6.170 Payut Pantawongdecha

Design Analysis For *TakeNotes*

April 8, 2013

Minimal Viable Product

1 Overview

1.1 Purpose and Goals

1.1.1 Brief Description

TakeNotes is a web application for taking online notes. Its interface is made similar to 3M's Post-it notes in the real world, or Apple's Stickies and Windows 7's Sticky Notes in the virtual desktop. Users are able to create, view and edit a collection of textual notes. Their notes are kept private on a server.

1.1.2 Key Goals

TakeNotes is meant to be a lightweight web application. It should allow users to manage their notes as if they were their 3M's Post-it notes. The notes should be kept on a server, and other users are not allowed to view notes of a user.

1.1.3 Motivation

A web application allows users to have their notes anywhere. The drawbacks of standalone softwares such as Stickies and Sticky Notes are that they are computer-specific (or platform-specific), and users cannot see their note when they use different computer (or different OS). On the other hand, TakeNotes utilizes web browsers to allow users to view and edit their notes on any machine.

1.2 Context Diagram

Figure 1 provides a high-level overview of the interactions between TakeNotes and its users.

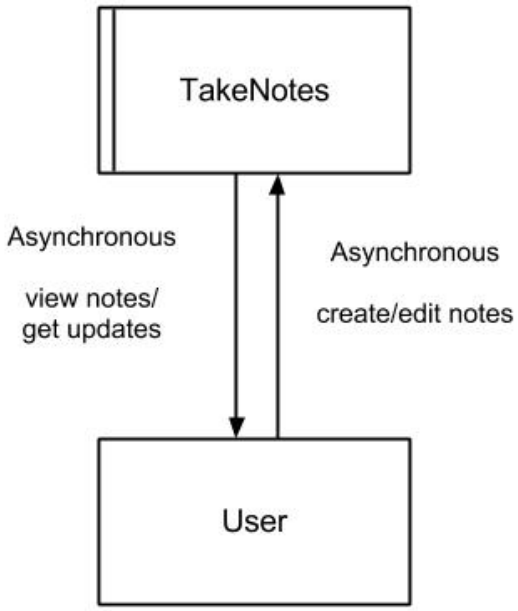

Figure 1: Context Diagram for TakeNotes. A stripped box represents a system component. A regular box represents an external components. Each edge is labeled with an event that represents interaction between two components; an arrow represents the direction of data flow.

2 Concepts

2.1 Key Concepts

The key concepts in the design of TakeNotes are **users** and **notes**. A user has many notes that are private to the user. Each user is identified by a unique e-mail address, and needs to provide a correct password to login. Note functionalities are implemented on the client side, and modified data will be send to the server asynchronously.

2.2 Object Model

Figure 2 shows the object model for the key concepts.

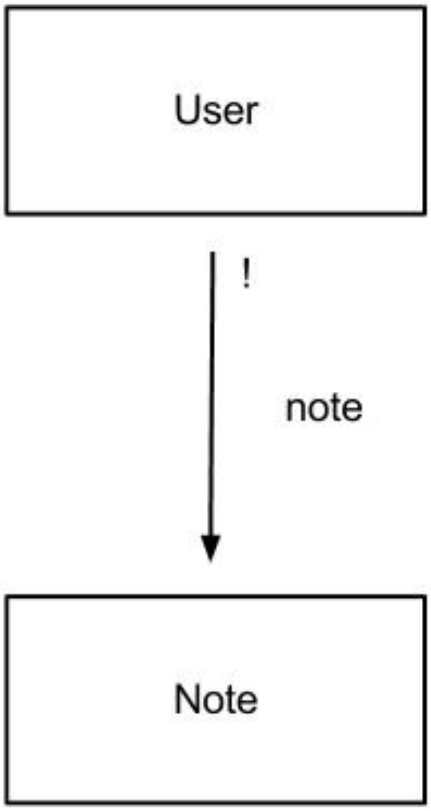

Figure 2: Object Model for TakeNotes.

3 Behavior

3.1 Feature Descriptions

TakeNotes provides following functionalities to the users:

- **Note creation:** A user can create a new note by pressing + button.
- **Note view:** A user can view only his/her notes.
- **Note edit:** A user can edit his/her notes. Clicking x button deletes the note.
- **Note update:** A user receives updates on notes by the same user in another session.
- **Note position:** A user can resize notes and move them around the page.

3.2 Security Concerns

TakeNotes has the following security requirement: It makes sure that the notes are accessible to only a user who creates them by requiring users to login to use the web application. The application may be vulnerable to the following potential security threats:

- **Injection Attack.** TakeNotes should not use use an interpreter such as eval or direct SQL queries.
- **Cross-site scripting attack.** An attacker might lure a user to enter malicious text into a note, which can lead to malicious script execution. Using sanitize method in rails on the text body can solve this problem.

