D424 – SOFTWARE ENGINEERING CAPSTONE TASK 3

Project Name: DC Management Dashboard

Student Name: Stephen Johnson

Student ID:010503054

GITLAB REPOSITORY

TASK 3 – PART A

•The GitLab Repository can be located by doing the following:

◦ Navigate to the GitLab account "WGU-GitLab-Environment"

◦ Click on the "Groups" tab

◦ Click on the "Student Repos" group

◦ Click on the "SJO3235" subgroup

◦ Click on the "D424-Software-Engineering-Capstone" project

• Commit messages can be found for each task within the repository's commit history.

• Direct repository URL:

• ◦ https://gitlab.com/wgu-gitlab-environment/student-repos/sjo3235/d424-software-engineering-capstone

FULL STACK SOFTWARE PRODUCT

TASK 3 – PART B

1. Object-Oriented Design:

• Inheritance, Polymorphism, and Encapsulation:

• Inheritance: The project exhibits inheritance via the RMATask and StandardTask classes, extending the Task class. These classes override specific methods to deliver task-specific functionalities.

• Polymorphism: The getDescription method in the Task class is overridden in its subclasses (RMATask and StandardTask) to present various descriptions based on the task type.

• Encapsulation: The TaskManager class encapsulates the task management logic, maintaining task-related data private and exposing vital methods to interact with tasks.

2. Search Functionality:

- The application contains a robust search functionality that permits users to filter tasks based on multiple criteria, such as task status, assigned technician, and date. This search returns various rows of results depicted in a user-friendly layout.

3. |Database Component:

- Firebase Real-time Database: The application employs Firebase Real-time Database to store and handle technician and manager data. It includes secure methods to add, adjust, and delete data, ensuring that all operations are conducted securely and efficiently.

- Local CSV Data Parsing: Task data is uploaded via CSV files, parsed utilizing PapaParse.js, and then processed within the application to classify tasks by data center and other attributes.

4. Report Generation:

- The application generates detailed reports that include multiple columns and rows, date-time stamps, and titles. These reports are generated based on the tasks managed within the application and can be exported for further analysis.

5. Validation Functionality:

- Data validation is implemented to ensure that all inputs meet specific criteria before processing. This prevents invalid or corrupted data from being stored or displayed, maintaining the integrity of the application.

6. Security Features:

- Authentication: Firebase Authentication is used to manage user access to the application. This ensures that only permitted users can interact with the application.

- Data Security: The application follows industry-standard security practices, including encryption and secure data handling through Firebase's built-in security rules.

7. Scalable Design Elements:

- The application follows a modular architecture, with components like task management, data processing, and reporting handled by separate modules. This design supports easy scalability, integrating new features or enhancements without affecting the existing functionality.

- TaskManager and TaskFactory classes are designed to handle various tasks, extending the application to support new task types or additional data centers effortlessly.

8. User-Friendly, Functional GUI:

- The GUI is crafted for intuitive, responsive use, ensuring seamless navigation and efficient task completion. By integrating Bootstrap and ApexCharts, the application offers a visually striking and interactive experience, complete with drag-and-drop functionality and real-time data updates.

DOCUMENTATION

TASK 3 - PART C

APPLICATION DESIGN

Class Diagram

The application is designed with a modular architecture to ensure scalability and maintainability, observing tenets like separation of concerns and single responsibility. Below is an overview of the key classes and their roles:

Authentication: Drives user authentication via Firebase, including methods for signing in and signing out and monitoring authentication state changes.

Parser: Handles the retrieval and loading of data from Firebase, particularly technician and manager data.

Task: Represents a generic task, encapsulating task data with methods to get and set properties, check if a task is resolved, and provide task descriptions.

RMATask and StandardTask: Inherit from Task and override the getDescription method to provide specific descriptions based on the task type.

TaskFactory: Creates instances of RMATask or StandardTask based on the task type.

TaskManager: Manages tasks across various data centers, providing methods to add tasks, retrieve resolved and unresolved tasks, and categorize functions by the data center.

TaskProcessor: Provides utility functions for processing tasks, including counting tasks by manager, retrieving resolved tasks, and generating task statistics.

Script: Central hub for data processing and visualization, including file uploads, report generation, and task analysis.

ChartGenerator: Generates visual charts using ApexCharts.

UIManager: Manages the user interface, including modal displays and employee details updates.

ShiftCalculator: Handles shift-related calculations and overtime categorization.

People: Manages the addition, update, and removal of managers and technicians, updating the UI as necessary.

