

# Software Design Document for the ConMan

Authors:

Version 0.1  
10/28/2013

# Contents

# 1 Introduction

This software design document provides a description of the **ConMan** application. In addition to describing the basic user interface design, the client-server interface and database description are also provided here.

## 1.1 Goal and Purpose

The purpose of **ConMan** is to provide a simple, web-based task-management application that can be used by both individuals and teams. The user interface should be easy to learn without hiding any of the core features of **ConMan**. The applications and its data should be protected with controls such as permission checking to ensure that data is not leaked and/or corrupted.

## 1.2 Core Features

The core features of **ConMan** include:

- Ability to register as a user
- Ability to define teams of users
- Ability to create, view, update, and delete project tasks
- Ability to create, view, update, and delete checklists associated with tasks
- Ability to register for notifications (via the application and via email) of upcoming deadlines
- Ability to view deadlines through a calendar interface

## 1.3 Major Constraints

The constraint of the largest magnitude is time. Due to time constraints, all development and testing must be completed within a 2 month period. A smaller constraint involves the teams web development experience. A portion of the team has little web development experience and even lower experience with the technologies that will be used (ASP.NET, jQuery, etc.).

## 1.4 Intended Audience and Reading Overview

This document is intended for individuals involved in the design and implementation of the **ConMan** web application. Stakeholders and customers may also want to read this document with the goal of ensuring that the product is being developed to their requirements. Readers of this document may read through this document in a sequential manner or they may read individual sections in any order. Below is a list of each section along with a short description:

- Architectural Design (Section 2)
  - The two major components of ConMan's Client-Server architecture are discussed.
- Data Design (Section 3)
  - The database tables are defined and the transaction of data is described.
- User Interface Design (Section 4)
  - The general layout for the main page of the web application is depicted through the use of multiple wireframes.
- Restrictions, Limitations, and Constraints (Section 5)
  - This section specifies the browsers to be supported by the initial release of ConMan and the applications reliance on internet connections.
- Testing Guidelines (Section 6)
  - A variety of test cases are presented along with their expected results.
- Appendices (Section 7)
  - This section will contain a list of definitions of terms used throughout the document.

## 2 Architectural Design

### 2.1 Overview

**ConMan** is divided into two major components: the client-side user interface, and a server-side application and database. The client-side communicates with the server using HTTP requests sent through an internet browser. **ConMan** utilizes the Client-Server architecture and allows multiple clients to concurrently access the server.

### 2.2 Client-Server Interaction

Data and user interactions are passed between the database, server, and clients through a MVC architecture.

The **ConMan** database will maintain the data for the **ConMan** application. The database will have no direct interaction with the clients and is only accessible through an interface with the server. Requests made to the database will occur in a First-Come, First-Served manner in order to preserve data consistency.

The **ConMan** server parses HTTP requests made by the clients and will interface with the database as needed. The server itself will be stateless and will not maintain user or application data after a request is handled. Data passed between the client and server will be in the JSON format.

### 2.3 Basic Context

The "basic context" of **ConMan** is where a user can only take actions that affect their personal account. Every **ConMan** user has a basic account, and consists of the tasks and checklists that are not a part of external teams or groups. When a user opens the **ConMan** application by pointing their browser to the **ConMan** URL, the initial user context is the basic context. The basic context is also the starting context when an account is created.

#### 2.3.1 Interface Description

Fields entirely enclosed in [] brackets are to be considered optional.

CreateAccount (	LastName: String ,
	FirstName: String ,
	E-mail: String ,
	Password: String );

ModifyAccount (	UserId: Int ,
	[LastName: String] ,
	[FirstName: String] ,
	[E-mail: String] ,
	[NewPassword: String] ,
	OldPassword: String );

```
CreateTask( UserId: Int ,
            TeamId: Int ,
            DueDate: date ,
            Description: String );
```

```
ModifyTask( UserId: Int ,
            TaskId: Int ,
            TeamId: Int ,
            [DueDate: date] ,
            [Description: String] ,
            [Completed: Boolean] ,
            [Note: String] );
```

```
CreateChecklist( UserId: Int ,
                 TeamId: Int ,
                 TaskId: Int ,
                 DueDate: date ,
                 Fields:[{ CheckListId: Int , FieldId: Int DueDate: date ,
                 Description: String }],
                 Description: String );
```

```
ModifyChecklist( UserId: Int ,
                 CheckListId: Int ,
                 TaskId: Int ,
                 [DueDate: date] ,
                 [Description: String] ,
                 [Fields:[{ CheckListId: Int , FieldId: Int DueDate: date ,
                 Description: String }]],
                 [CompletedField: Boolean []] ,
                 [Completed: Boolean] ,
                 [Note: String] );
```

