

Lab 1 Assignment: Using REST (Node.js) and React JS

Part -1: Simple Calculator Application using Node.js and React.js

Purpose:

To have proper understanding of RESTful Services and interaction between client and server using node.js as a backend server and react.js as frontend client.

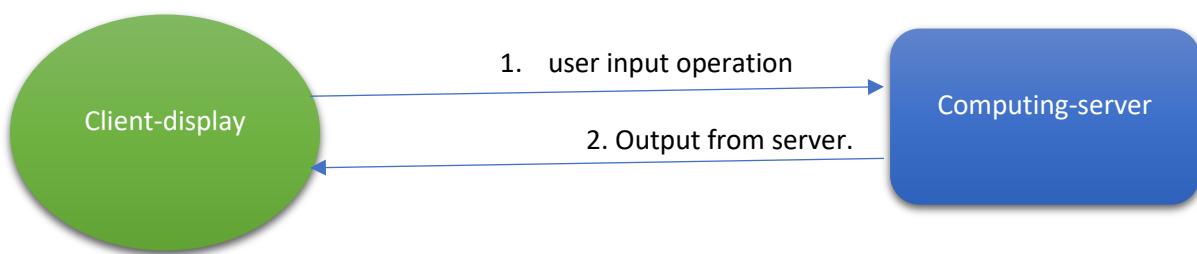
Goal:

Create a simple calculator application with the use of Node.js server and React.js as a client user interface.

The system should perform basic functionalities as below.

1. design a simple look alike online calculator using react.js
2. User can input the values in calculator display by clicking buttons.
3. As per the input the expression will be generated and on click of result button, user must be able to see the calculated result on screen.
4. user must be able to perform following functions using calculator.
 1. Addition 2. Subtraction 3. Multiplication 4. Division.
5. Client will make a request to node.js server where server will compute the result and respond to client with result. Client will display the result on screen to user.

Proposed System Design:



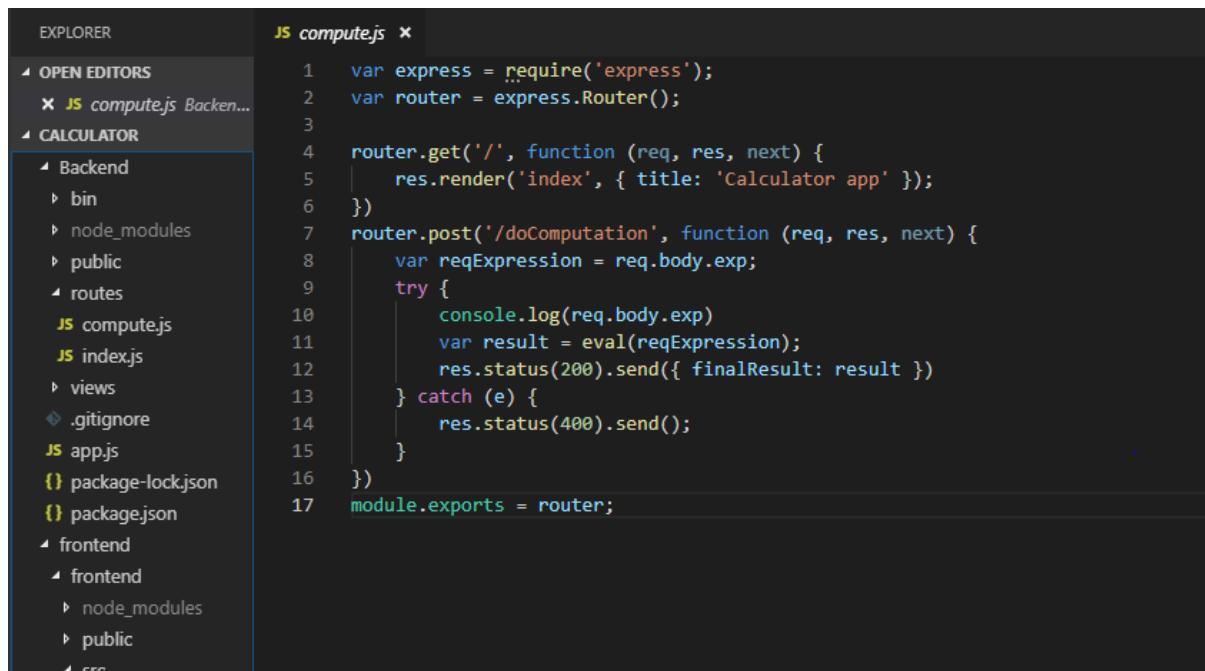
System Components:

1. Node.js Backend server using express.js :

Created a backend server which will serve client request and output the result back to client.

This contains a one RESTful service as computation API.

It runs on port 3005 and serves the client which is running on the port 3001 on localhost



```

EXPLORER JS compute.js ×
OPEN EDITORS
  JS compute.js Backend...
CALULATOR
  Backend
    bin
    node_modules
    public
    routes
      JS compute.js
      JS index.js
      views
      .gitignore
    JS app.js
    {} package-lock.json
    {} package.json
  frontend
    frontend
      node_modules
      public
      src
1 var express = require('express');
2 var router = express.Router();
3
4 router.get('/', function (req, res, next) {
5   res.render('index', { title: 'Calculator app' });
6 })
7 router.post('/doComputation', function (req, res, next) {
8   var reqExpression = req.body.exp;
9   try {
10     console.log(req.body.exp)
11     var result = eval(reqExpression);
12     res.status(200).send({ finalResult: result })
13   } catch (e) {
14     res.status(400).send();
15   }
16 })
17 module.exports = router;

```

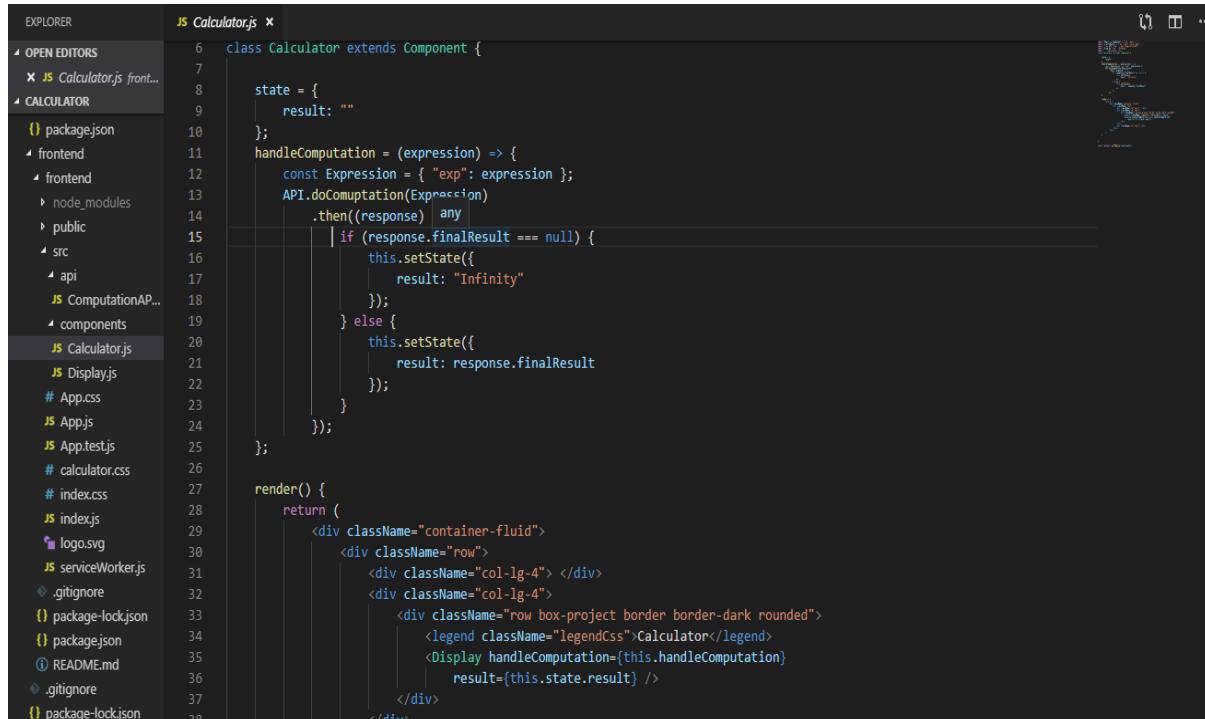
2. A calculator user interface using React.js:

A simple user interface for client that should be able to take input from client and display output to client.

Client API makes a call to server for computing the result. Server computes the result and send back a JSON response to client where client displays this result on the screen.

The frontend structure is described as below.

1. Calculator.js (display calculator)

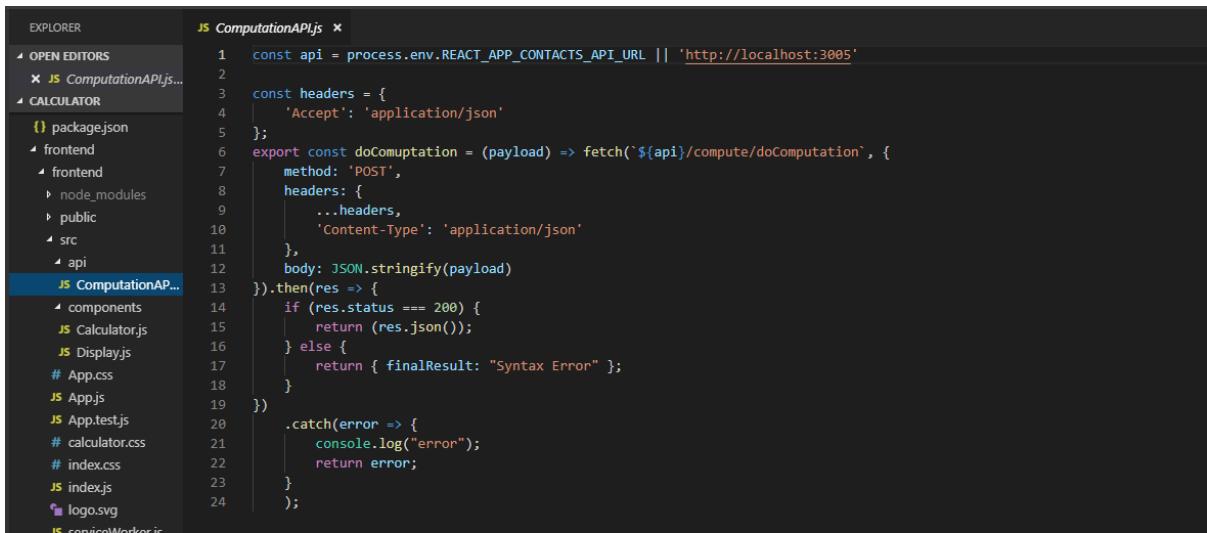


```

EXPLORER JS Calculator.js ×
OPEN EDITORS
  JS Calculator.js frontend...
CALULATOR
  package.json
  frontend
    node_modules
    public
    src
      api
        JS ComputationAP...
      components
        JS Calculator.js
        JS Display.js
        # App.css
        JS App.js
        JS App.test.js
        # calculator.css
        # index.css
        JS index.js
        logo.svg
        JS serviceWorker.js
        .gitignore
    {} package-lock.json
    {} package.json
    README.md
    .gitignore
    {} package-lock.json
6 class Calculator extends Component {
7   state = {
8     result: ""
9   };
10  handleComputation = (expression) => {
11    const Expression = { "exp": expression };
12    API.doComputation(Expression)
13      .then((response) | any |
14        if (response.finalResult === null) {
15          this.setState({
16            result: "Infinity"
17          });
18        } else {
19          this.setState({
20            result: response.finalResult
21          });
22        });
23  };
24  render() {
25    return (
26      <div className="container-fluid">
27        <div className="row">
28          <div className="col-lg-4"> </div>
29          <div className="col-lg-4">
30            <div className="row box-project border border-dark rounded">
31              <legend className="legendCss">Calculator</legend>
32              <Display handleComputation={this.handleComputation}
33                result={this.state.result} />
34            </div>
35          </div>
36        </div>
37      </div>
38    );
39  }
40}

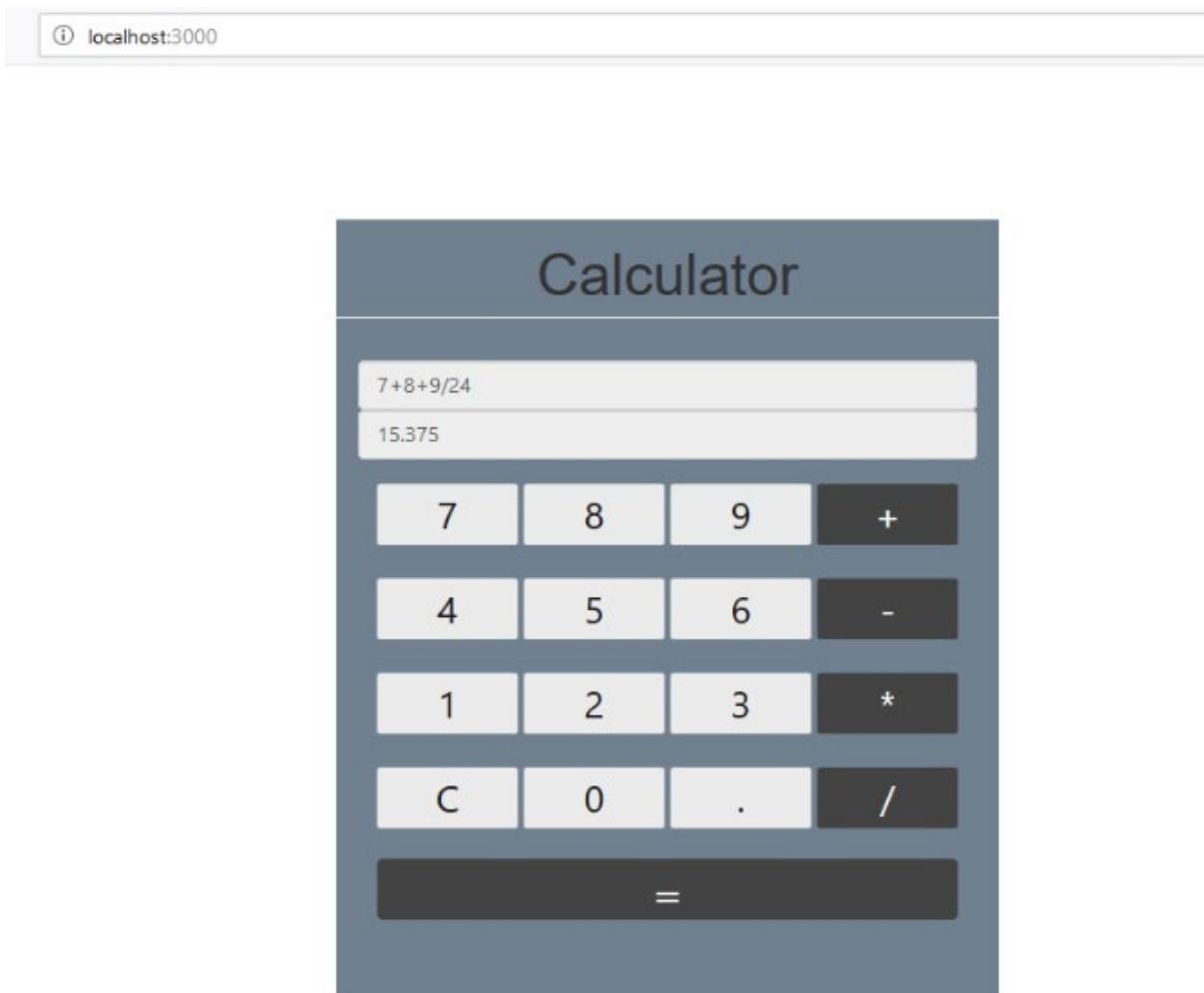
```

2. Computation API.



```
const api = process.env.REACT_APP_CONTACTS_API_URL || 'http://localhost:3005';
const headers = {
  'Accept': 'application/json'
};
export const doComputation = (payload) => fetch(`${api}/compute/doComputation`, {
  method: 'POST',
  headers: {
    ...headers,
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(payload)
}).then(res => {
  if (res.status === 200) {
    return res.json();
  } else {
    return { finalResult: "Syntax Error" };
  }
}).catch(error => {
  console.log("error");
  return error;
});
```

3. Calculator Display.



System outputs:

1. node.js server running on port 3005 and react app running on port 3000.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/frontend (master)
$ cd frontend

Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/frontend/frontend (ma
ster)
$ npm start

> frontend@0.1.0 start C:\Users\Rohan\Desktop\LAB1\calculator\frontend\fronte
nd
> react-scripts start
Starting the development server...
Compiled successfully!

You can now view frontend in the browser.

  Local:          http://localhost:3000/
  On Your Network:  http://10.0.0.201:3000/

Note that the development build is not optimized.

Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator (master)
$ cd backend

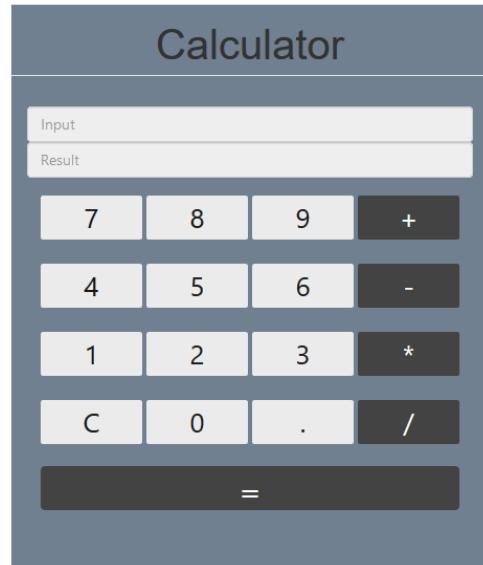
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/backend (master)
$ npm start

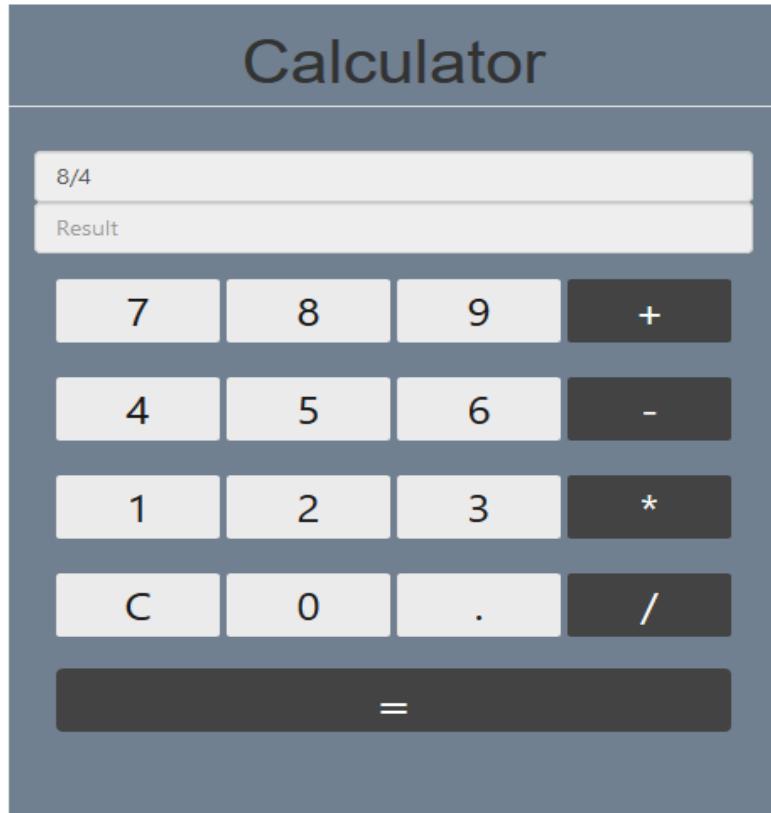
> backend@1.0.0 start C:\Users\Rohan\Desktop\LAB1\calculator\backend
> nodemon ./bin/www

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: ***!
[nodemon] starting `node ./bin/www'
listening on port 3005

```

2. React.js client



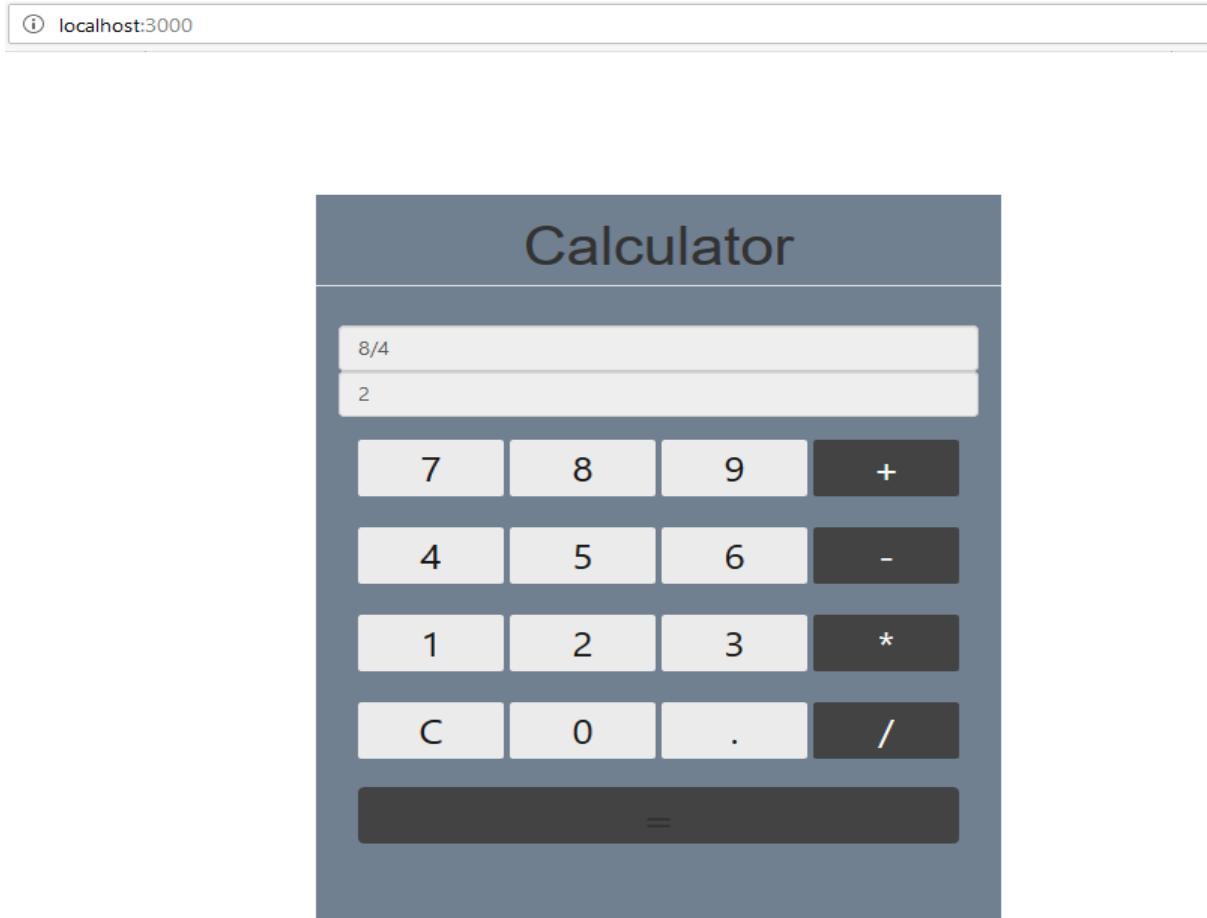
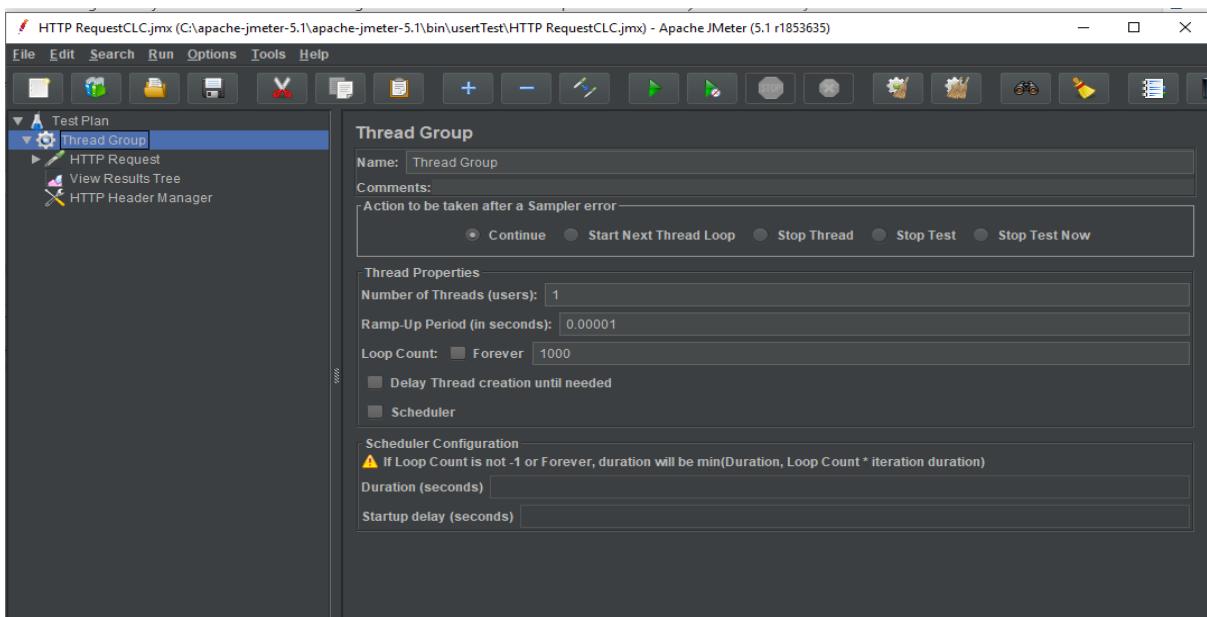
3. Input:**4. Request on server:**

```
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator (master)
$ cd backend

Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/backend (master)
$ npm start

> backend@1.0.0 start C:\Users\Rohan\Desktop\LAB1\calculator\backend
> nodemon ./bin/www

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: ***!
[nodemon] starting `node ./bin/www`
listening on port3005
7+9/2*6-4
POST /compute/doComputation 200 141.422 ms - 18
8/4
POST /compute/doComputation 200 1.443 ms - 17
```

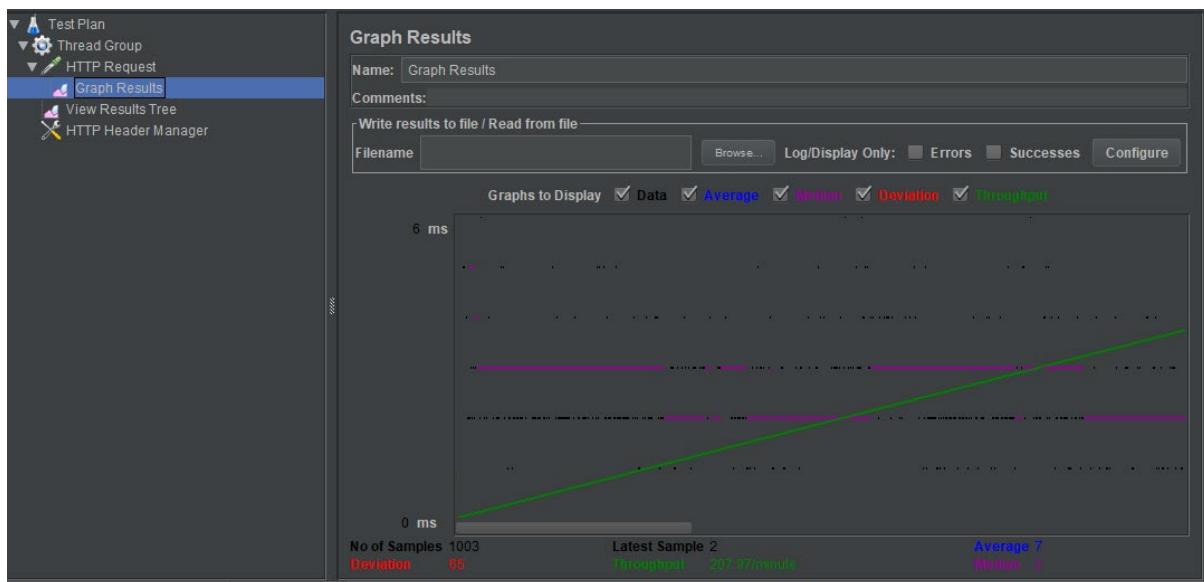
5. Output to client:**Testing Using Jmeter:****1. 1000 calculator calls.****Printed Results: 1000 calculator calls**

Printed results:

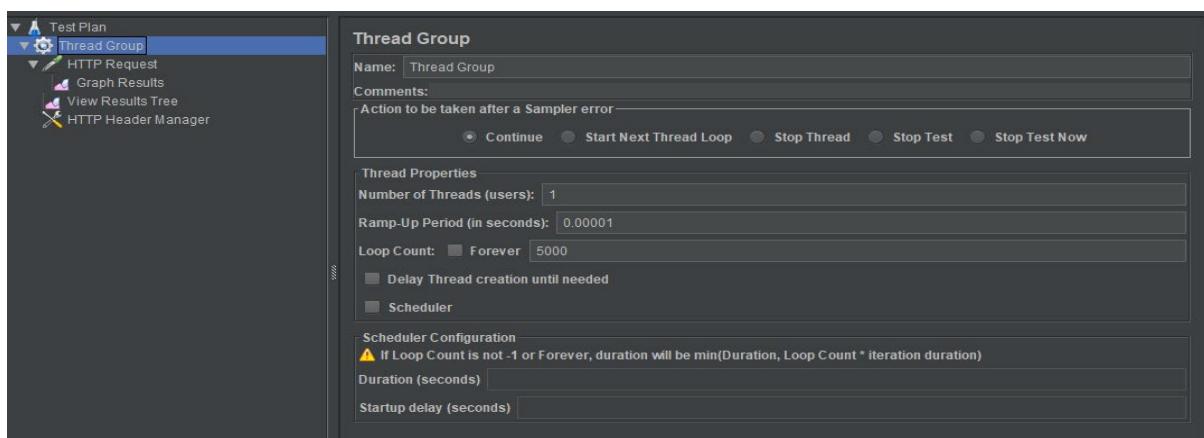
The screenshot shows the Apache JMeter interface and a terminal window. The JMeter window displays a 'View Results Tree' panel with a list of 15 'HTTP Request' entries, each marked with a green checkmark. The terminal window shows JMeter's summary results for a test run, including metrics like Average, Min, Max, and Error counts.

