Tasker Application Documentation

Table Of Contents

1. Application Functions

The purpose of this application is to provide a task management service for the user. The user will be able to:

1. User can see a list of tasks that they have (sorted by date - earliest at top)
2. User can add a new task to the list
3. User can tick off the task that they have completed

4. User can remove all the task they have completed

2. Endpoint REST Service

- Provides a REST service for use

Platform

- Developed using dropwizard.io framework with maven
- For setup refer to: https://www.dropwizard.io/en/latest/getting-started.html
- Initial maven setup using command line:

```
    mvn archetype:generate -DarchetypeGroupId=io.dropwizard.archetypes -
DarchetypeArtifactId=java-simple -DarchetypeVersion=2.0.0
```

- Details of the application

```
    <groupId>com.taskmanager.tasker</groupId>
    <artifactId>Tasker</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
```

- DataStorage:
  - h2 is used for data storage (development) as the application is simple with only 1 entity (see below)
    - System will need to configure for another database e.g. mysql or PostgreSQL for a more permenent solution
    - To change the configuration file and import the required dependencies
  - Persistance is done with the help of hibernate
  - Table creation upon start up of application

Dependencies

- The application uses the following dependencies
  - dropwizard-dependencies
  - dropwizard-core
  - dropwizard-db
  - dropwizard-hibernate
  - h2
  - mysql (not used but can be used if a more permenent database is required)

```
    <dependencies>
        <dependency>
            <groupId>io.dropwizard</groupId>
            <artifactId>dropwizard-core</artifactId>
        </dependency>
        <dependency>
            <groupId>io.dropwizard</groupId>
            <artifactId>dropwizard-db</artifactId>
        </dependency>
        <dependency>
            <groupId>io.dropwizard</groupId>
            <artifactId>dropwizard-hibernate</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
```

Setup and Configuration

- The application has a config.yml file that stores the configuration of the application

```
logging:
  level: INFO
  loggers:
    com.taskmanager.tasker: DEBUG

# Database settings.
database:
  # the name of your JDBC driver
  driverClass: org.h2.Driver

  # the username
  user: sa

  # the password
  password:

  # the JDBC URL
  url: jdbc:h2:mem:datajpa

  # any properties specific to your JDBC driver:
  properties:
    charSet: UTF-8
    hibernate.dialect: org.hibernate.dialect.H2Dialect
    hibernate.hbm2ddl.auto: create

  # the maximum amount of time to wait on an empty pool before throwing an
```

```
exception
  maxWaitForConnection: 3s

  # the SQL query to run when validating a connection's liveness
  validationQuery: "/* MyApplication Health Check */ SELECT 1"
```

- Configuration for the database is set up in TaskerConfiguration at the com.taskmanager.tasker package

```
public class TaskerConfiguration extends Configuration {

    @Valid
    @NotNull
    private DataSourceFactory database = new DataSourceFactory();

    @JsonProperty("database")
    public DataSourceFactory getDataSourceFactory() {
        return database;
    }

    @JsonProperty("database")
    public void setDataSourceFactory(DataSourceFactory dataSourceFactory) {
        this.database = dataSourceFactory;
    }
}
```

- Cross Origin Resource Sharing configuration
  - Control access to the endpoints
  - Set in the TaskerApplication

```
@Override
public void run(final TaskerConfiguration configuration,
                final Environment environment) {

    //Add the filter for cross origin
    final FilterRegistration.Dynamic cors =
            environment.servlets().addFilter("CORS",
CrossOriginFilter.class);

    // Configure CORS parameters
    cors.setInitParameter(CrossOriginFilter.ALLOWED_ORIGINS_PARAM, "*");
    cors.setInitParameter(CrossOriginFilter.ALLOWED_HEADERS_PARAM, "X-
Requested-With,Content-Type,Accept,Origin,Authorization");
    cors.setInitParameter(CrossOriginFilter.ALLOWED_METHODS_PARAM,
"OPTIONS,GET,PUT,POST,DELETE,HEAD");
    cors.setInitParameter(CrossOriginFilter.ALLOW_CREDENTIALS_PARAM,
"true");

    // Add URL mapping
    cors.addMappingForUrlPatterns(EnumSet.allOf(DispatcherType.class), true,
  "/*");
  }
```