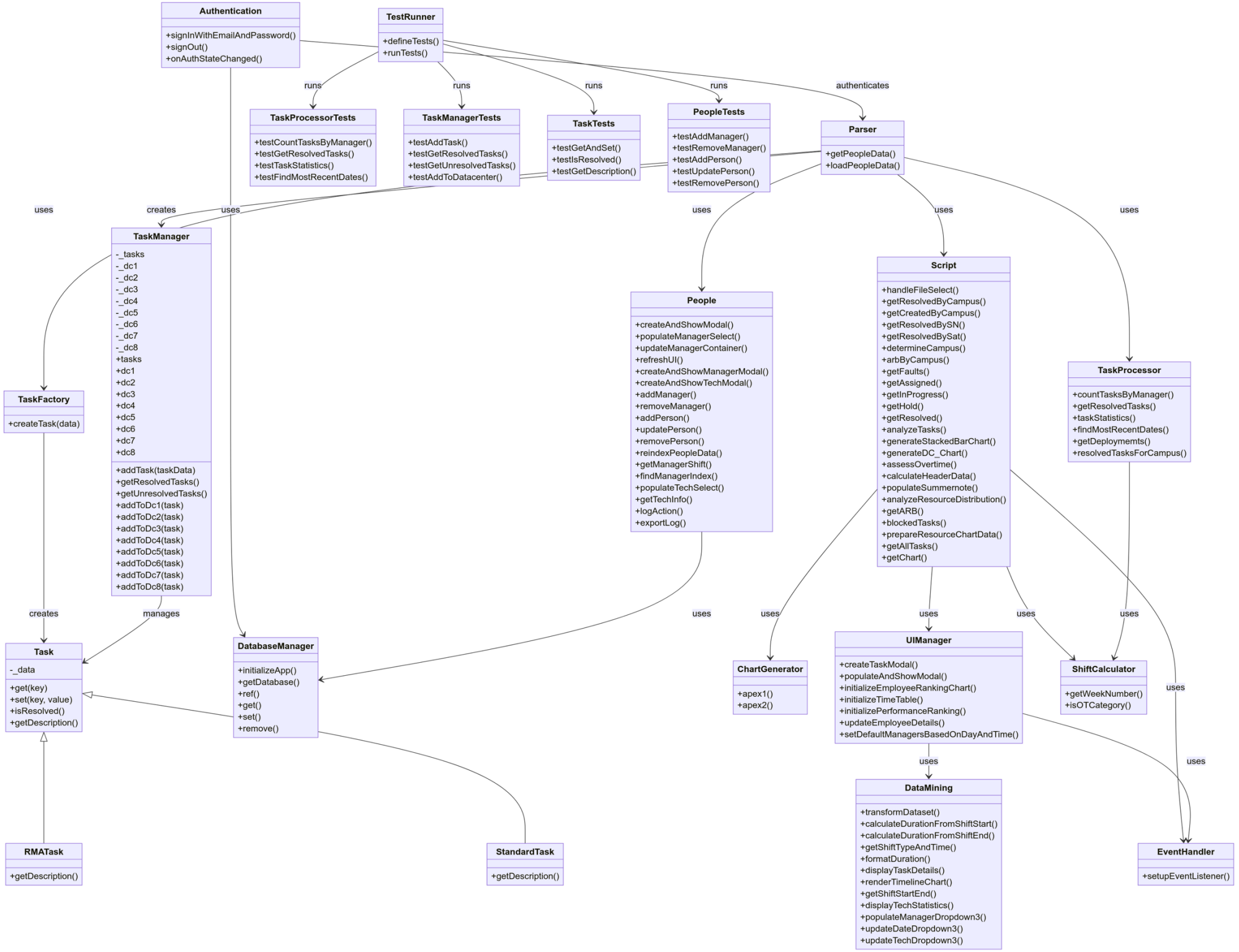
DatabaseManager: Interacts with Firebase for all database operations, including data retrieval, updates, and removals.

DataMining: Transforms and analyzes task data, rendering charts and updating UI elements like dropdowns.

EventHandler: Sets up event listeners for user interactions to trigger appropriate actions within the application.

