TODO Application

Jon Stoddart

Ey6685@wayne.edu

https://github.com/ey6685/toDoApp

January 14, 2019

TABLE OF CONTENTS

1. INTRODUCTION

The objective of this exercise is to practice your software engineering skills by developing and documenting a simple web-based ToDo Application. The exercise is a condensed, simplified version of all the tasks you will need to complete throughout the semester.

1.1PURPOSE OF THIS SPECIFICATION DOCUMENT

The purpose of this document is to record the information related to the architecture and system design of the product and its components.

1.2PROJECT REQUIREMENTS

The client has specified a few requirements for this application from which we have derived Functional and Non-Functional requirements.

Functional Requirements

- Ability to add tasks
- Ability to delete task
- Ability to view tasks

Non-Functional Requirements

- List must be persistent
- Database operations must be completed within 2 seconds
- Web application must have 99% uptime

2. ARCHITECTURE DESIGN

2.1 HARDWARE ARCHITECTURE

This product is demonstrating a proof-of-concept and is designed to be run on a local machine. The software is built to be compatible with any machine running Windows 10 with an i5 series Intel Processor having at least 4GB of RAM and 10GB of free storage space on the hard drive. Any other system setup may result in varying degrees of difference in performance.

2.2 SOFTWARE ARCHITECTURE

This web application is built using the XAMPP Stack. There are three components in the To Do application: the interface the user directly interacts with in their browser, which hands the requests off to the middleware, thereby authorizing the request and send the commands to the local database.

Since this is a proof-of-concept, the user interface is built using HTML5 with embedded PHP scripts. Requests are sent to the middleware, TomCat, which is running on port 80 for our application. In this application, requests are authorized using the default username and password for XAMPP. The PHP request is then sent to the MySQL database running on port 3306 and the embedded SQL statement is executed.

