

# **Todo**

## **Table of Contents**

### **1. Project Overview**

- 1.1. Team Members
- 1.2. Project Summary
- 1.3. Project Links
- 1.4. User Manual

### **2. Workflow Management App Details**

- 2.1. Strategic Analysis
- 2.2. Product Vision
- 2.3. Features

### **3. Customer Information**

- 3.1. Personas
- 3.2. User Manual Summary

### **4. Design and Architecture**

- 4.1. Styling
- 4.2. Architecture Design and System Architecture
- 4.3. Component Design and Specification
- 4.4. Data Structure Design and Specification
- 4.5. Algorithm Design and Specification

### **5. Scrum and Agile Process**

- 5.1. GitHub Scrum Board Configuration
- 5.2. Agile Process
- 5.3. Calculated Capacity and Velocity

5.4. Team Norms

## **6. Testing Documentation**

## **7. Sprint Updates**

7.1. Sprint 1

7.2. Sprint 2

7.3. Sprint 3

7.4. Sprint 4

7.5. Sprint 5

## **Section 1**

### **1.1. Team Members**

Leonardo Rodriguez - Team Leader

Adriana Kamor - JavaScript programmer

Ethan Ho - CSS programmer

Addy Krom - JavaScript programmer

Kamsiyochukwu Eneh - Database programmer

William Huang - Database programmer

### **1.2. Project Summary**

The objective of this project is to design, create, and implement a personal planner application geared toward students. Used by those who structure their day through simple tasks, this app offers a seamless form of organizing, scheduling, and visualizing their days. Todo combines the best aspects of a planner and a traditional checklist to make tasks more approachable for the user. This application will have three main features. Our first feature will be the core functionality of our planner application; the ability to create and delete tasks. Our second feature will be that tasks can be marked off as “completed” by a checkbox and a line through the task. Our final feature will be to sort tasks chronologically based on a given due date and time. Users will be able to create an

account and log in to access their past information and alter that information, which will then update in the application's database. In order to implement these features and allow users to access their information, there are multiple web application pages that users are required to go through before getting started. To register, there is a registration page to set up a username and password, a login page to access the account created, and a home to-do list page where users can see and edit items.

### 1.3. Project Links

GitHub Repository (The user manual is a part of the repository README file):

<https://github.com/cwru-courses/csds393-fall2022-five-guys-and-gal.git>

GitHub Project: <https://github.com/orgs/cwru-courses/projects/6>

Hosted Web Application: <https://whispering-gorge-40839.herokuapp.com/>

### 1.4. User Manual

#### Instructions

#### Overview

Welcome to Todo, the smarter planner application for students. This user manual has sections for setting up and using your account, troubleshooting, and information about the GitHub repository for this product. Todo's capabilities include the ability to add and delete tasks, assign and sort tasks by date and time, and label tasks under an event category. In order to save your data and keep your list private, you will need to register with an account and log in to the application.

#### Accessing the Application

To access the application, use the link <https://whispering-gorge-40839.herokuapp.com/> in your web browser. After a few seconds, the Todo application should load and be ready for use.

#### Creating an Account

Before being able to create your own planner list, you will need to register for an account on Todo. When opening Todo, for the first time, you will see that the home page displays the message "Please log in to see this page." This is because you are not logged into an account. To register, navigate to the Register page using the buttons located at the top of the page. This should bring you to a page with 2 fill-in fields, the first being for an email

and the second for a password. In the respective fields, enter an email you would like the account to be linked to and create a password for your account. Once typed in, hit the blue Register button at the bottom of the page to register your account.

### Logging In

To log in with your account information, navigate to the Login page using the buttons at the top of the page. Enter the email you used to register and the password you created for your account in the respective boxes, and click the blue Log In button at the bottom of the page. After clicking the Log In Button, the application will automatically bring you to the Home page, which should now have the title What todo? along with multiple boxes for adding task information.

### Navigating Application Functionality

#### Adding Tasks

Once logged in and on the Home page, you will now see various boxes. To create a task, you must fill in all of these boxes with their respective values in order to successfully add the task to your list. First, in the Description box, type in the name or label of your desired task. For the next box, you must choose a due date for your task. To add a due date, click the calendar icon on the right of your screen, and once clicked a date picker should appear on the left of your screen. From this point, click on your desired date and the date box will automatically update with your selection. In the next box, you will choose a due time for your task. Similar to the due date, you need to click the clock icon on the left side of the button which will pull up a time picker. Once you have chosen a time, you can exit the time picker by clicking any blank space on the application. The last box features a drop-down of multiple categories under which you can classify your task. Click anywhere in the box and select a category, after which the drop-down menu will automatically close. Once all of these fields are filled out, you may hit the blue Submit Task button to add your task to the list. Once clicked, you should see your task appear in the list below the entry fields.