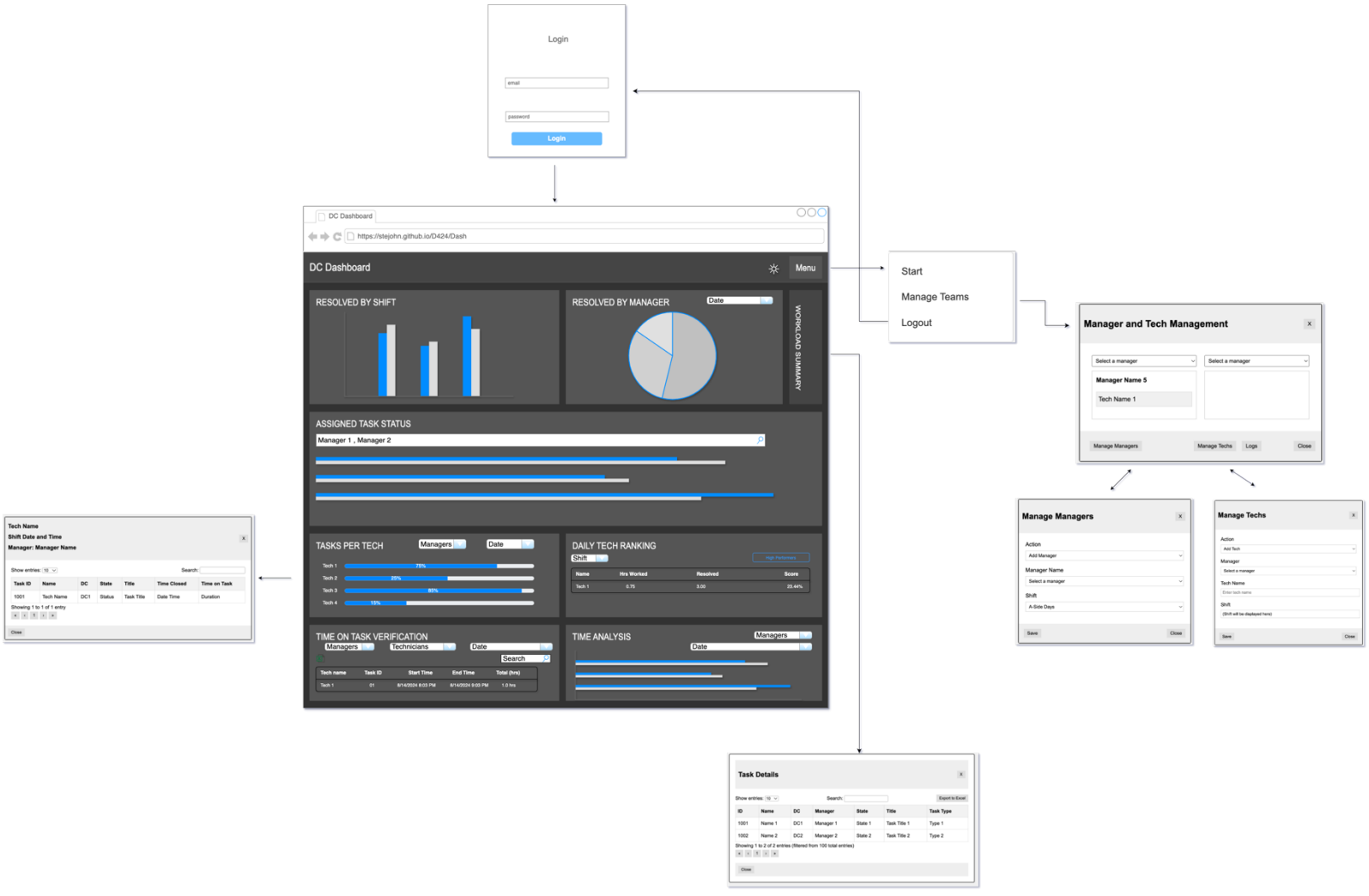
TestRunner: Executes unit tests for various components to ensure application reliability.

TaskProcessorTests, TaskManagerTests, TaskTests, PeopleTests: Contain unit tests for their respective components, verifying the functionality and correctness of critical methods.


Authentication
+signInWithEmailAndPassword()
+signOut()
+onAuthStateChanged()
TestRunner
+defineTests()
+runTests()
runs
runs
runs
runs
authenticates
TaskProcessorTests
+testCountTasksByManager()
+testGetResolvedTasks()
+testTaskStatistics()
+testFindMostRecentDates()
TaskManagerTests
+testAddTask()
+testGetResolvedTasks()
+testGetUnresolvedTasks()
+testAddToDatacenter()
TaskTests
+testGetAndSet()
+testIsResolved()
+testGetDescription()
PeopleTests
+testAddManager()
+testRemoveManager()
+testAddPerson()
+testUpdatePerson()
+testRemovePerson()
Parser
+getPeopleData()
+loadPeopleData()
uses
creates
uses
uses
uses
uses
TaskManager
-_tasks
-_dc1
-_dc2
-_dc3
-_dc4
-_dc5
-_dc6
-_dc7
-_dc8
+tasks
+dc1
+dc2
+dc3
+dc4
+dc5
+dc6
+dc7
+dc8
+addTask(taskData)
+getResolvedTasks()
+getUnresolvedTasks()
+addToDc1(task)
+addToDc2(task)
+addToDc3(task)
+addToDc4(task)
+addToDc5(task)
+addToDc6(task)
+addToDc7(task)
+addToDc8(task)
People
+createAndShowModal()
+populateManagerSelect()
+updateManagerContainer()
+refreshUI()
+createAndShowManagerModal()
+createAndShowTechModal()
+addManager()
+removeManager()
+addPerson()
+updatePerson()
+removePerson()
+reindexPeopleData()
+getManagerShift()
+findManagerIndex()
+populateTechSelect()
+getTechInfo()
+logAction()
+exportLog()
Script
+handleFileSelect()
+getResolvedByCampus()
+getCreatedByCampus()
+getResolvedBySN()
+getResolvedBySat()
+determineCampus()
+arbByCampus()
+getFaults()
+getAssigned()
+getInProgress()
+getHold()
+getResolved()
+analyzeTasks()
+generateStackedBarChart()
+generateDC_Chart()
+assessOvertime()
+calculateHeaderData()
+populateSummernote()
+analyzeResourceDistribution()
+getARB()
+blockedTasks()
+prepareResourceChartData()
+getAllTasks()
+getChart()
TaskProcessor
+countTasksByManager()
+getResolvedTasks()
+taskStatistics()
+findMostRecentDates()
+getDeploymemts()
+resolvedTasksForCampus()
TaskFactory
+createTask(data)
creates
manages
uses
uses
uses
uses
uses
uses
Task
-_data
+get(key)
+set(key, value)
+isResolved()
+getDescription()
DatabaseManager
+initializeApp()
+getDatabase()
+ref()
+get()
+set()
+remove()
ChartGenerator
+apex1()
+apex2()
UIManager
+createTaskModal()
+populateAndShowModal()
+initializeEmployeeRankingChart()
+initializeTimeTable()
+initializePerformanceRanking()
+updateEmployeeDetails()
+setDefaultManagersBasedOnDayAndTime()
ShiftCalculator
+getWeekNumber()
+isOTCategory()
uses
uses
uses
DataMining
+transformDataset()
+calculateDurationFromShiftStart()
+calculateDurationFromShiftEnd()
+getShiftTypeAndTime()
+formatDuration()
+displayTaskDetails()
+renderTimelineChart()
+getShiftStartEnd()
+displayTechStatistics()
+populateManagerDropdown3()
+updateDateDropdown3()
+updateTechDropdown3()
RMATask
+getDescription()
StandardTask
+getDescription()
EventHandler
+setupEventListener()

