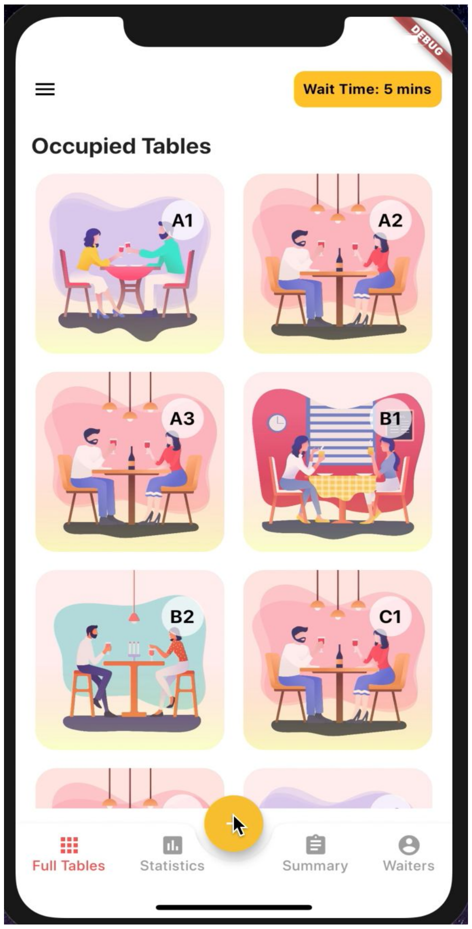
Testing and Inspection Report



A Report on Unit Testing and Code Inspections

Prepared by
Brian
Omar
Harsh
Emmanuel
***for use in* CS 440**
at the
University of Illinois Chicago

Spring 2020

Table of Contents

List of Figures 3

List of Tables 4

I Project Description 5

1 Project Overview 5

2 Project Domain 5

3 Relationship to Other Documents 5

4 Naming Conventions and Definitions 6

4a Definitions of Key Terms 6

4b UML and Other Notations Used in This Document 7

4c Data Dictionary for Any Included Models 8

II Testing 9

5 Items to be Tested 9

6 Test Specifications 11

7 Test Results 21

8 Regression Testing 25

III Inspection 25

9 Items to be Inspected 25

10 Inspection Procedures 28

11 Inspection Results 28

IV Recommendations and Conclusions 30

V Project Issues 31

12 Open Issues 31

13 Waiting Room 31

14 Ideas for Solutions 31

15 Project Retrospective 32

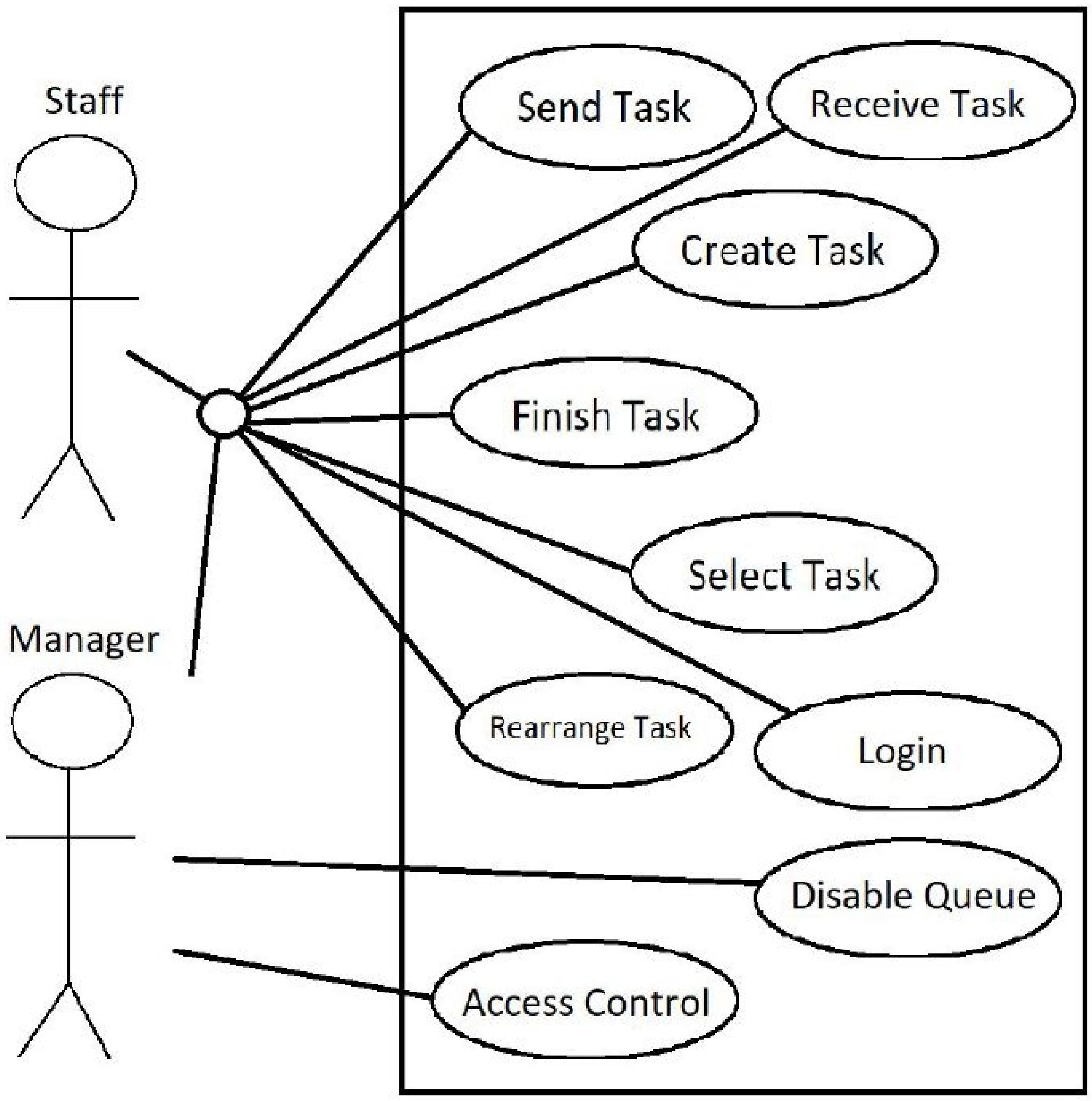
VI Glossary 32

VII References / Bibliography 33

VIII Index 33

List of Figures

Figure 1: Task Action Use Case Diagram



List of Tables

Table 1: Employee	Table 2: Dining Table	Table 3: Tasks
Employee_ID	Dining_Table_ID	Task_ID
FName	*Employee_ID*	*Dining_Table_ID*
LName	Seats	*Employee_ID*
Is_Manager	Is_Active	Status
Is_Logged_In	Is_Occupied	Title
Hire_Date	Seating_Time	Description
Birth_Date	Reservation_Name	Start_Time
Address	Has_Birthday	Finish_Time
Phone	Special_Request	Total_Time
Token		Task_Date
Salary		
Title		

I Project Description

1 Project Overview

Wait-Less is a restaurant management application that increases efficiency in restaurants by allowing managers and employees to send/receive tasks as a request or reminder. Employees are always stacked with different tables, guests, and general restaurant needs that having a system in which they are reminded of actions they should perform helps them throughout the night of service.