### 2.3.2 Processing Details

The **CreateAccount** interface is used when the user first creates an account with the **ConMan** system. The server will validate the input fields given by the user. If all fields are valid, the server will update the database records and add a new user. A team is automatically generated with a team-Id equivalent to the user-Id. If the server successfully validates the user fields and successfully updates the database, the user-Id generated by the database will be returned. Otherwise, an error code will be returned.

**ModifyAccount** functions similarly to **CreateAccount**, except fields that NULL can be given for fields that are not being updated and are not required to identify the action.

**CreateTask** adds a new task for the user. In the basic context, the **TeamId** will be the **UserId**. After verification by the server and a successful database update, the task-Id will be returned. Otherwise, an error code will be returned.

**ModifyTask**, like **ModifyAccount** allows textttNULL values to be given for fields that are not being updated. A success or error code will be returned.

`CreateChecklist` will add a checklist to a task owned by a user. In the basic context, the `TeamId` will be the `UserId`.

`ModifyChecklist` This interface function is invoked to modify an existing checklist.

### 2.3.3 Restrictions/Limitations

For every interface action, the server must validate the given fields against the database to ensure the user making the request has the appropriate permissions.

### 2.3.4 Performance Issues

Requests made by the clients to the server are subject to time-out restrictions. If the server takes too long to process an interface request and the client-server connection is ended, the client will assume the transaction failed. Under such a failure condition, the client will send a "revoke" notice to the server indicating that any database changes made on behalf of the timed-out request must be rolled-back.

### 2.3.5 Design Constraints

In order to maintain data consistency, either only one database should be used, or a very robust data-duplicating effort must be made to ensure consistency across multiple databases. Version one of `ConMan` will only support a single database back-end.

## 2.4 Team Context

The team context is conceptually very similar to the basic context, except administrator privileges are required to modify the team account. Administrator privileges are automatically given to the creator of a team. Tasks associated with the team can be modified by all team administrators and all users belonging to the team who have been assigned to the task.

### 2.4.1 Interface Description

Fields entirely enclosed in `[]` brackets are to be considered optional.

<code>CreateTeam(</code>	<code>TeamName: String ,</code> <code>E-mail: String ,</code> <code>Password: String );</code>
--------------------------	--

<code>ModifyTeam(</code>	<code>TeamId: Int ,</code> <code>[TeamName: String] ,</code> <code>[E-mail: String] ,</code> <code>[NewPassword: String] ,</code> <code>OldPassword: String );</code>
--------------------------	---

```
CreateTeamTask(  UserId: Int ,
                 TeamId: Int ,
                 AssignedUsers: Int [] ,
                 DueDate: date ,
                 Description: String );
```

```
ModifyTeamTask(  UserId: Int ,
                 TaskId: Int ,
                 TeamId: Int ,
                 [ AssignedUsers: Int [] ] ,
                 [ DueDate: date ] ,
                 [ Description: String ] ,
                 [ Completed: Boolean ] ,
                 [ Note: String ] );
```

```
CreateTeamChecklist(  UserId: Int ,
                     TeamId: Int ,
                     TaskId: Int ,
                     AssignedUsers: Int [] ,
                     DueDate: date ,
                     Fields: [{ CheckListId: Int , FieldId: Int DueDate: date ,
Description: String } ] ,
                     Description: String );
```

```
ModifyTeamChecklist(  UserId: Int ,
                     CheckListId: Int ,
                     TeamId: Int ,
                     TaskId: Int ,
                     [ AssignedUsers: Int [] ] ,
                     [ DueDate: date ] ,
                     [ Description: String ] ,
                     [ Fields: [{ CheckListId: Int , FieldId: Int DueDate: date ,
Description: String } ] ] ,
                     [ CompletedField: Boolean [] ] ,
                     [ Completed: Boolean ] ,
                     [ Note: String ] );
```

### 2.4.2 Processing Details

The processing details for the team context differ from the basic context only in the fact that an additional check must be performed to ensure the user making the request has the correct team permissions to do so.

### 2.4.3 Restrictions/Limitations

There are no additional restrictions or limitations.