User Interface Design Diagram

The UI design centers around a DC Dashboard that provides visual overviews and clear insights into data center tasks and performance metrics. Users can interact with the dashboard to drill into task details and manage teams through the Manager and Tech Management interface. The design prioritizes ease of use, allowing for efficient navigation and task management.



GitLab Repository Information

- URL:
  - https://gitlab.com/wgu-gitlab-environment/student-repos/sjo3235/d424-software-engineering-capstone

- Version Used for documentation:
  - 1.0

Maintenance User Guide

Introduction

This guide is designed to assist developers in setting up, running, and maintaining the Data Center Management Dashboard application. It provides detailed steps for configuring the development environment, running the application locally, and deploying it to Firebase. Additionally, it covers how to update dependencies, manage Firebase integration, and modify the application's core functionalities. For instructions on using the application from a user perspective, please refer to the User Guide.

Prerequisites

Before proceeding, ensure you have the following installed on your development machine:

- **Node.js**: Version 18.17.1 or later
- **npm**: Version 9.6.7 or later
- **Firebase CLI**: Latest version installed
- **Web browser**: Chrome, Firefox, or any modern browser
- **Text editor or IDE**: Visual Studio Code, Sublime Text, or any code editor of your choice
- **Git**: For version control and managing the source code repository

Steps to Set Up and Run the Application

1. **Clone the Repository:**
   - Clone the GitLab repository to your local machine using the following command:
     ```
     git clone https://gitlab.com/wgu-gitlab-environment/student-repos/sjo3235/d424-software-engineering-capstone.git
     ```
   - Navigate into the project directory:
     ```
     cd d424-software-engineering-capstone
     ```

2. **Install Dependencies:**

   - Install all necessary npm packages by running:
     ```
     npm install
     ```
   - This command will read the package.json file and install all dependencies required for the project, including papaparse.js, ApexCharts, and firebase.

3. **Firebase Setup (Optional but Recommended):**

   - If the application requires interaction with Firebase, ensure the Firebase CLI is installed and configured:
     ```
     npm install -g firebase-tools
     firebase login
     ```
   - Initialize Firebase within the project if not already done:
     ```
     firebase init
     ```
   - During initialization, select the appropriate Firebase services (e.g., Firestore, Authentication, Hosting).

4. **Running the Application:**

   - Start the development server:
     ```
     npm start
     ```
   - The application will compile and start a local server. By default, it should be accessible at http://localhost:3000 . The port number can differ depending on operating system and system configuration. Open this URL in your web browser to view and interact with the application.

5. **Modifying the Application:**

   - The main application logic are housed in scripts located in /Dash/assets/js/, which handles data parsing, task management, and chart generation.
   - To make changes, open the desired script in your text editor or IDE. For example:
     - Modify data parsing functionality in the Parser class.
     - Adjust task management logic in the TaskManager and TaskProcessor classes.
     - Customize chart rendering in the ChartGenerator class.
   - Any changes made to the code will automatically reload the application in your browser.

6. **Testing the Application:**

   - Unit tests are defined for critical functions within the TestRunner class and its associated test files (TaskProcessorTests, TaskManagerTests, etc.).
   - To run tests and verify the functionality:
     ```
     npm test
     ```

- Review the test results to ensure that all components are functioning correctly.

7. **Deploying the Application to Firebase:**

   - If you need to deploy the application, ensure your Firebase project is properly set up. Then, deploy using:
     ```
     firebase deploy
     ```
   - This command will deploy your application to Firebase Hosting and update any associated Firebase services.

Notes on Maintaining the Application

- **Updating Dependencies:**
  - Keep the project dependencies up to date by running:
    ```
    npm update
    ```
  - Regularly check for updates to critical packages such as firebase, ApexCharts, and papaparse.js.

- **Modifying Firebase Configurations:**
  - Firebase configurations (like API keys and project IDs) are stored in the firebaseConfig object within firebase-config.js. If you need to change these settings, update the corresponding values.

- **Adding New Features:**
  - When adding new features, adhere to the existing modular structure:
    - Use the TaskFactory class to create new task types.
    - Extend the ChartGenerator class to include additional charts or data visualizations.
    - Ensure that new features are tested by adding corresponding tests in the appropriate test class.

- **Version Control:**
  - Commit changes regularly and push them to the repository:
    ```
    git add .
    git commit -m "Description of changes"
    git push origin main
    ```
  - Use branches to manage new features or bug fixes, merging them into the main branch after testing.

User Guide

Introduction

Welcome to the Data Center Management Dashboard application. This guide supplies a clear outline of how to use the application, including logging in, navigating the dashboard, managing teams, and generating reports. The application is designed to help you efficiently manage and track data center operations, providing insightful analytics through various charts and tables.

