Technology Choices :

I have used following technologies in creating web service :

1. **Flask** - It is microframework for python which we used to build Restful API.
2. **SQlite** - As SQLite comes inbuilt with python, I thought its convenient to use it rather creating separate databases like mysql, postgres, oracle
3. **Flask SQLAlchemy** - SQLAlchemy is python SQL toolkit and ORM that gives us the same functionality as SQL.
4. **Flask Marshmallow** - I used this to render json response to user.
5. Python libraries : I used **Numpy, Scipy, Pandas, Scikit-learn** libraries to develop application to train, test and predict the model.

Design Choices:

My aim while creating web service was to implement rest-api functionality, not improving the accuracy of the model.

a) Datasets:
I have experimented this api on two sample classification datasets taken from Kaggle.com.
1. Glass Identification dataset
2. Breast Cancer Classification dataset

Both are included in folder named "**./rest-service-api/data**". We need to save datasets before running our service. Because I didn't find any way to preempt the location of dataset while running "curl" command.

b) Database: In database, we are not storing physical file of our dataset, but **pointer to location** where its stored. I have added extra field "Model" to store the location of model created during training.

c) Classification Algorithm used: **Random Forest Classifier**

d) Entrypoint for my application is "**Main.py**" python file. It is placed in "./rest-service-api/controllers" folder. It is placed this way, so as implement model-view-controller architecture.

e) **Dockerfile**: It is placed inside "./rest-service-api" folder, so that it can build docker image using "requirement.txt" file.

f) **requirement.txt**: It is placed inside "./rest-service-api" folder which contains all the dependencies required in our application.

g) **run.sh**: This file ensures creation of sqlite database before running our application and then routing to "main.py"

h) Our application run on host ip 0.0.0.0 to make it accessible using localhost instead of docker machine ip.

Task Performed:

1. First, we created database model for our application using SQL Alechemy and mentioned the field to expose to outside world.

2. For first part, we needed to perform basic crud opeartion on our database.

a. **Create Experiment**: In this, We first check if experiment id already exists in DB. If it is, then return json response immediately with appropriate message to handle the exception.
If it is not, we split dataset into training and test dataset to store in folder "./rest-service-api/data" and add the new row containing id, type, training & test file location to database and commit the transaction. Respond user back with success message.

b. **Retrieve Experiment:** There are two features we are providing. One, is to retrive exeperiment with given id or all experments in database.
To retrieve experiment with given id, we first checks if id is present in our database or not. If its not, then repond user back with appropriate message.

c. **Update Experiment**: Same goes for update and delete too. It checks if id is present in our database or not and then perform appropriate action.

d. **Delete Experiment:** It deletes the experiment by using id.

For each of CRUD, I have created separate endpoints.

3. For Second part, We needed to train, test the model and predict the values. Each has its own endpoint.

a. Train: This endpoint takes id as parameter, so to make sure which dataset to use for training(instead of sending json data to endpoint)
    -- Using this id, we first retrive training dataset from DB.
    -- Use randon forest classifier to fit the model and store the model in pickle file
    -- Save this pickle file location in database.

b. Test: This endpoint also takes id as parameter, so to make sure which dataset to use for testing.
    -- Using this id, we first retrive test dataset from DB.
    -- Using model saved in pickle file, we calculate accuracy and F1 score of our model.
    -- Save accuracy and F1 score to result column of database.

c. Predict: This endpoint takes id as parameter.
    -- It predicts the class based on model and data supplied.

Bonus Questions:

1. To handle multiple request, we have enabled option "**threaded = True"**.

However, to handle 1000 request/second, we can use dedicated web server like **Nginx** to deliver http response to user and application should be handled by a **WSGI application server** like **Gunicorn**.
Apart from that, The Flask-Cache extension allows you to reduce calls to database, additional computation.

2. To handle large data of 3GB to train, we can use **pandas library** that can load large CSV files in chunks. We need to load just one batch and process it, then discard that batch and move on.

3. If time to train each model is 3hr, I can increase number of estimators to make more trees for voting and which will take less time to compute result. Also, I can execute traning task in parallel threads.

4. To increase reponse time, I can use following flask extensions:
Flask-Compress extension compresses the application's response with gzip.
The Flask-Cache extension allows you to reduce calls to database, additional computation.
Flask-CDN extension provides developers with means to serve static content from CDNs
Flask-Assets,combining asset files reduces calls to the server, which leads to significant performance boost of your website.