#### Completing Tasks

If you have completed a task in your Todo list, you can mark the task off by clicking the square box that appears at the top center of each task. Once clicked, your task will be crossed off the box at the top of the task will be checked off, denoting that this task has been completed.

#### Deleting Tasks

If a task is no longer applicable, canceled, or you wish to get rid of it, you have the option to delete your tasks. In order to do this, click on the trash can icon that appears at the

bottom center of each task. Once clicked, your task will be deleted. Be careful, as this feature permanently deletes tasks from your list.

### Sorting Tasks

With Todo, you have two ways to view your Todo list. The first of which is by the time each task was created, and the second is in order of ascending due date. In order to sort your list, you should see the Sort By label next to a box that either says Time Created or Date. By clicking on this box, a drop-down menu will appear with both the Time Created and Date choices. To sort your list in the order in which you added tasks, choose the Time Created option. To sort your list in order of ascending due dates, choose the Date option. After clicking one of these options, the drop-down menu will automatically close and your list will automatically sort itself according to the sorting choice you made.

### Troubleshooting

#### Application Loading Issue

Because this application is hosted on a third-party website, there may be a delay in terms of the application loading on the user's end. However, if the application does not load after a minute, close out of the application completely. Make sure to check that you have a stable internet connection and retry the link to the application. If this happens repeatedly, please reach out to one of the contributors listed at the top of the user manual.

#### Task Not Being Added

If a task is not being added and you see error warnings appearing on your screen, this is because you did not fill out one of the necessary fields required to add a task. Todo requires you to fill out all fields including Description, Date, and Time in order to add a task to your list.

### Repository

#### Navigation

To access the repository, use the following link:

<https://github.com/cwru-courses/csds393-fall2022-five-guys-and-gal.git>. To view the user manual/README file in a separate window, click on README.md in the repository.

There are two main folders in our repository: api and client-app. As referenced by their names. api contains all necessary API, code, schema, and database information to create the application interface and base while client-app is where the styling and app functionality can be found. Within client-app, there are two folders. The first one, public, is imported files from the initial setup of the React app, and the second one, src, is where a majority of the product's code is found.

## Scrum Board

To view the Scrum board used throughout the creation of this application, you can view it through the following link: <https://github.com/orgs/cwru-courses/projects/6>. In this Scrum board, you are able to see the tasks, acceptance criteria, sprint layout, and story points of each user story that was made to complete this product's functionality.

## **Section 2**

### **2.1. Strategic Analysis**

Todo is a personal planner application that will help students organize and plan out both short-term and long-term tasks. Our application will solve the issue of prioritizing tasks based on their due date instead of a traditional to-do list, which has not traditionally assisted in the order of completion. Todo was created with students in mind and is appropriate for all levels of education. Todo is an application suitable for all of a student's needs because tasks are customizable to the student's input. Specifically, Todo has the ability to add and remove tasks from a to-do list, chronologically sort these tasks by their due date and time (assigned with the creation of a task), designate a task category, and mark the completion status of tasks. Tasks are also protected behind an account username and password in order to protect the user privacy and save task progress.

### **2.2. Product Vision**

Our product vision is to provide user-friendly, multi-purpose personal planning services for hardworking, responsibility-oriented students of all ages.

### **2.3. Features**

#### Feature #1: Core Product Functionality

This feature is the foundation of the product. It allows users with the ability to manipulate tasks to their desire. This includes the functionality to add, remove, and complete tasks. To add a task, the user can fill out the appropriate task fields at the top of the web application and add their task to the list by hitting a submit button. This will then be added to the user's account and saved after logging out. In order to delete a task, the user may press the "trash can" icon which will remove the task from the list. Along with this, the interface will provide clear information regarding the status of each task. The possible statuses for a given

task are denoted by a check box, which when clicked will cross the task off the list and therefore represent if a task is “complete” or not. These functionalities make up the core of ToDo.