Getting Started

Accessing the Application

- **URL:** https://stejohn.github.io/D424/Dash
- When you first access the application, you will be greeted with the Login page. The application automatically directs you to this page if it detects that you have not yet logged in. The login and authentication are managed via Firebase.

Logging In

1. **Sign In:**
    - Enter your email and password in the supplied fields.
    - Click the "Login" button to authenticate.
    - Upon successful login, you will be pivoted to the principal interface, the **DC Dashboard**.

Theme Toggle

- By default, the dashboard starts in dark mode. You can toggle between dark and light themes by clicking the theme icon in the navbar at the top right corner.

Using the DC Dashboard

Navigating the Menu

- **Menu Options:** Located in the top right corner of the navbar, the "Menu" button provides three options:
    1. **Start:** Initiates the data upload process and launches the DC Dashboard.
    2. **Manage Teams:** Opens the Manager and Tech Management interface.
    3. **Logout:** Logs you out of the application.

Starting the Application

1. **Click "Start":**
    - A modal window will appear, welcoming you to the DC Dashboard.

2. **Step 1 - Download Data File:**
    - Click the link labeled "**here**" under the "Download data file here" section to download the necessary data file.

3. **Step 2 - Upload Data File:**
    - Use the file upload element to locate and select the downloaded data file.
    - Once selected, the file will automatically load into the application, and the dashboard will update with the data.

Managing Teams

Manager and Tech Management

- **Accessing the Interface:**
    - Click "Manage Teams" from the menu to open the Manager and Tech Management interface.
- **Moving Technicians Between Teams:**

1. Use the left dropdown to select the manager from which you want to move a technician.

2. Use the right dropdown to select the manager you wish to transfer the technician to.

3. Drag the technician's name from the left side to the right side to move them. This action will update the Firebase Realtime Database.

4. **Note:** To see these changes reflected in the dashboard, either logout and log back in or make these changes before clicking "Start" in the menu.

- **Managing Managers:**
    - Click the “Manage Managers” button to add, remove, or edit managers.
    - Select the desired action (add, remove, edit), choose the manager’s name from the dropdown, and make any necessary changes, including altering their shift.

- **Managing Technicians:**
    - Click the “Manage Techs” button to add or remove technicians.
    - Select the manager under whom the technician works, then make the necessary changes, such as updating their assigned shift.

- **Viewing Logs:**
    - If any changes are made to the managers or technicians, click the “Logs” button to export a log of all actions taken. This log includes timestamps (in UTC format), the actions performed, and the user who performed them.

Dashboard Features

Workload Summary

- **Options Available:**
    - **Assigned, In Progress, Hold, Resolved:** Each option displays the current tasks under each state.
    - Click the info circle next to each option to open the “Task Details” modal, showing properties such as ID, Name, DC, Manager, State, Title, and Task Type.
- **Task Details:**
    - You can sort tasks within the modal by clicking on the property names. Use the search bar for multi-parameter filtering (e.g., filtering by title, name, task type, DC).
    - Export the task data to a CSV file by clicking the “Export to Excel” button. The exported file will include the date and time in its title.

Resolved by Manager

- **Overview:**
    - A pie chart displays the tasks resolved by each manager.
    - Use the dropdown to filter the data by day or shift.

Assigned Task Status

- **Overview:**
    - A horizontal pie chart displays the status of tasks (e.g., resolved, in progress, held) for each technician.
    - The chart can be filtered by clicking on the legend items to show only specific task states.
- **Manager Team Stats:**
    - Use the dropdown search bar to select or type in the manager's name to view their team's statistics.

Tasks per Tech

- **Overview:**
    - This section features a dropdown to select a manager and a shift date. It generates a list of resolved tasks for each technician during that shift.
    - Clicking on a technician's name opens a modal with detailed task information, including Task ID, Name, DC, State, Title, Time Closed, and Time on Task.
- **Task Sorting and Filtering:**
    - Sort properties by clicking on the property names. Use the search bar for filtering.
    - Export the task data to a CSV file with the date and time embedded in the title.

Daily Tech Ranking

- **Overview:**
    - Select a shift from the dropdown to render a table of technicians ranked by performance during that shift.
    - The table displays Name, Hours Worked, Resolved Tasks, and Score.
- **Ranking Algorithm:**
    - Technicians are ranked based on a minimum requirement of 2 tasks and 2 hours of actual work. Failure to meet these requirements negatively impacts their score, while exceeding them can result in a score above 100%.

Time on Task Verification

- **Overview:**
    - This view provides detailed insights into the actual time technicians spend on tasks.
    - The table displays properties like Name, Task ID, Start Time, End Time, and Total Time (hrs.).

- **Task Filtering and Exporting:**
    - Use the search bar to filter the table by any property.
    - Export the table data to a CSV file with the export date and time in the title.

Time Analysis

- **Overview:**
    - This horizontal bar graph shows the performance metrics of technicians under their managers and shifts.
    - Use the dropdown menus to select a manager and a change to render the data.

- **Metrics Displayed:**
    - The graph shows two categories, Total Tasks and Time on Tasks, represented by different colors as indicated in the legend.
    - Total Tasks represents the number of tasks resolved during the selected shift, and Time on Task represents the total hours spent on these tasks.