#### **2.4.4 Performance Issues**

There are no additional performance constraints.

#### **2.4.5 Design Constraints**

There are no additional design constraints.

### 3 Data Design

Server-side databases provide persistent data storage for the **ConMan** application. When a user connects to the **ConMan** application through a web-browser, the data associated with that user is retrieved by the server from the database and sent as a JSON encoded string back to the user for client side processing. To preserve consistency, client side processing will only include data representation (visual effects). Any create, update, or destroy operations on the data must be validated by a server request (interfaces defined below).

All data held outside of the database - client and server-side - will not be preserved by the **ConMan** application when the user navigates away from the **ConMan** page.

#### 3.1 Database Design

The following database table descriptions describe the data interactions and dependencies.

User Table
UserId: Int (unique)
LastName: String
FirstName: String
Email: String
Password: String

Team Table
TeamId: Int (unique)
TeamName: String
TeamEmail: String

Team Member Table
TeamId: Int
UserId: Int
Admin: boolean

Task Table
TaskId: Int
TeamId: Int
DueDate: date
Description: String

CheckList Table
ChecklistId: Int
TaskId: Int
DueDate: date
Description: String

CheckList Fields
ChecklistId: Int
FieldId: Int
DueDate: date
Description: String
Completed: boolean

CheckList Notes
ChecklistId: Int
FieldId: Int
Date: date
UserId: Int
Note: String

Task Notes
TaskId: Int
UserId: Int
Date: date
Note: String

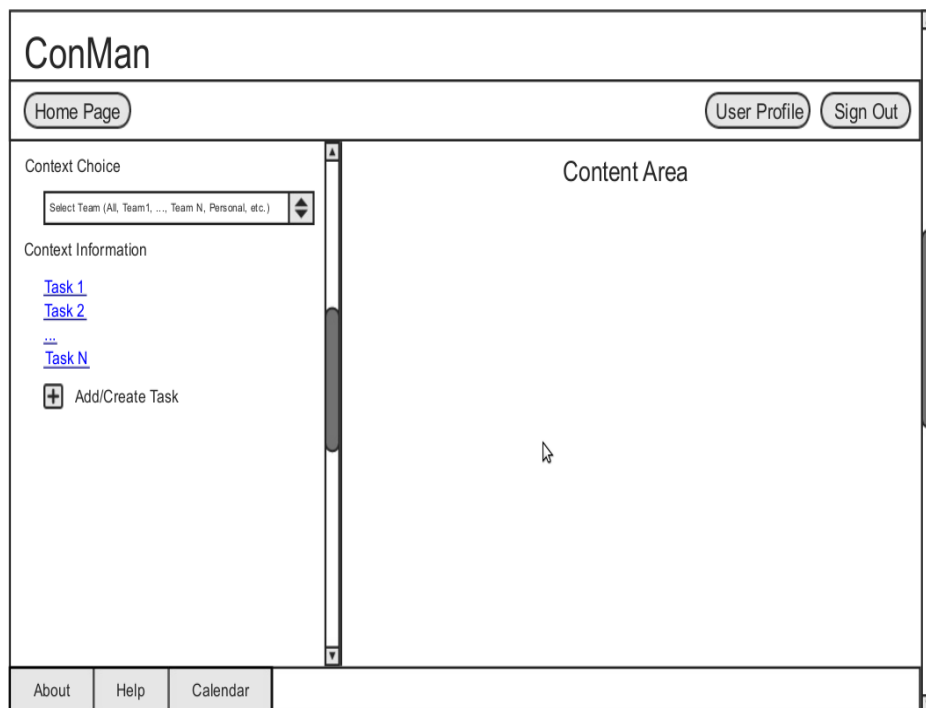
User Tasks
UserId: Int
TaskId: Int

User CheckLists
UserId: Int
ChecklistId: Int

## 4 User Interface

The user interface of **ConMan** is dominated by the use of the "main screen" (shown below). The main screen consists of four distinct parts, the top bar, the bottom bar, the left pane, and the content area. A general overview of some of the user interface designs follows.