### Feature #2: Chronological Date Organization

The user’s tasks can organized in chronological order. This feature builds on the information required from Feature #1 for task creation and reads a selected due date and due time for tasks. Using a dropdown menu, the user must choose a date and time for each of their tasks prior to submitting the task to their list. With the date and time parameters, users have the option to sort their tasks by creation or by ascending due date and time. This sort is initiated by the push of a “sort button” located under the “submit task” button. This allows the user to have a choice as to whether they wish for their tasks to be reorganized or not based on how they prefer to map out their tasks.

### Feature #3: Vague Event Categorization

This feature allows the user to mark important events/dates. While Feature #1 and Feature #2 are used for marking down specific task names and due dates/times, this feature is used to sort tasks into broader categories. Using a dropdown style menu below the name, date, and time parameters while creating a task, the user can designate a task as “General,” “Assignment,” “Exam,” or “Vacation.” Through categorization, the user can use it to pick up patterns in their workload or to see where a majority of their tasks lie.

## **Section 3**

### **3.1. Personas**

#### Persona #1: Jane - Undergraduate College Student

Jane, 19, is an undergraduate college student whose day-to-day is broken down into blocks of time commitments. Jane has five college courses, is in the university orchestra, and has a part-time job at Panera. Jane’s college courses are spread erratically Monday through Friday, her orchestra rehearsals on Tuesdays and Thursdays, and work shifts every other day. Jane is overall a great student and employee, but often forgets simple tasks she told herself she was going to complete earlier. This includes submitting certain assignments, doing laundry, taking out the trash at closing time, and planning time to relax. Jane seeks a way to stay on top of all of her commitments.

## Persona #2: Jonathan - High School Student

Jonathan, 17, is a high school senior amidst one of his most hectic school years. Jonathan is applying to colleges. Jonathan is balancing different deadlines for essays, financial-aid documents, and interviews. On top of Jonathan's school-commitments, he is the oldest of five at home. His parents both work and depend on him to take care of certain household and family responsibilities. Jonathan often has to pick up his younger siblings from school or soccer practice, prepare ingredients for his parents to come home and prepare dinner, and walk their family dog. Jonathan, being in one of the most crucial times of his life so far, needs to make sure he is organized and taking care of everything that needs to be done.

## **Section 4**

### **4.1. Styling**

For this product, our team felt that a simplistic and streamlined design would work best to make the user's tasks the focal point of the product. By making Todo easy to navigate and interpret, we believe that this will create a more seamless user experience and increase user satisfaction after discussing styling with multiple college students. For the application's color scheme, we are using a combination of the neutral colors black, gray, and white. The acceptable HEX colors are #878787, #0f0f0f, #303030, and #FFFFFF. The "submit button" is the only exception as its HEX color is #0a5acf, an electric blue, in order to highlight the main functionality of adding a task to the list. Text should appear lighter and the background should be darker, as the concept of "dark mode applications" has been studied to reduce glare, blue light, and eye strain. Since students will use this application at all points of the day, it was necessary to make sure that the app would also be easy to use in low-light environments as well. All fonts used should be sans serif for easier readability and the font style can be any of the following fonts: Segoe UI, Roboto, Helvetica Neue, Noto Sans, Liberation Mono and Sans, Arial, SFMono Regular, Menlo, Monaco, Consolas, and Courier New. All buttons should have rounded corners based on the design concept that rounded buttons direct the user's attention into the button and are found to be more friendly by users.

### **4.2. Architecture Design and System Architecture**

For Todo's high-level architecture design, a straightforward multi-tier architecture model was implemented. In summary, the hosted client end of Todo interacts with the combined server and database capabilities of MongoDB. The client is hosted using Heroku's third-party hosting capabilities. For the more specific components and architecture of the application, visual representations and their descriptions are available in 4.6, the UML diagram section of Design



and Architecture. There are three major architectural breakdowns for the structure of our application: creating a task, logging in, and the file setup of the application.

### **4.3. Component Design and Specification**

For the component architecture, 3 different classes were used to build the main functionality of the application: Register, Login, and Home. In addition to these 3, the App.js and App.css classes are used for the basic styling and structure of the application. Because JavaScript, the main language used to develop Todo, does not support interfaces, interfaces are not applicable to our product's architecture. However, we have still used a modified approach of modules to break down the Home class, while counting the other classes as modules. The Home class is broken down into the following modules: Add, Delete, Validate and Update, Field Information, and Map. Add covers the implementation of adding a task to the Todo list with the addTodo method and Delete covers the implementation of deleting a task in the Todo list with the deleteTodo method. Validate and Update includes the isValid and updateTodo methods which check if a new task is valid and update the list depending on the addition and deletion of tasks. Field Information includes all of the input options required in order to add a task, and Map maps all respective fields to class names, icons, and labels.