Testing

Introduction
URL: `https://stejohn.github.io/D424/test.html`

Purpose

- These unit tests ensure that the underlying functions within the TaskManager, TaskProcessor, and People classes produce the expected results. These classes were selected for unit testing because they are critical to the application's core functionality, including task management, data processing, and team management. Any unexpected results will be documented, and necessary changes will be implemented to resolve any issues.

Overview

- The unit test scripts focus on verifying the background logic of the application, ensuring that the various operations such as task creation, processing, and team management work as expected. Similar testing methods are applied across all core classes to thoroughly test the application's functionality.

Test Plan (Unit Test)

Items

- **JavaScript Environment** (Node.js)
- **Mocha** for running tests
- **Chai** for assertions
- **Sinon** for mocking functions and objects

Features

- Testing CRUD operations for the TaskManager and People classes.
- Verifying that the TaskProcessor correctly processes tasks and generates statistics.
- Ensuring that methods for task creation, data retrieval, and team management behave as expected.

Deliverables

- Unit test scripts.
- Documentation of the test results.
- Updated code that resolves any encountered issues.
- Summaries of the changes made due to completed tests.

Tasks

1. Write unit test scripts for the TaskManager, TaskProcessor, and People classes.
2. Run the test scripts.
3. Document the results of the test scripts.
4. Review the results and modify the code as necessary.

Needs

- **Mocha** (9.0.0)
- **Chai** (4.3.4)
- **Sinon** (12.0.1)
- **Node.js** (version 18.17.1 or later)

Pass/Fail Criteria

- **PASS:**
    - Unit test scripts run without any failures.
    - All verifications or assertions specified within the individual tests are met.
    - The TaskManager, TaskProcessor, and People classes behave as expected.
- **FAIL:**
    - Failures are encountered when running the unit test script.
    - Any verification or assertion within the test is not met.
    - The classes interact in an unexpected way or produce incorrect results.

Procedures

1. **Initialize the Testing Environment:**
    - Set up any necessary mocks and objects using Sinon.
    - Ensure that the testing environment closely mirrors the production environment.
2. **Run Initial Tests:**
    - Execute all unit tests using Mocha.
    - Note the result of the initial test (PASS or FAIL) in the test documentation.
3. **Document Failed Tests:**
    - For each failed test, document the expected result versus the actual result in the test documentation.
4. **Resolve Issues:**
    - Analyze the code corresponding to any failed tests.

    - Modify the code as necessary to address the issues.
    - Rerun all tests after making changes.
    - If the test passes and no other tests fail, proceed to the next step. If any tests fail, repeat this process until all tests pass.
5. **Document Iterations:**
    - Document the number of iterations required to resolve any errors.
    - Summarize the changes made to the code because of the tests.

Unit Test Results

TaskManagerTests.js

- **testAddTask**
    - Initial Test Result: PASS
- **testGetResolvedTasks**
    - Initial Test Result: PASS
- **testGetUnresolvedTasks**
    - Initial Test Result: PASS
- **testAddToDatacenter**
    - Initial Test Result: PASS

TaskProcessorTests.js

- **testCountTasksByManager**
    - Initial Test Result: PASS
- **testGetResolvedTasks**
    - Initial Test Result: PASS

- **testTaskStatistics**

    - Initial Test Result: PASS
- **testFindMostRecentDates**
    - Initial Test Result: PASS

PeopleTests.js

- **testAddManager**
    - Initial Test Result: PASS
- **testRemoveManager**
    - Initial Test Result: PASS
- **testAddPerson**
    - Initial Test Result: PASS
- **testUpdatePerson**
    - Initial Test Result: PASS
- **testRemovePerson**
    - Initial Test Result: PASS

Unit Test Code

Please see screenshots of the test classes below inside defineTests:

```
import {
  countTasksByManager,
  getResolvedTasks,
  findMostRecentDates,
} from "../assets/js/task_processor.js";
import {
  sanitizeName,
  Task,
  RMATask,
  StandardTask,
  TaskFactory,
  TaskManager,
} from "../assets/js/parser.js";

const { expect } = chai;
mocha.setup({
  ui: "bdd",
  cleanReferencesAfterRun: false,
});

function defineTests() {
  describe("Task Processor Functions with Mock People Data", () => {
    let taskManager;
    let originalPeople;

    const mockPeople = [
      { name: "Jane Doe", manager: "John Manager", shift: "A-Side Days" },
      {
        name: "Chris Evans",
        manager: "John Manager",
        shift: "A-Side Nights",
      },
    ];

    beforeEach(() => {
      originalPeople = window.people;

      window.people = [
        {
          name: "Jane Doe",
          manager: "John Manager",
          shift: "A-Side Days",
        },
        {
          name: "Chris Evans",
          manager: "John Manager",
          shift: "A-Side Nights",
        },
      ];

      taskManager = new TaskManager();
```