2.3 SYSTEM ARCHITECTURE

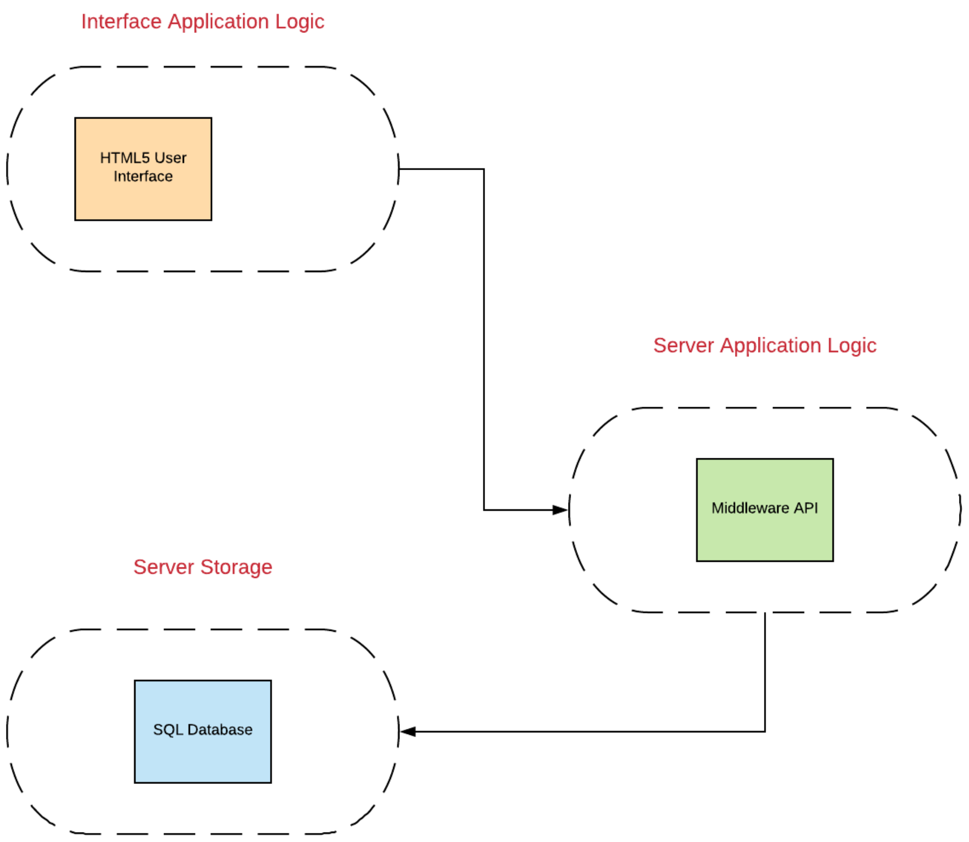


Figure 1: System Architecture Diagram

3. SYSTEM DESIGN

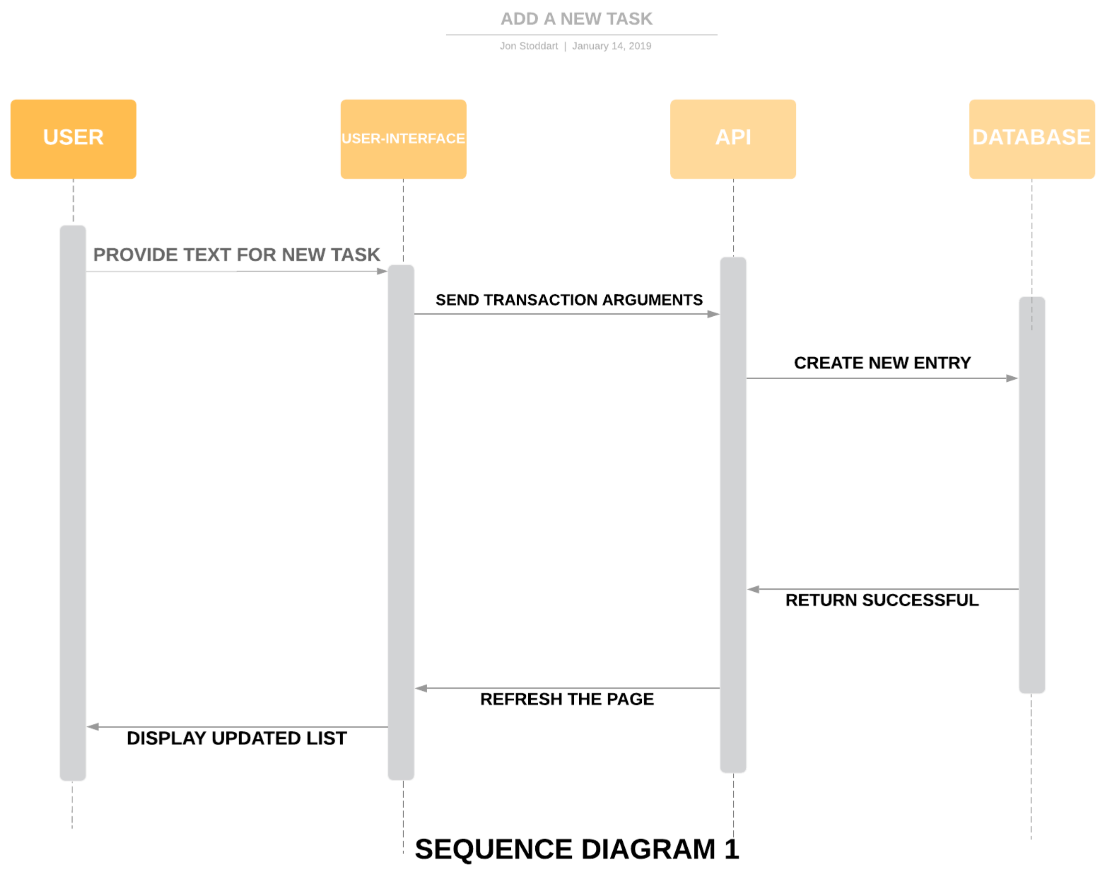
3.1 USE CASES

Use Case ID:	01	Functional Requirement:	Add a new task.
Use Case Name:	Create a new task.		
Description:	This use case describes a user that would like to create a new task in the database.		
Trigger:	User enters text in the “Type a new task here.” Field and clicks Add.		
Preconditions:	1. User needs to have navigated to localhost/toDoApp/viewtasks.php in their browser. 2. User needs to have typed a new task in the “Type a new task here.” Field and clicked the “Add” button.		
Postconditions:	1. The “Your Tasks” page should then refresh showing the new task has been created successfully.		
User Groups:	User1 as created when the database was initialized.		
Normal Flow:	1. User navigates to localhost/toDoApp/viewtasks.php after following the pre-conditions. 2. User enters the new task in the “Type a new task here.” Field and clicks the “Add” button. 3. Page will then refresh showing the new task has been added successfully.		
Alternative Flows:	1. If the user clicks the “Add” button without any text entered in the “New Task” field then an error message will be displayed asking for text to be entered beforehand.		
Frequency of Use:	The user will use this feature every time they enter a new task.		
Assumptions:	1. XAMPP is installed and running. 2. User needs to have the directory “toDoApp” downloaded to the path C:\ …\xampp\htdocs. 3. User needs to have navigated to localhost/toDoApp/index.php in their browser. 4. User needs to have clicked “Initialize Database”.		