3.3 User Interface

Figure 3 illustrates the interfaces of TakeNotes and the flow between them.

4 Design Challenges

Concurrency of note edit

When the same user edits a note from two different machines, concurrency issues may arise. The main solution is how the client and the server communicate on a note content. There are two options:

- Client send only changes (or diff) to the server.
- Client send the whole note to the server.

The first approach reduces the size of data transmitted through networks. TakeNotes is going to use a lot of updates through AJAX call, so minimizing the size of data will be nice. However, it creates a problem when a user sends in edits from two different machines because when machine 1 updates the note based on changes from the last saved notes, machine 2 which also sends in changes based on the last saved notes, but the actual saved notes already changes from the machine 1. This approach causes a note to not work well. Therefore, I choose the second approach. Sending the whole note to the server solves this problem as the last note is saved to the database. The drawback is that the transmission data is huge; however, the users are unlikely to create huge notes anyway, so I think it is not a big problem.

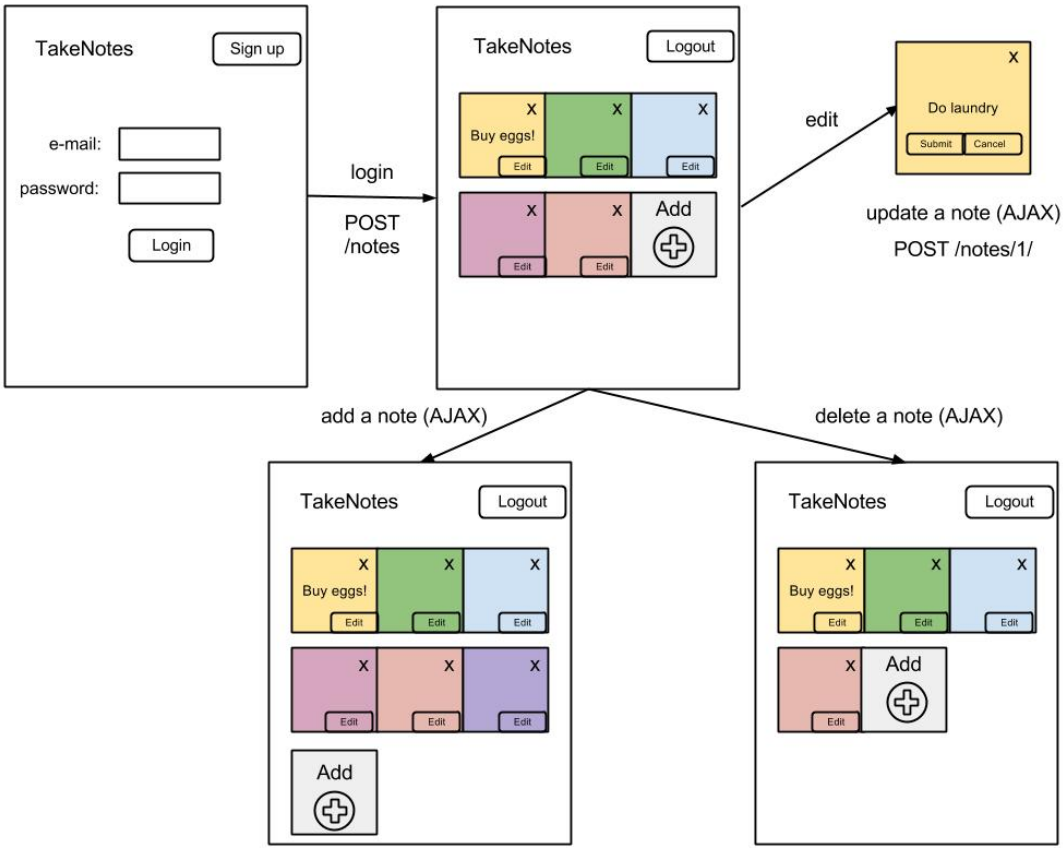

Figure 3: Wireframe for TakeNotes.

Does a user need to login?

There are two options to this question:

- Yes. This is what I choose and the whole design is based on this approach.
- No. Notes might be stored in a session. It also allows guests to try the website first.

I choose the first approach because first, the login process is quite easy. Next, the goal of TakeNotes is to allow users to have their notes anywhere. Therefore, if a session is used, I don't see a point of using the web app, because the users can used built-in sticky notes app in their computers.

When should notes be saved?

There are two options for this question:

- When a user edit text on a note.
- Saved with a submit button.