### **4.4. Data Structure Design and Specification**

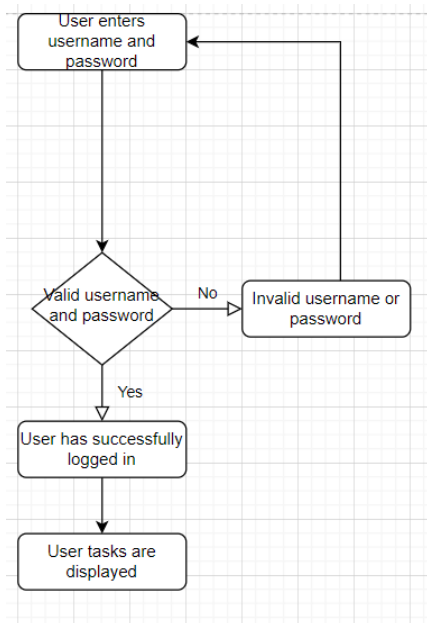
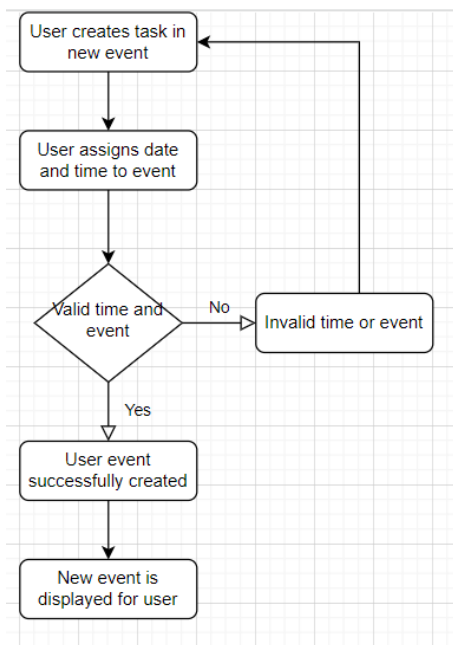
For this application, a nontraditional form of data structures and storage was used based on how MongoDB structures its data. MongoDB is a NoSQL database which would allow for easier scalability if this project was to be implemented to a wider audience, which in turn changes the structure of the application. MongoDB utilizes JSON-like structures and stores data in collections and documents, unlike tables and rows in SQL databases. Because the data for Todo is not relational, the algorithm design and schema are easier to implement because concentration is not being put as much into the relational nature of the data structure.

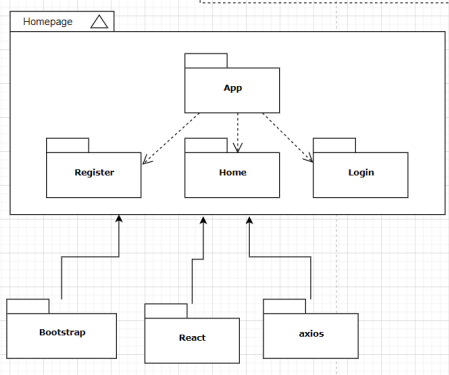
### **4.5. Algorithm Design and Specification**

Overall, the algorithm design of Todo is very straightforward and occurs in the order of addition for both tasks and user accounts. For the majority of the methods that add tasks and their parameters to the database and user interface, the "push" and "post" API methods are used to push values to the database, delete tasks, and update the interface. The main algorithm implemented is the sorting capabilities that are able to sort a user's Todo list by the creation or by due date and time. The sorting algorithm works in two different ways: by sorting by creation and by sorting by due date and time. Both sort algorithms are triggered by the press of the respective sort label in a drop-down menu present in the application. Both options take their respective sort variables, use a sorting structure similar to a node tree of the tasks, and push the new sorted list onto the user interface so the new order is displayed.

### **4.6 UML Diagrams**

Here are UML diagrams associated with specific systems in our project.





The first activity diagrams describe the steps of how a user is able to create a new event and log in. An event can only be created if there is a valid time and event, else the user will have to recreate the event. Similarly, a login can only occur if a valid username and password exists in our database, else the user will be asked to type in their login credentials again. The last diagram is a package diagram demonstrating which packages are needed in order to make the homepage work correctly. The homepage uses the files within its system along with the Bootstrap, axios, and React files.