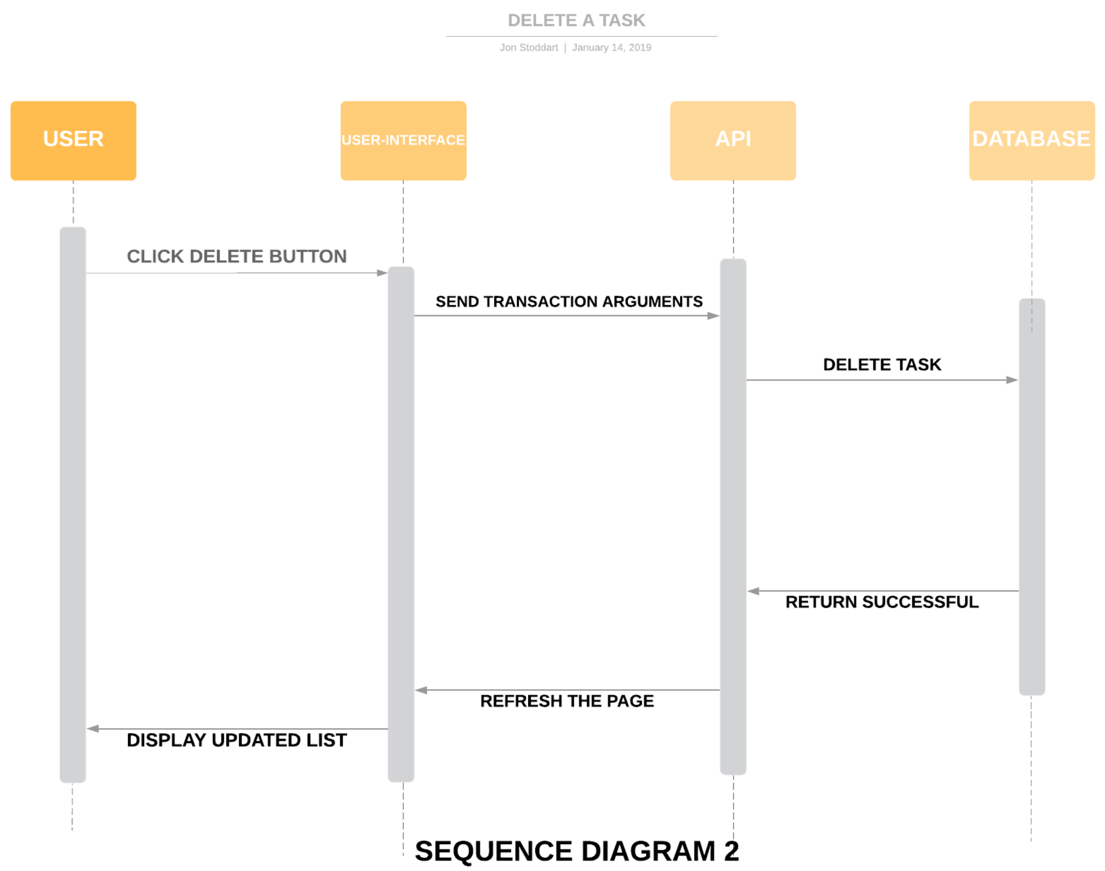
Use Case ID:	02	Functional Requirement:	Delete a task.
Use Case Name:	Delete an existing task.		
Description:	This use case describes a user that would like to delete a task from their list.		
Trigger:	User clicks the "Delete" button next to the task they would like to delete.		
Preconditions:	1. User needs to have navigated to localhost/toDoApp/viewtasks.php in their browser. 2. User then clicks the "Delete" button next to the task they would like to remove.		
Postconditions:	1. The "Your Tasks" page should then refresh showing the task has been deleted.		
User Groups:	User1 as created when the database was initialized.		
Normal Flow:	1. User navigates to localhost/toDoApp/viewtasks.php after following the pre-conditions. 2. User clicks the "Delete" button next to the corresponding task they would like to remove from their list.		
Alternative Flows:	1. If the user has not added a task to their list then they will not see a "Delete" button.		
Frequency of Use:	The user will use this feature every time they remove a task.		
Assumptions:	1. XAMPP is installed and running. 2. User needs to have the directory "toDoApp" downloaded to the path C:\ …\xampp\htdocs. 3. User needs to have navigated to localhost/toDoApp/index.php in their browser. 4. User needs to have clicked "Initialize Database". 5. User needs to have entered at least one task into their list.		

