Password Safe

With WebDAV Sync

by

Jake Schuenke

Matt Spurr

Tyler Shelton

Kegan Kaiser

Contents

[1. Project Schedule 3](#_Toc348048549)

[2. Domain Model 4](#_Toc348048550)

[3. Layered Architecture 5](#_Toc348048551)

[4. Interaction Diagrams 6](#_Toc348048552)

[4.1 System Sequence Diagrams 6](#_Toc348048553)

[4.1.1 Set up Master Password 6](#_Toc348048554)

[4.1.2 Set Notes 6](#_Toc348048555)

[4.1.3 Add Password 7](#_Toc348048556)

[4.1.4 Data Sync 7](#_Toc348048557)

[4.1.5 Set Password Standards 8](#_Toc348048558)

[4.1.6 Set Connection to WebDAV Server 8](#_Toc348048559)

[4.1.7 Automatic Sync 9](#_Toc348048560)

[4.2 Operation Contracts 9](#_Toc348048561)

[4.3 Sequence Diagrams 11](#_Toc348048562)

[4.3.1 Create Note 11](#_Toc348048563)

[4.3.2 Create Account 12](#_Toc348048564)

[4.3.3 Generate Password 12](#_Toc348048565)

[5. Design Class Diagram 13](#_Toc348048566)

[5.1 Grasp Principles 13](#_Toc348048567)

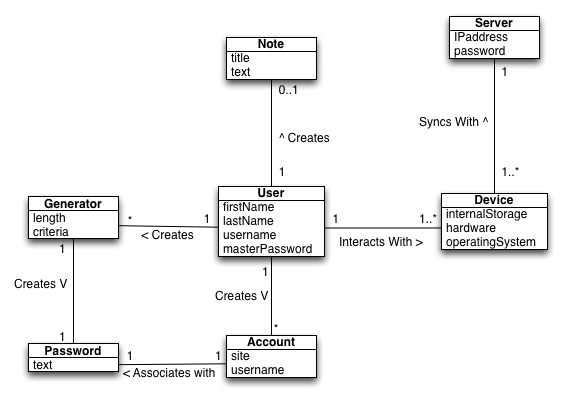
[5.2 GoF Principles 14](#_Toc348048568)

[6. Who-Done-What Table 16](#_Toc348048569)

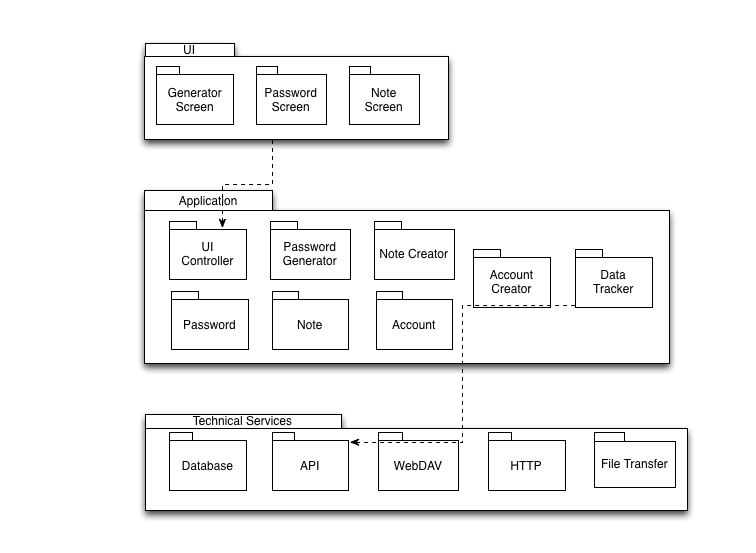
# 1. Project Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Iteration Complete | Milestone | Milestone Complete |
| 0 | 12/512 | Learn basics of Objective C  Learn about WebDAV technology  Learn about Apple framework  Learn WebDAV frameworks | 12/5/12  12/5/12  12/5/12  12/5/12 |
| 1 | 12/14/12 | Generate Password  Create Note  Domain Model Created  SSDs and OCs completed  Logical Architecture and Package Diagrams  Behavioral models  Store Password  Criteria for Generator  Present | 12/6/12  12/6/12  12/7/12  12/9/12  12/11/12  12/11/12  12/13/12  12/14/12  12/14/12  12/21/12  12/21/12 |
| 2 | 1/11/13 | Associate Information  Connect to Server  Send Password to Server  Connect To WebDAV  Update Milestone 3 documentation  Present | 1/6/12  1/6/12  1/8/13  1/11/13  1/10/13  1/11/13 |
| 3 | 2/15/13 | Secure Transfer  Master Password  Connect to any WebDAV  Update Milestone 4 documentation  Final Presentation | 1/28/13  1/25/13  2/2/13  2/15/13  2/15/13 |
|  | **Project Complete** | **2/15/13** |  |