Entities

- Entities store in com.taskmanager.tasker.api
- The application has 1 entity:
- Task
  - Purpose:
    - To hold the task infomation
  - Attributes:
    - id: UUID (PK) - (Autogenerate with uuid2)
    - taskDescription: String
    - taskDate: Date
    - taskCompleted: boolean (identify if the task has been completed)
  - Constructors:
    - Task(taskDescription, taskDate, taskCompleted)
    - Task()
  - Getters/Setters:
    - Getters and setters for all attributes.
    - Has annotation @JSONProperty

Data Access Object (DAO)

- DAO stored in com.taskmanager.tasker.db
- The application has 1 DAO:
- TaskDAO
  - Purpose:
    - To access the database table which maps the entity task
  - Extends:
    - AbstractDAO

    ```
        public class TaskDAO extends AbstractDAO<Task>{}
    ```

  - Attribute:
    - sessionFactory : SessionFactory
  - Constructor:

    ```
    public TaskDAO(SessionFactory sessionFactory) {
        super(sessionFactory);
        this.sessionFactory = sessionFactory;
    }
    ```

  - Methods:

- findById()
  - Description: To obtain a Task object by its Id
  - Input Parameters: id (id of object in UUID)
  - Returns: Task object

```
public Task findById(UUID id)
```

- findAll()
  - Description: To obtain a list of all task
  - Input Parameters: null
  - Returns: List<Task>

```
public List<Task> findAll()
```

- save()
  - Description: To persist the task into the database
  - Input Parameters: task (a Task object)
  - Returns: UUID (the id of the saved task)

```
public UUID save(Task task)
```

- deleteCompletedTask()
  - Description: To delete Tasks object with isCompleted == true from database
  - Input Parameters: null
  - Returns: null

```
public void deleteCompletedTask()
```

Resources and Endpoints

- Resources stored in com.taskmanager.tasker.resources
- The application has 1 Resource:
- TaskManagerResource:
  - Purpose : To provide endpoints related to the task object
  - Produces : MediaType.APPLICATION_JSON
  - Attribute:
    - taskDAO : TaskDAO (stores the DAO)
  - Constructor:

- TaskManagerResource(TaskDAO)
  - Takes a TaskDAO as a an argument

```
public TaskManagerResource(TaskDAO taskdao) {
    this.taskdao = taskdao;
}
```

- Endpoints:
  - getTaskList()
    - Path: /api/taskmanager/list
    - Type: GET
    - Desciption: To get a list of task (sorted by date)
    - Input Parameters: null
    - Returns: array with Json object (task)

    ```
    public List<Task> getTaskList()
    ```

  - saveTask(Task task)
    - Path: /api/taskmanager/save
    - Type: POST
    - Desciption: To save a new task (if no task id)or to update the isCompleted status of the current task
    - Input Parameters: task (Task object)
    - Returns: task (if saved)
    - WebApplicationException: throws when task id is not null but is invalid

    ```
    Task saveTask(Task task)
    ```

  - getTaskById(String id)
    - Path: /api/taskmanager/get/{id}
    - Type: GET
    - Desciption: To get a task from the id provided in the path parameter
    - Input Parameters: id (UUID)
    - Returns: task (if found)
    - WebApplicationException: throws when task id is not null but is invalid

    ```
    public Task getTaskById(@PathParam("id") String id)
    ```

  - clearCompleteTask()
    - Path: /api/taskmanager/clear
    - Type: DELETE

    - Desciption: To delete the task which are completed from the database
    - Input Parameters: null
    - Returns: Response

```
public Response clearCompleteTask()
```

  - Other methods:
    - sortTaskList(List tasklist)
      - Description: To sort a list of task by date
      - Input Parameters: tasklist
      - Returns: null

```
private void sortTaskList(List<Task> tasklist)
```