Use Case ID:	03	Functional Requirement:	View Tasks
Use Case Name:	View existing tasks.		
Description:	This user case describes a user who would like to view the tasks in their list.		
Trigger:	User navigates to “localhost/toDoApp/viewtasks.php” in their browser.		
Preconditions:	1. User needs to have already added tasks into their list.		
Postconditions:	1. The “Your Tasks” page displays any tasks you have previously added.		
User Groups:	User1 as created when the database was initialized.		
Normal Flow:	1. User navigates to “localhost/toDoApp/viewtasks.php” in their browser which queries the database and prints out any tasks they have in their list.		
Alternative Flows:	1. If the user navigates to “localhost/toDoApp/index.php” in their browser, they can also navigate to their list by clicking the “View List” button.		
Frequency of Use:	The user will use this feature every time they want to view their list.		
Assumptions:	1. XAMPP is installed and running. 2. User needs to have the directory “toDoApp” downloaded to the path C:\ …\xampp\htdocs. 3. User needs to have navigated to localhost/toDoApp/index.php in their browser. 4. User needs to have clicked “Initialize Database”. 5. User needs to have entered at least one task into their list.		

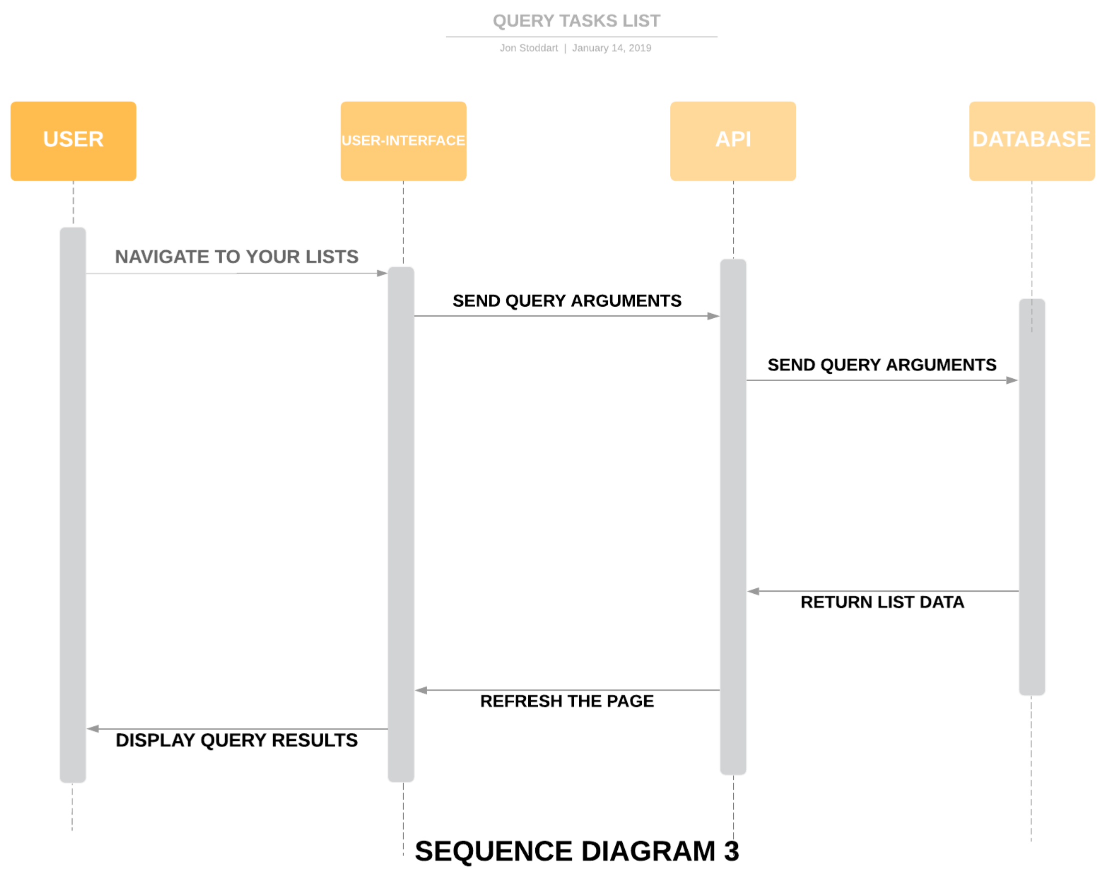
3.2 SEQUENCE DIAGRAMS



SEQUENCE DIAGRAM 1

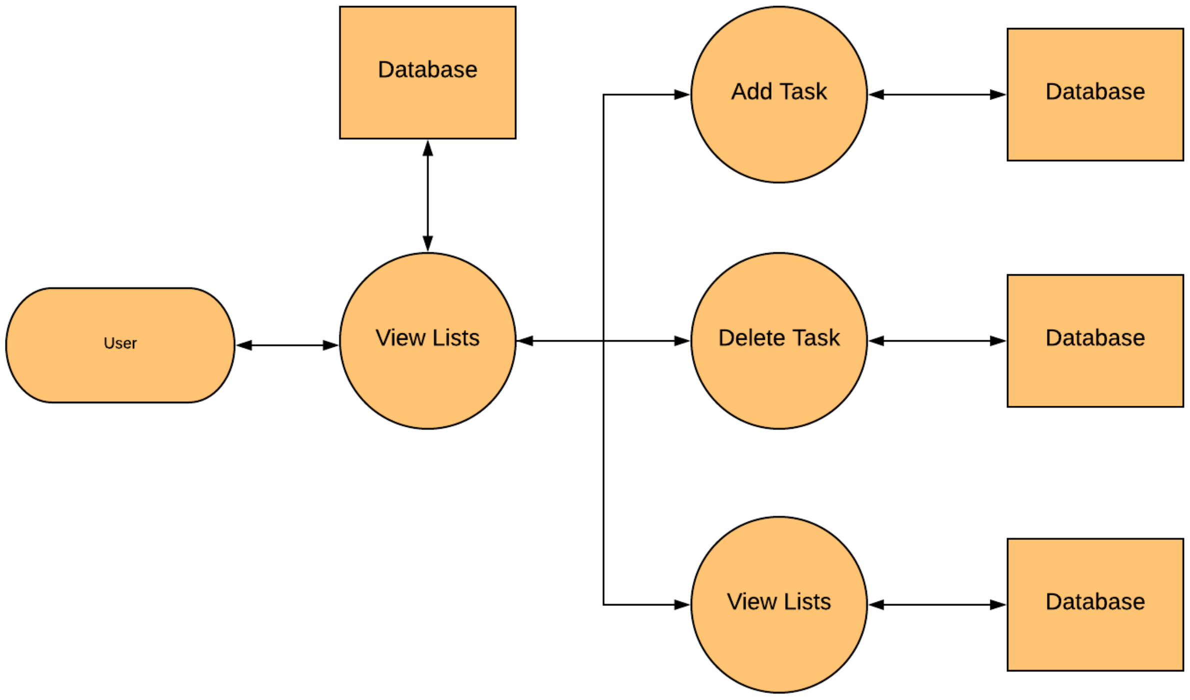


SEQUENCE DIAGRAM 2



SEQUENCE DIAGRAM 3

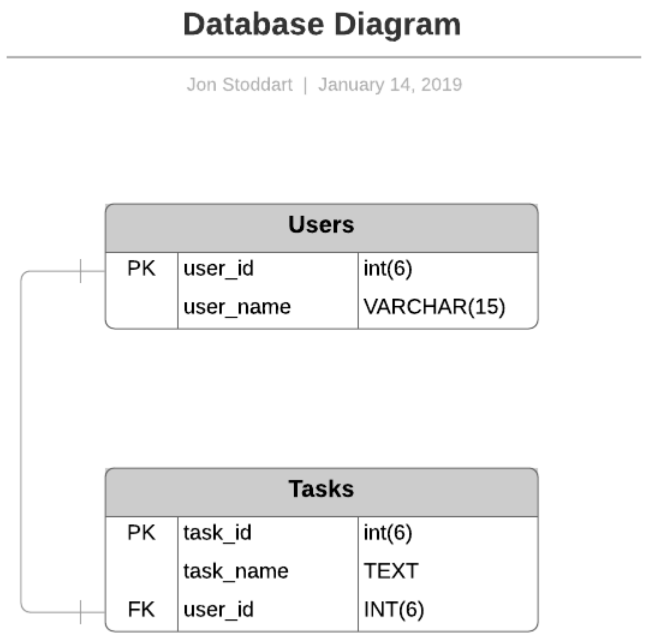
3.3 DATA FLOW DIAGRAM



DATA FLOW DIAGRAM

3.4 DATABASE DESIGN DIAGRAM

For this implementation I decided to go with only two tables since a third table would be unnecessary. I hypothesized a scenario where a DESCRIPTION table includes a longer description of the task, but this seemed to add unnecessary complexity for the desired result.



3.5 CLASS DIAGRAM

Since this exercise is meant to be very simple there are no classes that were used. I felt that trying to implement classes in this application would be adding unnecessary complexity.

3.6 TEST CASE

Based upon the test case descriptions and examples I found online, the layout and content of these test cases would look very similar to my use cases listed in Section 3.1. For this reason, I've decided to simply reference them instead of listing the same information in a slightly different wording.