BIKEQUEST

Top-Level Design

Dan Spinosa (dspinosa)
22 February 2005

PREFACE

I have not previously created a requirements or specifications document for BikeQuest. As such, this top-level design document is roughly based on the BikeQuest specifications by Redha Elminyawi (available at http://www.cs.brown.edu/courses/cs190/2005/asgns/2-11/relminya.pdf). Although this is still a tentative design, enumerated below are some noteworthy design choices:

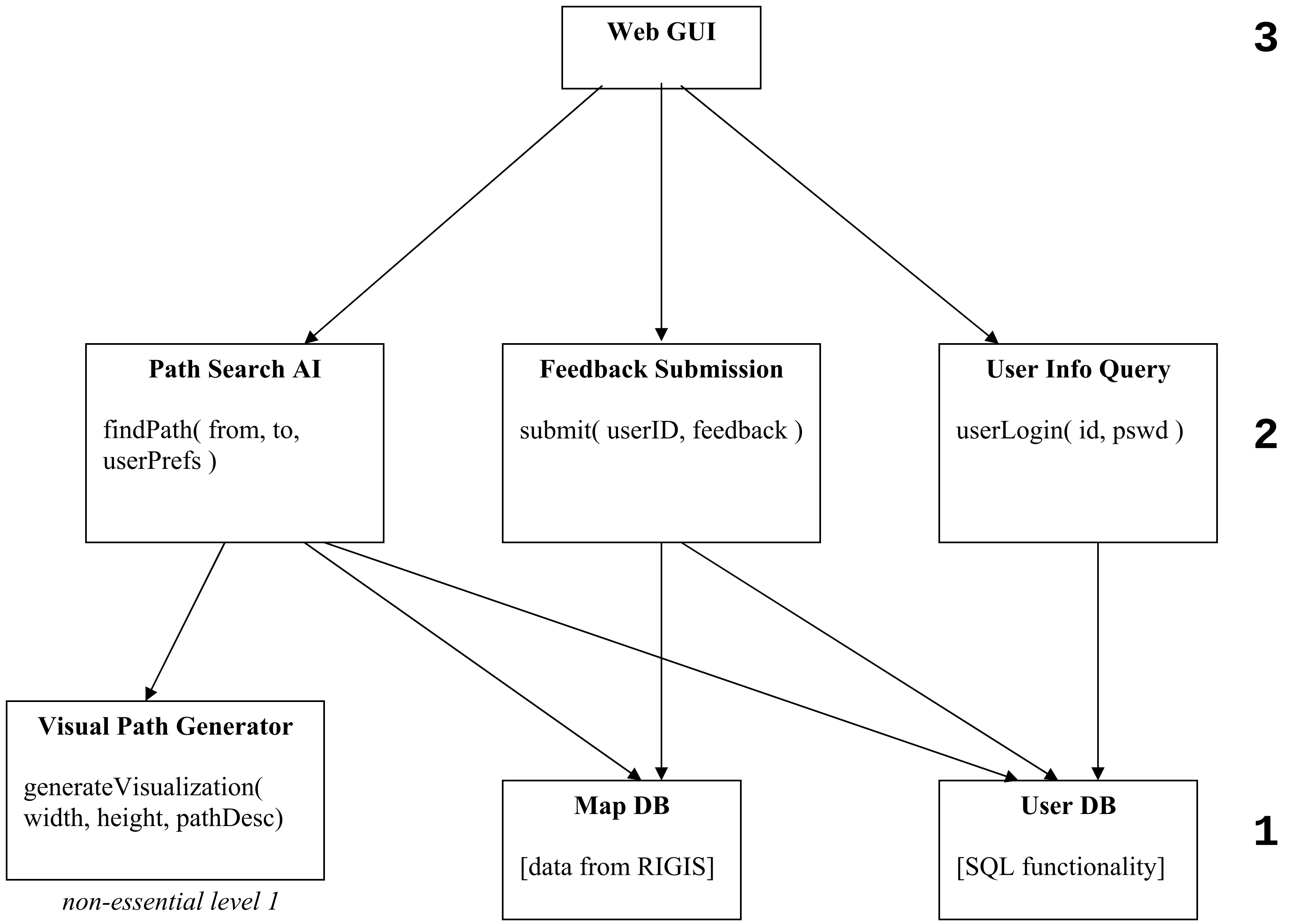
* No blog-like features
* User levels/ratings used for members
* Feedback is tagged with id and date/time
* A to B and A to A route finding is included
* Paths are defined by small sections, not entire routes
* Paths can be created interactively in web GUI

Although listed, the group organization is very tentative. It is difficult to get a grasp of what everyone would be best suited for from the course website. Once the team gets together these things can be worked out much more effectively.

DESCRIPTION OF PROJECT

Drivers have maps.google, MSNmaps, and MapQuest. Bikers will soon have BikeQuest. BikeQuest however goes beyond the scope of the traditional road trip planning service. Users will form a community whereby they can not only find a bike path, but also make recommendations and create roundtrip or one-way paths that take into account others' experiences.

HIGH LEVEL COMPONENTS



COMPONENT DESCRIPTION

Web GUI

Will be making calls on the path search algorithm, the feedback submission system, and the user info/login components. Stubs that return test data every time can be created by the Web GUI programmer. These can be used to hammer out the look and feel of BikeQuest while the mechanics to produce that actual data are still being produced. The stubs will also server as guides for the creators of the components that will be sending formatted data to the GUI. Although it is a level 3, work can begin on the GUI immediately.