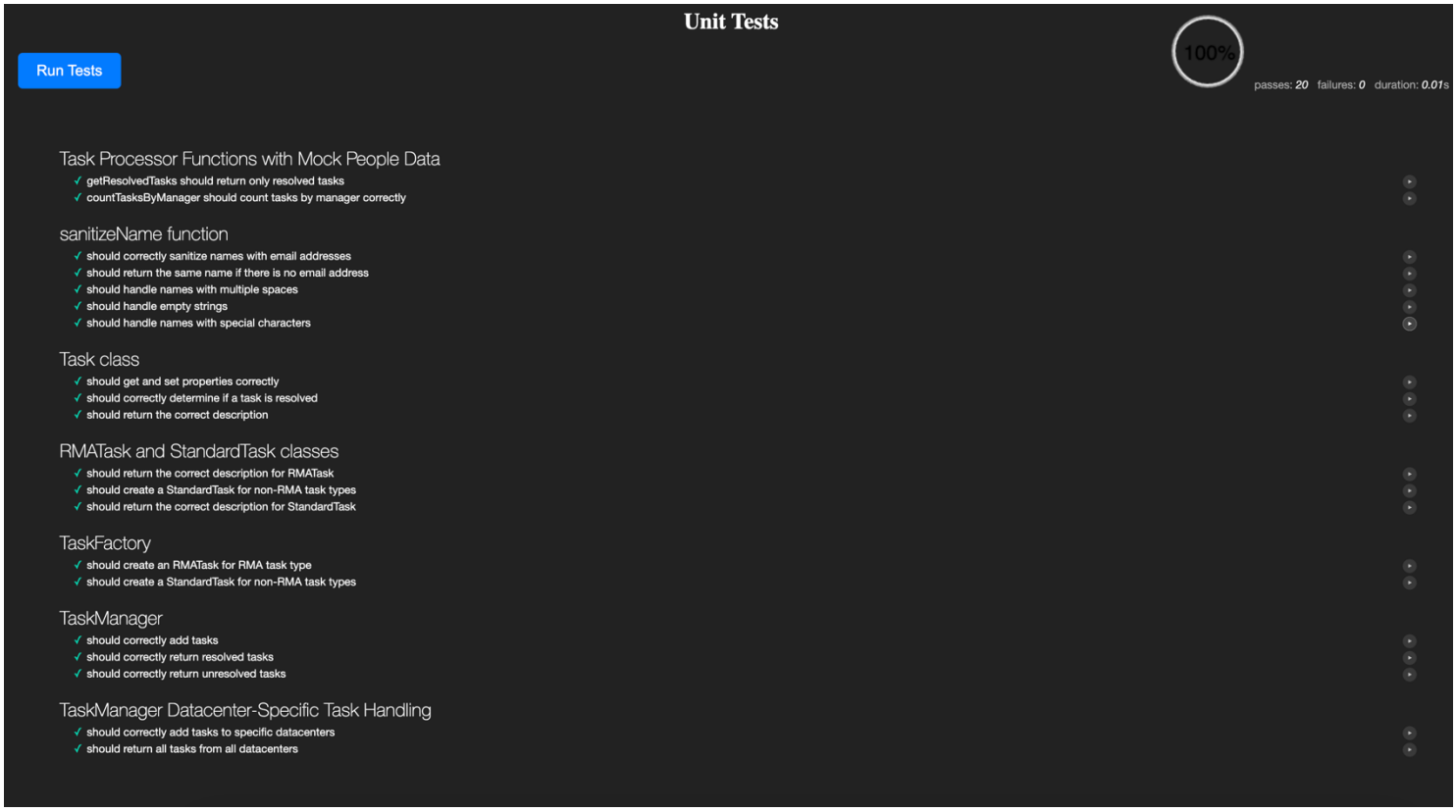
```
    const sampleTasks = [
      {
        "Datacenter Code": "DC1",
        Id: "1000",
        Title: "Disk Replacement: Faulty Drive",
        "Task Type": "BreakFix",
        "Fault Description": "Storage: Disk Drive Failure",
        "Created Date": "7/29/24 10:00",
        State: "Resolved",
        "Assigned To": "Jane Doe <jdoe@abc-dc.com>",
        "Work End Date": "7/29/24 17:00",
        "Work Start Date": "7/29/24 10:30",
        Expedite: "y",
      },
      {
        "Datacenter Code": "DC1",
        Id: "1004",
        Title: "Rack Installation: New Equipment",
        "Task Type": "Deployment",
        "Fault Description": "Hardware: Rack Installation",
        "Created Date": "7/29/24 14:00",
        State: "Resolved",
        "Assigned To": "Chris Evans <cevans@abc-dc.com>",
        "Work End Date": "7/29/24 16:00",
        "Work Start Date": "7/29/24 14:00",
        Expedite: "y",
      },
    ];

    sampleTasks.forEach((task) => taskManager.addTask(task));
  });

  afterEach(() => {
    window.people = originalPeople;
  });

  it("getResolvedTasks should return only resolved tasks", () => {
    const resolvedTasks = taskManager.getResolvedTasks();
    expect(resolvedTasks).to.have.lengthOf(2);
    expect(resolvedTasks[0]._data).to.have.property("State", "Resolved");
    expect(resolvedTasks[1]._data).to.have.property("State", "Resolved");
  });

  it("countTasksByManager should count tasks by manager correctly", () => {
    const resolvedTasks = taskManager.getResolvedTasks();

    const input = {
      "Shift 1": { dates: { "7/29/24": { tasks: resolvedTasks } } },
    };

    const debugCountTasksByManager = (testArr) => {
      let result = {};
```

```
  for (const shift in testArr) {
    for (const date in testArr[shift].dates) {
      for (const task of testArr[shift].dates[date].tasks) {
        const assignedToName = task._data["Assigned To"]
          .split("<")[0]
          .trim();

        const employee = window.people.find(
          (person) => person.name === assignedToName
        );

        if (employee) {
          const managerName = employee.manager;

          if (!result[managerName]) {
            result[managerName] = { dates: {} };
          }
          if (!result[managerName].dates[date]) {
            result[managerName].dates[date] = {
              total: 0,
              employees: {},
            };
          }
          if (
            !result[managerName].dates[date].employees[assignedToName]
          ) {
            result[managerName].dates[date].employees[assignedToName] = {
              total: 0,
              tasks: [],
            };
          }
          result[managerName].dates[date].total++;
          result[managerName].dates[date].employees[assignedToName]
            .total++;
          result[managerName].dates[date].employees[
            assignedToName
          ].tasks.push(task);
        }
      }
    }
  }

  return result;
};

const tasksByManager = debugCountTasksByManager(input);

const expectedTasksByManager = {
  "John Manager": {
    dates: {
      "7/29/24": {
```

```
            total: 2,
            employees: {
              "Jane Doe": {
                total: 1,
                tasks: [
                  {
                    _data: {
                      "Datacenter Code": "DC1",
                      Id: "1000",
                      Title: "Disk Replacement: Faulty Drive",
                      "Task Type": "BreakFix",
                      "Fault Description": "Storage: Disk Drive Failure",
                      "Created Date": "7/29/24 10:00",
                      State: "Resolved",
                      "Assigned To": "Jane Doe <jdoe@abc-dc.com>",
                      "Work End Date": "7/29/24 17:00",
                      "Work Start Date": "7/29/24 10:30",
                      Expedite: "y",