```
C:\WINDOWS\system32\cmd.exe
=====
Don't use GUI mode for load testing !, only for Test creation and Test debugging.
For load testing, use CLI Mode (was NON GUI):
jmeter -n -t [jmx file] -l [results file] -e -o [Path to web report folder]
& increase Java Heap to meet your test requirements:
  Modify current env variable HEAP="-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m" in the jmeter batch file
Check : https://jmeter.apache.org/usermanual/best-practices.html
=====
Warning: Nashorn engine is planned to be removed from a future JDK release
Generate Summary Results + 5 in 00:00:14 = 0.4/s Avg: 2006 Min: 2003 Max: 2014 Err: 5 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results + 10 in 00:00:31 = 0.3/s Avg: 2005 Min: 2003 Max: 2012 Err: 10 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 15 in 00:00:45 = 0.3/s Avg: 2006 Min: 2003 Max: 2014 Err: 15 (100.00%)
Generate Summary Results + 9 in 00:00:28 = 0.3/s Avg: 2005 Min: 2001 Max: 2013 Err: 9 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 24 in 00:01:12 = 0.3/s Avg: 2005 Min: 2001 Max: 2014 Err: 24 (100.00%)
Generate Summary Results + 10 in 00:00:31 = 0.3/s Avg: 2003 Min: 2001 Max: 2006 Err: 10 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 34 in 00:01:43 = 0.3/s Avg: 2005 Min: 2001 Max: 2014 Err: 34 (100.00%)
Generate Summary Results + 10 in 00:00:31 = 0.3/s Avg: 2003 Min: 2002 Max: 2006 Err: 10 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 44 in 00:02:14 = 0.3/s Avg: 2004 Min: 2001 Max: 2014 Err: 44 (100.00%)
Generate Summary Results + 10 in 00:00:31 = 0.3/s Avg: 2003 Min: 2001 Max: 2006 Err: 10 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 54 in 00:02:44 = 0.3/s Avg: 2004 Min: 2001 Max: 2014 Err: 54 (100.00%)
Generate Summary Results + 10 in 00:00:30 = 0.3/s Avg: 2004 Min: 2002 Max: 2011 Err: 10 (100.00%) Active: 1
Started: 1 Finished: 0
Generate Summary Results = 64 in 00:03:15 = 0.3/s Avg: 2004 Min: 2001 Max: 2014 Err: 64 (100.00%)
Generate Summary Results + 9 in 00:00:27 = 0.3/s Avg: 2003 Min: 2002 Max: 2007 Err: 9 (100.00%) Active: 1
```

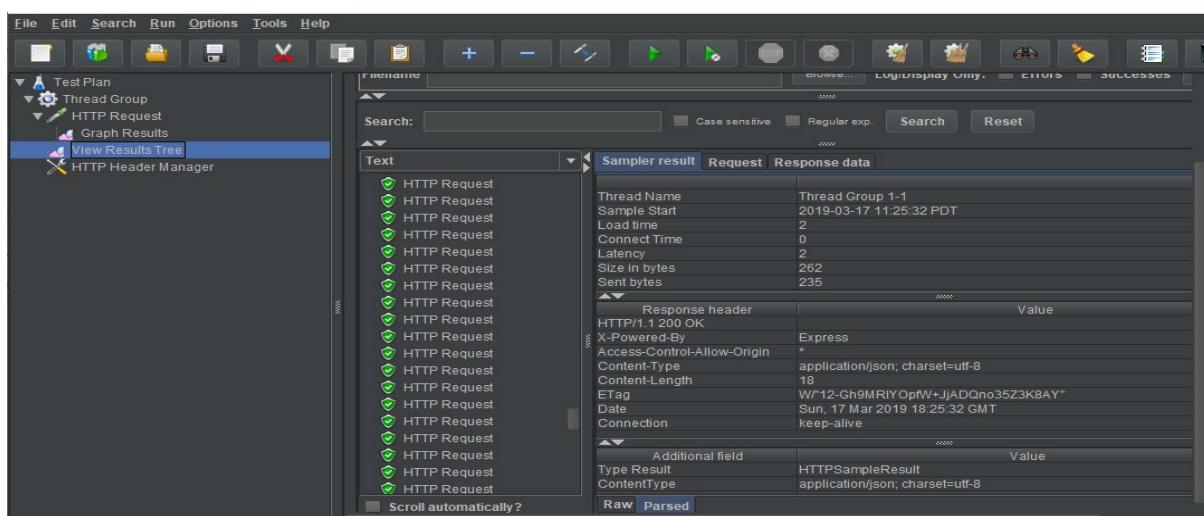
Graph result:



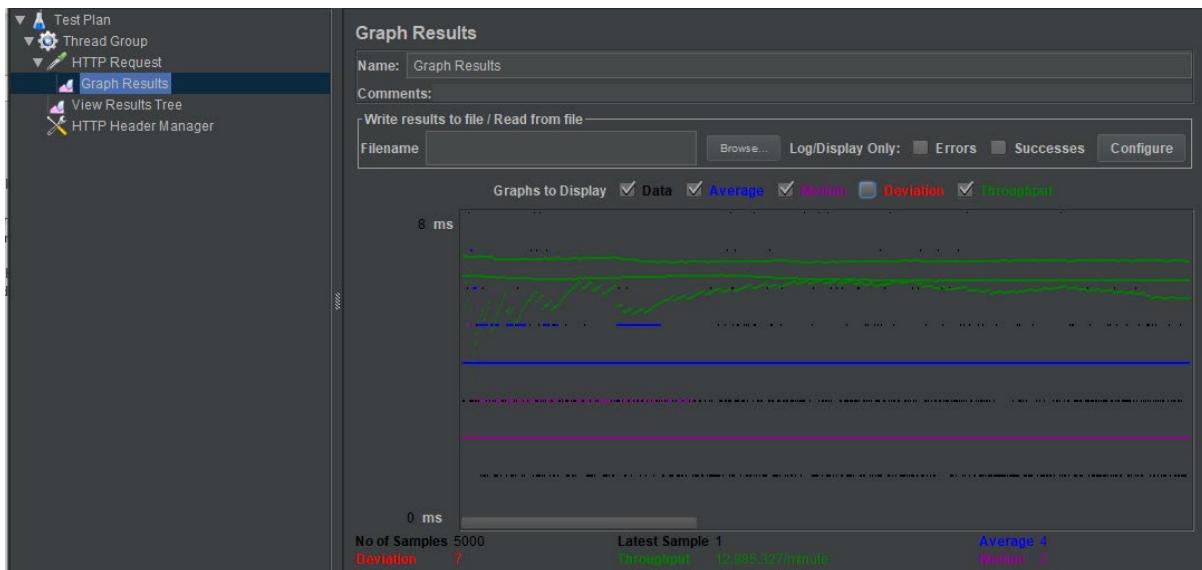
Test 2: 5000 calculator calls



Printed average time



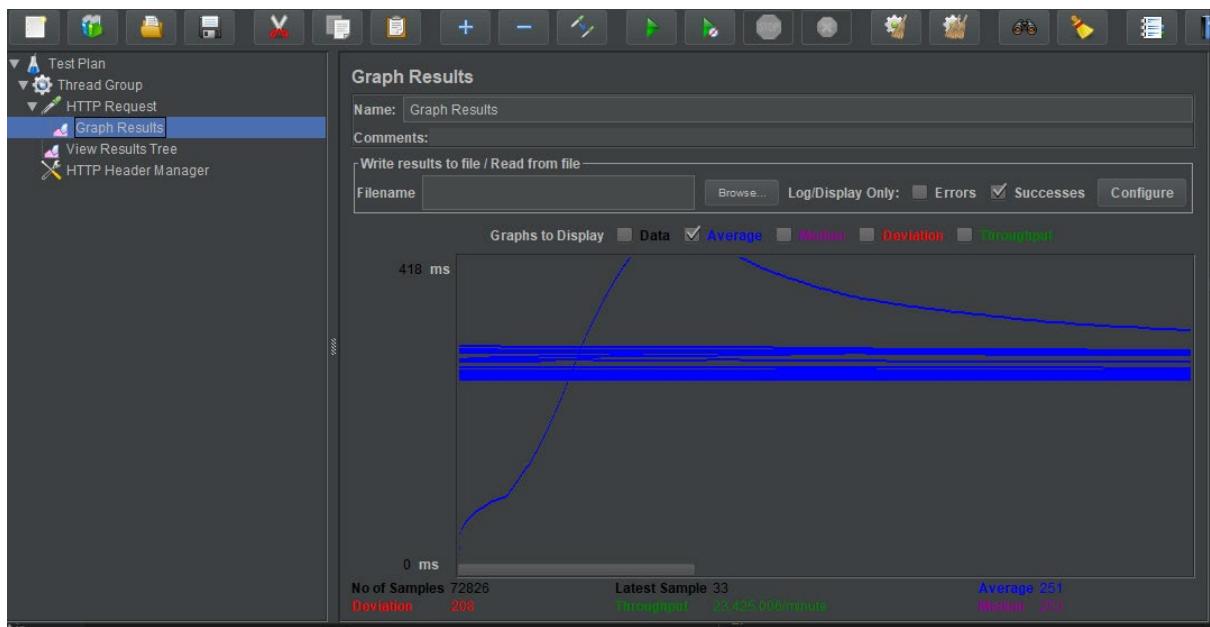
Graph Results:



Test:3 (100 users with 1000 calls each)

The screenshot shows the JMeter interface with two main panels. The top panel displays the Thread Group configuration for a test plan. The 'Number of Threads (users)' is set to 100, 'Ramp-Up Period (in seconds)' is 0.00001, and 'Loop Count' is set to 'Forever'. The bottom panel shows the 'View Results Tree' results for the same test plan. The results table lists 1000 'HTTP Request' entries, indicating 1000 calls made by each of the 100 threads. The table includes columns for Thread Name, Sampler result, Request, Response data, Response header, and Additional field.

Graph results:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator (master)
$ cd frontend
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/frontend (master)
$ cd frontend
Rohan@LAPTOP-99SVDF75 MINGW64 ~/Desktop/LAB1/calculator/frontend/frontend (master)
$ npm start
> frontend@0.1.0 start C:\Users\Rohan\Desktop\LAB1\calculator\frontend\frontend
> react-scripts start
Starting the development server...
Compiled successfully!
You can now view frontend in the browser.

1: bash, bash
{ exp: '8+9' }
17
POST /compute/doComputation 200 5.587 ms - 18
{ exp: '8+9' }
17
POST /compute/doComputation 200 0.724 ms - 18
{ exp: '8+9' }
17
POST /compute/doComputation 200 0.668 ms - 18
{ exp: '8+9' }
17
POST /compute/doComputation 200 25.554 ms - 18
{ exp: '8+9' }
17
POST /compute/doComputation 200 0.524 ms - 18
{ exp: '8+9' }
17
POST /compute/doComputation 200 6.310 ms - 18

```

Comparison of server performance:

Graph results shows that server can easily handle 1000 user calls synchronously.

Whereas it takes more time to perform 5000 calls and graph results vary because of the variations in throughput.

For performing 100 concurrent users with 1000 API calls each, the server gets so many deviations and the average throughput time is increased. This is because of scalability and weak asynchronous request handling.

Performance can be improved by giving more memory and connection pooling to server that can reduce throughput and content.

Part-2: Prototype of SJSU Canvas Application using MERN (MySQL, Express.js, React.js, Node.js) .

Purpose:

Developing a full stack single page application (SPA) to have proper understanding of the concept of MERN application development. By developing this application developer will be able to understand react and node environment and the structure of MVC and interaction with MySQL database.

Goal:

Design a single page web application of SJSU Canvas (a student portal) using MySQL as a model database, Express.js and node.js as server(controller) and React.js (view) as a client user interface.

Primary Structure:

1. Node.js server for handling server request
2. MySQL schema for storing data of application
3. React client user interface for application view on client side.

Users of application:

1. faculty (with admin rights)
2. Student

Functional Requirements:

1. RESTful service on server side:

Server should be able to perform below tasks.

1. Basic user functionalities

1. Sign up new user (Name, Email and password)
2. Sign in existing user
3. Sign out.
4. Profile (Profile Image, Name, Email, Phone Number, About Me, City, Country, Company, School, Hometown, Languages, Gender)
5. Users can update Profile anytime.

2. Students

1. Student should be able to search for all the courses by term, by id or by course name. Should have filter like id greater than a value.
2. Student should be able to enroll for a course, drop a course and waitlist a course when full.

3. Faculty:

1. Faculty should be able to create a course with following fields
 - a. CourseId
 - b. CourseName
 - c. Course Dept
 - d. CourseDescription
 - e. CourseRoom
 - f. CourseCapacity
 - g. Waitlist capacity
 - h. courseTerm
2. Faculty should be able to give permission codes for the waitlisted students.

4. Course Details:**Student:**

Student can view his grades in the course.

Student should be able to submit his assignment.

Student should be able to view all his submissions per assignment.

Student should be able to take quiz.

Student should be able to view all announcements

Student should be able to view all the people under that course.

Student should be able to view and download lecture notes and files

Faculty:

Faculty should be able to create Assignments and Quizzes.

Faculty should be able to view and download submissions from students.

Faculty should be able to make announcements

Faculty should be able to view all students registered for that course.

Faculty should be able to grade assignment submitted by students.

Faculty should be able to upload lecture notes and files

Faculty should be able to remove a student from the course.

5. Home

1. Student should be able to view all the courses he/she has registered.

2. Faculty should be able to view all the courses created by them.

6. important Note:

User must login to have access to application. and password must be encrypted.

Should perform connection pooling (in-built in MySQL) for database access.

Proposed System Design:



Implementation:

1. Database Structure:

Schema: Canvas

Data Dictionary:

announcemnet

Column	Type	Null	Default	Comments
announcemnetId <i>(Primary)</i>	int(255)	No		
announcementTitle	varchar(255)	Yes	NULL	
courseId	int(255)	Yes	NULL	
announcemnetDetail	varchar(10000)	Yes	NULL	
userId	int(255)	No		
announcemnetDate	date	No		

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	announcemnetId	0	A	No	
courseId	BTREE	No	No	courseId	0	A	Yes	
userId	BTREE	No	No	userId	0	A	No	

assignment

Column	Type	Null	Default	Comments
assignmentId <i>(Primary)</i>	int(255)	No		
assignmentTitle	varchar(255)	Yes	NULL	
assignmentDetail	varchar(1000)	Yes	NULL	
courseId	int(255)	No		
assignmentType	varchar(255)	No	assignment	
points	int(255)	Yes	NULL	
dueDate	varchar(255)	Yes	NULL	

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	assignmentId	3	A	No	
courseld	BTREE	No	No	courseId	1	A	No	

course

Column	Type	Null	Default	Comments
courseId (<i>Primary</i>)	int(45)	No		
courseDept	varchar(255)	Yes	NULL	
courseDescription	varchar(255)	Yes	NULL	
courseRoom	varchar(255)	Yes	NULL	
courseCapacity	int(255)	Yes	NULL	
waitlistCapacity	int(255)	Yes	NULL	
courseTerm	varchar(255)	Yes	NULL	
lectureTime	datetime(5)	Yes	NULL	
courseName	varchar(255)	Yes	NULL	

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	courseld	2	A	No	

enrollment

Column	Type	Null	Default	Comments
enrollmentId (<i>Primary</i>)	int(255)	No		
userId	int(255)	No		
courseld	int(255)	No		
type	varchar(255)	No	student	

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	enrollmentId	2	A	No	
courseld	BTREE	No	No	courseId	1	A	No	

grades

Column	Type	Null	Default	Comments
userId	int(255)	No		
assignmentId	int(255)	No		
score	int(255)	Yes	0	
courseld	int(255)	No		

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
studentuser	BTREE	No	No	userId	0	A	No	
courseConstraint	BTREE	No	No	courseId	0	A	No	
assignmentIdConstraint	BTREE	No	No	assignmentId	0	A	No	

permissonenumber

Column	Type	Null	Default	Comments
courseId	int(11)	No		
permissonenumber	int(11)	No		

submission

Column	Type	Null	Default	Comments
userId	int(255)	No		
assignmentId	int(255)	No		
grades	decimal(65,2)	No	0.00	
courseId	int(255)	No		
submissionDetails	varchar(1000)	Yes	NULL	
submissionDate	varchar(1000)	No		

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
userId	BTREE	No	No	userId	1	A	No	
staticCourseId	BTREE	No	No	courseId	1	A	No	
assignmentIdConstraintSubmission	BTREE	No	No	assignmentId	2	A	No	

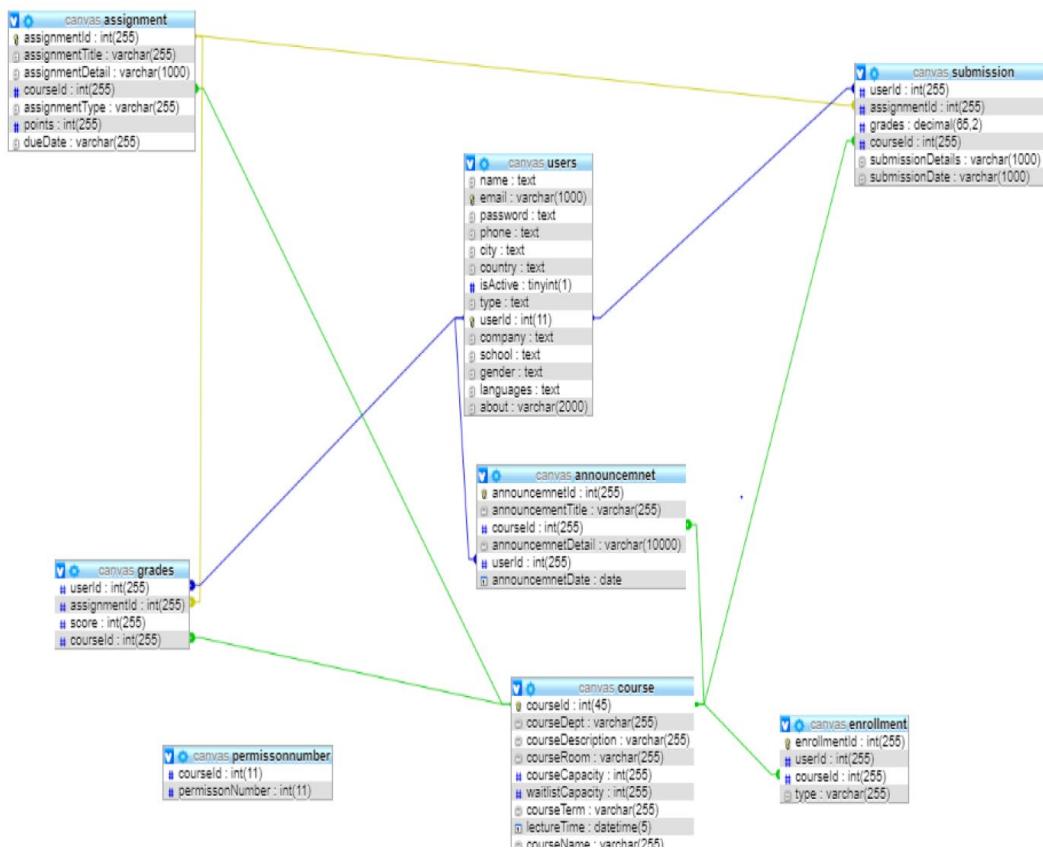
users

Column	Type	Null	Default	Comments
name	text	Yes	NULL	
email	varchar(1000)	No		
password	text	Yes	NULL	
phone	text	Yes	NULL	
city	text	Yes	NULL	
country	text	Yes	NULL	
isActive	tinyint(1)	Yes	NULL	
type	text	Yes	NULL	
userId <i>(Primary)</i>	int(11)	No		
company	text	Yes	NULL	
school	text	Yes	NULL	
gender	text	Yes	NULL	
languages	text	Yes	NULL	
about	varchar(2000)	Yes	NULL	

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	userId	0	A	No	
userId	BTREE	Yes	No	userId	0	A	No	
email	BTREE	Yes	No	email	0	A	No	

Schema Structure:



2. RESTfull Services node.js server:

1. index.js

```

js index.js ✘
1  var express = require("express")
2  var cors = require("cors")
3  var bodyParser = require("body-parser")
4  var app = express()
5  var methodOverride = require('express-method-override')
6  var port = process.env.PORT || 3001
7  app.use(methodOverride('_method'))
8  app.use(bodyParser.json())
9  app.use(cors())
10 app.use(bodyParser.urlencoded({ extended: false }))
11
12 var Users = require('./routes/Users')
13 var courses = require('./routes/courses')
14 var permissionNumber = require('./routes/permissionNumber')
15 var enrollment = require('./routes/enrollment')
16 var grades = require('./routes/grades')
17 app.use('/users', Users)
18 app.use('/courses', courses)
19 app.use('/permissionNumber', permissionNumber)
20 app.use('/enrollment', enrollment)
21 app.use('/users/courses/grades', grades)
22 app.listen(port, () => {
23   console.log("Server is running on port: " + port)
24 })
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: AdobeCollabSync ▾

```

Server is running on port: 3001
connected
  
```

2. Users Signup:

Create a signup form which will take input from client and add the user in database if not already exists.

This will contain basic user details like Name, Email, password, type etc..

It encrypts the password using bcrypt and generates a Hash value and stores the Hash value in database.

```
1  const express = require("express")
2  const users = express.Router()
3  const bcrypt = require('bcrypt')
4  const mysql = require('mysql')
5  var con = require("../database/db")
6
7  users.post('/signup', (req, res) => {
8      bcrypt.hash(req.body.password, 10, (err, hash) => {
9          const sql = "INSERT INTO users (name,email,password,type)VALUES(" +
10              mysql.escape(req.body.name) + "," +
11              mysql.escape(req.body.email) + "," +
12              mysql.escape(hash) + "," +
13              mysql.escape(req.body.type) +
14              ")";
15          con.query(sql, function (err, result) {
16              if (result) {
17                  res.send({
18                      status: "user registered"
19                  })
20              } else {
21                  res.status(400).send({
22                      "message": err.sqlMessage
23                  });
24              }
25          })
26      })
27  })
```

3. Login:

When users enter the site address it will directly land on the login page where he will enter a username and password.

On entering user credentials, server will generate a hash value of entered password and match it with the saved hash value of the password in database.

If the credentials match, then user will land on his home page or Dashboard page where he will see all his registered courses.

Use of Json-web-tokens: for making our session strategy horizontal I have used json web tokens which will generate a web token when user signs in and store that value in its local storage. Now every time when a user sends any request, it will have a JWT with it. The server matches the token with its own token if both values match then only server will respond. This way the session can be maintained and user authorization is also possible.