# 2. Domain Model



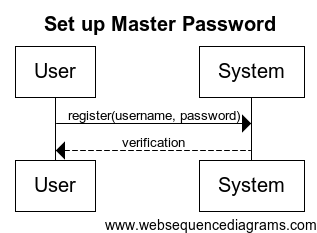
# 3. Layered Architecture



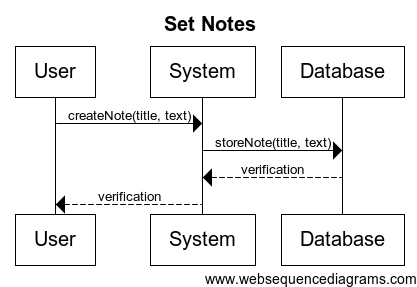
# 4. Interaction Diagrams

## 4.1 System Sequence Diagrams

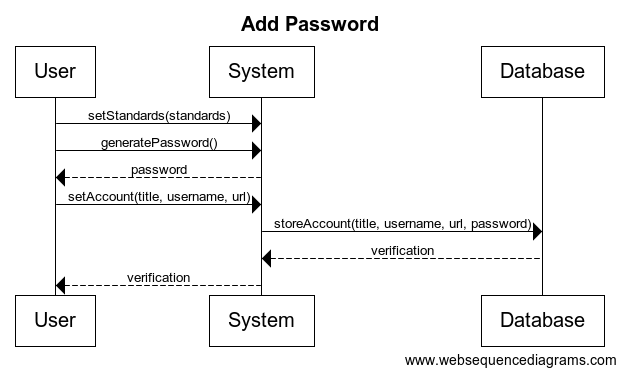
This section details three system sequence diagrams for the application

4.1.1 Set up Master Password  


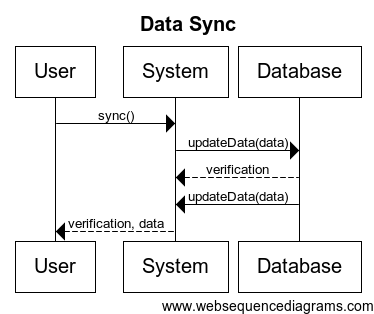
### 4.1.2 Set Notes



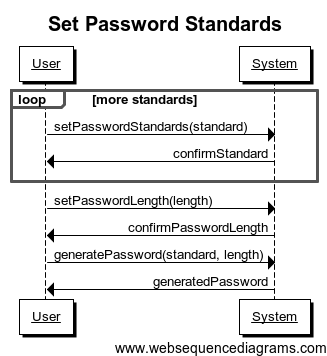
### 4.1.3 Add Password



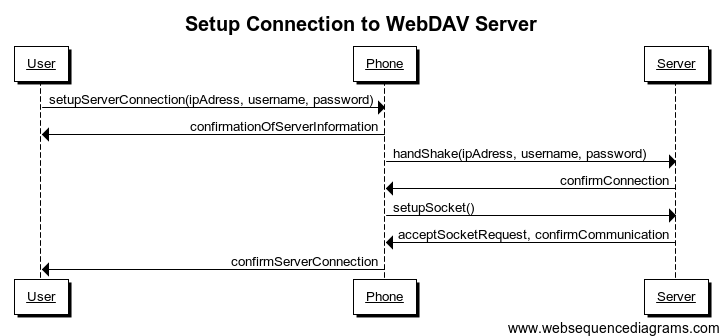
### 4.1.4 Data Sync



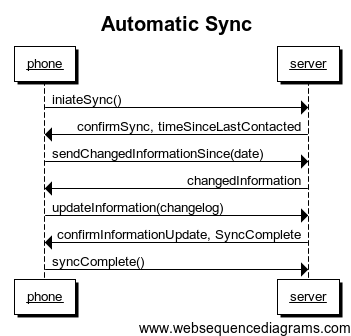
### 4.1.5 Set Password Standards



### 4.1.6 Set Connection to WebDAV Server



### 4.1.7 Automatic Sync



## 4.2 Operation Contracts

**Operation**: register(username, password)

**Cross-references**: UC1

**Preconditions**: User has started application

**Postconditions**: Master account with username and password is created

**Operation**: setLength(value)

**Cross-references**: UC 3

**Preconditions**: The User has initiated the process of creating a new password.

**Postconditions**: The value for length has been set and this value will be used for the next generated password.

**Operation**: setNumCaps(value)

**Cross-references**: UC 3

**Preconditions**: The User has initiated the process of creating a new password.

**Postconditions**: The number of capital letters has been set and this value will be used for the next generated password.

**Operation**: setNumLower(value)

**Cross-references**: UC 3

**Preconditions**: The User has initiated the process of creating a new password.

**Postconditions**: The number of lowercase letters has been set and this value will be used for the next generated password.