- Creating a new TaskManagerResource in the TaskerApplication and registering it to the jersey

```
@Override
public void run(final TaskerConfiguration configuration,
                final Environment environment) {
    final TaskDAO dao = new TaskDAO(hibernate.getSessionFactory());
    environment.jersey().register(new TaskManagerResource(dao));

    // Configure CORS parameters
    // *** codes omitted for brevity ***

}
```

Running application locally

- Navigate to the folder containing the application
- Package code in distributed format

```
mvn package
```

- Use the cmd java -jar <package java file> server <config file>

```
java -jar Tasker-1.0-SNAPSHOT.jar server config.yml
```

- The application should start on the http://localhost:8080 by default

3. User Interface

Platform

- The user interface is built using ReactJS
- Refer to the application functions for the features present

Dependencies

- "axios": "^0.21.1"
- "@fortawesome/free-regular-svg-icons": "^6.1.1"
- "@fortawesome/react-fontawesome": "^0.2.0"
- "react": "^18.2.0",

Services

- Consist of 1 service:
- taskService
  - Uses axios as a means of interacting with API
  - Interacts with the endpoint (previous section)
  - Methods:
    - getTaskList()
      - Method: GET
      - Input Parameters: null
      - API called: "/api/taskmanager/list"
    - addTask(taskDescription, taskDate)
      - Method: POST
      - Input Parameters:
        - taskDescription : String
        - taskDate : long (unix timestamp)
      - API called: "/api/taskmanager/save"
    - changeStatus(task)
      - Method: POST
      - Input Parameters:
        - task : a json object with the names of id, taskDescription, taskDate, isCompleted
      - API called: "/api/taskmanager/save"
    - deleteCompleted()
      - Method: DELETE
      - Input Parameters: null
      - API called: "/api/taskmanager/clear"

Components

- Consist of 1 component:
- Home
  - CSS file: Home.css
  - Description: Display a page handling the task management

        - Obtain a list of task upon landing on page : findData
        - Displays the form for new task by clicking on button triggering : displayForm
        - Saves the new task by clicking on save button triggering : addTask
            - Description field is validated to be not null
            - Date field is validated to be in the form yyyy-mm-dd
        - Task can be checked (meaning completed) or unchecked (meaning not completed) : selectTask
        - Completed task can be deleted triggering : deleteCompletedTask
    - Methods:
        - findData()
            - Description: To obtain a list of task with the help of taskService
        - displayForm()
            - Description: To display the new task form
        - selectTask()
            - Description: To check a task (completed) or uncheck it (not completed)
        - addTask()
            - Description: Add a new task with the use of taskService
            - Uses: obtainValidDate() to check date, uses clearNewTask() to clear the fields
        - deleteCompletedTask()
            - Description: To call the taskService to have it remove all tasks that are completed
            - Uses: findData() to regenerate list
        - obtainValidDate()
            - Description: Checks if the date input in the new task form is valid
        - getDate()
            - Description: Converts a long unix timestamp to a valid string date yyyy-mm-dd
        - clearNewTask()
            - Description: clear the description and date fields of the new task form

Running application locally

- Navigate to the folder containing the application
- Install the necessary packages

```
npm install
```

- Starting the application

```
npm start
```

- The application will start on http://localhost:3000
- The application default api server is http://152.67.99.60:8085
- To specify the api server during start

```
# windows command prompt:
set "REACT_APP_APIURL=<server>" && npm start

# windows powershell:
($env:REACT_APP_APIURL="<server>") -and (npm start)

# Linux
REACT_APP_APIURL=<server> npm start
```

4. Containerizie solution

Building Docker Images

- The steps taken is similiar for both the endpoint REST service and the UI

1. Navigate to the root folder of the respective component
    - Tasker (endpoint REST service)
    - Tasker-Client (React UI)
2. Ensure that Dockerfile is present (Create if missing)
    - For Tasker

    ```
    #Set the image file
    FROM openjdk:8-jre-alpine

    # Create new app directory (at the image side)
    Run mkdir /app

    CMD ["export JAVA_HOME=`which java`"]