My first design was to save a note when a user type any text into the note. This behavior is similar to Stickies in OS X or Sticky Notes in Windows 7. It is more convenient for users because they don't have to click edit and submit buttons in order to edit or save notes. They can just type anything in and their notes are autosaved. However, I find that this approach makes AJAX call every time there is a letter changed, so the website might become irresponsive or delayed. One way to fix this approach is to send an ajax call every 5-10 seconds for updates. This method is feasible, but there are some checks to perform. For example, the site has to check if users close web browser or go to another page so that it sends the most recent change to the server so that the server doesn't remain at the data 5-10 seconds ago. To avoid these extra checks, I choose to use a relatively simple approach, which is to implement an edit and a submit buttons. This saves a lot of codes to write, and ensures that notes are saved as the users want. Users won't lose their saves.

Notes on code design

E-mail and password are stored as attributes for each user. Note only has content, and belongs_to a user. However, to make implementation of extension easier, note might be contained in another object **folder** instead. Note objects are on the client side, and send updates to the server via AJAX call.

Extension: Folders

The content of the extension is the same as the minimal viable product with some specified additions.

1 Overview

1.1 Purpose and Goals

1.1.1 Brief Description

In addition to mvp, TakeNotes allows users to have folders to keep different types of notes. It allows users to categorize their notes for convenience. For example, a user may have folders **DEFAULT**, **TODO** and **IMPORTANT**, etc.

1.1.2 Key Goals

The goal of the extension is to allow a user to classify notes by themselves.

1.1.3 Motivation

When a user creates more and more notes, it gets harder to maintain and find notes. Therefore, note categorization is a necessary feature to have for a user to handle his/her notes.

1.2 Context Diagram

Figure 1 provides a high-level overview of the interactions between TakeNotes and its users.

2 Concepts

2.1 Key Concepts

The key concepts in the design of TakeNotes are **users**, **folders** and **notes**. A user has many folders that are private to the users. Each user is identified by a unique e-mail address, and needs

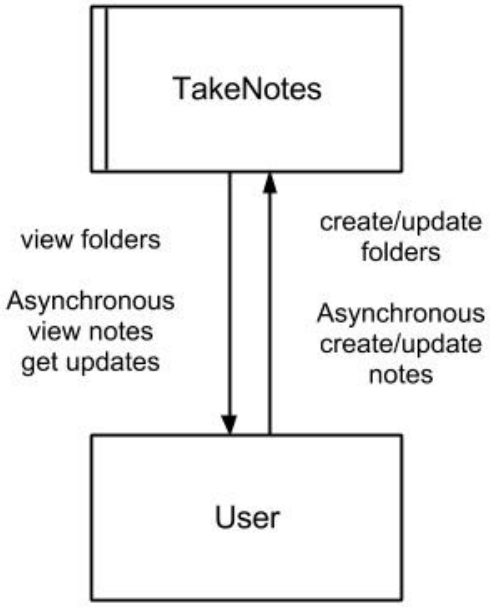

Figure 4: Context Diagram for TakeNotes. A stripped box represents a system component. A regular box represents an external components. Each edge is labeled with an event that represents interaction between two components; an arrow represents the direction of data flow.

to provide a correct password to login. A folder has many notes that can be moved between folders. The folder page will load once, and notes are created and updated via AJAX call inside the page. Note functionalities are implemented on the client side, and modified data will be send to the server asynchronously.

2.2 Object Model

Figure 5 shows the objet model for the key concepts.

3 Behavior

3.1 Feature Descriptions

TakeNotes provides following functionalities to the users:

- **Folder creation/edit:** A user can create/edit a folder to store notes. Default folder is created for every user, and cannot be edited.

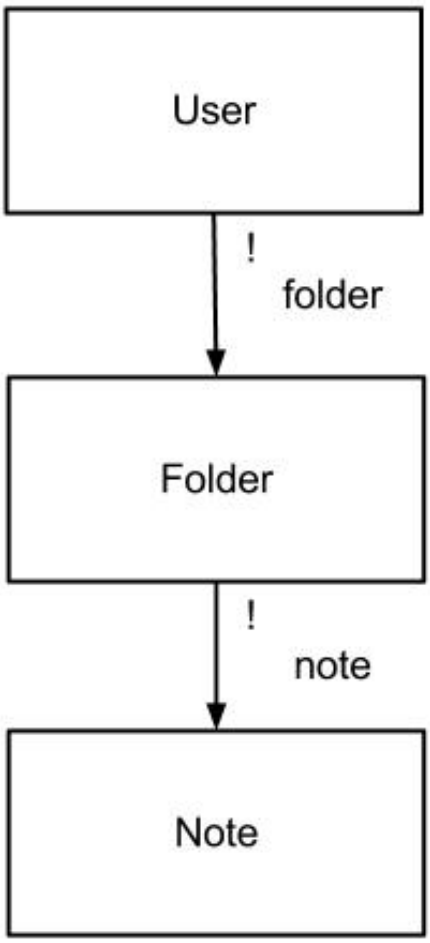

Figure 5: Object Model for TakeNotes.

