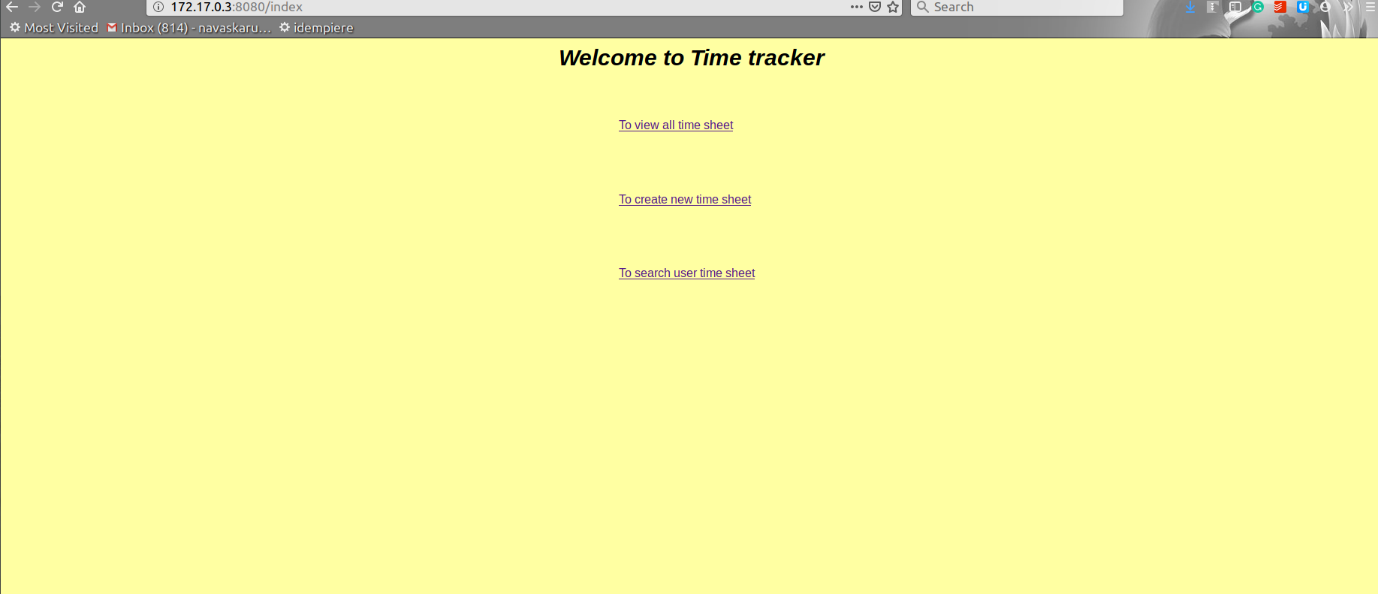
**Time Tracker application Overview:**

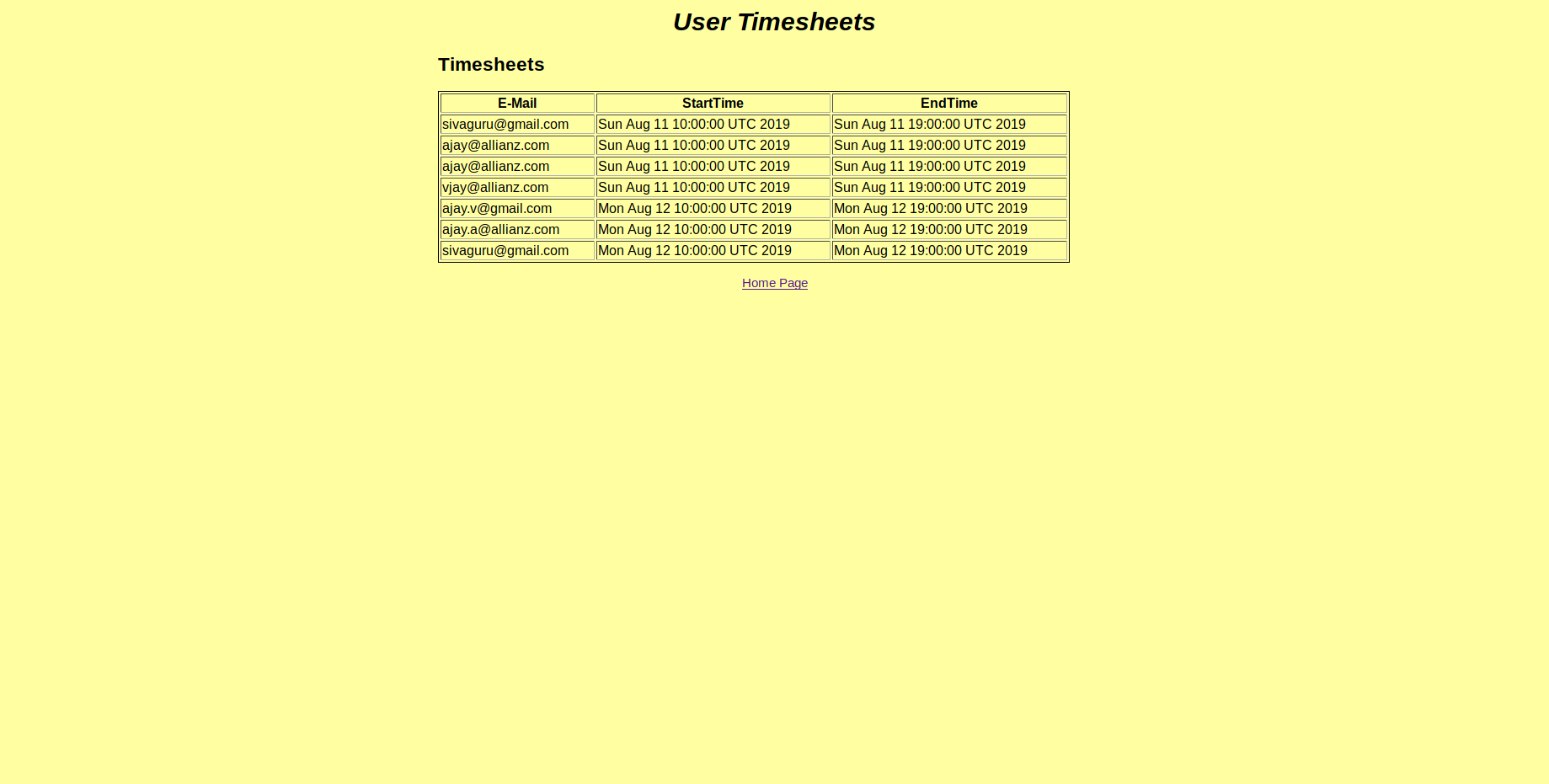
Time tracker application helps to track the user’s time sheet, create time sheet and search for time sheet using the user’s email id. The application source code is available in below GitHub repository.