    # To select the port
    EXPOSE 8080

    #Copy from host machine to the image
    COPY ["/target/Tasker-1.0-SNAPSHOT.jar","/app"]
    COPY ["/config.yml","/app"]

    # The app directory
    WORKDIR /app

    # To add an entry point
    ENTRYPOINT exec java -jar Tasker-1.0-SNAPSHOT.jar server config.yml
    ```

    - For Tasker-Client
        - There should be a .dockerignore with (node_modules)

- Specify the ENV: REACT_APP_NOT__APIURL= if not the default

```
FROM node:alpine

# Setting up variables for change
ARG epURL=http://152.67.99.60:8085
ENV epURL_env =$epURL

# Create new app directory (at the image side)
Run mkdir /app

# Copy the pakage.json
COPY /package.json /app
COPY ./ /app

# The app directory
WORKDIR /app

# Install the necessary files
RUN npm install

# To select the port
EXPOSE 3000

# Default executable command

CMD REACT_APP_APIURL=${epURL_env} npm start

# CMD["npm","start"]
```

3. In the command line, build the docker image

   - docker build -t <repo/file>:<tag>
   - Tasker:

```
    docker build -t leebaojin/tasker:v1.0
```

   - Tasker-Client:
     - There is an option set for the client app to specify the url of the endpoint
     - For default, which will use the deployed end point

```
    docker build -t leebaojin/taskerclient:v1.0
```

     - To run on localhost for testing
       - use: docker build --build-arg epURL= -t :

```
    docker build --build-arg epURL=http://localhost:8080 -t
test/clientapp:v0.1 .
```

4. Check the images

```
docker images
```

5. The docker images should be displayed
6. To run the docker image in a container
    - Use: docker run -d -p=<container_port>:<tcp_port> --name <docker_image_name>

```
docker run -d -p=8080:8080 --name tasker leebaojin/tasker:v1.0
```

7. Check the container running

```
docker ps
```

8. To stop the docker container

```
docker stop <Container_ID>
```

9. Remove container

```
docker rm <Container_ID>
```

Pushing docker images to DockerHub

1. Ensure that docker hub has the appropriate repository
2. Use docker push <repo/file>:<tag>

```
docker push leebaojin/tasker:v1.0
```

5. Endpoint Testing

Frameworks

- Use mocha-awesome and chai for the test
- Test written in javascript
- To setup an empty envrionment:
    1. Navigate to the file for the test
    2. (optional if already have the folder) Clone a project

```
    git clone https://github.com/mitchallen/autom8able-mochajs-starter.git
mocha-awesome
```

3. Install mocha-awesome
    - Ref: https://scriptable.com/blog/mocha-awesome

```
    npm install
    npm install --save-dev mochawesome
```

4. Install chai (used for custom)
    - Ref: https://www.chaijs.com/guide/installation/

```
    npm install chai-http
```

Test Sequence

1. Check server is present
2. Attempt to save a new task ("/api/taskmanager/save")
3. Attempt to get the saved task ("/api/taskmanager/get/{id}")
4. Repeat task 2 and 3
5. Get a list of task ("/api/taskmanager/list")
6. Modify task 1 to completed and save ("/api/taskmanager/save")
7. Delete completed task ("/api/taskmanager/clear")
8. Verify task has been deleted ("/api/taskmanager/get/{id}")
9. Change task 2 status and delete

- Test details can be found: https://docs.google.com/spreadsheets/d/1VRKMAjyNwBoEFTmCUQ-iuRqF9w4R5S0r/edit?usp=sharing&ouid=111269330940438351616&rtpof=true&sd=true