The backend system and central database of the application will keep track of the number of tasks completed by the employees, as well as the time it takes to complete these tasks. With the stored information, the application will display this given information to the managers on a graph so they can analyze the consistency of efficiency throughout a given time period. Knowing these statistics, managers can look at which days have had a slower average task completion time over how many tasks were completed during that day.

2 Project Domain

There are different testing environments being used for both frontend and backend. The frontend tests include widget testing in Flutter as well as manual testing. In manual testing the developer tries to run the UI in different environments, platforms and devices. Some of these devices include different dimensions and provide a better clarity on the application's usability by users across these platforms.

3 Relationship to Other Documents

Throughout the document we used the Wait-Less project description document[1] to better understand the project. We referenced the Wait-Less Project Design document [2]when creating the models for our project. Finally, when writing the testing portion of the document we referenced the Wait-Less Project requirement document[3].

[1] University of Illinois At Chicago. "Wait-Less Project Description", Fall 2019

[2] University of Illinois At Chicago. "Wait-Less Project Design", Fall 2019

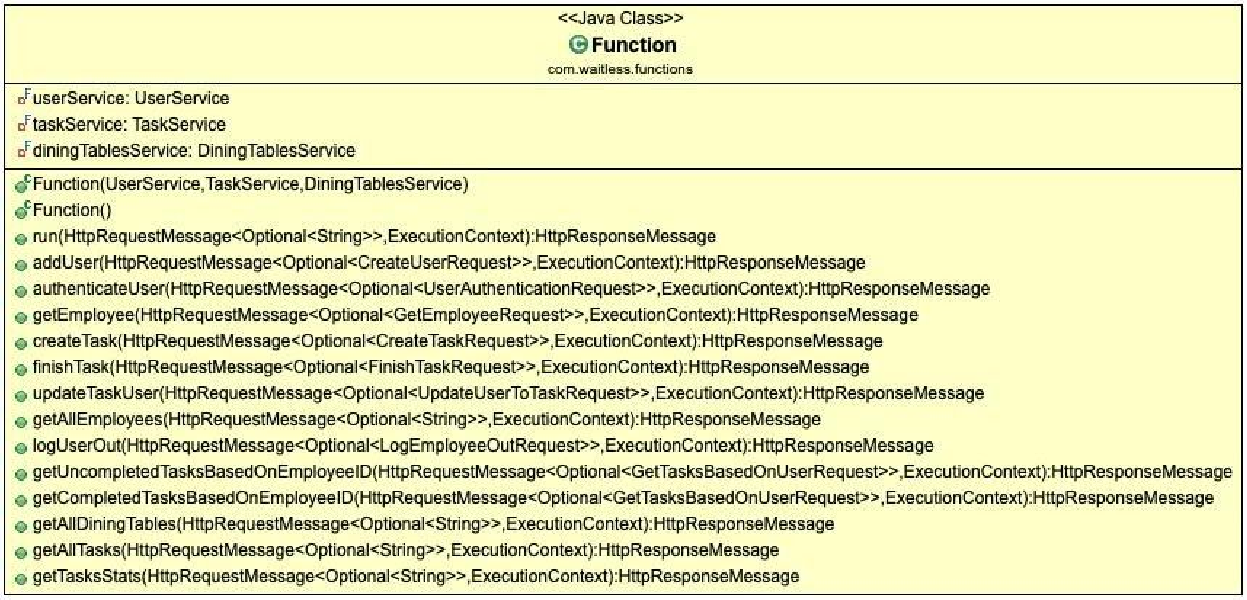
[3] University of Illinois At Chicago. "Wait-Less Project Requirements", Fall 2019

4 Naming Conventions and Definitions

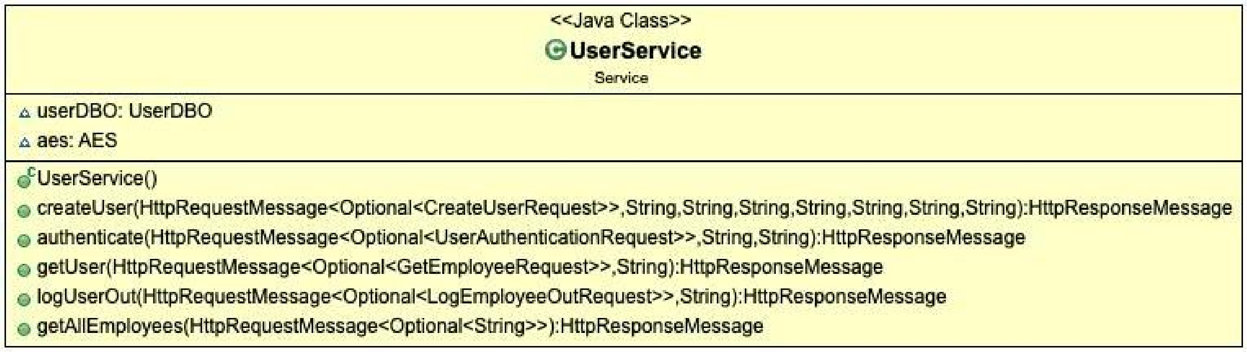
4a Definitions of Key Terms

Term	**Definition**
Wait-less	Restaurant workflow and productivity application
Flutter	A versatile software development kit created by Google. Utilizes Dart to produce multi platform applications.
Dart	An object oriented language that is class-based and C styled.
Java	Object oriented programming language that runs on top of a virtual machine. Objects are garbage collected and versatile.
Azure	Cloud computing and hosting services provided by Microsoft. Popular industry solution, with very low downtime.