<https://github.com/sivagurunathbabu/userTimeTrackeApp.git>

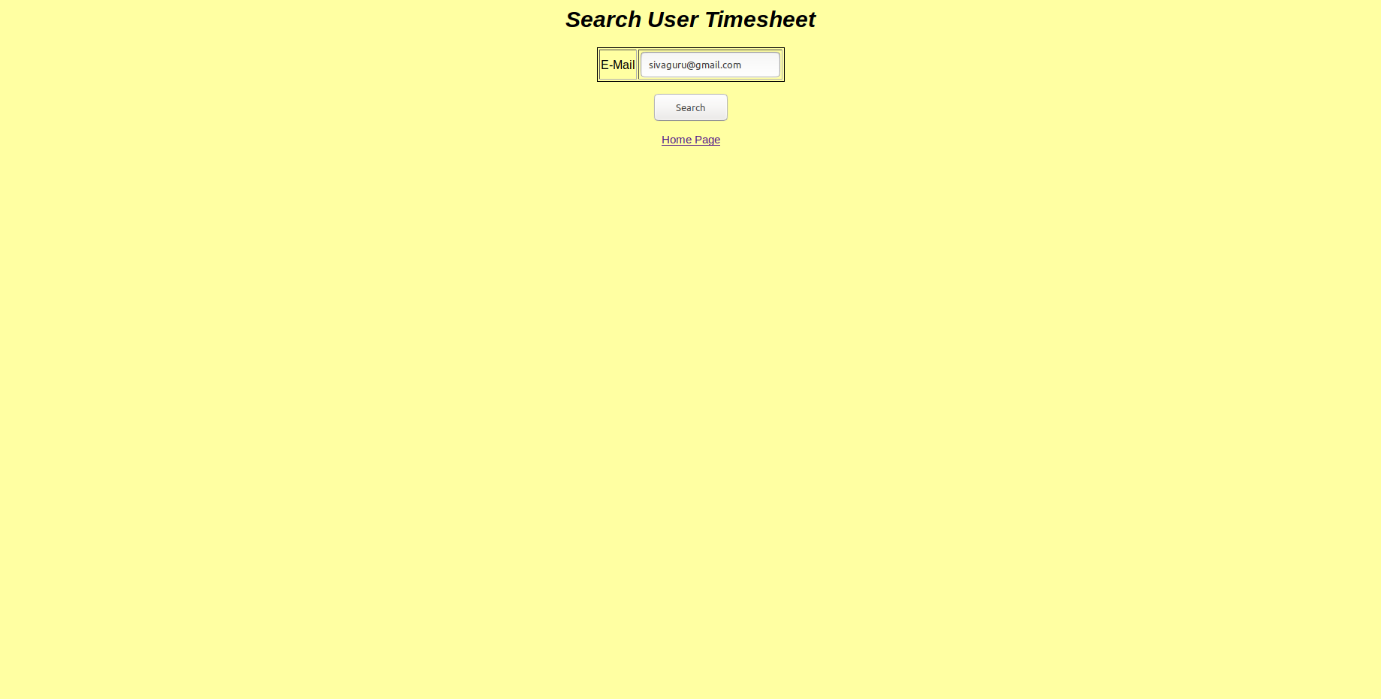
Welcome screen which list the user menus.



