**Django ToDoList Project Specification – 2 hour project**

|  |  |
| --- | --- |
|  | Jacob Kiggins |

# **Communications**

Develop a Todo List web app using python and django. This Todo List will allow for basic operations (add delete modify) and hold the values status, description, and due date.

Extended features will also be implemented. The revision history of a specific Todo Item is available for viewing. From the main table containing the Todo List, each todo item may be sorted by a particular column name.

Below is a much more detailed list of requirements, Green marked items are included features, Red marked are not included. Bold and Green text was a non-negotiable requirement from the customer.

**Requirements:**

1. **Create a Django application that manages a To-Do List for one user.**
2. **Implement the software using the versions of Django and Python required by your customer**
   1. **python 3.4.3**
   2. **django 1.9.1**
3. **The fields to capture for each to-do list item are Description, Due Date, and Status.**
4. **Allow the user to Create, Review, Update, and Delete a to-do list item.**
5. Implement the application via the Admin Console.
6. Implement via a custom webpage
7. When initially viewing the to-do items, show all of them in list form with column headings.  Each to-do list item shown in list form can be edited (all 3 fields) by clicking on the view link.  You will then see a web page with all 3 fields that can be changed for the to-do item previously selected.
8. When creating a new to-do item, load the Status field with a default value of "New".
9. Optional requirements.  The following are "nice to have" and can be included, or not, in the application:
   1. Add authorization security to get into the to-do application
   2. Allow all or a selected number of to-do items to be deleted at once
   3. When displaying to-dos in list form, allow the to-dos to be sorted by clicking on the field/column headers
   4. Allow Time to be included as part of the Due Date.  Show a calendar icon lookup for selecting the date.
   5. For validation, require that all fields/columns have a value.
   6. Show the history of changes to each to-do item.
10. If you are spending over 2 hours on this application from start to finish, you are doing too much (non-functional requirement).
11. After the application has been tested, zip the Django application files (from the first *mysite* folder down). Include this spec document under the first *mysite* folder.
12. **Provide instructions on how to run the application in a brief README.txt file.**

# **Planning**

This document will serve as the project plan for the ToDo List activity.

Each item marked green in the requirements section will be implemented in the finished application. Each item marked red was a possible feature that was not implemented due to time constraints. Each item bolded was a non-negotiable requirement.

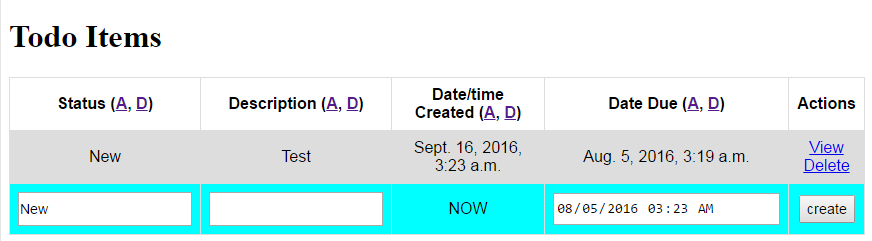
For every item marked green in the requirements section their is a corresponding model in the Modeling section.

With the models complete the project may be implemented as described in the Construction section. Once construction is complete the project will be deployed as described in the Deployment section.

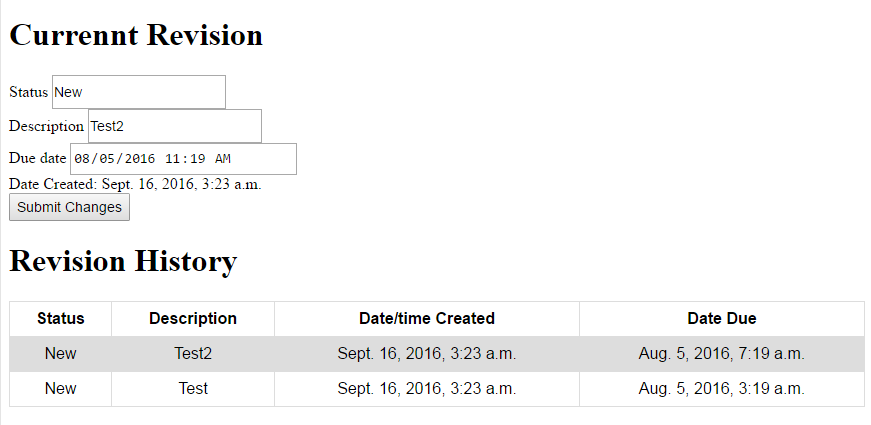
The estimated time to complete this project is 2 hours and it will be finished and handed in by 9/16/2016 at 11:59 pm.