```

50  users.post("/login", (req, res) => {
51
52      var email = req.body.email;
53      var password = req.body.password;
54      con.query('SELECT * FROM users WHERE email = ?', [email], function (error, results, fields) {
55          if (error) {
56              res.json({
57                  status: false,
58                  message: 'there are some error with query'
59              })
60          } else {
61              if (results.length > 0) {
62                  bcrypt.compare(password, results[0].password, function (err, ress) {
63                      if (!ress) {
64                          res.json({
65                              status: false,
66                              message: "Email and password does not match"
67                          });
68                      } else {
69                          res.json({
70                              status: true,
71                              message: "Successfully Login"
72                          })
73                      }
74                  });
75              } else {
76                  res.json({
77                      status: false,
78                      message: "Email does not exists"
79                  });
80              }
81          }
82      })
83  })

```

4. User profile:

User can update his profile at any time by going to profile page and update the details button.

This is a very basic details form on which the post request will be made, and user will be able to update his details accordingly.

User can not update his password from this page.

```

users.post('/profile', (req, res) => {
    let sql = "UPDATE users SET name=" + mysql.escape(req.body.name) +
        ",phone=" + mysql.escape(req.body.phone) +
        ",city=" + mysql.escape(req.body.city) +
        ",country=" + mysql.escape(req.body.country) +
        ",school=" + mysql.escape(req.body.school) +
        ",company=" + mysql.escape(req.body.company) +
        ",languages=" + mysql.escape(req.body.languages) +
        ",about=" + mysql.escape(req.body.about) +
        " WHERE email=" + mysql.escape(req.body.email);
    con.query(sql, function (err, result) {
        if (result) {
            res.send({
                message: "profile updated"
            })
        } else {
            console.log(err)
        }
    })
})

```

Dashboard:

Student will be able to see all the courses he has enrolled for and faculty will be able to see all the courses he/she has created on dashboard.

```
users.get("/courses", (req, res) => {
  const sql = "SELECT * from enrollment e,course c WHERE e.userId=" + mysql.escape(req.body.userId) + " AND e.courseId=c.courseId";
  con.query(sql, (err, results) => {
    if (results) {
      res.status(200).json(results)
      console.log(results)
    } else {
      res.status(400).json({ message: err })
    }
  })
}

module.exports = users
```

Student can see all the available courses:

```
JS courses.js  ✘
1  const express = require("express")
2  const route = express.Router()
3  const bcrypt = require('bcrypt')
4  const mysql = require('mysql')
5  var con = require("../database/db")
6
7  route.get('/', function (req, res) {
8
9    const sql = "SELECT * FROM course"
10
11    con.query(sql, function (err, result) {
12      if (result) {
13        res.status(200).json(result)
14        console.log("result")
15      } else {
16        res.status(400).send({ message: err })
17        console.log(err)
18      }
19    })
20  })
21
22})
```

Enrollment:

Student can enroll in desired course from the list and remove his self from the list.

```
JS enrollment.js  ✘
21  route.post("/", (req, res) => {
22    const sql = "select * from enrollment where userId=" + mysql.escape(req.body.userId) + "and courseId=" + mysql.escape(req.body.courseId);
23    con.query(sql, (err, result) => {
24      if (!result.length == 0) {
25        res.json({ message: "already exists" })
26      } else {
27
28        const sql = "INSERT INTO enrollment(courseId,userId) values(" + mysql.escape(req.body.courseId) + "," + (req.body.userId) + ")";
29        con.query(sql, (err, result) => {
30          if (err) {
31            res.json({ mesasge: err })
32          } else {
33            res.json({ message: "user added" })
34          }
35        })
36      }
37    })
38  })
39})
```

Enrolled students:

Student and faculty can see all the registered people for that course in the enrolled students tab.

```
JS enrollment.js ×
1  const express = require("express")
2  const route = express.Router()
3  const bcrypt = require('bcrypt')
4  const mysql = require('mysql')
5  var con = require("../database/db")
6
7  route.get("/", (req, res) => {
8
9      const sql = "SELECT * from enrollment e,users u WHERE e.courseId=" + mysql.escape(req.body.courseId) + " AND e.userId=u.userId";
10
11     con.query(sql, (err, result) => {
12         if (err) {
13             res.status(400).send({ message: err })
14         } else {
15             res.status(200).json(result);
16         }
17     })
18 })
19 }
```

Remove enrollment:

Faculty can remove student from the course and student can remove his self from the course as and when needed.

```
JS enrollment.js ×
41 route.delete("/", (req, res) => {
42     const sql = "delete from enrollment where userId=" + mysql.escape(req.body.userId) + " and courseId=" + mysql.escape(req.body.courseId);
43     con.query(sql, (err, result) => {
44         if (err) {
45             res.status(400).send(err)
46         }
47         else {
48             res.status(400).json({ mesasge: "removed from course" })
49         }
50     })
51 })
```

Announcements:

Faculty must be able to create announcements under any course.

```
JS permissionNumber.js   JS announcements.js ×
18
19 route.post("/announcements", function (req, res) {
20     const today = new Date();
21     const sql = "INSERT INTO announcement (announcementTitle,courseId,announcemnetDetail,userId,announcemnetDate) VALUES (" +
22         mysql.escape(req.body.courseId) + "," +
23         mysql.escape(req.body.announcementDetail) + "," +
24         mysql.escape(req.body.userId) + "," + today + ")";
25     con.query(sql, (err, result) => {
26         if (err) {
27             res.status(400).send({ message: err })
28         } else {
29             res.status(200).send({ message: "announcement created" })
30         }
31     })
32 })
33
34 module.exports = route;
```

View announcements:

Students and faculty must be able to see all the announcement made under course.

```

7   route.get("/announcements", function (req, res) {
8
9     const sql = "SELECT * FROM announcement WHERE courseId=" + mysql.escape(req.body.courseId);
10    con.query(sql, (err, result) => {
11      if (result) {
12        res.status(200).json(result)
13      } else {
14        res.status(400).send({ message: err })
15      }
16    })
17  })
18

```

Permission Number:

Faculty must be able to generate random and unique permission number for every course based on the waitlist student capacity.

```

js permissonNumber.js ✘
5  > route.get("/", (req, res) => {
6
7   const sql = "SELECT waitlistCapacity FROM course WHERE courseId=" + mysql.escape(req.body.courseId);
8   con.query(sql, (err, result) => {
9     if (result) {
10       if (result[0].waitlistCapacity) {
11         const waitlist = result[0].waitlistCapacity;
12         for (i = 0; i < waitlist; i++) {
13           let r = Math.floor(Math.random() * 100) + 1;
14           const sql1 = "INSERT INTO permissonNumber(courseId,permissonNumber) VALUES (" +
15             + mysql.escape(req.body.courseId) + "," + mysql.escape(r) + ")";
16
17           con.query(sql1, (err, result1) => {
18             if (result1) {
19               if (result1[0].permissonNumber) {
20                 res.status(200).json(result1[0])
21               } else {
22                 res.status(400).send(err)
23               }
24             })
25           }
26         }
27       }
28     }
29   } else {
30     res.status(400).send({ message: err })
31   }
32 }
33 })
34 module.exports = route

```

Show Assignments:

User must be able to see all the assignments posted under the course

```

route.get('/assignments', function (req, res) {
  const sql = "SELECT * FROM assignment WHERE courseId=" + mysql.escape(req.body.courseId) + " AND assignmentType='assignment'";
  con.query(sql, (err, result) => {
    if (result) {
      res.status(200).json(result)
    } else {
      res.status(400).json({ message: err })
    }
  })
}

```

Create an Assignment:

Faculty must be able to create a new assignments and add the values like assignment title, assignment type, assignment details, due date, total points etc.

```
route.post('/assignments', function (req, res) {
  const sql = "INSERT INTO assignment (assignmentTitle,assignmentDetail,courseId,assignmentType,points,dueDate) VALUES (" +
    mysql.escape(req.body.assignmentTitle) + "," +
    mysql.escape(req.body.assignmentDetail) + "," +
    mysql.escape(req.body.courseId) + "," +
    mysql.escape(req.body.assignmentType) + "," +
    mysql.escape(req.body.points) + "," +
    mysql.escape(req.body.dueDate) +
  ")";
  con.query(sql, (err, result) => {
    if (result) {
      res.send({ status: "assignment added" })
    } else {
      res.status(400).send({ message: err.sqlMessage })
    }
  })
}

module.exports = route;
```

Submission:

Student must be able to submit his assignments by uploading file or any suggested method given in assignment details. This should contain details like assignment Id, student Id, submission date etc.

```
const express = require("express")
const route = express.Router()
const fileUpload = require('express-fileupload')
route.use(fileUpload())
route.use('/public', express.static(__dirname + '/public'))
route.post('/file', (req, res, next) => {
  let uploadFile = req.files.file
  const fileName = req.files.file.name
  uploadFile.mv(
    `${__dirname}/public/files/${fileName}`,
    function (err) {
      if (err) {
        return res.status(500).send(err)
      }

      res.json({
        file: `public/${req.files.file.name}`,
      })
    },
  )
}

module.exports = route;
```

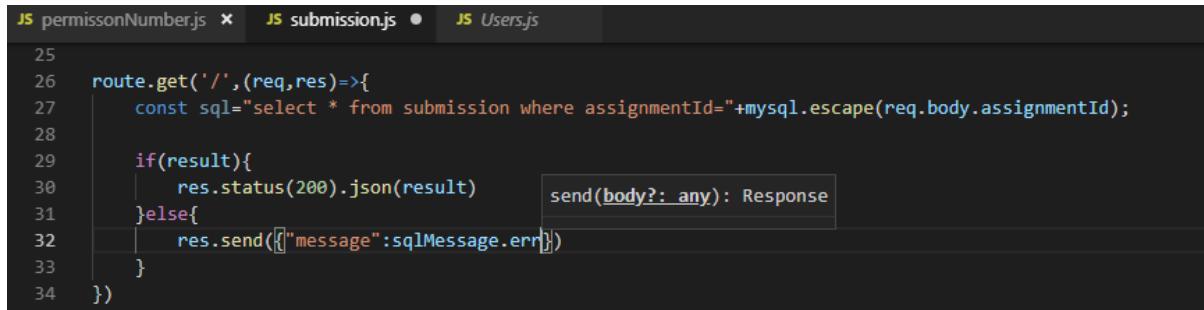
```

1  const express = require("express")
2  const route = express.Router()
3  const bcrypt = require('bcrypt')
4  const mysql = require('mysql')
5  var con = require("../database/db")
6
7  route.post('/', (req, res) => {
8      const date = new Date();
9      const sql = "insert into submission(userId,courseId,assignmentId,submissionDetails,submissionDate)Values(" +
10          mysql.escape(req.body.userId) + "," +
11          mysql.escape(req.body.courseId) + "," +
12          mysql.escape(req.body.assignmentId) + "," +
13          mysql.escape(req.body.submissionDetails) + "," +
14          mysql.escape(date) + ")";
15
16      con.query(sql, (err, result) => {
17          if (result) {
18              res.send({ status: "submitted successfully" })
19          } else {
20              res.status(400).send({ "message": err.sqlMessage })
21          }
22      })
23  })

```

View submissions:

Faculty must be able to see all the submission done by students by assignment Id .And student should be able to see all his submissions.



```

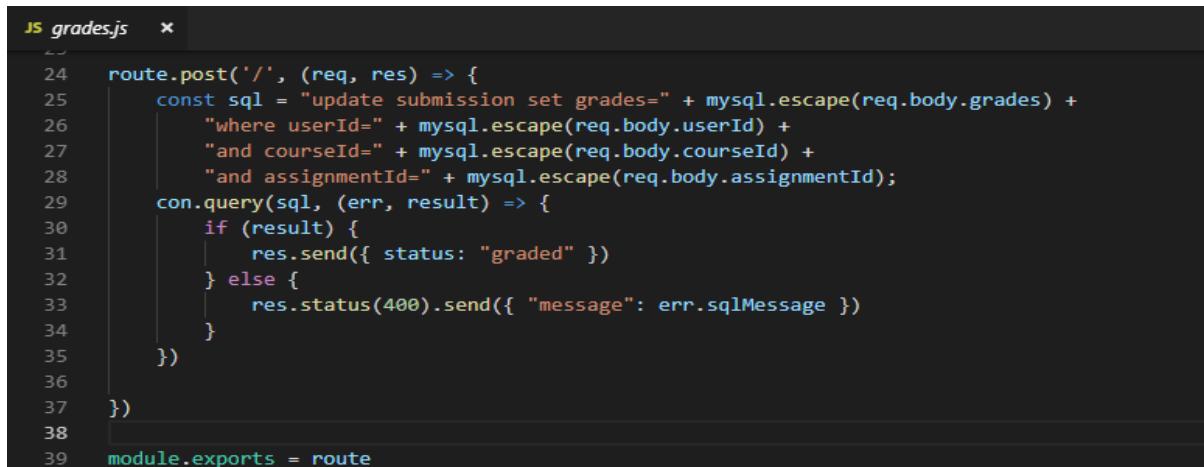
JS permissionNumber.js x JS submission.js ● JS Users.js
25
26 route.get('/',(req,res)=>{
27     const sql="select * from submission where assignmentId="+mysql.escape(req.body.assignmentId);
28
29     if(result){
30         res.status(200).json(result)
31     }else{
32         res.send([{"message":sqlMessage,err}])
33     }
34 })

```

Grades:

Grading assignment:

Faculty must be able to grade an assignment submitted by students.



```

JS grades.js x
24  route.post('/', (req, res) => {
25      const sql = "update submission set grades=" + mysql.escape(req.body.grades) +
26          "where userId=" + mysql.escape(req.body.userId) +
27          "and courseId=" + mysql.escape(req.body.courseId) +
28          "and assignmentId=" + mysql.escape(req.body.assignmentId);
29      con.query(sql, (err, result) => {
30          if (result) {
31              res.send({ status: "graded" })
32          } else {
33              res.status(400).send({ "message": err.sqlMessage })
34          }
35      })
36  })
37 })
38
39 module.exports = route

```

View Grades:

Student must be able to see his grades in grades Tab where he can see all his submitted assignments and its grades.

```
JS grades.js ×
1  const express = require("express")
2  const route = express.Router()
3  const mysql = require('mysql')
4  var con = require("../database/db")
5  //all grades for one course and student
6  route.get('/', function (req, res) {
7
8      const sql = "SELECT * FROM submission s where s.userId=" + mysql.escape(req.body.userId) +
9      " and s.courseId =" + mysql.escape(req.body.courseId);
10
11     con.query(sql, function (err, result) {
12         if (result) {
13             res.status(200).json(result)
14             console.log("result")
15         }
16         else {
17             res.status(400).send({ message: err })
18             console.log(err)
19         }
20     })
21
22 })
```

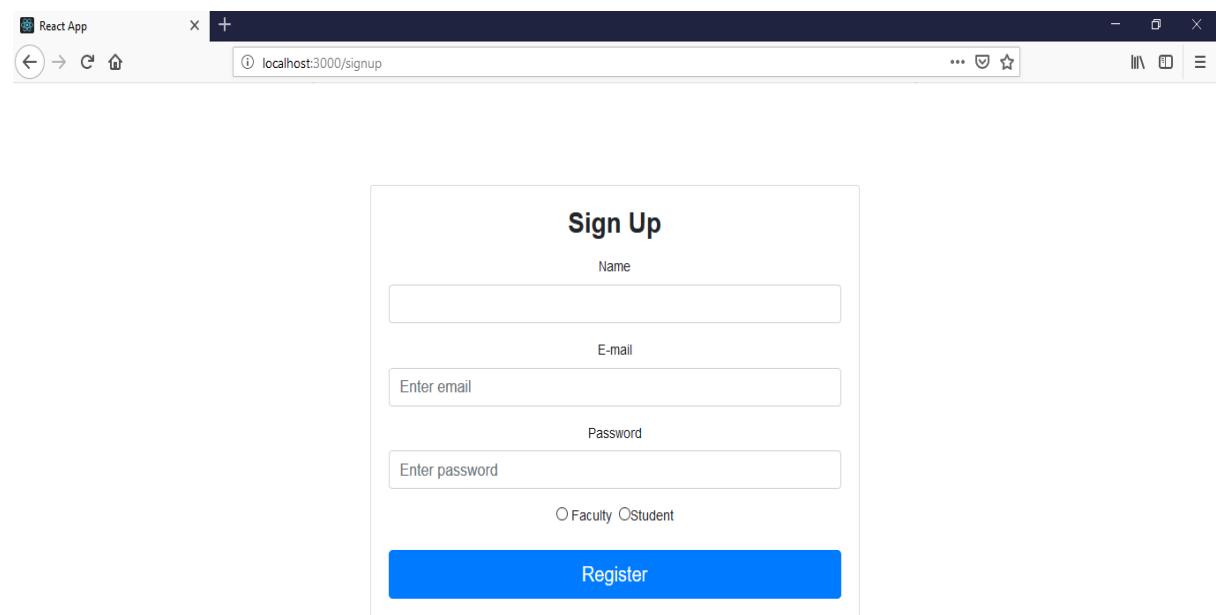
Connection pooling:

```
JS grades.js JS db.js ×
29 });
30 |
31 var db = mysql.createPool({
32   connectionLimit: 100,
33   host: 'localhost',
34   user: 'root',
35   password: '',
36   database: 'canvas'
37 })
38
39 db.connect(function (err) {
40   if (err) throw err;
41   console.log("connected");
42 })
43 module.exports = db;
```

Encryption:

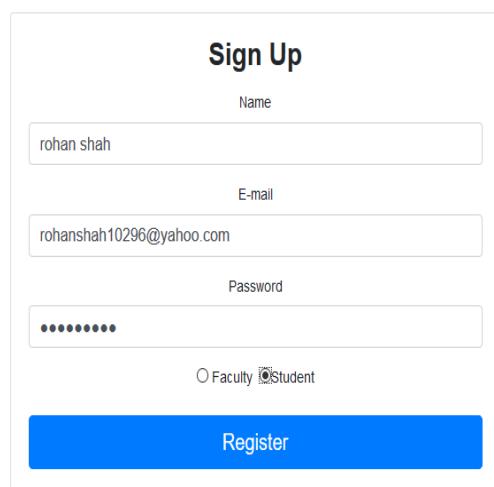
```
const express = require("express")
const users = express.Router()
const bcrypt = require('bcrypt')
const mysql = require('mysql')
var con = require("../database/db")

users.post('/signup', (req, res) => {
    bcrypt.hash((req.body.password), 10, (err, hash) => {
        const sql = "INSERT INTO users (name,email,password,type)VALUES(" +
            mysql.escape(req.body.name) + "," +
            mysql.escape(req.body.email) + "," +
            mysql.escape(hash) + "," +
            mysql.escape(req.body.type) +
            ")";
    })
})
```

React Client UI:

The screenshot shows a 'Sign Up' form in a browser window titled 'React App'. The URL in the address bar is 'localhost:3000/signup'. The form consists of the following fields:

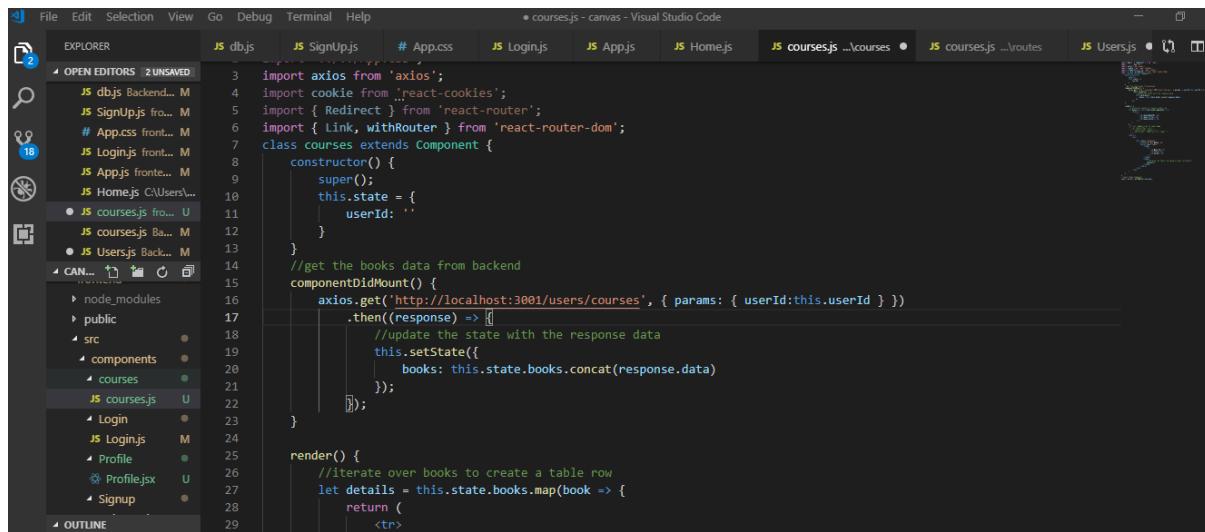
- Name: An input field containing the placeholder 'Name'.
- E-mail: An input field containing the placeholder 'Enter email'.
- Password: An input field containing the placeholder 'Enter password'.
- Role Selection: A radio button labeled 'Faculty' and another labeled 'Student'.
- Register Button: A large blue button with the text 'Register'.



The screenshot shows the same 'Sign Up' form as before, but with the 'Name' field populated with the value 'rohan shah'. The other fields and the 'Register' button remain the same.

The screenshot shows a web browser window with the URL `localhost:3000/login` in the address bar. The page title is "Connecting to one.SJSU Spartan App Portal". The main content is a "Sign In" form. It contains two input fields: "Enter email" with the value `rohanshah10296@yahoo.com` and "Enter password" with the value `*****`. Below the password field is a blue "Sign In" button. At the bottom of the form is a link "New user? signup here".

This screenshot is identical to the one above, showing the same web browser interface and "Sign In" form. The "email" field now contains the value `rohanshah10296@yahoo.com` and the "password" field contains the value `*****`.