## Section 5

### 5.1 Team Norms

- Exhibit respect for other group members and their opinions.
- Actively communicate with the team with questions, status updates, and any information relevant to the project.
- Contribute your part of the project in a timely manner based on user story and sprint deadlines.
- Trust your team members.
- When confronted with a problem, share it with other team members to avoid having “blocked” user stories.
- Work towards team goals, not individual goals.

### 5.2 Agile Process

Utilizing the scrum framework, we have planned a process centered around change and adaptability. At the start of each sprint, we will agree upon appropriate user stories to implement during the sprint. These user stories will be taken from the sprint backlog and picked based on our initial calculated velocity and capacity for the average sprint. Team members have generally been split into roles between 3 categories: database, JavaScript, and CSS. All team members collaborate in these respective roles for each user story to complete the tasks required under each story. Each user story will be worked on until certain, defined acceptance criteria is met. During this period of time, the user story will be under the “In Progress” section on the scrum board. If there are obstacles

preventing the user story from being completed, the user story is moved to the “Blocked” section. Once the acceptance criteria is met, the user story will be moved to the “In Testing” section of the board. The functionality will be thoroughly tested to confirm that the user story requirements have been met completely. From there, the user story will be sent to the “Done” section. As the end of the sprint gets closer, we will establish if the user stories are “Done Done,” implying that they are fully implemented and presentable. While there will be no daily scrum, the team will meet in person twice per sprint to work on and assign larger task areas for the sprint. In addition to this, there is regular discussion over messaging platforms and emails to define and discuss any blockers or potential issues in the sprint. At the end of the sprint, the team will have a sprint retrospective to analyze what needs to be changed in the upcoming sprint based on the performance of the previous one and update the respective sprint interim report.

### **5.3. Calculated Capacity and Velocity**

#### Velocity:

Total Story Points: 21

Number of Sprints: 5

Velocity: 4.2 → 4 points per sprint

#### Capacity:

Number of Group Members: 6

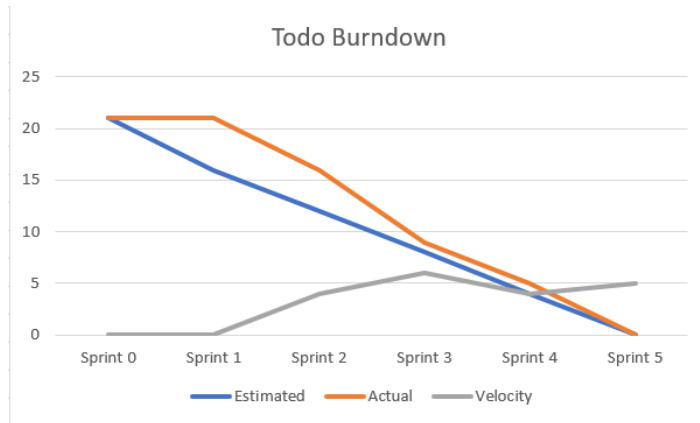
Hours to Work Per Day: 1

Number of Days Per Sprint: 11 days

Capacity: 66 hours per sprint

Point Capacity Per Sprint: 7

Below is a burndown chart showing the projected burndown versus the actual burndown. The velocity is also shown in the graph. In the beginning sprints, our user story point velocity was not as high as we anticipated. This is because we were mostly focusing on researching the necessary frameworks and utilities to complete the project. After these tools were established, our velocity began to increase in order to complete all of the scrum user story points.



## 5.4. GitHub Scrum Board Configuration

GitHub Project Link:

<https://github.com/orgs/cwru-courses/projects/6>

Brainstorming:

Using the in-class activity with post-it notes, we were able to separate tasks from user stories and begin to assign them to their respective sprints. We began by writing out all of our user stories and then sorting them into sprints to ensure that we had all user stories covered before figuring out how we wanted to divide up the work between sprints. We consistently revisit our tasks, user stories, and to-do lists on our Scrum board after each sprint cycle and during each cycle while completing tasks. During the sprint cycle, we move the user stories that are being worked on through the Scrum board. After each sprint, we reevaluate any remaining user stories and update their details accordingly, including story points, criteria, and sprint planning.

To-do:

The "To-do" section of our scrum board contains all our user stories that have yet to be started. These are organized in order of precedence. User stories that are required to be done earlier are placed higher in this section.