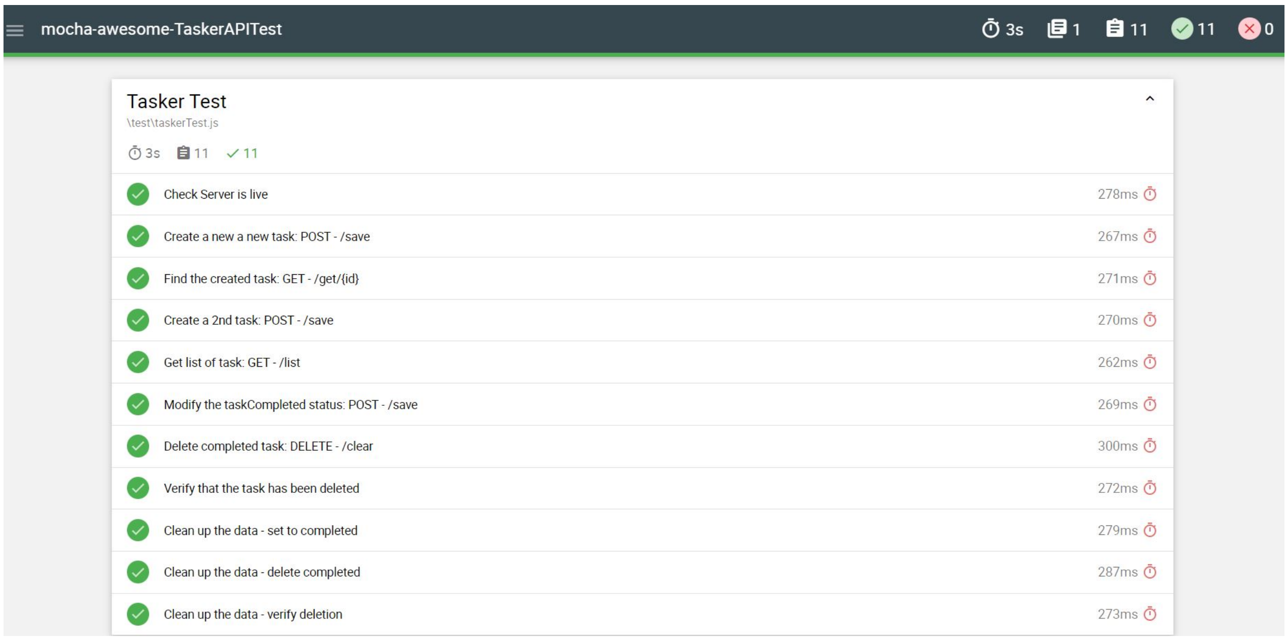
Running test

1. Navigate to the folder : TaskerAPITest-mocha-awesome
2. Check that the package.json file (should have test:awesome under scripts)
3. Place the test script in the "test" folder
4. Run the test in the folder

```
    npm run test:awesome
```

Getting Result

1. Open the "mochawesome-report" folder
2. Open the mochawesome.html
3. Html provide test result


mocha-awesome-TaskerAPITest 3s 1 11 11 0

Tasker Test
\test\taskerTest.js
3s 11 ✓ 11

✓	Check Server is live	278ms
✓	Create a new a new task: POST - /save	267ms
✓	Find the created task: GET - /get/{id}	271ms
✓	Create a 2nd task: POST - /save	270ms
✓	Get list of task: GET - /list	262ms
✓	Modify the taskCompleted status: POST - /save	269ms
✓	Delete completed task: DELETE - /clear	300ms
✓	Verify that the task has been deleted	272ms
✓	Clean up the data - set to completed	279ms
✓	Clean up the data - delete completed	287ms
✓	Clean up the data - verify deletion	273ms

6. UI Testing

Frameworks

1. Test written in Java using maven, selenium and testNG
2. Dependencies

```
    <dependencies>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-
java -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.2.2</version>
        </dependency>
        <!--
https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.2.1</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-
api -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.8.2</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.testng/testng -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.9.10</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.github.stephenc.monte/monte-
screen-recorder -->
        <dependency>
            <groupId>com.github.stephenc.monte</groupId>
            <artifactId>monte-screen-recorder</artifactId>
            <version>0.7.7.0</version>
        </dependency>
    </dependencies>
```

3. Other code source:
    - uses MyScreenRecorder (class file)from: https://github.com/naveenanimation20/ScreenRecorder
        - To help create method for recording the test as a video