User timesheet screen to view the employee’s timesheet.



Search screen to search for a specific user timesheet.



Create timesheet screen to save new Time sheet.



**Technology and IDE used:** Java, Spring Boot, Maven, Mongo database, Eclipse, Docker Engine.

**Prerequisite**

* A linux operating system installed with Docker engine. For installation please refer the below URL.

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

* A non-root user account with sudo privilege set up on your server.

**Docker Overivew:**

Docker container wraps the application in a complete file system that contains everything it needs to run: code, runtime and system libraries.

The application (Time tracker) and database (Mongo database) images are stored in Docker hub registry at the public repository location ***siva-repo*** with the user account ***sivagurunathbabu*** (Docker id).

The images are created from the linux base images so it will work on linux docker container.

**Steps to containerize the application.**

Detailed description of the docker commands are mentioned in the section “Docker commands”.

1. Build the application jar using maven command (mvn install)
2. Create a directory in which we will write Dockerfile.txt and keep the maven generated application jar userTimeTracker-1.0.
3. Add following content into the Dockerfile.
   1. FROM openjdk:8
   2. LABEL maintainer="sivagurunathbabu@gmail.com"
   3. COPY userTimeTracker-1.0.jar userTimeTracker-1.0.jar
   4. EXPOSE 8080
   5. ENTRYPOINT ["java","-Dspring.data.mongodb.uri=mongodb://oreo/dev","-jar","userTimeTracker-1.0.jar"]

**Description of docker file instructions**

The FROM instruction specifies the base image from which we will be building.

The MAINTAINER instruction specifies the author field of the generated images.

The COPY instruction copies the files (userTimeTracker-1.0.jar) to the file system of the container.

The EXPOSE instruction instructs the container to expose the given port to the host system.

The ENTRYPOINT instruction is set to run the SpringBoot application with the JVM argument -Dspring.data.mongodb.uri=mongodb://oreo/dev when the container starts.

1. The image of the Time tracker application can be built with the below command.

sudo docker build -f ./Dockerfile.txt -t sivagurunathbabu/siva-repo:time-tracker-1.0

1. The mongo database can be launched with the below command. This starts the mongo database in the detached mode.

Ensure that the host system has /mongodb/data directory in the working directory folder. If not available create new directory /mongodb/data for container to share the mongo database data.

sudo docker run --name oreo -d -v /mongodb/data:/data/db -p 27017:27017/tcp sivagurunathbabu/siva-repo:mongodb

1. The time tracker application can be launched with the below command. This starts the time tracker application in the detached mode.

sudo docker run --name springdocker -d -v /mongodb/data:/data/db -p 8090:8080/tcp --link oreo:mongodb -sivagurunathbabu/siva-repo:time-tracker-1.0

1. The running containers can be displayed with the below command.

Verify the running containers with name oreo and springdocker with status UP.

sudo docker ps

1. Identify the IP details of the container to get access to containerized application (User Tracker). The IP details of the containers can be displayed with the below command.

sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' springdocker

1. Access the containerized the application with the URL of the identified IPAddress. The currently containerized application IPAddress is 172.17.0.3.

<http://172.17.0.3/index>

**Docker commands**

The docker build command builds the container image.

The docker pull command downloads the image from the registered repository.

The docker run command instantiates the container from the image if available, otherwise it downloads like pull command and instantiates the container.

The docker push command pushes the image into the registered repository.

The docker ps command lists the running container details.

The docker inspect command shows the detailed information of the container.

The docker logs command shows the console output of the running container.

**Build** *command arguments descriptions.*

Example:

sudo docker build **-f** ./Dockerfile.txt **-t** sivagurunathbabu/siva-repo:time-tracker-1.0

**-f** specifies the path of the Dockerfile.txt which is used to build the image.

**-t** builds the image with the given tag. The repository name is ***sivagurunathbabu/siva-repo*** and tag is ***time-tracker-1.0***

**Run** *command arguments descriptions.*

Example:

sudo docker run **--name** oreo **-d** **-v** /mongodb/data:/data/db **-p** 27017:27017/tcp sivagurunathbabu/siva-repo:mongodb

**--name** specifies the name of the container. The name helps in connecting the container from the other container.

**-p** publishes the port from the host to container.

**-d** specifies the mongo database container to run in detached mode. The container runs in the background. The output from the running mongo database can’t be seen in the terminal session.

**-v** specifies the mapping of your local host mongodb/data directory with the /data/db directory within the container.

**--link** adds links to the container.

**Ps** *command arguments descriptions.*

Example:

sudo docker ps -a

**-a** specifies the docker to list all the container details irrespective of their status (running/up, exited and etc).

**Inspect** *command arguments descriptions.*

Example:

sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' springdocker

--format specifies the docker to extract the IP address information from the container named springdocker