Path Search AI

This level two component is the core of the system. The call to the visual path generator can only be made after the path search has completed running, and as such this component has literally no dependency on it. It relies heavily on the Map DB and the User DB. Map DB data is needed to determine what routes actually exist. This will be correlated with the User DB data which will provide user supplied information on paths. Conceptual work on the AI can begin immediately but the core of the coding must wait for working (if sparse) map and user databases.

Visual Path Generator

This component is almost more of a level 3 than a 1 as it will not be put into action until the level 2 path search AI is complete. It can be created immediately however. The visual path can be one or both of the following: 1) a 1:1 map path that follows routes from the physical map DB or 2) an MSNmap-like route map that emphasizes the concept and readability of the path over its physical structure (see cs224 route maps available at http://www.cs.brown.edu/courses/cs224/2004/hw/routemaps.pdf).

Feedback Submission

The web GUI will submit info to this component which will then enter it into the map and user databases. This encapsulates the community idea of rating paths and providing feedback after one rides a given path. All feedback will be tagged by user id, date and time such that searches can be selective on how they make use of what feedback information.

User Info Query

This component is the basis for the community of BikeQuest. By allowing users to choose to create a login their feedback can be tagged and used selectively. Users will be rated such that a dedicated administrative staff is not needed; the community will have the onus of policing itself. The web GUI will call this if/when a user logs in and use data returned to provide users with past route recalling (and easy rating of those routes) and preferences when searching (regarding to who they trust/like or other options). Work cannot begin on this module until the user DB interface is complete.

Map DB

Without knowledge of the RIGIS data format it is unclear how this database will be implemented at this time.

User DB

An SQL database will suffice for storage of user information, including login, password hash, search preferences, past searches, and feedback history.

EXTERNAL DEPENDENCIES

Information from the RIGIS database as well as other biking sources (including authors and biker contacts) will be required for this project. The most important of these is the RIGIS data and sufficient comprehension of it such that the map DB can be created and utilized.

Releasing BikeQuest into the wild will also rely on external assistance. For one, it must run on a server. Although heavy traffic is not expected the server will be performing intensive tasks including database queries, search AI and image retrieval. Also, a system administrator will be needed, although their need will be mitigated by the community building aspect of the project.

Also needed are apache, mySQL, a cgi processor or equivalent application running on the server.

TASK BREAKDOWN

Web GUI designer

Create a google mapsesque interface to the BikeQuest system. This is a lot of work and getting the browser to do the backflips that google has is no simple task. This task also involves usability research and testing.

AI coder

A two or more person job, the AI has a lot of work. This is the crux of the system and depending on the coders' familiarity with AI (taken cs141?) may require outside help. These coders must also become intimately familiar with the map DB and at least one of the coders must know the user DB inside and out.

Feedback unit coder

A smaller project, this coder need only take a submit from the web and add it to the user DB appropriately. The coder on this will likely also take on testing and/or more user system tasks.

User Info coder

Another smaller project, possibly to be coded by the feedback coder, the executer of this task write the code that interacts with the user DB and authenticates users. They must also provide the GUI with user preferences and history.

Visual Path coder

This task consists of taking data about a route and creating its visual appearance. Depending on the GUI implementation this may consist only of creating a png overlay for a map. This task also involves implementation of the hand-drawn style route map creation and its delivery to the GUI.

User/Map DB

The coders on the databases must create and hammer down an interface for getting information. Documentation of the database must be created and these coders will be responsible for maintaining integrity of the DB design and use by the AI and user query components.

General Manager (GM)

This is the dictator. For a solid system, a clear design must created and its integrity enforced by the GM. The GM should also be a hacker with knowledge of the entire system and its workings, although knowledge of every line in every component is not required. Furthermore, the GM must continually monitor progress and make the necessary adjustments before things get out of control, off course, or out of the design.

Librarian / Documentation

One of the most demanding jobs, this worker has the responsibility of creating all documentation of the system such that an outsider could walk in and begin work on a component just by reading the docs. Extensive help manuals should not need to be created if the design is followed the GUI hackers do their job.

GROUP ORGANIZATION

aavila – web GUI
crschmid - Map/user DB
dkarr – Librarian/Documentation
dspinosa – GM? webGUI?
Htse - visual path coder
relminya – map DB/AI
vywu - User Info, feedback
wcabral – AI

SCHEDULE

Now - 3/4	First meeting, design choice, group organization
3/4 – 3/11	Finalize top-level design
3/11 – 3/16	First draft of interfaces
3/16 – 3/21	Comment on and update interfaces, create A,B,C lists of features
3/21 – 3/25	Finalize Interfaces
3/25 – 4/10	coding begins, GM takes a good hard look at project
4/11	No work
4/12 – 4/18	Group meets to discuss project and decide its future, <u>GM has final word</u>
	Prepping for integration and initial integration
4/19	Full Integration / Feature Freeze
4-19 – 4/25	Bug fixing and component finalizing,
4/25 – 4/29	Hard core testing of system, bug fixing
4/29	In class first demo
4/29 – 5/4	Bug fixing, more testing, final system and its demo development
5/4 – 5/15	Final system development, finish everything still on our plates
5/16	Present the system and remove beta moniker