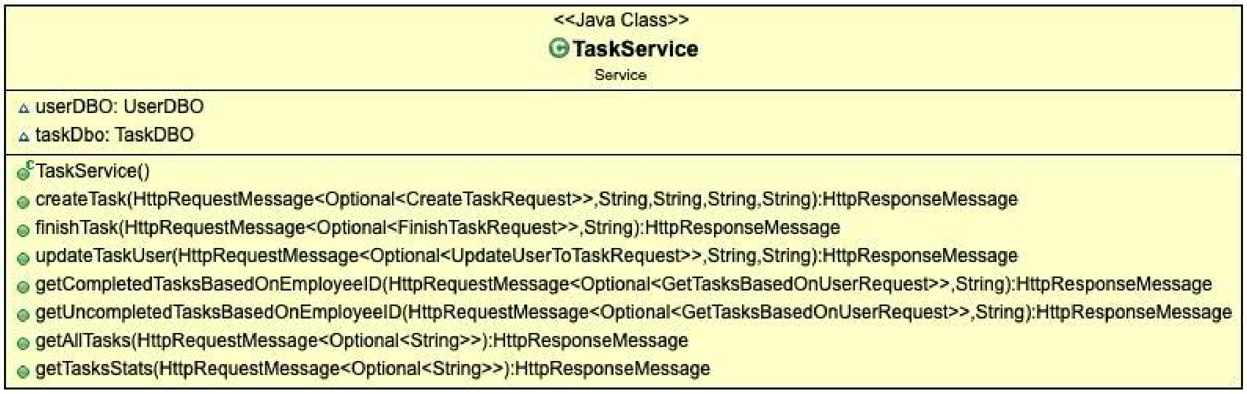
4b UML and Other Notations Used in This Document



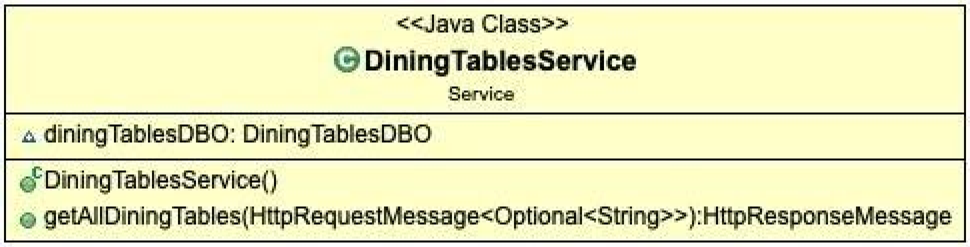
```
<<Java Class>>
Function
com.waitless.functions

userService: UserService
taskService: TaskService
diningTablesService: DiningTablesService

Function(UserService,TaskService,DiningTablesService)
Function()
run(HttpRequestMessage<Optional<String>>,ExecutionContext):HttpResponseMessage
addUser(HttpRequestMessage<Optional<CreateUserRequest>>,ExecutionContext):HttpResponseMessage
authenticateUser(HttpRequestMessage<Optional<UserAuthenticationRequest>>,ExecutionContext):HttpResponseMessage
getEmployee(HttpRequestMessage<Optional<GetEmployeeRequest>>,ExecutionContext):HttpResponseMessage
createTask(HttpRequestMessage<Optional<CreateTaskRequest>>,ExecutionContext):HttpResponseMessage
finishTask(HttpRequestMessage<Optional<FinishTaskRequest>>,ExecutionContext):HttpResponseMessage
updateTaskUser(HttpRequestMessage<Optional<UpdateUserToTaskRequest>>,ExecutionContext):HttpResponseMessage
getAllEmployees(HttpRequestMessage<Optional<String>>,ExecutionContext):HttpResponseMessage
logUserOut(HttpRequestMessage<Optional<LogEmployeeOutRequest>>,ExecutionContext):HttpResponseMessage
getUncompletedTasksBasedOnEmployeeID(HttpRequestMessage<Optional<GetTasksBasedOnUserRequest>>,ExecutionContext):HttpResponseMessage
getCompletedTasksBasedOnEmployeeID(HttpRequestMessage<Optional<GetTasksBasedOnUserRequest>>,ExecutionContext):HttpResponseMessage
getAllDiningTables(HttpRequestMessage<Optional<String>>,ExecutionContext):HttpResponseMessage
getAllTasks(HttpRequestMessage<Optional<String>>,ExecutionContext):HttpResponseMessage
getTasksStats(HttpRequestMessage<Optional<String>>,ExecutionContext):HttpResponseMessage
```