Test Sequence

1. Test that the correct site has been navigated to
2. Click on New button to display the new task form - Verify that the form is displayed
3. Enter invalid date on new task form (with description blank). Click save - Verify that an error is displayed
4. Enter the valid date and a description. Click save
5. Verify that the new task appear at the top of task list
6. Click on the "-" button to close the new task form - Verify that the form is hidden
7. Click on checkbox of the newly created task - Verify that it is checked
8. Click on the "clear completed" button below - Verify that the checked task has been removed

- Test details can be found: https://docs.google.com/spreadsheets/d/1VRKMAjyNwBoEFTmCUQ-iuRqF9w4R5S0r/edit?usp=sharing&ouid=111269330940438351616&rtpof=true&sd=true

Running Test

1. Navigate to the folder : TaskerUI-testng
2. Ensure command prompt at the correct folder
3. To run the test
    - On the webserver hosting page (default)

    ```
    mvn test
    ```

    - An alternate url to be used

        - mvn test "-DUI_URL=(your url)"

    ```
    mvn test "-DUI_URL=http://localhost:3000/"
    ```

4. Wait for the test to complete

Getting Result

1. from curent path open: target/surefire-reports/Surefire suite
2. Open the report: Surefire test.html
3. Report display the results of test UI Test Image
4. To view a video recording:
    - Go back to : TaskerUI-testng
    - open : recordings
    - The video recordings are stored in this location
5. Sample recording: https://drive.google.com/file/d/1jtJW33NxN0P1qdXtBykaqabwsnO8uFo9/view?usp=sharing

7. Deployment

Platform

1. Uses Oracle Cloud as deployment platform
2. The UI and the endpoint REST service are containerize with as docker images
3. Docker images are deployed to cloud
    - Endpoint Rest Service: http://152.67.99.60:8085
    - REACT UI: http://150.230.10.235:3000

Details

This provides a brief summary of the setup of the compute instance for the docker images

Create a compute instance

1. Create an account at oracle cloud and login
2. Click on the menu icon at the top left
    - Select compute -> instances
3. Click on create instance
    - Fill up the necessary infomation
    - Add an ssh public key (generate one if necessary)
        - This ssh key is essencial to access the compute instance
    - Press create
4. Wait for instance to set up and run

Enter instance through command prompt

1. Open command prompt

2. Login to the instance created as opc user
   - Use ssh opc@<ip address> -i <path to your private ssh key>

```
    ssh opc@160.213.15.80 -i ".ssh\mykey"
```

3. Upon login, you will see
   - xxxxxxxx -> date e.g. 20220601

```
    [opc@instance-xxxxxxxx-yyyy~]$
```

Pull docker image

1. Check if docker is present

```
    docker version
```

2. If no docker is present, proceed to install
   1. Go to root

```
    sudo -s
```

   2. Will see display

```
   [root@instance-xxxxxxxx-yyyy~]$
```

   3. Install utils and download docker

```
    dnf install -y dnf-utils zip unzip
    dnf config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo
```

   4. Install docker

```
    dnf remove -y runc
    dnf install -y docker-ce --nobest
```

   5. Grant Docker Priviliges to opc user

```
usermod -aG docker opc
```

6. Enable and start docker

```
systemctl enable docker
systemctl start docker
```

7. Switch to opc user

```
su - opc
```

- If unsure which user, use: whoami, which will show the user

```
whoami
```

8. Check docker version and images

```
docker version
docker images
```

3. Pulling docker images from docker hub
   - Use: docker pull :

```
docker pull tasker leebaojin/tasker:v1.0
```

4. Run the docker images
   - Follow instruction from earlier (Containerizie solution)

Set up the port

1. Return back to the oracle cloud website
2. Go to the list of instance
3. Setup the ingress rules for the security list

8. Folders

1. Tasker (API endpoints with java script - dropwizard.io)
2. Tasker-Client (Reactjs UI)
3. TaskerAPITest-mocha-awesome (API endpoint test for Tasker)

4. TaskerUI-testng (UI testing for Tasker-Client)
    - An alternative UI testing framework using BrowserStack is in folder: TaskerUI-testng-browserstack

9. Additional