**Operation**: setNumNums(value)

**Cross-references**: UC 3

**Preconditions**: The User has initiated the process of creating a new password.

**Postconditions**: The number of numbers has been set and this value will be used for the next generated password.

**Operation**: setNumSpecial(value)

**Cross-references**: UC 3

**Preconditions**: The User has initiated the process of creating a new password.

**Postconditions**: The number of special characters has been set and this value will be used for the next generated password.

**Operation**: saveConnections(hostname, password)

**Cross-references**: UC 5

**Preconditions**: The User has access to a WebDAV server and knows the hostname and password needed to connect to that server.

**Postconditions**: The connection is saved and the device will now sync passwords with this server.

**Operation**: createNote(title, text)

**Cross-references**: UC6

**Preconditions**: User is on the note creation screen

**Postconditions**: A note with a title and text is created and stored on the local device

**Operation**: storeNote(title, text)

**Cross-references**: UC6

**Preconditions**: User has created a note on the local device

Device is connected to the server

**Postconditions**: The newly created note on the local device is stored on the server as well

**Operation**: setStandards(standards)

**Cross-references**: UC8

**Preconditions**: User is on the password generator screen

**Postconditions**: The standards for password generation are set

**Operation**: setAccount(title, username, url)

**Cross-references**: UC8

**Preconditions**: User has generated a password and is on the account creation screen

**Postconditions**: An account is created with the given title, optional username, optional url, and the password that was already generated.

**Operation**: storeAccount(title, username, url, password)

**Cross-references**: UC8

**Preconditions**: User has created an account on the local device

Device is connected to the server

**Postconditions**: The newly created account on the local device is stored on the server as well

**Operation**: setAutoSync(value)

**Cross-references**: UC 9

**Preconditions**: N/A

**Postconditions**: The system will or will not automatically sync all passwords on startup based on whether “value” was “yes” or “no”.

**Operation**: updateData(data)

**Cross-references**: UC10

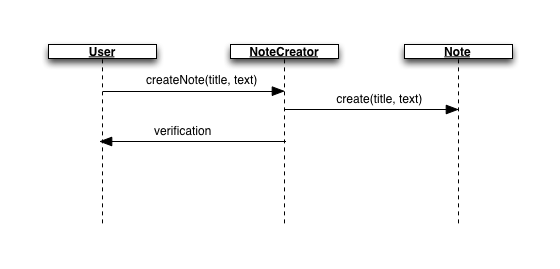
**Preconditions**: User has told the device to sync with the server

Device is connected to the server

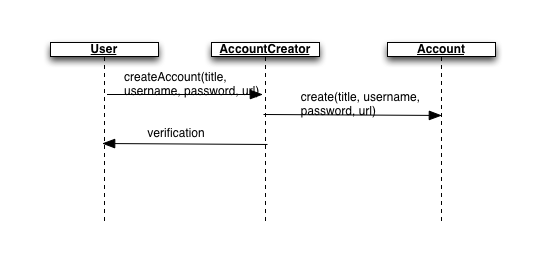
**Postconditions**: System compares timestamps for each entry and sets them to the most recent on the device and the database

## 4.3 Sequence Diagrams

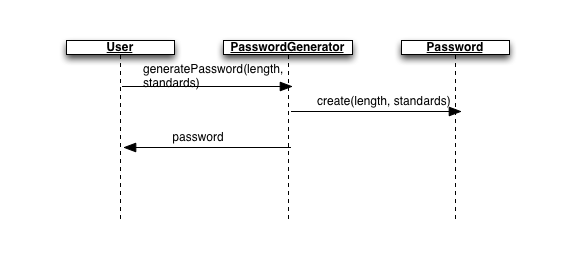
### 4.3.1 Create Note



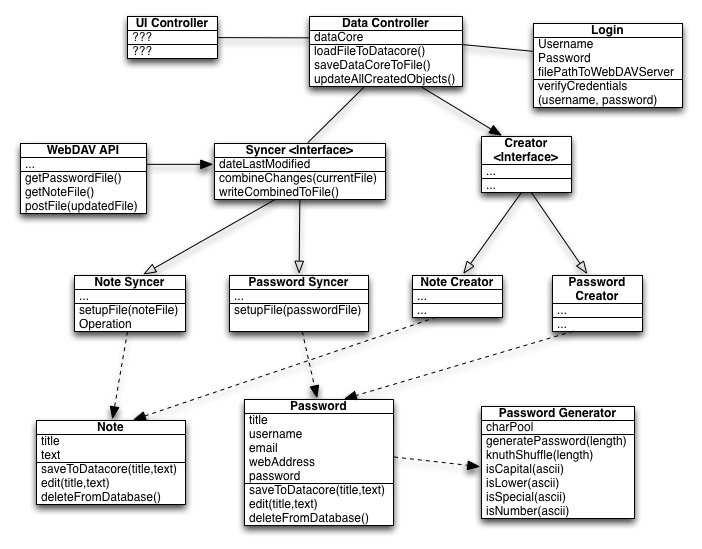
### 4.3.2 Create Account



### 4.3.3 Generate Password



# 5. Design Class Diagram



## 5.1 Grasp Principles

**Creator**

Apple’s core data allows for the Creator principle to be followed. It creates objects and then puts them in persistent storage. Core data will be linked with the data controller to manage the objects stored as persistent data.