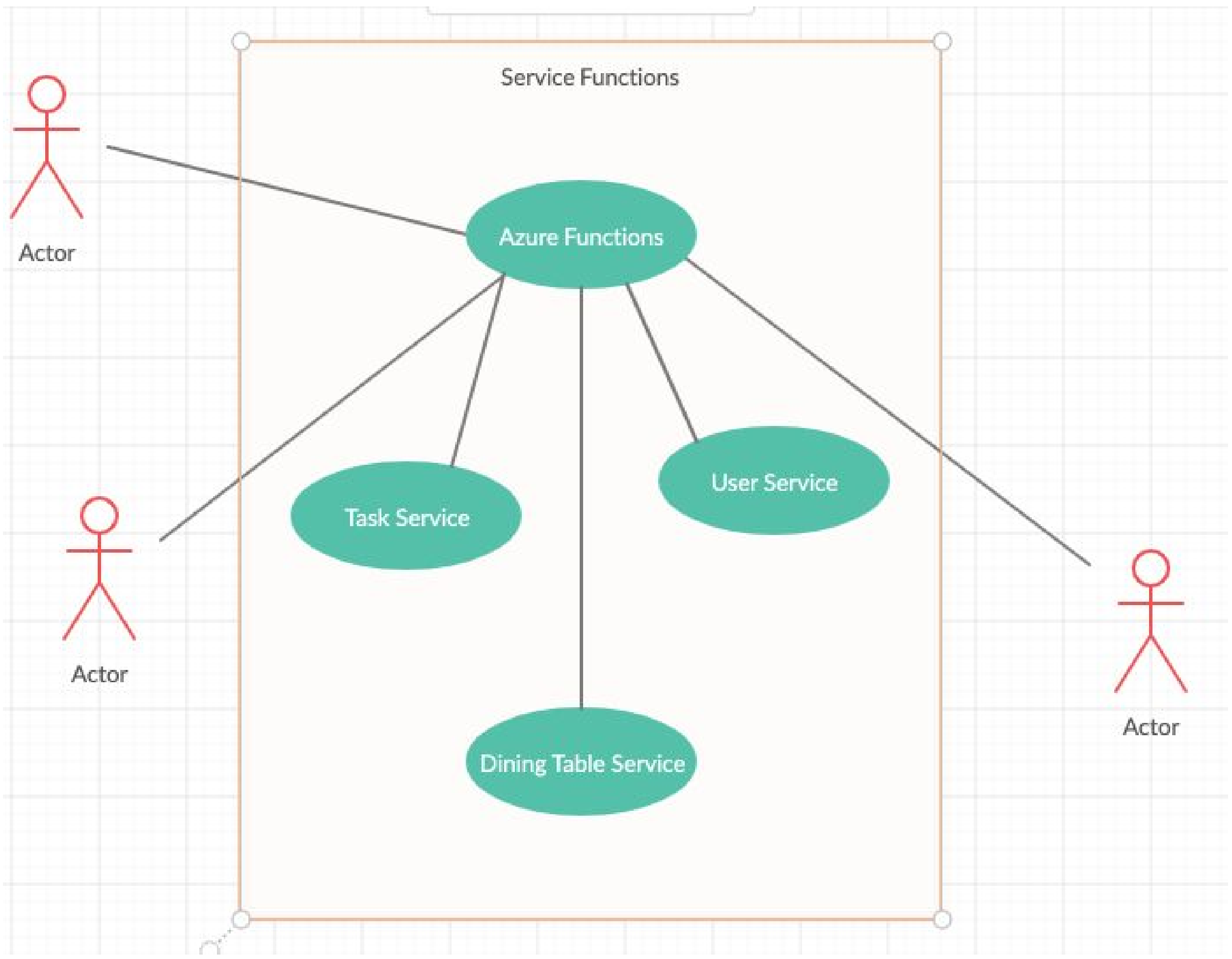
```
<<Java Class>>
UserService
Service

userDBO: UserDBO
aes: AES

UserService()
createUser(HttpRequestMessage<Optional<CreateUserRequest>>,String,String,String,String,String,String,String):HttpResponseMessage
authenticate(HttpRequestMessage<Optional<UserAuthenticationRequest>>,String,String):HttpResponseMessage
getUser(HttpRequestMessage<Optional<GetEmployeeRequest>>,String):HttpResponseMessage
logUserOut(HttpRequestMessage<Optional<LogEmployeeOutRequest>>,String):HttpResponseMessage
getAllEmployees(HttpRequestMessage<Optional<String>>):HttpResponseMessage
```



```
<<Java Class>>
TaskService
Service

userDBO: UserDBO
taskDbo: TaskDBO

TaskService()
createTask(HttpRequestMessage<Optional<CreateTaskRequest>>,String,String,String,String):HttpResponseMessage
finishTask(HttpRequestMessage<Optional<FinishTaskRequest>>,String):HttpResponseMessage
updateTaskUser(HttpRequestMessage<Optional<UpdateUserToTaskRequest>>,String,String):HttpResponseMessage
getCompletedTasksBasedOnEmployeeID(HttpRequestMessage<Optional<GetTasksBasedOnUserRequest>>,String):HttpResponseMessage
getUncompletedTasksBasedOnEmployeeID(HttpRequestMessage<Optional<GetTasksBasedOnUserRequest>>,String):HttpResponseMessage
getAllTasks(HttpRequestMessage<Optional<String>>):HttpResponseMessage
getTasksStats(HttpRequestMessage<Optional<String>>):HttpResponseMessage
```



```
<<Java Class>>
DiningTablesService
Service

diningTablesDBO: DiningTablesDBO

DiningTablesService()
getAllDiningTables(HttpRequestMessage<Optional<String>>):HttpResponseMessage
```



4c Data Dictionary for Any Included Models

Entity Name	**Entity Description**
Azure Functions	Holds all endpoints for the various application API functions.
User Service	Handles service calls that manage user data.
Task Service	Handles service calls that manage task data.
Dining Table Service	Handles service calls that manage dining table data.

II Testing

5 Items to be Tested

ID	Author	Code to be Tested

O	Omar	*// Pre-Condition: getTasks() function should set the variable listComTasks to the retrieved list*
		Widget _buildCompletedList() {
		  return FutureBuilder(
		    future: getTasks(), *// Future which returns a list of completed tasks*
		    builder: (context, snapshot) { *// Build widget based off the getTask info*
		      return snapshot.hasData ?
		                listComTasks.length == 0 ?
		                new Container(child: Text('No Completed Tasks!')) *// No tasks*
		              : new ListView.builder(*// Has Tasks*
		                  itemCount: listComTasks.length,
		                  itemBuilder: (BuildContext content, int index){ *// Build individual task items*
		                      Task task = listComTasks[index];
		                      return CompletedList(task, content);
		                  },
		             )
		            : new Center(child: SpinKitWave(color: Colors.*cyan*, size: 50)); *// Loading icon*
		      },
		    );
		}

H	Harsh	dialogContent(BuildContext context){ *// Dialog Content for the popup*
		 return Stack(
		   children: <Widget>[
		     Container(
		       padding: EdgeInsets.only(*// specify the dimensions and position*
		           top: 200,
		           bottom: 16,
		           left: 16,
		           right: 16
		       ),
		       margin: EdgeInsets.only(top: 16),
		       decoration: BoxDecoration(
		         color: Colors.white,
		         shape: BoxShape.rectangle,
		         borderRadius: BorderRadius.circular(17),
		         boxShadow: [
		           BoxShadow(
		             color: Colors.black87,
		             blurRadius: 10.0,
		             offset: Offset(0.0, 10.0),
		
		               )
		         ]

		),

B	Brian	*//Precondition createUser() return the result of the create user operation*
		// Function should output the correct httpResponse
		// 200 - User Created
		// 400 - Bad Request
		// 500 - Internal Server Error
		@FunctionName("Add-User")
		    public HttpResponseMessage addUser(@HttpTrigger(name = "req", methods = {HttpMethod.POST}, authLevel = AuthorizationLevel.ANONYMOUS) HttpRequestMessage<Optional<CreateUserRequest>> request,
		                                final ExecutionContext context) {
		        CreateUserRequest createUserRequest = request.getBody().orElse(null);
		        if (createUserRequest != null)
		            return userService.createUser(request, createUserRequest.firstName, createUserRequest.lastName,
		                    createUserRequest.birthday, createUserRequest.address,
		                    createUserRequest.phone, createUserRequest.title, createUserRequest.encryptedPassword);
		        else {
		            return request.createResponseBuilder(HttpStatus.BAD_REQUEST).body("Please input a valid username and password").build();
		        }
		    }