```

File Edit Selection View Go Debug Terminal Help
OPEN EDITORS 2 UNSAVED
EXPLORER JS db.js JS SignUp.js # App.css JS Login.js JS App.js JS Home.js JS courses.js ...courses JS courses.js ...routes JS Users.js
3 import axios from 'axios';
4 import cookie from 'react-cookies';
5 import { Redirect } from 'react-router';
6 import { Link, withRouter } from 'react-router-dom';
7 class courses extends Component {
8     constructor() {
9         super();
10        this.state = {
11            userId: ''
12        }
13    }
14    //get the books data from backend
15    componentDidMount() {
16        axios.get('http://localhost:3001/users/courses', { params: { userId: this.userId } })
17        .then((response) => [
18            //update the state with the response data
19            this.setState({
20                books: this.state.books.concat(response.data)
21            });
22        ]);
23    }
24
25    render() {
26        //iterate over books to create a table row
27        let details = this.state.books.map(book => {
28            return (
29                <tr>

```

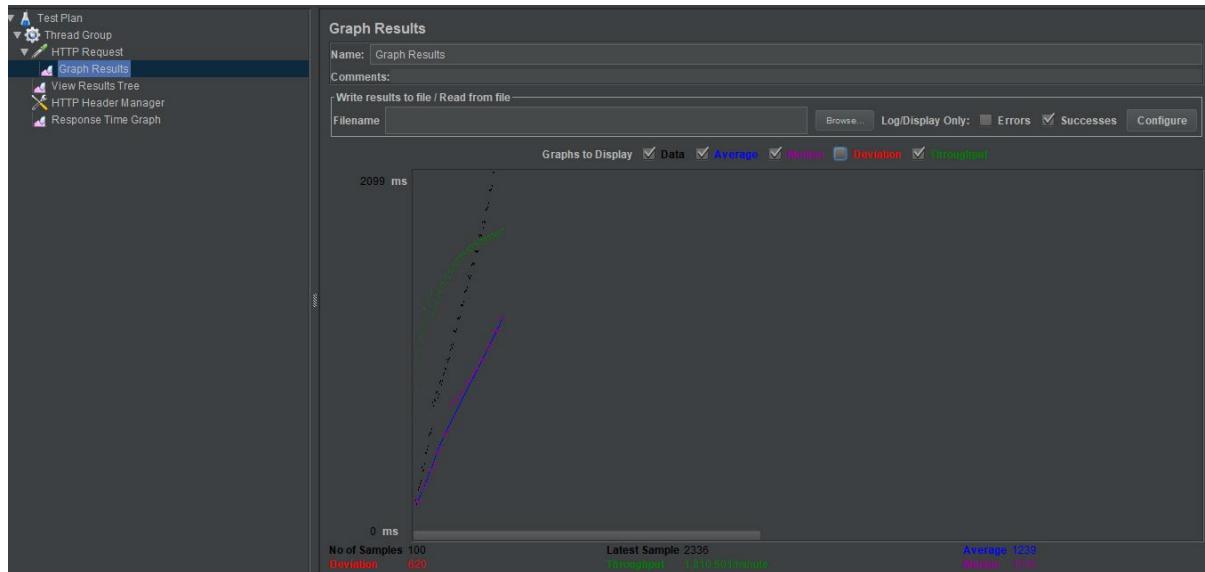
TESTING:

JMETER Testing.

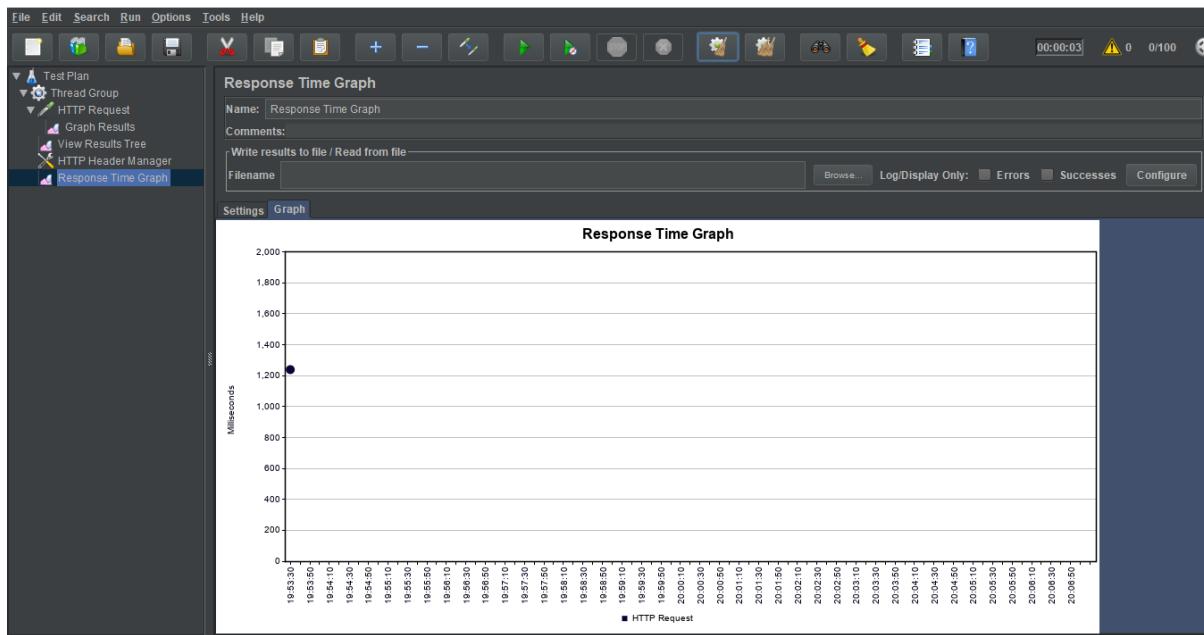
1. With Connection pooling.(pool size 1000)

1. 100 concurrent users:

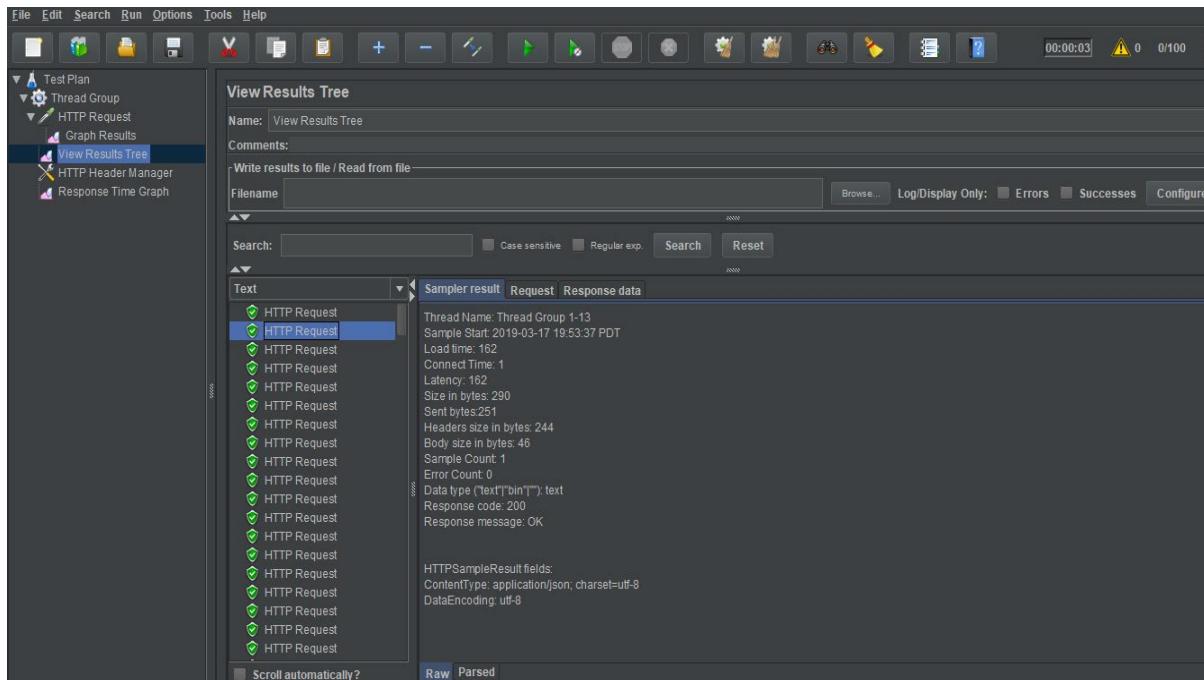
Graph result:



Response time Graph:

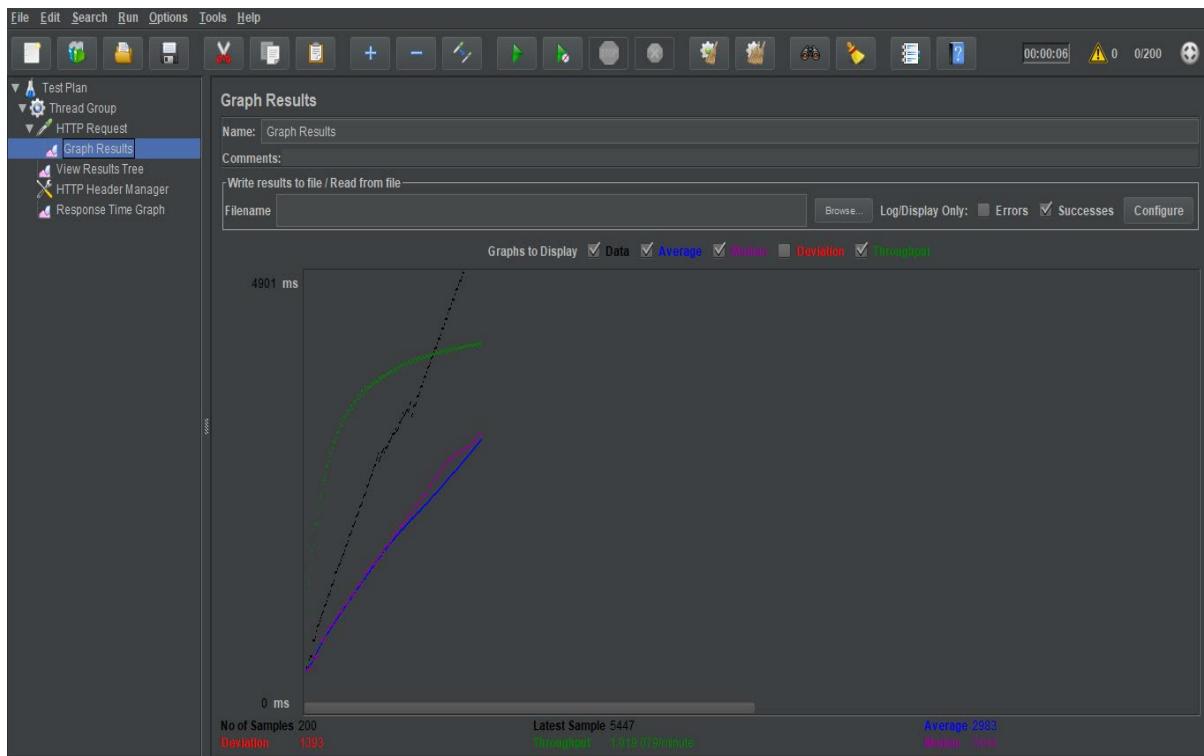


Result tree:

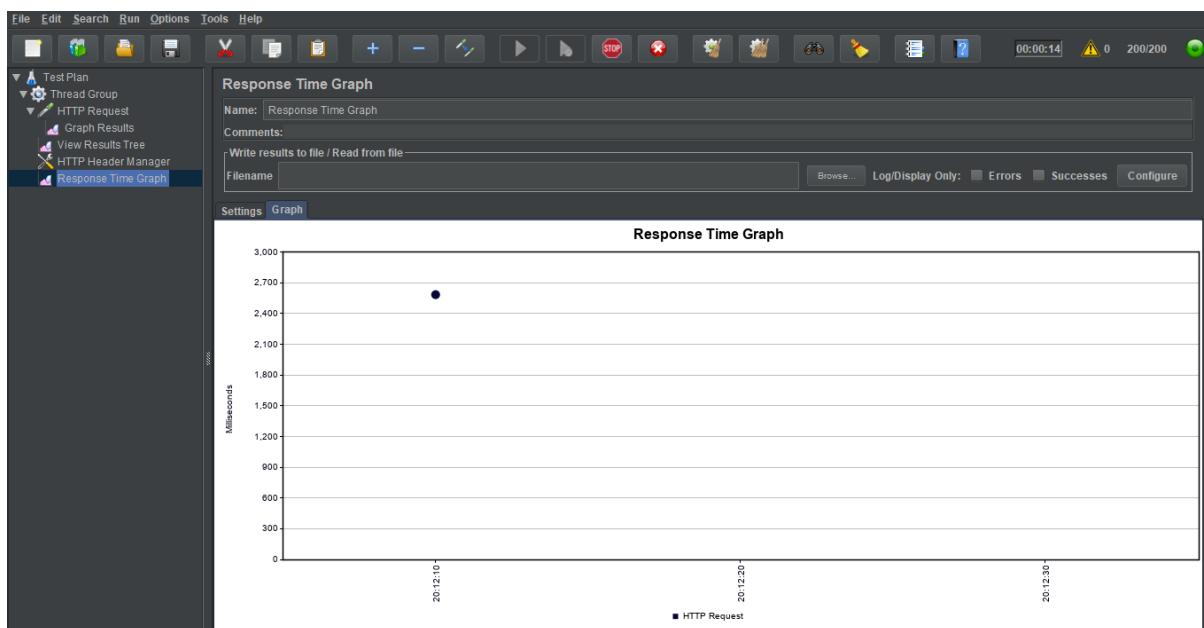


200: Concurrent users:

Graph results:



Response Time Graph:

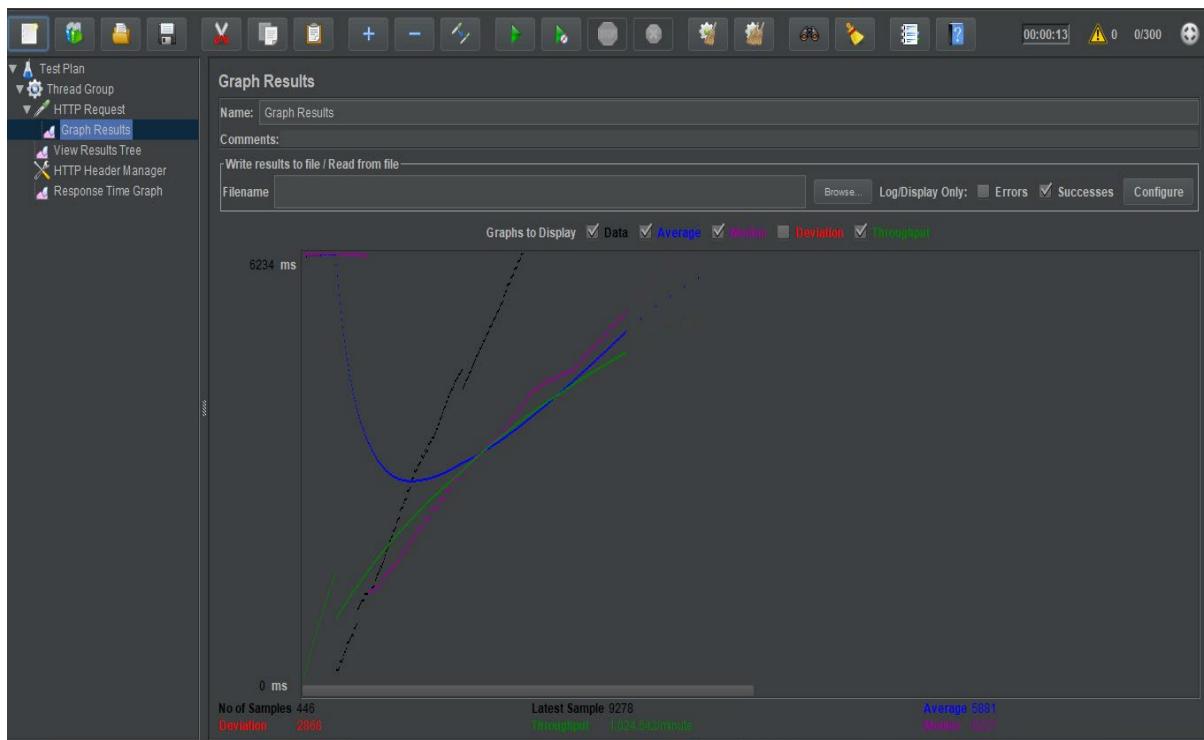


Result Tree:

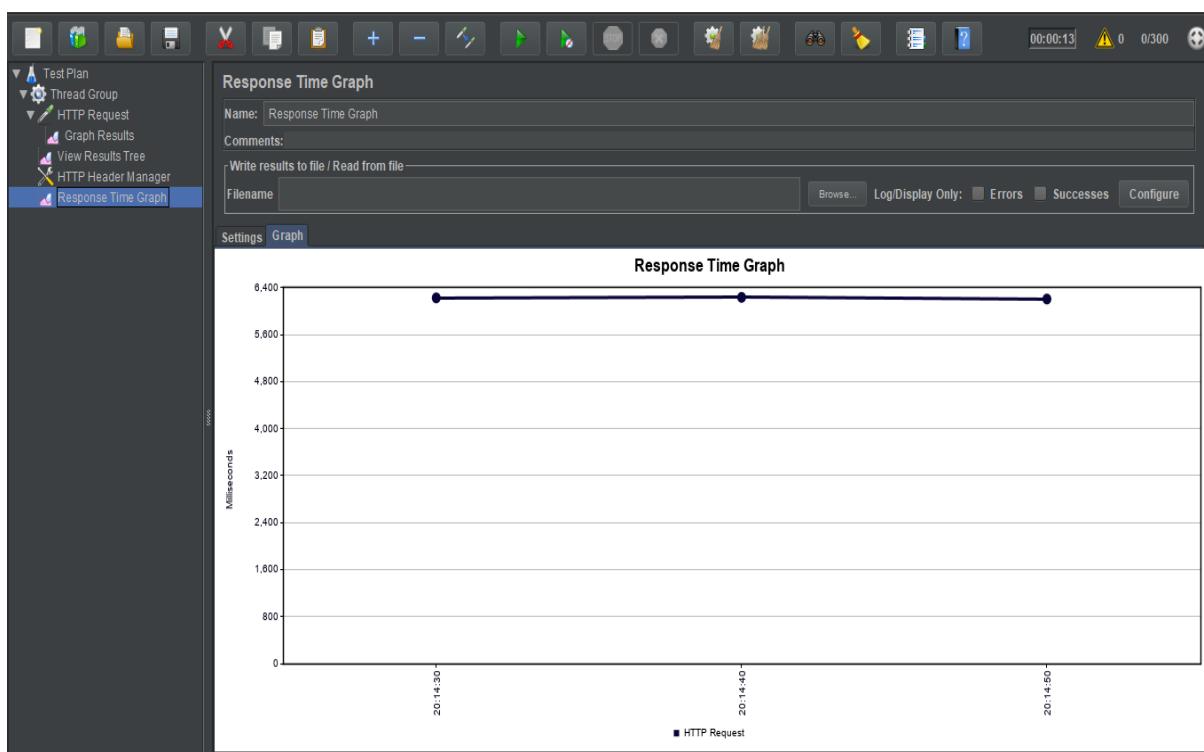
The screenshot shows the JMeter interface with the 'View Results Tree' listener selected in the left sidebar under the 'Test Plan' section. The main pane displays a tree view of 'HTTP Request' sampler results. One specific request is expanded, showing detailed metrics: Thread Name: Thread Group 1-75, Sample Start: 2019-03-17 20:13:17 PDT, Load time: 6142, Connect Time: 0, Latency: 6142, Size in bytes: 290, Sent bytes: 251, Headers size in bytes: 244, Body