### 4.1 Main Screen



As can be seen here, the top and bottom bars are populated with navigational tools. The left pane allows the user to select which context they would like to operate in (personal or team context) and displays options relating to the selected context. The content area then has three primary views

## 4.2 Content Area


### 4.2.1 Calendar View

### Calendar View

Range Selection:

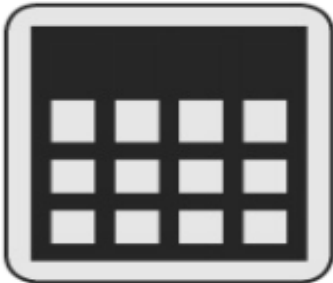
☐ Day  
☐ Week  
☐ Month

Select Date:



---

Showing Task Calendar for <context selected in side pane above>



The calendar view gives the user an opportunity to see when the tasks and checklists he is assigned are due. If the user has selected the personal context in the left pane, then only the personal context tasks will be displayed. The tasks and checklists appearing in the calendar will be hyper-links that navigate the content area to the task or checklist view respectively.

#### 4.2.2 Task View

### Task View

Task Name:

<task name appears here>

Task Description:

<task description appears here>


Task Checklist List:

[Checklist 1](#)

[Checklist 2](#)

...


[Checklist M](#)


 Add/Create Checklist


Task Notes:

<task notes appear here>

Task Actions:

 Edit Task

 Delete Task

 Sign-up for notifications

The task view will show the task name, description, checklists, notes, and available actions for the selected task. For tasks which are assigned to teams, non-administrative users will not be able to perform every action (namely deleting a task). Non-administrative users will still be able to see the administrative actions, but the action link will be obscured (usually grayed out).

#### 4.2.3 Checklist View


### Checklist View

Checklist Name: <checklist name appears here>

Checklist Description: <checklist description appears here>


Checklist Parent Task: [Parent Task Name](#)


Checklist Items: [Item 1](#)  
[Item 2](#)  
...  
[Item X](#)


 Add/Create Checklist Item

Checklist Notes: <checklist notes appear here>

Checklist Actions:

 Edit Checklist

 Delete Checklist

 Sign-up for notifications

A checklist is conceptually very similar to a task and displays similar items. Again, not every action will be available to non-administrative users.

## 5 Restrictions, Limitations, and Constraints

As detailed in the SRS, the user interface will only be supported for the Chrome and FireFox desktop web-browsers. Future versions of **ConMan** could add support more browsers and devices.

An internet connection is also required to use **ConMan**, and there are no plans to implement off-line support. Because **ConMan** is intended to assist coordination between teams (particularly software development teams), a network connection is essential.

## 6 Testing Guidelines

Testing will be integrated throughout the development of **ConMan**. Components will be tested individually before being integrated into the system for system testing. A database of test cases, test results, and action required will be maintained by the developers to ensure **ConMan** satisfies all of its major requirements.

### 6.1 Testing Cases and Expected Results

#### 6.1.1 Account Creation

**Motivation:** A user, Mr. Fat-Fingers, decides that **ConMan** sounds really cool and decides to give it a try. Unfortunately, Mr. Fat-Fingers has difficulty typing. After finally navigating to the account creation page, Mr. Fat-Fingers tries to create an account.

Potential cases:

1. E-mail field is incorrectly entered.
  - Input: An invalid e-mail (a description of a valid e-mail will be determined later).
  - Output: An error message will be displayed to the user prompting them to enter a valid e-mail address.
2. Account creation fields are left blank.
  - Input: Any of the fields for first name, last name, e-mail, or password are left blank.
  - Output: An error message appears prompting the user to populate all of the fields.
3. An e-mail already in use has been entered.
  - Input: An e-mail already in the **ConMan** database.
  - Output: An error message prompting the user to either enter a different e-mail or log onto existing account.
4. All fields are correctly entered.
  - Input: A unique and correctly formatted e-mail address, and all other fields are populated.
  - Output: A user account is created and user re-directed to the **ConMan** home page.

#### 6.1.2 Account Login

**Motivation** A returning user, Ms. Forgetful, is trying to log in to her **ConMan** account.

Potential cases:

1. E-mail field is incorrectly entered.



- Input: An improperly formatted e-mail.
  - Output: An error message prompts Ms. Forgetful to check the formatting of her e-mail address.
2. E-mail is not contained in the **ConMan** database.
    - Input: A correctly formatted e-mail that is not associated with a user account.
    - Output: An error message that prompts the user that her information could not be validated.
  3. The password and e-mail combination do not match.
    - Input: A correctly formatted e-mail that is not associated with the given password.
    - Output: An error message that prompts the user that her information could not be validated.
  4. The password or e-mail fields are left blank.
    - Input: Either password or e-mail fields are left blank.
    - Output: An error message that prompts the user to fill all fields.
  5. The password and e-mail combination are valid.
    - Input: A correctly formatted e-mail that miraculously matches the associated password entered by Ms. Forgetful.
    - Output: A user account is created and user re-directed to the **ConMan** home page.