E	Emmanuel	//Precondition createUser() return the result of the create user operation
		// Function should output the correct httpResponse
		// 200 - Task marked as done
		// 400 - Bad Request
		// 404 - Task not found
		// 500 - Internal Server Error
		
		@FunctionName("Finish-Task")
		    public HttpResponseMessage finishTask(@HttpTrigger(name = "req", methods = {HttpMethod.POST}, authLevel = AuthorizationLevel.ANONYMOUS) HttpRequestMessage<Optional<FinishTaskRequest>> request,
		                                final ExecutionContext context) {
		        FinishTaskRequest finishTaskRequest = request.getBody().orElse(null);
		        if (finishTaskRequest != null)
		            return taskService.finishTask(request, finishTaskRequest.taskID);
		        else {
		            return request.createResponseBuilder(HttpStatus.BAD_REQUEST).body("Please input a valid username and password").build();
		        }
		    }

6 Test Specifications

ID	Specifications	
H1	**Description**	Test that the created Stack widget has a container with the correct padding and margin
	Items covered	● Container ● Padding ● Margin
	Requirements Addressed	N/A
	Environmental Needs	● The latest version of the Flutter SDK must be installed so the correct widget libraries are called.
	Intercase Dependencies	N/A
	Test Procedures	● The main build function of a Flutter application must be executed ● The proper BuildContext object is passed into the test function ● Utilize object fields to determine the value of the Container, padding, and margin
	Input Specification	● Valid working BuildContext object for the widget building
	Output Specifications	● Container is non-null ● Padding is non-zero ● Margin is non-zero
	Pass/Fail Criteria	● Container Padding must match the specified values ● Container Margin must match the specified values

H2	**Description**	Test that the created Container widget contains a BoxDecoration object with a rounded corner white rectangle
	Items covered	● Container ● Box Decoration ● Shape ● Color

		● Border Radius
	Requirements Addressed	N/A
	Environmental Needs	● The latest version of the Flutter SDK must be installed so the correct widget libraries are called.
	Intercase Dependencies	● H1
	Test Procedures	● The main build function of a Flutter application must be executed ● The proper BuildContext object is passed into the test function ● The BuildContext successfully builds the Container widget (H1) ● Utilize object fields to determine the shape, color, and border radius values of the Container's BoxDecoration object
	Input Specification	● Valid working BuildContext object for the widget building ● Non-Null Container widget (H1)
	Output Specifications	● BoxDecoration is non-null ● Color is non-null ● Shape is non-null ● Border Radius is non-zero
	Pass/Fail Criteria	● Color must be white ● Shape is a Rectangle ● Border Radius has circular corners

H3	**Description**	Test that the created BoxDecoration widget contains a BoxShadow object of color black, blurRadius, and offset
	Items covered	● Box Decoration ● Box Shadow ● Color ● Blu Radius ● Offset
	Requirements Addressed	N/A
	Environmental	● The latest version of the Flutter SDK must be installed so the

	Needs	correct widget libraries are called.
	Intercase Dependencies	● H1 ● H2
	Test Procedures	● The main build function of a Flutter application must be executed ● The proper BuildContext object is passed into the test function ● The BuildContext successfully builds the Container widget (H1) ● The BuildContext successfully builds the BoxDecoration widget (H2) ● Utilize object fields to determine the color, blu radius, and offset values of the BoxDecoration's BoxShadow object
	Input Specification	● Valid working BuildContext object for the widget building ● Non-Null Container widget (H1) ● Non-Null BoxDecoration widget (H2)
	Output Specifications	● BoxShadow is non-null ● Color is non-null ● Blur Radius is non-zero ● Offset is non-zero
	Pass/Fail Criteria	● Color must be black ($\pm$ opacity) ● Blur Radius must match the specified value ● Offset must match the specified value

O1	**Description**	Test that the list of tasks is populated with the recent data
	Items covered	● Task List ● Task Name ● Task Description
	Requirements Addressed	It takes into account the heavy dependence of the application on backend and data provided
	Environmental Needs	Most recent Flutter SDK and required libraries should be imported.
	Intercase Dependencies	There are no dependencies related to testcase but there can be a dependency on the validation of the data fetched.

	Test Procedures	● First the data is fetched from the online database ● Next the data is stored into the list ● A manual check as well as an automated test can be done to see if the current data displays the most recent data in the database
	Input Specification	● Valid data provided
	Output Specifications	● Data is populated into the list
	Pass/Fail Criteria	The list should include all the data which should be in the task list according to the database

O2	**Description**	Test that the list of tasks stores data with the correct data type
	Items covered	● Task list has valid data objects and type
	Requirements Addressed	It addresses the requirement of having the correct data in order for the application to function smoothly
	Environmental Needs	Most Recent Flutter SDK
	Intercase Dependencies	O1
	Test Procedures	● Once the data is fetched ● A simple test to find the data type of the objects in the list and if each has a table, name and description
	Input Specification	Task list with task stored
	Output Specifications	If the tasks match with the task in the task list, a value of true or false
	Pass/Fail Criteria	Each task will have three objects that are table, name and description

O3	**Description**	Test that future builder builds the same amount of widgets as the length of the task list

	Items covered	● Future Builder ● Task List ● UI
	Requirements Addressed	The UI follows the same pattern, and displaying the correct number of tasks from the backend
	Environmental Needs	The latest Flutter UI should be running and all the import statements for the libraries used should be written.
	Intercase Dependencies	O1
	Test Procedures	● Once the recent data is fetched ● Test compares the length of the List with the number of widgets displayed ● Future Builder is invoked with respect to the items in the task list
	Input Specification	● Task List is non-null
	Output Specifications	● UI displayed on screen ● Widgets follow correct dimension (Can be dependent on other portion of the code not covered here)
	Pass/Fail Criteria	Future Builder is successfully displaying widgets with respect to the task list items

E1	**Description**	Tests that finish task function will output a 200 if taskService.finishTask() works correctly
	Items covered	● taskService ● 200 Response Message ● Finish Task Request
	Requirements Addressed	Address the requirement of users being able to finish a requested task
	Environmental Needs	Java 1.8, and most recent maven

	Intercase Dependencies	Taskservice.finishTask(), DBO.finishTask() both need to be functional to work properly.
	Test Procedures	● Retrieve API url from Azure Function portal ● Create a curl request to the api with an input input of a valid taskId to be finished ● Ensure that taskService.finishTask() is invoked
	Input Specification	● taskId is a valid task
	Output Specifications	● A 200 status code
	Pass/Fail Criteria	Pass: A 200 status code is returned, and the database should have the corresponding taskId marked as finished with the time of completion corresponding to the time the api was called Fail: Any other status code is returned or the database is unchanged or database is not exact format as pass case