size in bytes: 46, Sample Count: 1, Error Count: 0, Data type ("text/plain"): text, Response code: 200, Response message: OK. Below the tree, the 'HTTPSampleResult fields:' section lists ContentType: application/json; charset=utf-8 and DataEncoding: utf-8. At the bottom of the pane, there are tabs for 'Raw' and 'Parsed' data.

300 concurrent users:

Graph Results:



Response Time Graph:



Result tree

View Results Tree

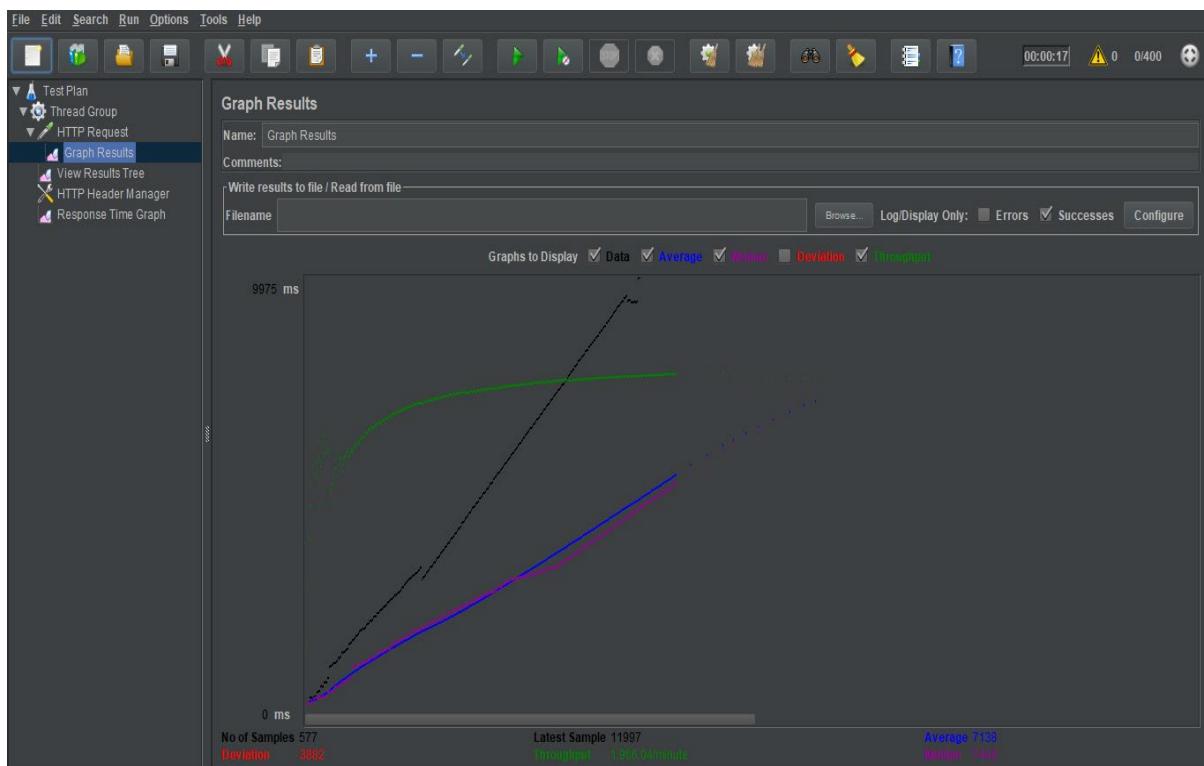
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename Browse... Log/Display Only: Errors Successes Configure

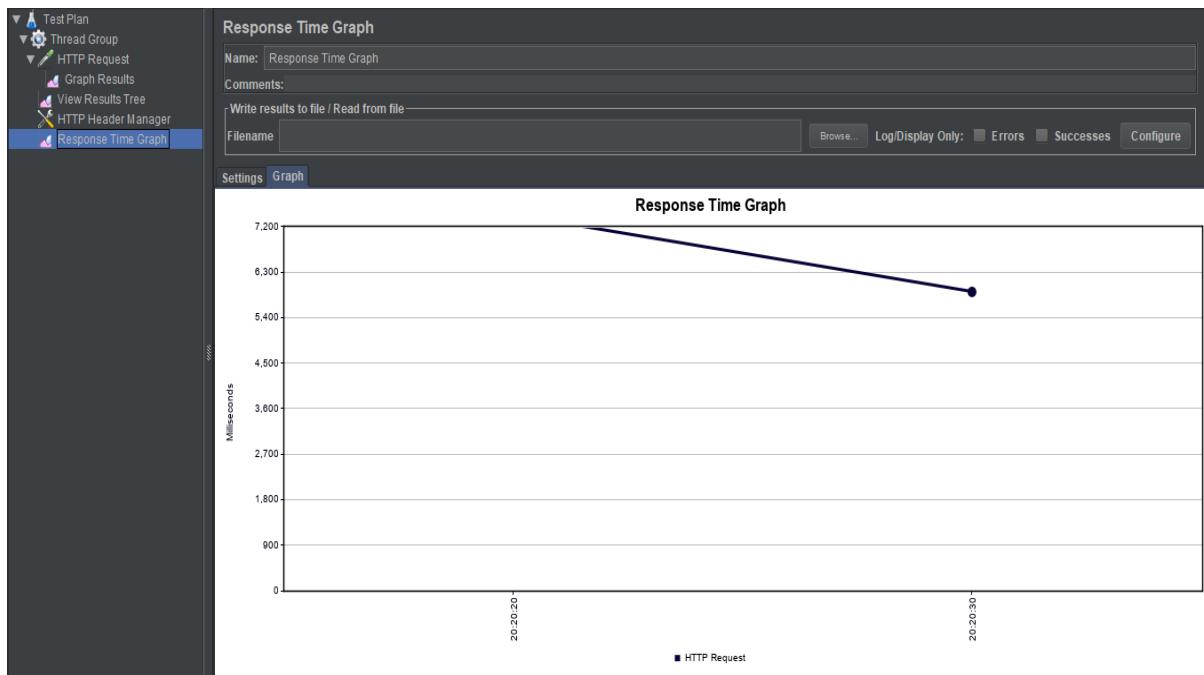
Search: Case sensitive Regular exp.

Text Sampler result Request Response data

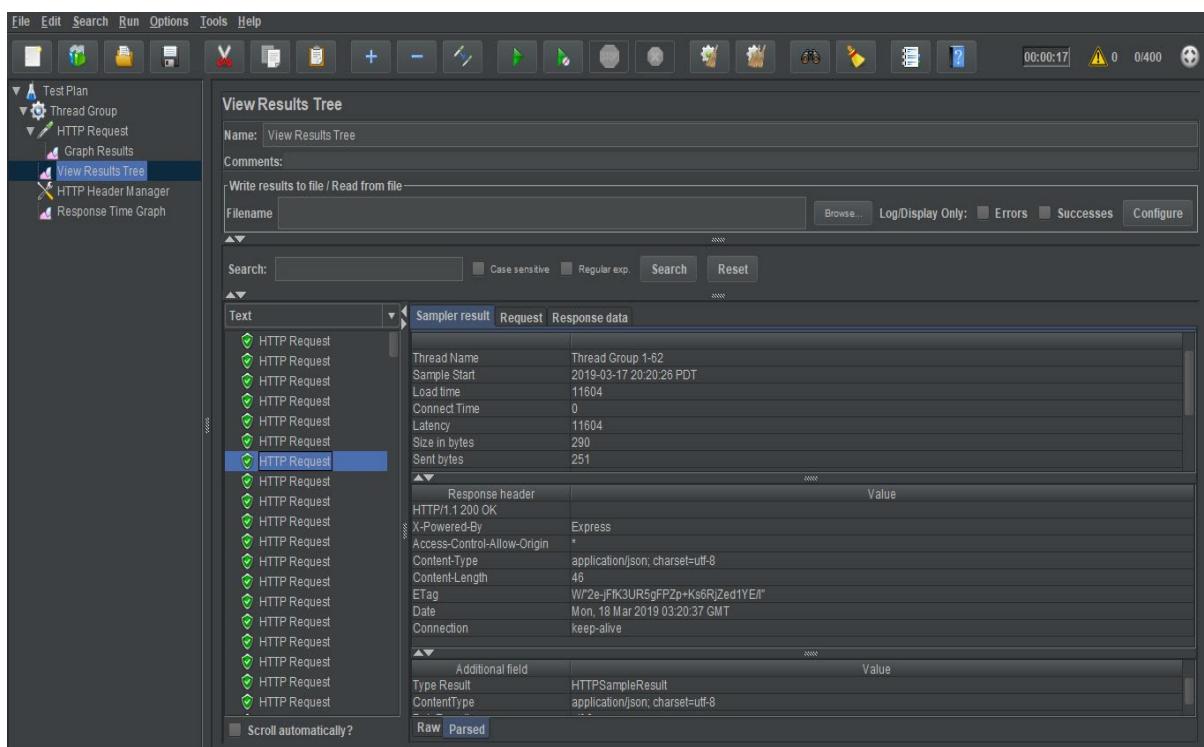
HTTP Request	Thread Name	Thread Group 1-225
HTTP Request	Sample Start	2019-03-17 20:14:53 PDT
HTTP Request	Load time	6359
HTTP Request	Connect Time	0
HTTP Request	Latency	6359
HTTP Request	Size in bytes	290
HTTP Request	Sent bytes	251
HTTP Request	Response header	HTTP/1.1 200 OK
HTTP Request	X-Powered-By	Express
HTTP Request	Access-Control-Allow-Origin	*
HTTP Request	Content-Type	application/json; charset=utf-8
HTTP Request	Content-Length	46
HTTP Request	ETag	W/"2e-JFk3UR5gFPZp+Ks6RjZed1YE/l"
HTTP Request	Date	Mon, 18 Mar 2019 03:15:00 GMT
HTTP Request	Connection	keep-alive
HTTP Request	Additional field	Type Result
HTTP Request	Type Result	HTTPSampleResult
HTTP Request	Content-Type	application/json; charset=utf-8

Scroll automatically?

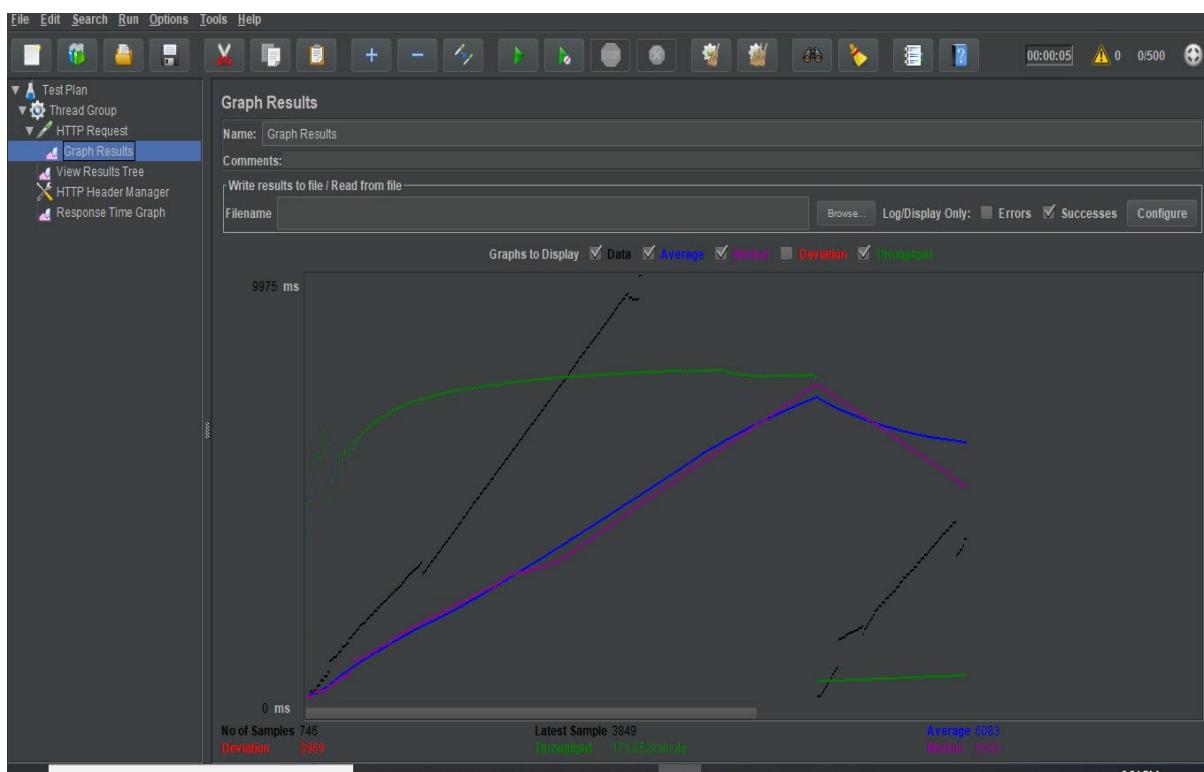
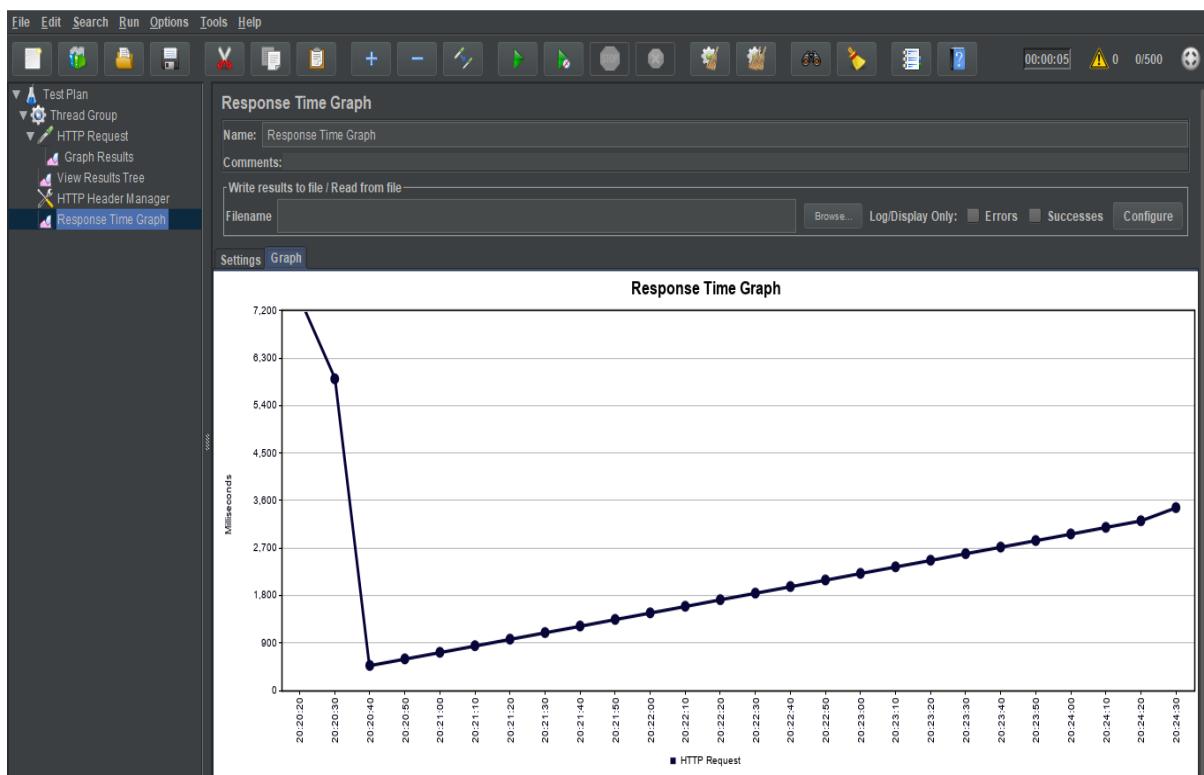
400 Concurrent users:**Graph results:****Response Time Graph:**



Result tree:



500 Concurrent users:

Graph Results:**Response time graph:****Result tree:**

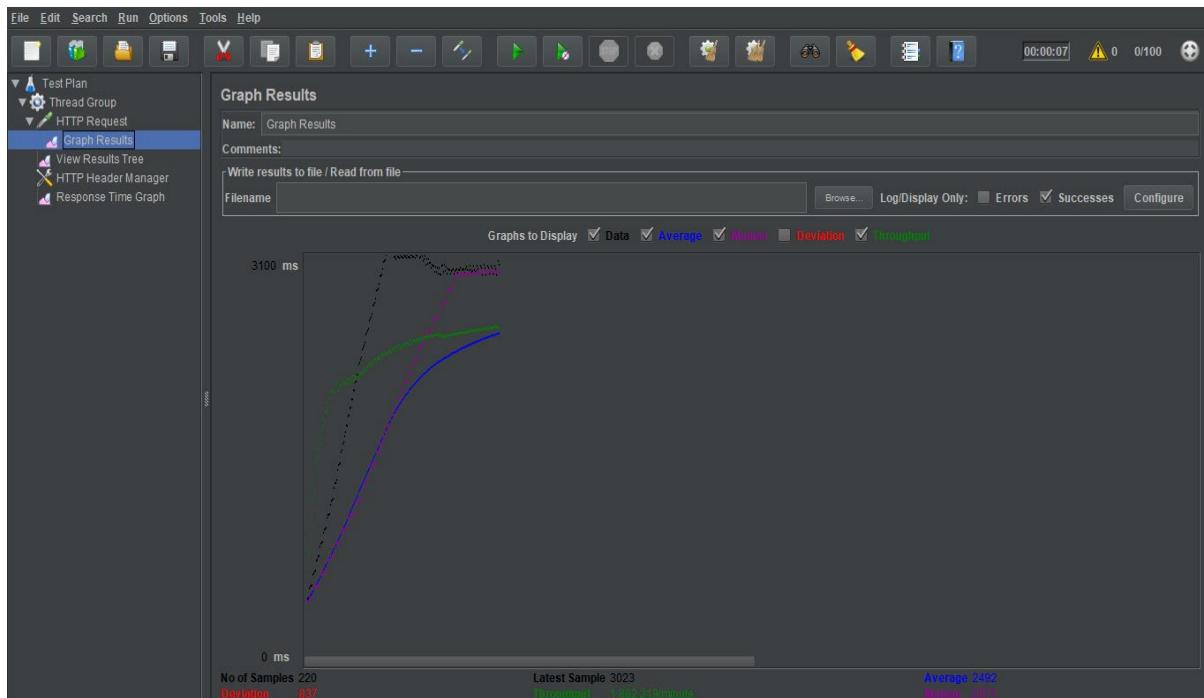
Screenshot of JMeter's View Results Tree listener. The tree view shows a single node under 'HTTP Request' labeled 'View Results Tree'. The details panel displays the following data:

Text	Sampler result	Request	Response data
