client web browser (1). The server will invoke the Login JSP page, which will pass the information to the Login Handler (1.1). The Login Handler will attempt to construct a UserPermissions object (1.1.1), which is the “Permissions Oracle” referred to in the component design section. The constructor for this object will query the database (1.1.1.1) and if no matches are found, an exception is thrown indicating logon failure (1.1.1.2). Otherwise the database is then queried for permissions (1.1.1.3). Upon receiving a valid UserPermissions object, the Login Handler will then store this object in the HttpSession (1.1.2). JSP provides such a service to keep data persistent between individual HTTP transactions associated with a single user session.

Figure 5 shows a typical sequence where a user has already logged in and wishes to interact with the system. In this case, the user would like to modify his/her account information stored in the database. The web browser will issue a request to fetch the edit page on behalf of the user (1). The Account Info JSP will invoke the Account Info Handler (1.1) in order to generate a page displaying the user’s current account information. The handler will not assume that the current session is valid thus it will fetch the UserPermissions object from the HttpSession (1.1.1). If the UserPermissions object does not exist, the user will be immediately directed to the login page thus the edit operation is canceled. Otherwise the Account Info Handler will get the id of the user from the UserPermissions object (1.1.2) and create a User object by passing this id to the constructor (1.1.3). The User object constructor will query the database in order to fill out the data items of the object corresponding to fields in the database. With the User object created, the Account Info Handler now generates a personalized page with options to edit the information.

After the user has edited the fields in the web browser and submitted the information, the Account Info JSP page will receive the information and repeat the same steps above to verify the integrity of the current session. Changes are made to the User object (2.2), and the Account Info Handler will ensure that this requested operation is valid by checking the permissions and integrity of the changed data items (2.3.1, 2.3.2). If certain fields are invalid, a page is generated to inform the user of the problems (2.4), otherwise the updated object is stored in the database (2.3.3.1).
5.2 Other Scenarios

Other use cases in the system follow the same general sequence of events that occur when editing account information. Therefore, these diagrams are omitted from this document.

FIGURE 5.2: Sequence Diagram for logging onto the system
FIGURE 5. Sequence Diagram for editing account information

1.1: Get Account Info
1.2: Get User Permissions
1.3: Create User Object using the ID

2.1: Get Account Info
2.2: Make Changes
2.3: Submit User Object
2.4: If failed, get field validation info and mark bad fields

2.1.1: Repeat the steps above
2.1.2: Get User Permissions
2.1.3: Create using the ID
2.1.4: Query
2.1.5: Verify that they can edit this user
2.3.1: Verify that the data is valid
2.3.2: Save
2.3.3: Submit Changes
2.3.3.1: Update
2.3.3.2: Update
2.3.3.3: Update

Http Session
User Permissions
Hoofers DB
User Object
Account Info Handler
Account Info JSP
6.0 User Interface Flow

The primary design goal for the user interface was simplicity and functionality. Low levels of page nesting is desired so that a user does not get “lost” while navigating. Figure 6 shows the logical pages that the user will see with arrows denoting navigation paths between pages.

The first page a user will see is splash.jsp. This page is also returned to the user whenever a session is deemed invalid. At this point, the user can either register a new account or log in with an existing account. Upon successful login, the user will reach welcome.jsp.

Welcome.jsp will contain a link to modify the user’s account information, and to each club page. Most pages in the system, including welcome.jsp, will contain a banner containing the following information and links:

- username of logged in user
- link back to welcome.jsp
- clerk.jsp and administration.jsp links if the user has those privileges

Each club will have a separate page such that club-specific links and information can be added in the future.

FIGURE 6. User interface flow with navigation paths
FIGURE 7. Components of a club page (sailing.jsp)

- sailing.jsp
- display_activity.jsp
- main_banner.jsp
- create_act.jsp
- display_activity_list.jsp
- include_club_common.jsp
- include_mem_info.jsp
- include_clsu.jsp
- include_edit_ratings.jsp
- include_join_activity.jsp
- include_manage_account_info.jsp
- include_manage_rating_options.jsp
- include_manage_membership_options.jsp
- create_act.jsp

Most of the functionality will be contained within the club pages. For the sake of modularity, a club page is constructed of numerous sub-pages. This will be achieved through the use of the JSP include directive which is similar to a preprocessor directive. Therefore, certain included pages expect variables to be already defined for their use. The downside of this approach is that variable name clashes can occur if a Java variable is already defined in the page that included it.

Figure 7 shows how a club page, sailing.jsp, is constructed. Each component will now be briefly described:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sailing.jsp</td>
<td>Top-level club page</td>
</tr>
<tr>
<td>main_banner.jsp</td>
<td>Constructs a banner for the top of the page. Includes logged in username, a logout link, and administrative links if appropriate for the user</td>
</tr>
<tr>
<td>include_club_common.jsp</td>
<td>Constructs the navigation page and information panes of page. Includes all other functionality common to most clubs</td>
</tr>
<tr>
<td>display_activity_list.jsp</td>
<td>Queries and creates a table for the type of activity requested</td>
</tr>
<tr>
<td>display_activity.jsp</td>
<td>Displays detailed information about a selected activity</td>
</tr>
<tr>
<td>create_act.jsp</td>
<td>Handles all the forms necessary for creating an activity.</td>
</tr>
<tr>
<td>include_join_activity.jsp</td>
<td>This will be used as a return point for buy_activity.jsp. It informs user of the result of the transaction</td>
</tr>
<tr>
<td>include_mem_info.jsp</td>
<td>Default page when user first enters club page. Displays information about the user’s club membership</td>
</tr>
<tr>
<td>include_clsu.jsp</td>
<td>Manage Club link will invoke this component to generate the CLSU page. The following subsystems are invoked by include_clsu.jsp</td>
</tr>
<tr>
<td>include_edit_ratings.jsp</td>
<td>Handles the CLSU edit ratings function</td>
</tr>
<tr>
<td>include_manage_ratings_options.jsp</td>
<td>Handles the CLSU function of adding/editing/removing club ratings options</td>
</tr>
<tr>
<td>include_manage_membership_options.jsp</td>
<td>Handles the CLSU function of adding/editing/removing membership options</td>
</tr>
<tr>
<td>include_manage_account_info.jsp</td>
<td>Handles the CLSU function of editing a users account information</td>
</tr>
</tbody>
</table>
Figure 8 shows an example of how sailing.jsp will look with the current prototype of the user interface.

FIGURE 8. Screenshot of the current prototype of sailing.jsp