E2	**Description**	Test that finish task will output a 404 if taskServie.finishTask() cannot find the task
	Items covered	● 404 HttpResponse ● Finish Task Request ● taskService
	Requirements Addressed	Address the requirement of users being able to finish a requested task, ensuring the finish task will not work when not used properly
	Environmental Needs	Java 1.8, and most recent maven
	Intercase Dependencies	Taskservice.finishTask(), DBO.finishTask() both need to be functional to work properly.
	Test Procedures	● Retrieve API url from Azure Function portal ● Create a curl request, using this url, to the api with an input parameter of an invalid taskId to be finished ● Ensure that taskService.finishTask() is invoked
	Input Specification	● taskId is an invalid task

	Output Specifications	● A 404 status code
	Pass/Fail Criteria	Pass: A 404 status code is returned, database is unchanged Fail: Any other status code is returned or database is changed

E3	**Description**	Test that finish task will output a 500 if taskServie.finishTask() cannot connect to the SQL database
	Items covered	● 500 HttpResponse ● Finish Task Request ● TaskService
	Requirements Addressed	Address the requirement of users being able to finish a requested task, ensuring the finish task will not work if the backend services are not working
	Environmental Needs	Java 1.8, and most recent maven
	Intercase Dependencies	Taskservice.finishTask(), DBO.finishTask() both need to be functional to work properly.
	Test Procedures	● Retrieve API url from Azure Function portal ● Create a curl request to the api with a parameter input of a taskId into the function ● Turn SQL database off or change SQL url to simulate an down SQL server ● Ensure that taskService.finishTask() is invoked
	Input Specification	● A task Id must be provided
	Output Specifications	● A 500 status code
	Pass/Fail Criteria	Pass: A 500 status code is returned database is unchanged Fail: Any other status code is returned or database is changed

E4	**Description**	Test that finish task will output a 400 no parameters a specified

	Items covered	● FinishTaskRequest ● 400 Response Message
	Requirements Addressed	Address the requirement of users being able to finish a requested task, ensuring the finish task will not work when not used properly
	Environmental Needs	Java 1.8, and most recent maven
	Intercase Dependencies	N/A
	Test Procedures	● Retrieve API url from Azure Function portal ● Create a curl request to the api with an input of blank parameter for the taskId
	Input Specification	● An empty parameter
	Output Specifications	● A 400 status code
	Pass/Fail Criteria	Pass: A 400 status code is returned and database is unchanged Fail: Any other status code is returned or database is changed

B1	**Description**	Tests that add user function will output a 200 if userService.createUser() works correctly
	Items covered	● userService ● 200 Response Message ● Create User Request
	Requirements Addressed	Address the requirement of users being able to register an account within the application
	Environmental Needs	● Java 1.8, and most recent maven
	Intercase Dependencies	● userService.createUser(), UserDBO.createUser() both need to be functional to work properly. Or the use of a mock library to simulate these functions is necessary.

	Test Procedures	● Input a valid request to be handled ● Ensure that userService.createUser() returns a 200 ● Ensure that the Create User Request Api properly returns a 200
	Input Specification	● request is an valid instance of CreateUserRequest
	Output Specifications	● A 200 status code
	Pass/Fail Criteria	Pass: A 200 status code is returned Fail: Any other status code is returned

B2	**Description**	Tests that add user function will generate the corresponding user profile with proper records. userService.createUser() must work correctly
	Items covered	● userService ● Create User Request ● UserNotFoundException
	Requirements Addressed	Address the requirement of users being able to have their data stored securely and with high integrity within the application
	Environmental Needs	● Java 1.8, and most recent maven
	Intercase Dependencies	● userService.createUser(), UserDBO.createUser() both need to be functional to work properly. Or the use of a mock library to simulate these functions is necessary. ● Access to a local Azure test environment
	Test Procedures	● Launch a local sandbox environment that can host the SQL DB (live data) ● Input a valid request to be handled ● Ensure that userService.createUser() returns a 200 ● Ensure that the Create User Request Api properly returns a 200
	Input Specification	● request is a valid and specified instance of CreateUserRequest
	Output	● Encrypted password is stored on remote server

	Specifications	● Employee with respective Name exists
	Pass/Fail Criteria	Pass: A 200 status code is returned Fail: Any other status code is returned, which would be generated by UserNotFoundException

B3	**Description**	Tests that add user function will output a 400 if userService.createUser() works incorrectly
	Items covered	● userService ● 400 Response Message ● Create User Request
	Requirements Addressed	Address the requirement of users not being able to create an invalid account
	Environmental Needs	● Java 1.8, and most recent maven
	Intercase Dependencies	● userService.createUser(), UserDBO.createUser() both need to be functional to work properly. Or the use of a mock library to simulate these functions is necessary.
	Test Procedures	● Input an invalid request to be handled ● Ensure that userService.createUser() returns a 400 ● Ensure that the Create User Request Api properly returns a 400
	Input Specification	● request is an invalid instance of CreateUserRequest
	Output Specifications	● A 400 status code
	Pass/Fail Criteria	Pass: A 400 status code is returned Fail: Any other status code is returned

7 Test Results

ID	**Specifications**	
H1	**Date of**	4/24/20

	Execution	
	Tester	Omar
	Expected Results	● Padding: top=200; bottom=16; left=16; right=16 ● Margin: top=16; bottom=0; left=0; right=0
	Actual Results	● Padding: top=200; bottom=16; left=16; right=16 ● Margin: top=16; bottom=0; left=0; right=0
	Test Status	Pass

H2	**Date of Execution**	4/24/20
	Tester	Omar
	Expected Results	● Color = white ● Shape = Rectangle ● Border Radius = 17
	Actual Results	● Color = white ● Shape = Rectangle ● Border Radius = 17
	Test Status	Pass

H3	**Date of Execution**	4/24/20
	Tester	Omar
	Expected Results	● Color = black (± opacity) ● Blur Radius = 10 ● Offset: dx = 0.0; dy = 10.0
	Actual Results	● Color = black87 ● Blur Radius = 10 ● Offset: dx = 0.0; dy = 10.0
	Test Status	Pass

O1	**Date of Execution**	4/24/20
	Tester	Harsh
	Expected Results	● 10 Tasks in Task List ● Tasks match with their respective descriptions (correctly mapped)
	Actual Results	● There are 10 tasks in the list ● Each task has correct task description
	Test Status	Pass