### 6.1.3 Create a new team

**Motivation** Mr. Ringleader, would like to create a new team, the "Task-buster Trio." He has two friends who make up the trio, Mr. Fat-Fingers and Ms. Forgetful.

Potential cases:

1. Team e-mail field is incorrectly entered.
  - Input: An improperly formatted e-mail.
  - Output: An error message prompts Mr. Ringleader to check the formatting of his e-mail address.
2. The password or e-mail fields are left blank.
  - Input: Either password or e-mail fields are left blank.
  - Output: An error message that prompts the user to fill all fields.
3. The e-mail field is valid and all other entries are included.
  - Input: A correctly formatted e-mail and all other fields are entered. The given password and e-mail for the Task-buster Trio does not need to be the same as the e-mail or password combo for Mr. Ringleader.
  - Output: A new team is created and Mr. Ringleader is automatically added as the administrator. Mr. Ringleader can now switch between his team and personal contexts within the application.

#### 6.1.4 Add team members

**Motivation** Now that Mr. Ringleader has created the "Task-buster Trio," he would like to add his two friends, Mr. Fat-Fingers and Ms. Forgetful.

Potential cases:

1. E-mail field is incorrectly entered.
  - Input: An improperly formatted e-mail.
  - Output: An error message prompts Mr. Ringleader to check the formatting of the e-mail address.
2. The e-mail field is valid but no user with that e-mail is associated with a **ConMan** account.
  - Input: A correctly formatted e-mail with no corresponding account.
  - Output: An error message appears indicating that the e-mail is not found within the database.
3. The e-mail field is valid and all other entries are included.
  - Input: A correctly formatted e-mail and all other fields are entered.
  - Output: The users associated with the e-mail addresses (Mr. Fat-Fingers and Ms. Forgetful in this case) are added to the team member list.  
Mr. Fat-Fingers and Ms. Forgetful can now see the "Task-buster Trio" context within their **ConMan** accounts.

#### 6.1.5 Add tasks

**Motivation** Ms. Forgetful cannot remember what she was supposed to do, except that she must add a task to her **ConMan** account. She opens the **ConMan** application and tries to add a task named "placeholder."

Potential cases:

1. Ms. Forgetful is in her basic context when she clicks "add task."
  - (a) Ms. Forgetful leaves a required field blank.
    - Input: An task-creation entry with a blank required field.
    - Output: An error message prompts Ms. Forgetful to include all required fields.
  - (b) The date manually entered by Ms. Forgetful was for some point in the past.
    - Input: A task creation request with an already expired due date.
    - Output: An error message appears telling her to pick a future date.
  - (c) The e-mail field is valid and all other entries are included.
    - Input: A correctly formatted e-mail and all other fields are entered.
    - Output: A task that is only visible to Ms. Forgetful is created.
2. Ms. Forgetful is in her "Task-buster Trio" context when she clicks "add task."

- (a) Ms. Forgetful leaves a required field blank.
  - Input: An task-creation entry with a blank required field.
  - Output: An error message prompts Ms. Forgetful to include all required fields.
- (b) The date manually entered by Ms. Forgetful was for some point in the past.
  - Input: A task creation request with an already expired due date.
  - Output: An error message appears telling her to pick a future date.
- (c) The e-mail field is valid and all other entries are included.
  - Input: A correctly formatted e-mail and all other fields are entered.
  - Output: A task that is visible and editable to Mr. Taskmaster and Mr. Fat-Fingers is created.

## 7 Appendices

### 7.1 Definitions

- Client-Server: A system architecture where a central server provides data to one or many networked computers (these are the clients).
- MVC: A design pattern commonly used for building interactive applications. The Model, View, and Controller components form the structure of the MVC pattern. The Model is a representation of the underlying application data, the View is a collection of objects representing the user interface elements (buttons, list, etc.), and the Controller is a connection between the Model and the View.
- HTTP: Hypertext Transfer Protocol (HTTP) is a standard for communication between servers and web browsers.
- URL: Uniform Resource Locator (URL) is a string of characters denoting the location of an entity (e.g., a website or an image) within the Internet.
- JSON: JavaScript Object Notation (JSON) is a text-based standard for data interchange in a human-readable format.