Blocked:

The "Blocked" section of our scrum board contains all user stories that we currently cannot complete for any reason. Any user story here signifies that we cannot work on this user story until we find an alternative way to get around the condition stopping us from doing so.

#### In Progress:

The "In Progress" section of our scrum board contains all the user stories that are currently being worked on. These are progressing mostly smoothly, with no blockers that render us incapable of continuing to make progress.

#### In Testing:

The "In Testing" section of our scrum board contains code that is no longer being developed and is being tested. Testing confirms that all tasks and acceptance criteria are functioning correctly as specified in the user story.

#### Done:

The "Done" section of our scrum board means that this user story is complete. It meets all acceptance criteria and functions properly. All tests have passed and this code no longer needs to be altered.

#### Done Done:

The "Done-Done" section of scrum board means that this user story is complete, meets all acceptance criteria, functions properly, passes all tests, and is approved by the team to deploy. At this point the user story is completely deployed into our source code and is not something we should work on again.

### **5.5 - Scrum Ceremonies**

These are the scrum ceremony progress reports from each sprint.

#### Sprint 1:

Our team held a sprint retrospective at the end of the sprint. In this sprint, we established the necessary tools and frameworks to build our project. On top of this, we have created a plan of action for the upcoming sprints. For future sprints, we will begin the actual development process and implementation of the application. From this sprint, we learned that in order for us to succeed, we need to be much more communicative than we already are. Communication within the group is paramount for progress. Additionally, we addressed the holes in some of our user stories, and the additional tasks that are necessary to ensure the quality and completion of each user story, especially the initial user stories. Although we have found that our initial project timeline was not a good fit in this review, we have made some adjustments to our approaches to database implementation, user stories, and group meetings which will have a positive impact on future progress.

#### Sprint 2:

As a team, we met in-person twice during the week and kept consistent communication via text throughout the entire sprint. Our first meeting was at the beginning of the sprint. In this ceremony, we confirmed the tools and frameworks necessary to complete the project and assigned roles for each member of the team with accompanying due dates. Assigned roles are mentioned in the following section. The second meeting took place near the end of the sprint. The purpose of this meeting was to ensure that the progress of the sprint was moving forward and clarify any obstacles we faced as a team. As a result of the meeting, we established that we were progressing at a reasonable pace and there were no major impediments in our way, though we were still becoming familiar with the MERN stack.

### Sprint 3:

As a team, conflicts pushed us to only be able to meet once this sprint, instead of twice. Regardless, we were still able to produce the functional features we intended to complete this sprint. The single meeting took place in the middle of the sprint to organize the team and ensure we were all clear on what we needed to get done this sprint, and who was responsible for which part. Along with this, we kept consistent communication via text to ensure each member completed their task.

### Sprint 4:

Due to conflicted time schedules and Thanksgiving break, we were only able to meet once as a team this sprint. In this ceremony, we were able to establish team roles and requirements for this sprint and tasks for the upcoming sprint. We also began discussing how we were going to present our project for the final presentation. On top of this, we kept consistent communication via a group chat to ensure everyone was on the same page and aware of the deadlines we had firmly established for this sprint.

### Sprint 5:

We were able to meet once this sprint combined with consistent communication via text. In this meeting, we assigned roles for establishing the final tests and implementation of design. We also created a plan of action for our executive presentation.

## **Section 6**

This section outlines the testing documentation/screenshots throughout the sprints.

### Sprint 2:

Home Login Register

ethan@example.com

.....

Register

Home Login Register

ethan@example.com

.....

Log in

Home Logout

What do you want to do?


☐ here's a task


☐ here's another task

☒ here's a third task


### Sprint 3:

What todo?


☐ here's a task 


☒ here's another task 

What todo?

☐ here's a task 

What todo?

☐ here's a task 

☐ here's another task 



## Sprint 4:

[Home](#) [Logout](#)

What todo?

Description

mm/dd/yyyy

--:-- --

General

Submit Task

Sort By↑ Time Created

General

12/16/2022 11:00

☐ load this

Assignment

12/14/2022 11:00

☐ load this after

[Home](#) [Logout](#)

What todo?