O2	**Date of Execution**	4/24/20
	Tester	Harsh
	Expected Results	● Each task has three objects ● Task Name, Table and Description ● They each have correct data types
	Actual Results	● Each task has three objects with correct data types ● Task Name, Table and Description
	Test Status	Pass

O3	**Date of Execution**	4/24/20
	Tester	Harsh
	Expected Results	● Future Builder is invoked with respect to the length of the list ● Currently there are 10 tasks in the list ● There should be 10 task widgets with the corresponding task details like table number, name and description
	Actual Results	● Every task matches with the correct description and details ● There are 10 widgets available on the screen
	Test Status	Pass

E1	**Date of Execution**	4/24/20
	Tester	Brian
	Expected Results	● Http message should return a status code of 200 indicating is successfully finished the task
	Actual Results	● A Http message with a status code of 200 was returned
	Test Status	Pass

E2	**Date of Execution**	4/24/20
	Tester	Brian
	Expected Results	● Http message should return a status code of 404 indicating the task to finish could not be found
	Actual Results	● A Http message with a status code of 404 was returned
	Test Status	Pass

E3	**Date of Execution**	4/24/20
	Tester	Brian
	Expected Results	● Http message should return a status code of 500 indicating the SQL server is down
	Actual Results	● A Http message with a status code of 500 was returned
	Test Status	Pass

E4	**Date of Execution**	4/24/20

	Tester	Brian
	Expected Results	● Http message should return a status code of 400 indicating a bad request from the user
	Actual Results	● A Http message with a status code of 400 was returned
	Test Status	Pass

B1	**Date of Execution**	4/24/20
	Tester	Emmanuel
	Expected Results	● Http message should return a status code of 200 indicating a good request from the user
	Actual Results	● A Http message with a status code of 200 was returned
	Test Status	Pass

B2	**Date of Execution**	4/24/20
	Tester	Emmanuel
	Expected Results	● Http message should return a status code of 200 indicating a good request from the user ●
	Actual Results	● A Http message with a status code of 200 was returned ● Specified user data was retained securely
	Test Status	Pass

B3	**Date of Execution**	4/24/20

	Tester	Emmanuel
	Expected Results	● Http message should return a status code of 400 indicating a bad request from the user
	Actual Results	● A Http message with a status code of 400 was returned
	Test Status	Pass

8 Regression Testing

For the purposes of this project no regression tested was needed as it applies for CS 440.

III Inspection

9 Items to be Inspected

ID	Author	Code to be Inspected

1	**Omar**	
		```
		*// Pre-Condition: getTasks() function should set the variable listComTasks to the retrieved list*
		Widget _buildCompletedList() {
		  return FutureBuilder(
		    future: getTasks(), *// Future which returns a list of completed tasks*
		    builder: (context, snapshot) { *// Build widget based off the getTask info*
		      return snapshot.hasData ?
		              listComTasks.length == 0 ?
		              new Container(child: Text('No Completed Tasks!'))  *// No tasks*
		            : new ListView.builder( *// Has Tasks*
		                itemCount: listComTasks.length,
		                itemBuilder: (BuildContext content, int index){  *// Build individual task items*
		                  Task task = listComTasks[index];
		                  return CompletedList(task, content);
		                },
		            )
		            : new Center(child: SpinKitWave(color: Colors.*cyan*, size: 50)); *// Loading icon*
		      },
		    );
		}

		    );
		```

```
}
```

2 **Harsh**

```
dialogContent(BuildContext context){ // Dialog Content for the popup
 return Stack(
   children: <Widget>[
     Container(
       padding: EdgeInsets.only( // specify the dimensions and position
           top: 200,
           bottom: 16,
           left: 16,
           right: 16
       ),
       margin: EdgeInsets.only(top: 16),
       decoration: BoxDecoration(
         color: Colors.white,
         shape: BoxShape.rectangle,
         borderRadius: BorderRadius.circular(17),
         boxShadow: [
           BoxShadow(
             color: Colors.black87,
             blurRadius: 10.0,
             offset: Offset(0.0, 10.0),

               )
         ]
       ),
```

3 **Brian**

```
  /**
   * @param request http request to send and receive
   * @return 200 - Statistics of completed tasks based on the day 500 - Error connecting
to database
   *        500 - Error parsing for average time
   */
  public HttpResponseMessage getTasksStats(HttpRequestMessage<Optional<String>>
request) {
    try {
      Map<Date, ArrayList<Task>> datesToTask = new HashMap<>();
      List<Task> allTasks = taskDbo.getAllCompletedTasks();
      allTasks.stream()
          .filter(task->task.completionTime!=null)
          .forEach(
              task -> {
                  if (datesToTask.containsKey(task.taskDate))
                    datesToTask.get(task.taskDate).add(task);
                  else
                    datesToTask.put(
                        task.taskDate, new ArrayList<>(Collections.singletonList(task)));
              });
      List<TaskStats> taskStats = new ArrayList<>();
      for (Map.Entry<Date, ArrayList<Task>> entry : datesToTask.entrySet()) {
        Date key = entry.getKey();
        ArrayList<Task> value = entry.getValue();
        OptionalDouble averageTime =
            value.stream().mapToLong(task ->
Time.valueOf((task.completionTime)).getTime()).average();
        int numberOfTasks = value.size();
        taskStats.add(new TaskStats(key.toString(), numberOfTasks, averageTime));
      }
      taskStats.forEach(task -> System.out.println(task.averageTimeTaken));
      taskStats.sort(Comparator.comparing((TaskStats taskOne) -> taskOne.date));
      return request.createResponseBuilder(HttpStatus.OK).body(taskStats).build();
    } catch (ObtainingAverageException e) {
      return
request.createResponseBuilder(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
parsing average time ").build();
    } catch (SQLException e) {
      return request
          .createResponseBuilder(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
connecting to SQL database").build();
    }
  }
```

4 **Emmanuel**

```
  /**
    *
    * @param request http request to send and receive
    * @param employeeID employeeID to authenticate with
    * @param password unencrypted password
    * @return 200 if valid username and password
    *        401 if invalid password
    *        404 if username not found
    *        500 if internal server error
    */


 public HttpResponseMessage
authenticate(HttpRequestMessage<Optional<UserAuthenticationRequest>> request,
String employeeID, String password) {
        try{
            UserAuthenticationDBO userAuthenticationRequest =
userDBO.userAuthenticate(employeeID);
            if(userAuthenticationRequest.passwordtoken.equals(password)){
                userDBO.logUserIn(employeeID);
                return request.createResponseBuilder(HttpStatus.OK).body(new
UserAuthenticateResponse(userAuthenticationRequest.firstName,userAuthenticationReq
uest.lastName,userAuthenticationRequest.isManager)).build();
            }
            else
                return
request.createResponseBuilder(HttpStatus.UNAUTHORIZED).body("Valid user but
incorrect password").build();
        }
        catch (UserNotFoundException e){
            return request.createResponseBuilder(HttpStatus.NOT_FOUND).body("Could not
find user").build();
```

		}
		catch (SQLException e){
		return
		request.createResponseBuilder(HttpStatus.INTERNAL_SERVER_ERROR).body("Error connecting to SQL database").build();
		}
		}

10 Inspection Procedures

The inspection checklist that we used for this section helped us look at the basic programming practices [4]. We went this route since this project incorporates several languages and didn't want to have a different checklist for every language used.