# **Modeling**

The design will be based around two web pages. One is the list view which displays the Todo items for the user. The items will be formatted as below:



The actions column will contain two links, View and Delete. View will take the user to a page modeled below:



From this page the user has the ability to modify a todo item and view the revision history for a todo item.

The delete link simply deletes that particular item and all of its history.

More information can be found in the construction section of this document.

Use Case Diagram

use_case.png

Data Models

todo_models.png

# **Constructio**n

As the requirements suggest this project will be implemented via python using the django web framework. It was constructed using PyCharm and Git was used for version control.

The various use cases are outlined in detail below:

Use(Test) Case Descriptions

|  |  |
| --- | --- |
| **Use Case Number:** | UC-01 |
| **Use Case Name:** | Create Todo Item |
| **Overview:** | The User will input status, description and due date and prompt the system to create a new todo item |
| **Actor(s):** | User |
| **Pre-condition(s):** | - System has been setup and configured.  - System is running  - User is on the index of the todo application |
| **Scenario Flow:** | Main (success) Flow:   1. User inputs status, description, and due date on the main page 2. The user presses the create button 3. The system creates the todoitem and redirects the user back to the main page, where they can view the new item 4. POST CONDITION: There is a new Todo Item in the system, The user is on the todo index page and the new todo item input fields are clear |
| **Alternate Flows:** | Alternate Flow #1: At step 1 the user wants to reset the todo input fields   1. The user reloads the page 2. POST CONDITION: Fields are clear |

|  |  |
| --- | --- |
| **Use Case Number:** | UC-02 |
| **Use Case Name:** | Modify Todo Item |
| **Overview:** | The User will view the details of a todo item and use the provided input fields to modify the item |
| **Actor(s):** | User |
| **Pre-condition(s):** | - System has been setup and configured.  - System is running  - User is on the index of the todo application |
| **Scenario Flow:** | 1. The user clicks the view link next to a todo item in the actions column 2. The system sends the user to the todo item detail page 3. The user modifies the todo item by changing the values of the input fields displayed 4. The user clicks the submit button 5. The system modifies the todo item 6. The system redirects the user to the main page 7. POST CONDITION: The user is on the index page, a todo item has been modified and can be viewed |
| **Alternate Flows:** | Alternate Flow #1: The user decides they don’t want to modify the todo item before step 4   1. The user hits the back button 2. The user is brought to the index page 3. POST CONDITION: Nothing within the system has changed |

|  |  |
| --- | --- |
| **Use Case Number:** | UC-03 |
| **Use Case Name:** | View Todo Item revision History |
| **Overview:** | The User will click on the detail link and view the revision history on the detail page. |
| **Actor(s):** | User |
| **Pre-condition(s):** | - System has been setup and configured.  - System is running  - User is on the index of the todo application |
| **Scenario Flow:** | Main (success) Flow:   1. The user clicks the view link next to one of the todo items in the actions column 2. The system directs the user to the todo item detail page 3. The user observes the revision history 4. POST CONDITION: the user is on the todo item detail page for a particular todo item |
| **Alternate Flows:** | Alternate Flow #1: There are no todo items   1. The user can not view any revision history 2. POST CONDITION: Nothing has changed |

|  |  |
| --- | --- |
| **Use Case Number:** | UC-04 |
| **Use Case Name:** | Delete a todo item |
| **Overview:** | The User will click the delete link in the actions column next to a particular todo item, this item will be deleted. |
| **Actor(s):** | User |
| **Pre-condition(s):** | - System has been setup and configured.  - System is running  - User is on the index of the todo application |
| **Scenario Flow:** | Main (success) Flow:   1. The user clicks the delete link in the actions column next to a todo item 2. The system deletes the todoitem and redirects the user back to the main page 3. POST CONDITION: the item is deleted, the user is on todo app index |
| **Alternate Flows:** | Alternate Flow #1: There are no todo items   1. The user can not delete an item 2. POST CONDITION: Nothing has changed |

Each use case matched system functionality for all flows.

# **Deployment**

This document is available at the root (mysite) folder of the django todo list project.

To deploy the server first the user must

1. download the djangoTutorial.zip file.
2. unzip the project into a directory of your choosing
3. open a cmd window in the current directory
4. Type: python manage.py runserver

This assumes you have the correct python and django versions outlined in the requirements section installed and configured correctly.

To view the todo list application navigate to <http://127.0.0.1:8000/todo/> after the above steps are followed.

This document along with the project will be zipped and submitted on the myCourses dropbox