Description

mm/dd/yyyy

--:-- --

General

Submit Task

Sort By↑ Date

Assignment

12/14/2022 11:00

☐ load this after

General

12/16/2022 11:00

☐ load this

## Test Execution Details

- Informal Reviews

- Our team's informal reviews occurred whenever a team member was writing code and required assistance such as an idea or solution to a problem. If an issue came up while we were all working separately, this would happen over the phone. If an issue occurred while we were working altogether, one or more of us would look at the code and discuss possible solutions.
- The objective of this testing was to ensure that our code lacked any obvious errors before being committed into the main branch which improved the overall initial quality of our software and reduced the amount of work needed later to fix any bugs that may have occurred if we did not review our code before committing it. Nearly 80% of the issues found were related to a syntax error.
- Technical Reviews
  - Our technical reviews occurred near the end of each sprint, typically before class on the Thursday before the sprint was due. During these reviews, each individual who developed code during the sprint would present their code to the rest of the development team in order to determine if the code fits its intended purpose and fulfills the functional requirement.
  - The objective of this testing is to ensure that the code does not include any errors through examining the code line by line. Additionally, during these reviews we provided feedback on each other's code and helped each other better understand the entire code base. After each technical review we began noticing a significant improvement in our code as we implemented techniques suggested by other team members.
- Field Testing - Alpha Testing
  - Alpha testing began occurring once we deployed our application. This testing does not cover any code individually and instead ensures that the functional requirements and overall system are working as expected through the user interface. We conducted these tests by accessing the application through Google Chrome after each commit to test that the new changes work as expected, and that the new changes did not break the rest of the application.
  - On December 2nd, 2022 we detected an error in which the "you need to be logged in to see this page" text is being displayed as the same color as the background making the text impossible to read. We resolved this bug by pushing a new commit which fixed the text color.
- Field Testing - Beta Testing
  - Beta testing began December 3rd with a set of five students at Case Western Reserve University. This testing was used to seek feedback from a wider range of testers, specifically users who are not part of our development team in order to get a more diverse and comprehensive view on how our software application is functioning.

- On December 4th, one of the beta testers reported a bug where the application would crash when attempting to register an account using an email that has already been registered. Immediately afterwards we pushed a commit to fix this bug.

## **Section 7**

### **7.1 Sprint 1**

#### **7.1.1 User Story Progress Analysis**

Currently, our team is still in the planning and research stages of our project because in order to further implement a database along with code, we needed to establish the foundational aspects of database implementation and programming language. After examining various commonly used languages in reference to web application design, we have decided that Javascript would be the best fit for this project. Because Javascript is productive for both front and back-end development and we wanted to streamline our project design using one language, Javascript has the most benefits. Throughout this sprint, we began learning JavaScript syntax and doing research on how to reference objects(in this case, tasks) to a main database. We have also decided to use MERN stack which is an acronym for its components: MongoDB, ExpressJS, ReactJS, and NodeJS. With MERN, we have the ability to seamlessly combine the interaction between the front end, back end, and database of our application.

In the initial assignment of user stories to respective sprints, the user stories of adding tasks, removing tasks, and viewing tasks were supposed to be completed during the first sprint. However, our group had not recognized the necessary preparation that went into the tasks required to complete each of the user stories. For adding and removing assignments for the first two user stories, the creation, configuration, and implementation of the database is required. Because we were concurrently doing research on what database type we wanted to implement, it was not feasible to begin configuring the database amidst ongoing research. Because there is no data or a database to configure the user interface around, we were unable to jump ahead to this sprint and carry on with development. As it currently stands, we had only assigned user stories to 3 out of 5 of our sprints to create some leeway in our progression. Therefore, we have made the decision to push each user story back by 1 sprint to account for this sprint's research and education.

Regarding risks that have become apparent during this sprint, time management and lack of detailed tasks per user story have become a part of our most noticeable risks. We have initially accounted for the time management risk and any problems with tasks related to user stories by allotting all of our work between 3 sprints. This has given us space to change the type and amount of user stories per sprint as well as fix any issues with the user stories themselves. Because this project is being completed in 5 sprints, we have very little concern about the risk of

not meeting the final deadline. Another risk that was exposed by this sprint was lack of communication. Establishing a firm communication standard within the group is key for success in the development group. By having a physical meeting with team members instead of our past format of conference calls, this will solve most of our meeting issues including external influences from our environments, internet connection and multiple people speaking at the same time, and will help establish group deadlines within each sprint.