[1] New York University. "Code Review Checklist." *Effective Code Reviews: Code Review Checklist*, nyu-cds.github.io/effective-code-reviews/03-checklist/.

11 Inspection Results

Code ID	**Inspector**	**Date and Time Inspected**	**Discoveries**
2	**Omar**	**4/24/20**	● Function is incomplete, it's missing more components (braces, brackets, and parenthesis do not match) ● Code is easily understood, but can become difficult for programmers who are new to Dart and Flutter since there are no comments
3	**Omar**	**4/24/20**	● Code is easily understood, but can become difficult to follow for programmers who are new to Java and its data structures since there are no comments ● Input and Output is well documented ● Using "for each" loops was a good feature since it eliminates out of bound errors ● Memoization of the dates in the datesToTask HashMap was a good performance enhancer
4	**Omar**	**4/24/20**	● Code is easily understood, but can become difficult to follow for programmers who are new to Java and its data structures since there are no comments

			● Input and Output is well documented
1	**Harsh**	**4/24/20**	● Code is very well written and follows the regular rules of documentation ● The dimensions and container sizes are predefined and can be changed according to the devices if needed
3	**Harsh**	**4/24/20**	● Code is very well written and requires a good understanding of Java to understand ● The objects follow proper programming naming conventions making the code readable.
4	**Harsh**	**4/24/20**	● Code is very well written and requires a good understanding of Java ● It follows proper programing rules and is readable
1	**Brian**	**4/24/20**	● Code is clear and easy to understand ● Comments throughout code allow for clear understand even without a dart foundation ● Proper documentation to specify exact inputs and outputs at beginning of the function would make it more clear
2	**Brian**	**4/24/20**	● Use of function is not completely clear without specific comments telling us its use. Dialog content for popup is not clear. ● Code within function to specify what each number means would make it more clear ● Code is easy to follow
4	**Brian**	**4/24/20**	● Input and outputs for code are very clear ● Code is not optimized for asynchronous and multiple threads. Code will likely not be able to support all necessary users
1	**Emmanuel**	**4/24/20**	● Code is well commented and concisely organized, may require some reasonable knowledge to understand ● Inputs and outputs are clearly distinguishable in code

2	**Emmanuel**	**4/24/20**	● Code is very clear and easy to follow ● Included comments offer good information ● Values in containers are predefined
3	**Emmanuel**	**4/24/20**	● For loop algorithm is justified for the use, containers follow a design that could be easily expanded ● Good use of error handling to cover nuance cases

IV Recommendations and Conclusions

Tester/Inspector	**Comments**
Omar	● Tests H1, H2, and H3 all passed with success ● Code ID 2 should include the full working code and should have comments ● Code ID 3 and 4 should include comments
Harsh	● Tests O1, O2, and O3 all passed without any issues ● All Code IDs are well commented and follows regular documentation rules ● It is a good practice to make Containers with defined dimensions or with respect to the device.
Brian	● Tests E1,E2,E3,E4 all passed successfully ● Code 1,2,3,4 is overall well documented ● Code 4 should be better optimized for asynchronous run inorder to support more users
Emmanuel	● Tests B1, B2, and B3 all passed with success ● All Code IDs followed good design structures and were clear and concise. ● Code 3 and 4 could use a bit more comments

V Project Issues

12 Open Issues

No offline support - Currently, the application needs to have an active internet connection to work since it's constantly communicating with the backend system for updates.

Many users support - Currently, the backend code is single threaded and will likely not be able to support the proper amount of users.

13 Waiting Room

Remember me and Forgot Password - One feature in the frontend that should be implemented is the "remember me" and "forgot password" in the login screen for all employees. This will allow them to quickly log into the application and also have the option to reset, or recover, their forgotten password.

Add and Delete Dining Table - One feature for the frontend, backend, and database is to add support for adding or deleting a dining table from the application. This is needed since restaurants should have the ability to alter tables that reflect changes in the restaurant during a given night.

14 Ideas for Solutions

Offline support solution 1 - One solution to this issue could be the support for task management offline and auto update the database when the application comes online again. This way employees can continue to self assign tasks so they can continue working.

Offline support solution 2 - Another solution to this issue could be to have bluetooth or LAN capabilities. This allows for the main features of sending and receiving tasks to continue. The manager will hold all the local data and can auto update the database when the manager application comes online again.

Many users support 1 - Implement the use of java futures and other asynchronous features built into java 8 in order to fix this issue

Many users support 2 - Rather than using Azure function which only supports java 8, create a web application in Azure allowing us to upgrade to the latest version of java. This will allow us to use even more, easier to use java features like async/await for parallel programming .

15 Project Retrospective

One method that really works is ice scrum and the general agile methodology for the coding project since it provides the teams a way to visually look at the tasks at hand. Likewise, we are also able to see the deadlines for all the releases throughout the sprints.

VI Glossary

Dining Table:	Used for customers to be seated
Token:	This is the encrypted password for a given employee used in the database
Primary Keys (Database):	Fields which are underlined
Foreign Keys (Database):	Fields which are underlined and italicized
Is Active (Database):	A dining table which is able to seat customers
Is Occupied (Database):	A dining table which currently has customers seated
Dialog (Flutter):	Using material design from Flutter
Future Builder (Flutter):	Widget that builds itself once it receives the data

VII References / Bibliography

[1] University of Illinois At Chicago. “Wait-Less Project Description”, Fall 2019

[2] University of Illinois At Chicago. “Wait-Less Project Design”, Fall 2019

[3] University of Illinois At Chicago. “Wait-Less Project Requirements”, Fall 2019

[4] New York University. “Code Review Checklist.” *Effective Code Reviews: Code Review Checklist*, nyu-cds.github.io/effective-code-reviews/03-checklist/.

VIII Index

N/A