                    },
                  },
                ],
              },
              "Chris Evans": {
                total: 1,
                tasks: [
                  {
                    _data: {
                      "Datacenter Code": "DC1",
                      Id: "1004",
                      Title: "Rack Installation: New Equipment",
                      "Task Type": "Deployment",
                      "Fault Description": "Hardware: Rack Installation",
                      "Created Date": "7/29/24 14:00",
                      State: "Resolved",
                      "Assigned To": "Chris Evans <cevans@abc-dc.com>",
                      "Work End Date": "7/29/24 16:00",
                      "Work Start Date": "7/29/24 14:00",
                      Expedite: "y",
                    },
                  },
                ],
              },
            },
          },
        },
      },
    };

    expect(tasksByManager).to.deep.equal(expectedTasksByManager);
  });
});
```

```
describe("sanitizeName function", () => {
  it("should correctly sanitize names with email addresses", () => {
    expect(sanitizeName("John Doe <john@example.com>")).to.equal("John Doe");
  });

  it("should return the same name if there is no email address", () => {
    expect(sanitizeName("Jane Smith")).to.equal("Jane Smith");
  });

  it("should handle names with multiple spaces", () => {
    expect(sanitizeName("Bob   Johnson <bob@example.com>")).to.equal(
      "Bob   Johnson"
    );
  });

  it("should handle empty strings", () => {
    expect(sanitizeName("")).to.equal("");
  });

  it("should handle names with special characters", () => {
    expect(sanitizeName("Jöhn Döe <john.doe@example.com>")).to.equal(
      "Jöhn Döe"
    );
  });
});

describe("Task class", () => {
  let task;

  beforeEach(() => {
    task = new Task({
      Id: "1",
      Title: "Test Task",
      State: "In Progress",
    });
  });

  it("should get and set properties correctly", () => {
    expect(task.get("Id")).to.equal("1");
    expect(task.get("Title")).to.equal("Test Task");

    task.set("State", "Resolved");
    expect(task.get("State")).to.equal("Resolved");
  });

  it("should correctly determine if a task is resolved", () => {
    expect(task.isResolved()).to.be.false;
    task.set("State", "Resolved");
    expect(task.isResolved()).to.be.true;
  });

  it("should return the correct description", () => {
```

```
    expect(task.getDescription()).to.equal("Task 1: Test Task");
  });
});

describe("RMATask and StandardTask classes", () => {
  it("should return the correct description for RMATask", () => {
    const rmaTask = new RMATask({
      Id: "2",
      Title: "RMA Task",
      "Fault Code": "FC001",
    });
    expect(rmaTask.getDescription()).to.equal("RMA Task 2: RMA Task - FC001");
  });

  it("should create a StandardTask for non-RMA task types", () => {
    const taskData = {
      "Task Type": "BreakFix",
      Id: "5",
      Title: "BreakFix Task",
    };
    const task = TaskFactory.createTask(taskData);
    expect(task).to.be.an.instanceOf(StandardTask);
    expect(task.getDescription()).to.include(
      "Standard Task 5: BreakFix Task"
    );
  });

  it("should return the correct description for StandardTask", () => {
    const standardTask = new StandardTask({
      Id: "3",
      Title: "Standard Task",
      "Priority Bucket": "High",
    });
    expect(standardTask.getDescription()).to.equal(
      "Standard Task 3: Standard Task - High"
```



Testing Summary

- All unit tests passed successfully, indicating that the core functionalities of the TaskManager, TaskProcessor, and People classes are working as intended. The thorough manual testing conducted during development contributed to this outcome. These unit tests will be invaluable for future maintenance and development, ensuring that new features or changes do not introduce regressions.