### **7.1.2 Overview of Database Schema**

Having decided to use MongoDB as our database, we have moved on to creating an overview of our database schema as our first step. Since we have users with multiple tasks, we have a one-to-many relationship. Our schema for users consists of their user email String field (that must be unique) and a password String field. Our schema for ToDo tasks will consist of a String text field for the user-inputted task, a Boolean describing whether the task is done or not, and an ObjectId referring to the user's email. Now that we have our schemas, once we are done implementing them and setting up our authentication, we have a way to find the ToDo tasks of each user through their references. As we move along in our sprints, we will modify these schemas to encompass other user stories, such as the ability to set a due date.

## **7.2 Sprint 2**

### **7.2.1 Progress Completion Report**

This sprint, we were able to accomplish creating the fundamental groundwork of the application, including primary components of the full stack. Our database, provided by MongoDB, has been implemented to account for users' email addresses and associated passwords with the help of the server-side component of the application. The client-side layer of the application has been implemented with the help of React and Bootstrap. As a team, we were able to complete first-step user stories, such as creating tasks with accompanying names, viewing created tasks, and checking off completed tasks.

#### **7.2.2. Team Roles**

Oct. 20-23 - Yimin Huang

Import necessary MERN-related files and dependencies.

Oct. 23-26 - Kamsiyochukwu Eneh Yimin Huang

Set up the database and server-side layer of the application.

Oct. 26-29 - Ethan Ho Addy Krom Adriana Kamor Leonardo Rodriguez

Set up the necessary JavaScript and Bootstrap files to create the client-side component of the application.

### 7.3 Sprint 3

#### 7.3.1 Progress Completion Report

This sprint, we were able to accomplish creating a functional remove feature to the application. As intended, this feature effectively removes any desired task from the list of Todos. This change is shown visually (on the front-end) and is reflected in the deletion of the task in the database. The remove feature's GUI is enhanced with Bootstrap to improve the appearance of the button.

#### 7.3.2 Team Roles

Adriana Kamor

Implement client side functionality of the remove feature

Leonardo Rodriguez

Decorate the remove feature with Bootstrap and CSS. Implement server file functionality for deleting from the database.

Ethan Ho

Diagram the design of the feature

### 7.4 Sprint 4

#### 7.4.1 Progress Completion Report

This sprint, our team was able to fully structure feature 2, which allows users to associate a time and date with a given created task. Using the Tempus Dominus date-time-picker Bootstrap dependency, users can easily assign a deadline for their task. Along with this, users now have the ability to sort tasks based on their due dates, with the closest due date being at the top of the list. This functionality is enabled through a button click event. In our GitHub files, the code structure for the "sort button" and the "sort function" can be found in the Home page file, but have not been completely implemented. Because the date and time attribute for tasks has not been implemented successfully, the implementation of the sorting algorithm and button are blocked but exist within the code. Lastly, we now have our project hosted online for users to more easily access the application, compared to the previous need to set up the application locally.

To deploy our project, we used Heroku to host our front/back end and MongoDB Atlas to host our database. Since to host on Heroku our code had to be refactored, we decided to have a

branch dedicated only to deployment. Here is where code will be pushed after being tested on our main branch. Our site lacks sort functionality because not all functionality has been implemented since the last deployment. On the home page JS file, the sort functionality and button have been structurally set up, but are not approved for implementation and usage by the user due to an issue in the creation of a datetimepicker. Our site is hosted at <https://whispering-gorge-40839.herokuapp.com/>. This is the default URL when using the free service on Heroku.

One obstacle we are facing with our deployment is the fact that Heroku's free services are now being made part of a premium plan. We are monitoring our website to upgrade our plan when necessary. We would also like to shorten the URL to improve the user experience.

#### **7.4.2 Team Roles**

Leonardo Rodriguez - Host project online

Kamsiyochukwu Eneh - Structure datetimepicker

Adriana Kamor - Structure sort by date/time functionality

Ethan Ho - Draw diagrams and write report

Addy Krom - Explain diagrams and write report

### **7.5 Sprint 5**

#### **7.5.1 Progress Completion Report**

This sprint, we completed the implementation and testing of the final user stories we needed to complete. These user stories included a sort by group/event feature, labeling tasks with certain categories, and validation for tasks input. On top of this, we were able to make the overall user interface more user-friendly and better suited for the student environment.

#### **7.5.2 Team Roles**

Leonardo Rodriguez - Final feature and sort

Adriana Kamor - User interface updates

Ethan Ho - Scrum metric calculations and diagrams

Addy Krom - Scrum metric calculations and testing