- **Folder update:** A user receives updates on folders by the same user in another session.
- **Note creation:** A user can create a new note within a folder by pressing + button.
- **Note view:** A user can view only his/her notes within a folder.
- **Note edit:** A user can edit his/her notes within a folder. Clicking x button deletes the note.
- **Note update:** A user receives updates on notes by the same user in another session.
- **Note position:** A user can resize notes and move them around the page.

3.2 Security Concerns

This section is the same as mvp.

3.3 User Interface

Figure 6 illustrates the interfaces of TakeNotes and the flow between them. It is similar to figure 3, but has folders page between login page and notes page. There is a move button added to each note (edit button is not shown here). Moving a note between folders is the same as deleting it in one folder and add it to another folder.

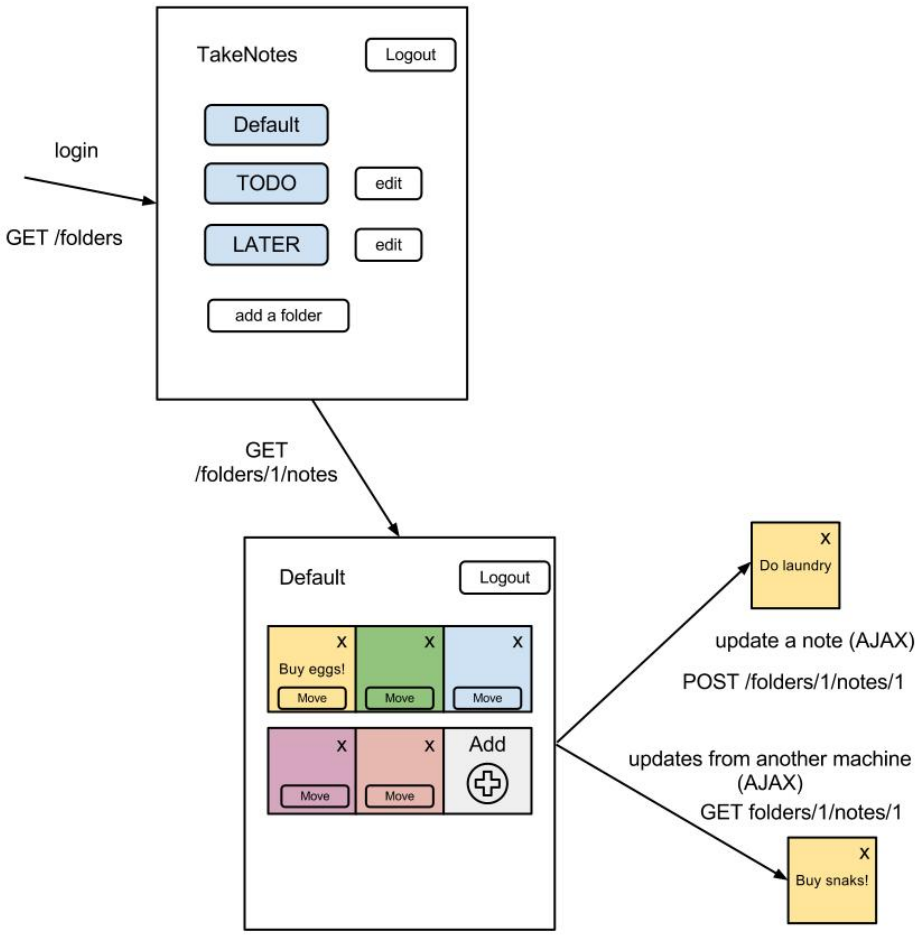

Figure 6: Wireframe for TakeNotes with login page omitted.

4 Design Challenges

Addition to mvp challenges.

"Done" folder

When a note is deleted, the server may delete it from the database, or keep it in the recycle bin. Some softwares that allow users to keep their "done" thing in TODO list by moving an element of a todo list to a "done" folder. This gives a user an option to view old elements of todo list. In this project, I need to decide if I should implement a "done" button to move a note to "done" (or "archive") folder. There are two options:

- **Implement "archive" folder.** I can implement an "archive" folder and "archive" button on each note, which moves the note to the archive folder. This is convenient for users to handle their todo list, because they can move notes between "archive" folder and other folders by one click.
- **No extra implementation.** Let the users handle everything by themselves. No need to put "archive" buttons in every note. A delete button deletes the note permanently.

I choose not to implement an "archive" folder and "archive" buttons because I want TakeNotes to be web app for general note taking instead of keeping todo list specifically. Moreover, a user can create his own "archive" folder anyway, and putting "archive" buttons in every note will reduce the space that a user can view a note.

Notes on code design

E-mail and password are stored as attributes for each user. Default folder is auto-generated for each user. Folder only has name as its attribute, and rails features such as has_many notes and belongs_to user are used. Similarly, note only has content, and belongs_to a folder.