**Information Expert**

The Information Expert principle is demonstrated by the Password and Note classes. These classes are responsible for initializing and updating the information that they need, such as a title and description.

**Controller**

This design has a data controller as well as an interface controller. The data controller is responsible for controlling the back end, including password generation and syncing, while the interface controller is responsible for controlling the front end such as updating tables on the screen that display the current passwords.

**Low Coupling**

This design seeks to have the lowest level of coupling possible. To do this, each class was analyzed carefully to determine which classes it needed to depend on. Most classes depend on two or fewer other classes.

**High Cohesion**

This design seeks to have the highest level of cohesion possible. To do this, each class was carefully evaluated to determine if it needed to be broken up. For example, the syncer was divided into a syncer for passwords and for notes since the data about them will need to be retrieved differently. Most classes have five or fewer methods to ensure that too much responsibility is not given to a single class.

**Polymorphism**

Our design utilizes polymorphism by implementing interfaces. The Syncer and Creator interfaces each define basic methods that the classes require, while the Password and Note versions of each fill in extra details unique to the synchronization or creation of passwords and notes.

**Pure Fabrication**

The PasswordSafe application allows users to save Accounts, which have the following attributes:

* Username (string) – The name, email, etc. used to log in to the account.
* Password (string) – The password used to log in to the account.
* Title (string) – Represents what this account is for. Example: Facebook, Email

However, there is no actual Account object. The idea of an account is simulated by saving usernames alongside their corresponding passwords and titles. The title is displayed so that the users know what the password and username are for. The matching password and username are displayed alongside the correct title.

**Indirection**

Our design utilizes Indirection by not allowing the Data Controller to directly access Notes or Passwords. Instead the Data Controller talks to a Creator, when it wants to generate a note or password, or a Syncer, when it wants to sync data with the server. This prevents Note and Password from being flooded with commands and allows them to focus completely on defining notes and passwords.

**Protected Variation**

The Data Controller is protected from any variation in passwords and notes due to the indirect way it accesses them. The Note and Password variations of Syncer and Creator will work in different ways, but the Data Controller does not have to be concerned with this.

## 5.2 GoF Principles

**Adapter**

The PasswordSafe system implements two adapter classes: the Syncer and the Creator. Passwords and Notes stored by the system are very similar. They both store text based data that can be set and retrieved. The Syncer and Creator interfaces were created to take advantage of these similarities, while allowing Password and Note objects to retain some level of uniqueness. The Data Controller will need to communicate with the Password and Note versions of both the Syncer and Creator class. Instead directly communicating with all four classes the Data Controller will communicate with two adapter classes. One of these will handle all Creator calls and will send the requests to the appropriate class, while the other handles all calls to the Syncer classes.

**Strategy**

Strategy is the idea that you should define each algorithm or policy in a separate class with a common interface. This is the pattern we use when syncing notes and passwords. Both share a common interface called Syncer, but we define the differences in syncing in the subclasses PasswordSyncer and NoteSyncer. These include the algorithms specifically for each class.

**Factory**

A Factory is a design pattern that is used to create objects that should be cached or has complex creation logic. In our design, the generator class will act as a factory that outputs the password class. Passwords are complex objects that use different criteria to generate, and then link themselves to other information. Also, the generator will handle the local caching of the created password by handling the logic of inserting the created password into the local copy of core data.

**Observer**

The observer patterns which allows for low coupling between objects that need to be informed when events happen from one another is used in the design in the GUI controller. The controller is a subscriber to the listeners of the actual User Interface. The listeners subscribe to the actual actions of the user, and the controller subscribes to the listeners. This way when a user inputs actions, the listeners respond, and make proper calls using the controller.

# 6. Who-Done-What Table

|  |  |  |  |
| --- | --- | --- | --- |
| Who Done It:  Team Member Names | Section/Part Completed | Task/Comments | # of hours effort |
| Entire Team | Graphs | Check out all the graphs | 1.5 hours per member |
| Entire Team | GoF principles | Update information presented | 2 hour |
| Jake and Tyler | Activity Diagram | Create the activity diagram | 3 hours |
| Entire Team | Develop Code | Develop the source code | 20 hours per member |