HTTP Request	Thread Name: Thread Group 1-62 Sample Start: 2019-03-17 20:20:26 PDT Load time: 11604 Connect Time: 0 Latency: 11604 Size in bytes: 290 Sent bytes: 251		
HTTP Request	Response header: HTTP/1.1 200 OK X-Powered-By: Express Access-Control-Allow-Origin: *		
HTTP Request	Content-Type: application/json; charset=utf-8 46		
HTTP Request	ETag: W/2e-jFK3UR5gFPZp+Ks6R/Zed1YE/I Date: Mon, 18 Mar 2019 03:20:37 GMT		
HTTP Request	Connection: keep-alive		
HTTP Request	Additional field: Type Result: HTTPSampleResult Content-Type: application/json; charset=utf-8		
HTTP Request	Raw Parsed		

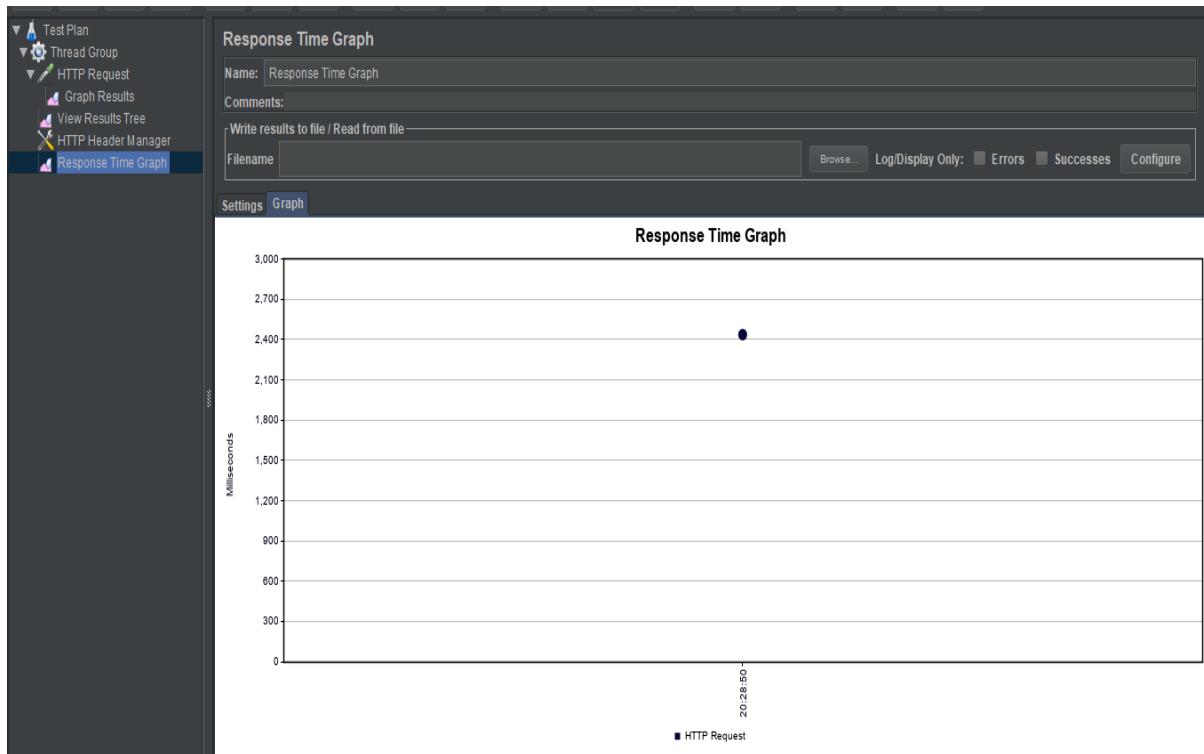
Without connection Pooling:

100 Concurrent users:

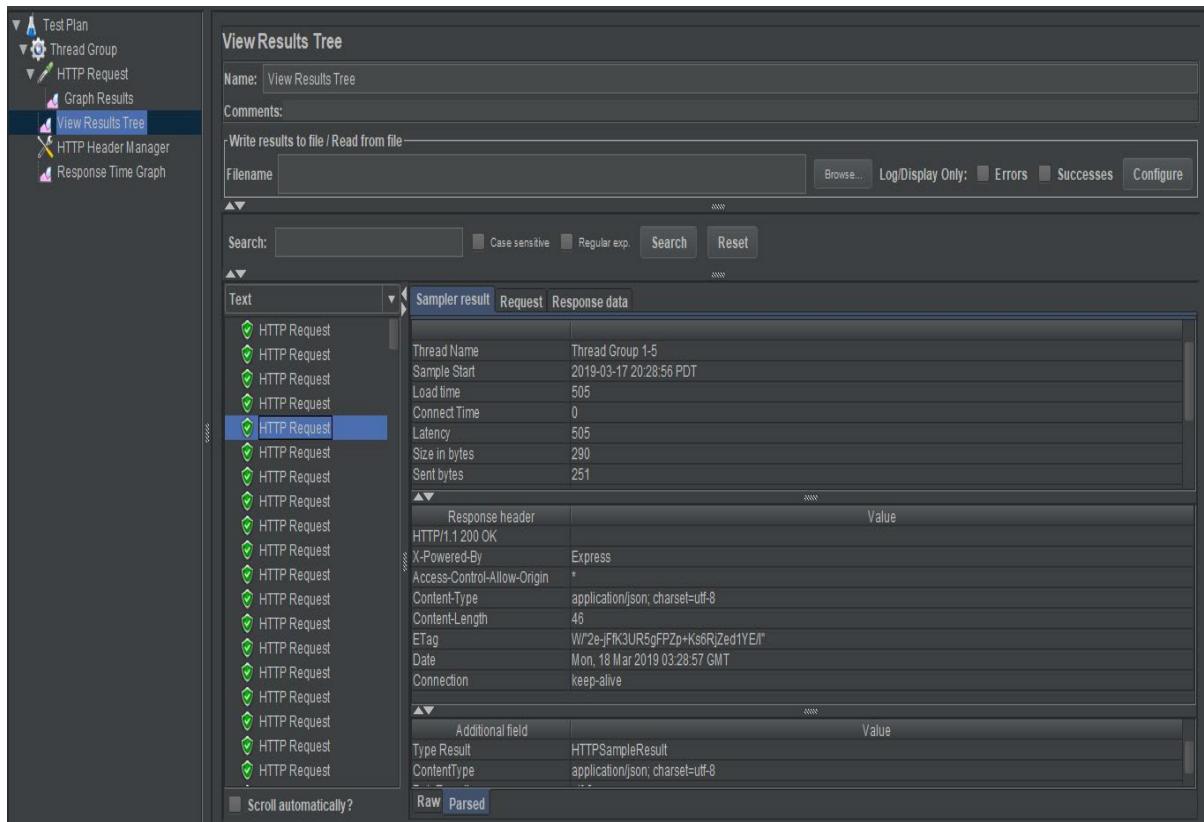
Graph Results:



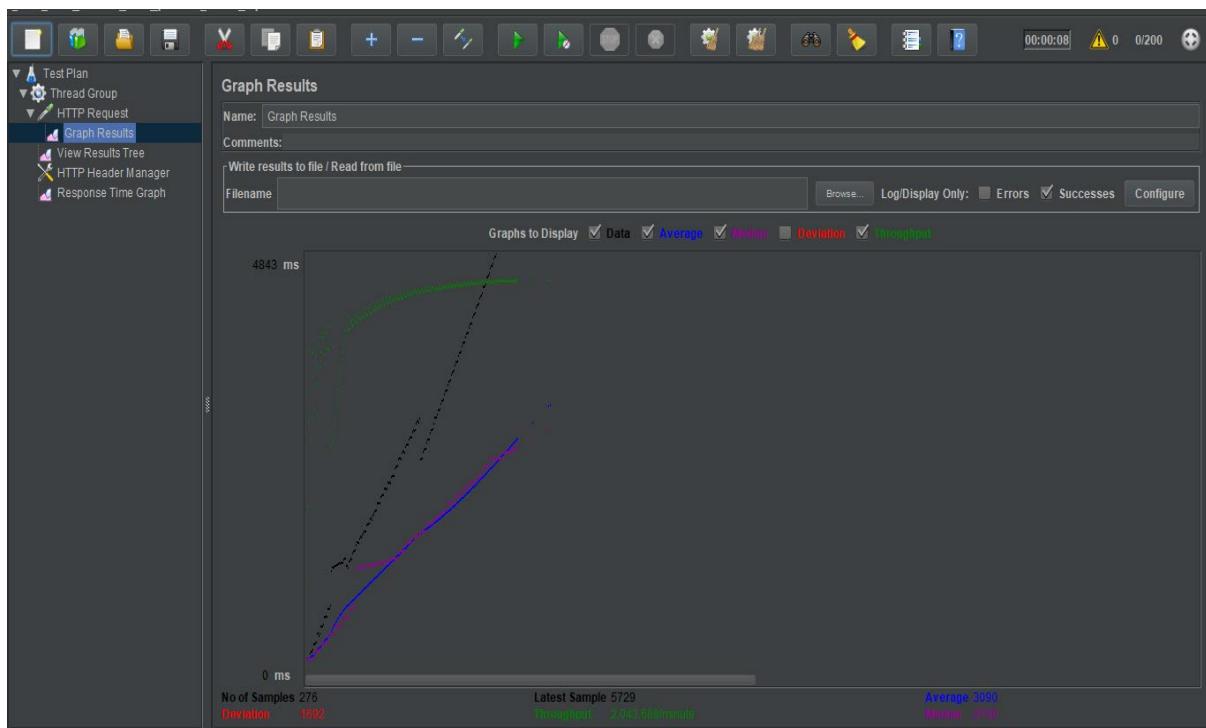
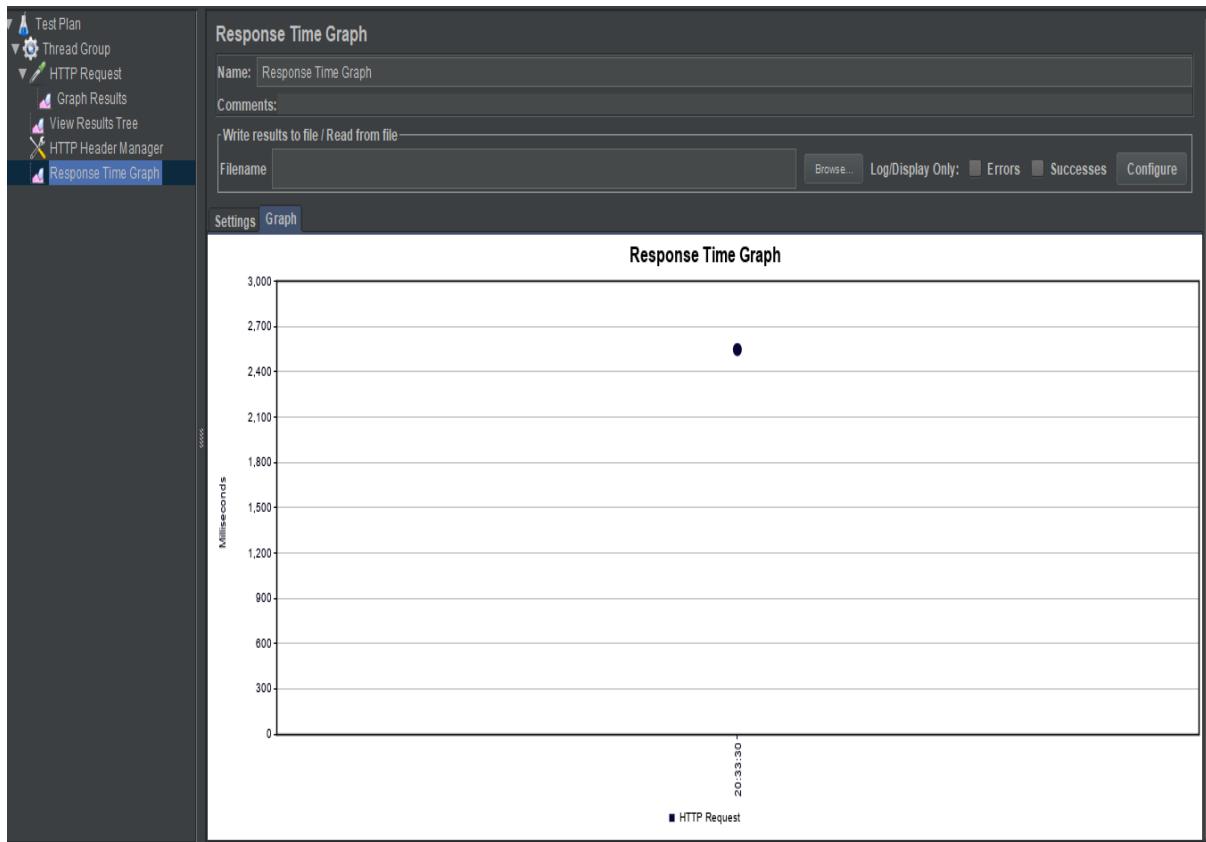
Response time graph:



Result tree:



200 Concurrent users:

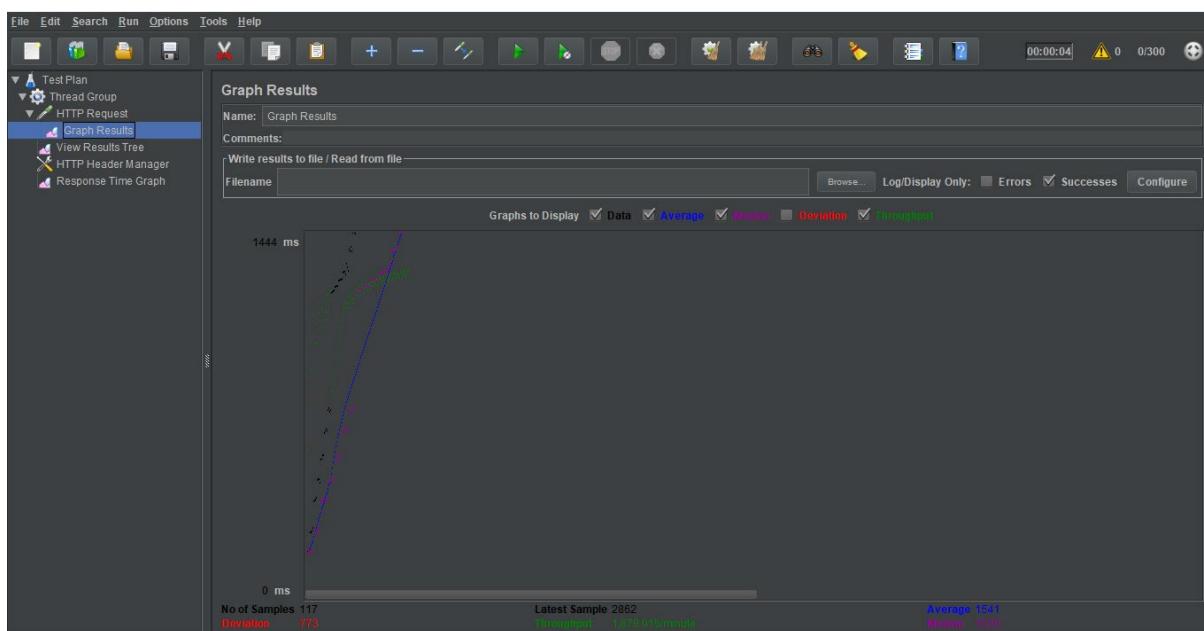
Graph results:**Response Time Graph:****Result tree:**

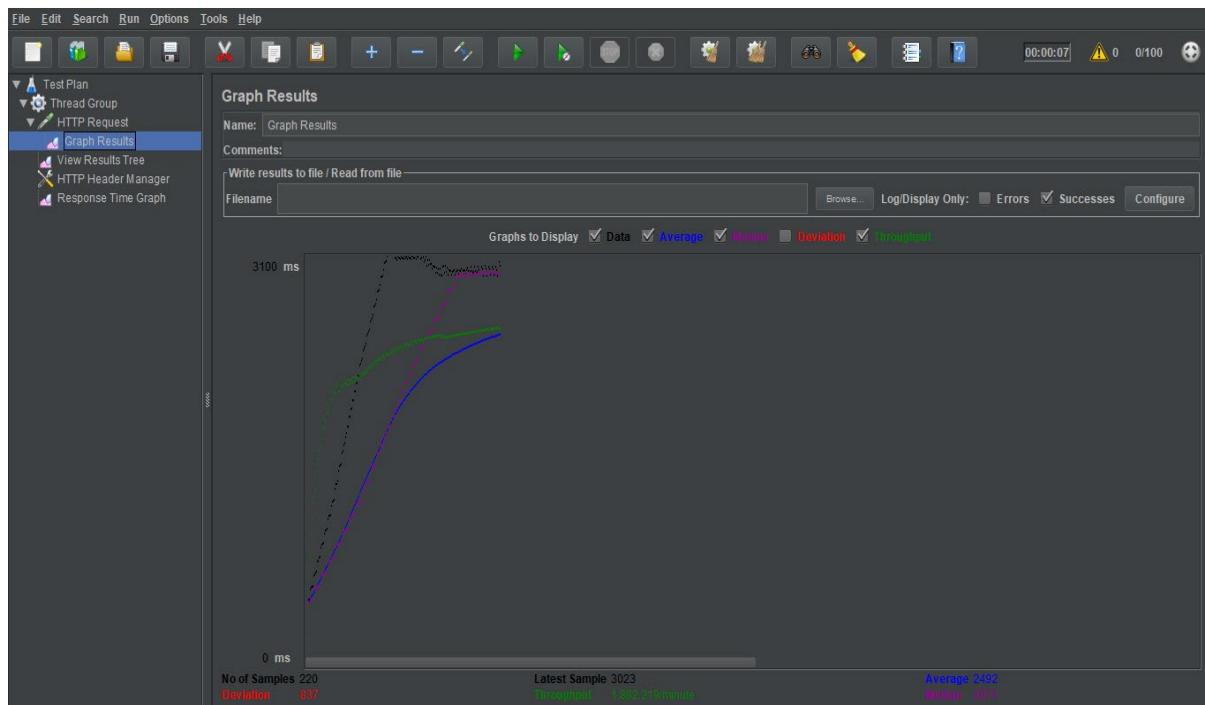
The screenshot shows the JMeter interface with the 'View Results Tree' listener selected in the left sidebar under 'Test Plan > Thread Group > HTTP Request > View Results Tree'. The main panel displays a single request details table:

	Sampler result	Request	Response data
HTTP Request	Thread Name: Thread Group 1-2 Sample Start: 2019-03-17 20:32:11 PDT Load time: 143 Connect Time: 0 Latency: 143 Size in bytes: 290 Sent bytes: 251		
HTTP Request	HTTP/1.1 200 OK X-Powered-By: Express Access-Control-Allow-Origin: *Content-Type: application/json, charset=utf-8 Content-Length: 46 ETag: W/2e-jFk3UR8gFP2o+Ks0RZsd1YE/I/ Date: Mon, 18 Mar 2019 03:32:11 GMT Connection: keep-alive	Response header	Value
HTTP Request	Type Result: HTTPSampleResult Content-Type: application/json, charset=utf-8	Additional field	Value
HTTP Request	Raw Parsed		

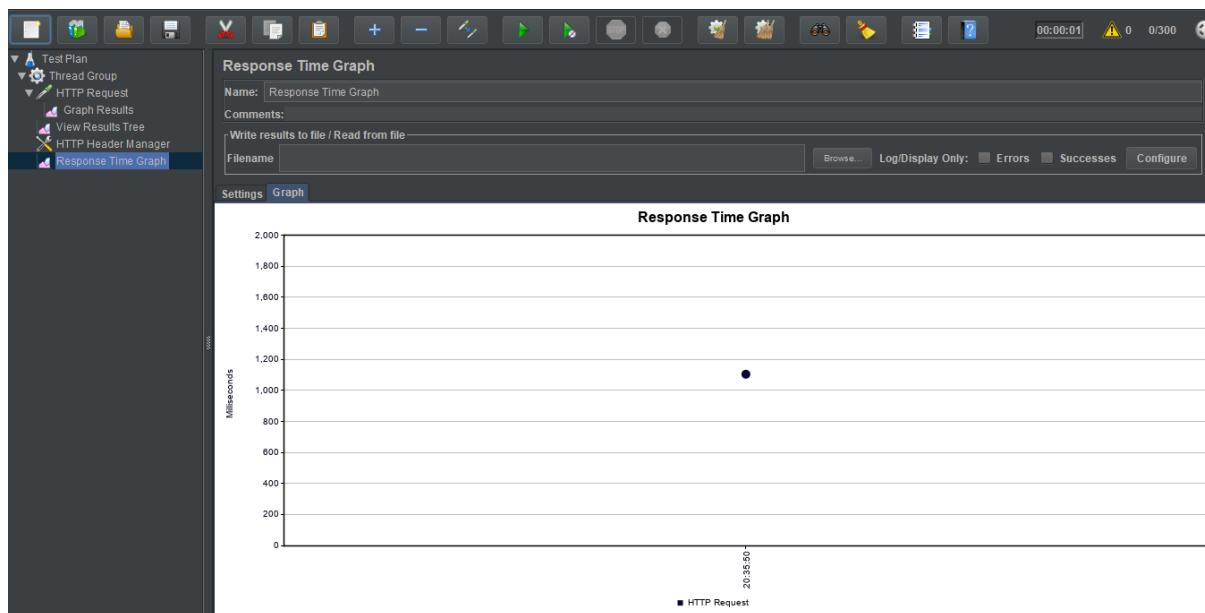
300 concurrent users:

Graph Results:



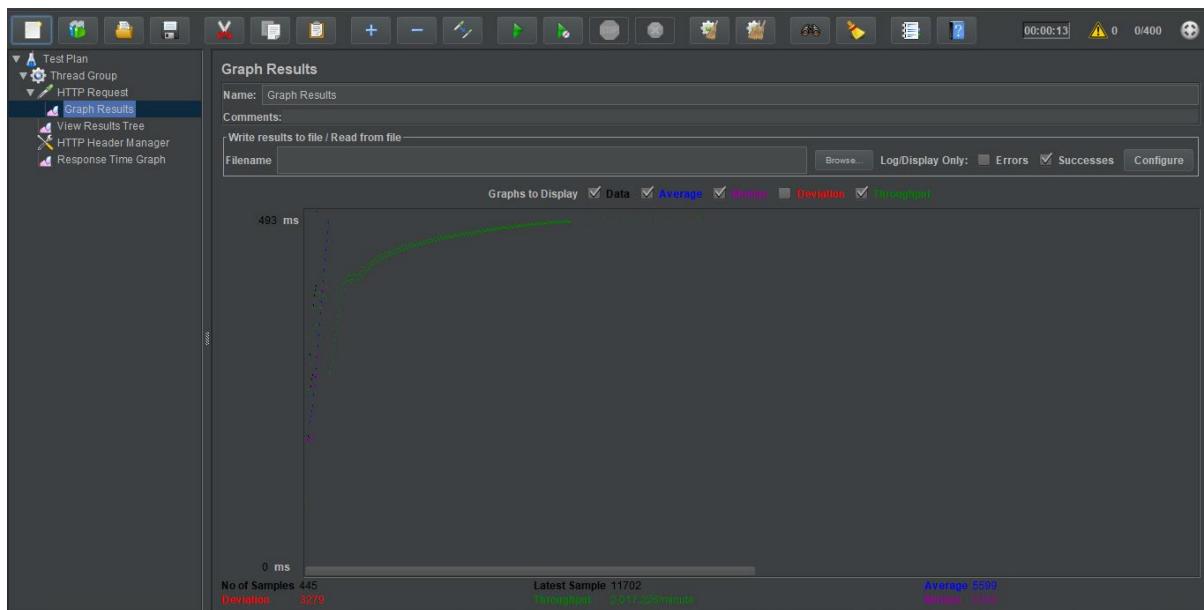


Response Time Graph:

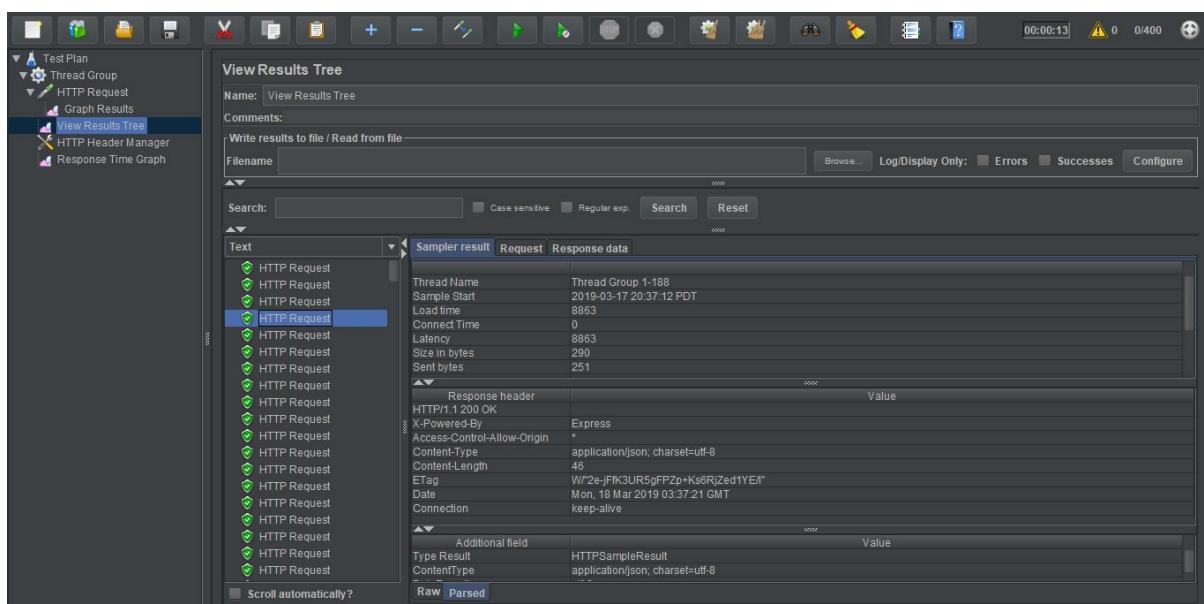


400 Concurrent users:

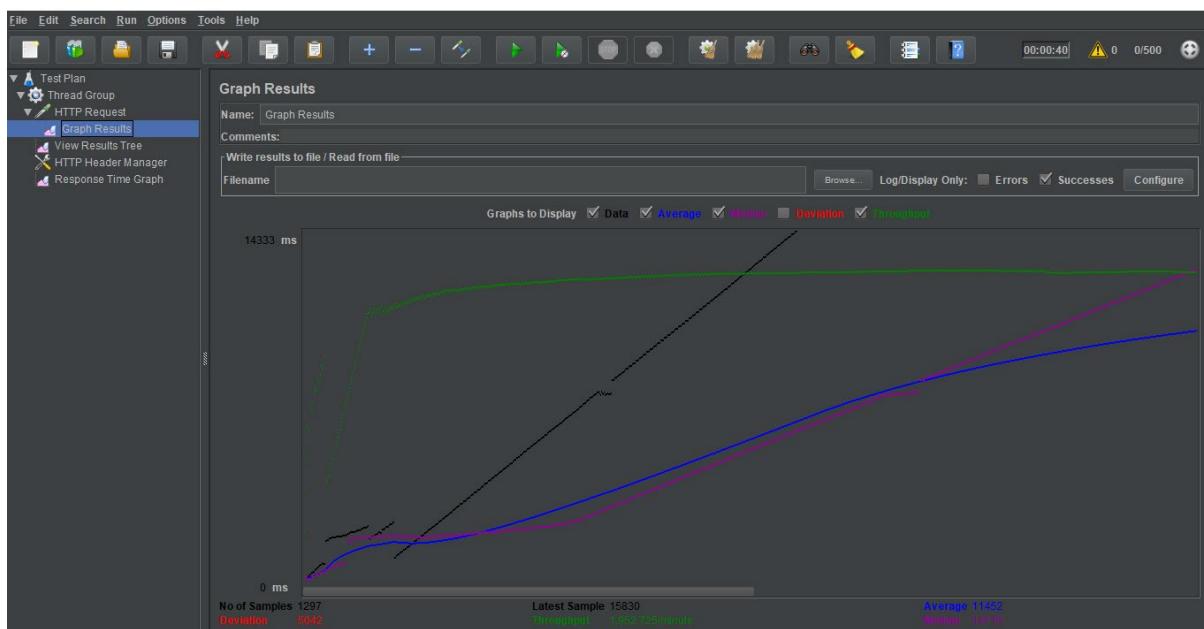
Graph Results:



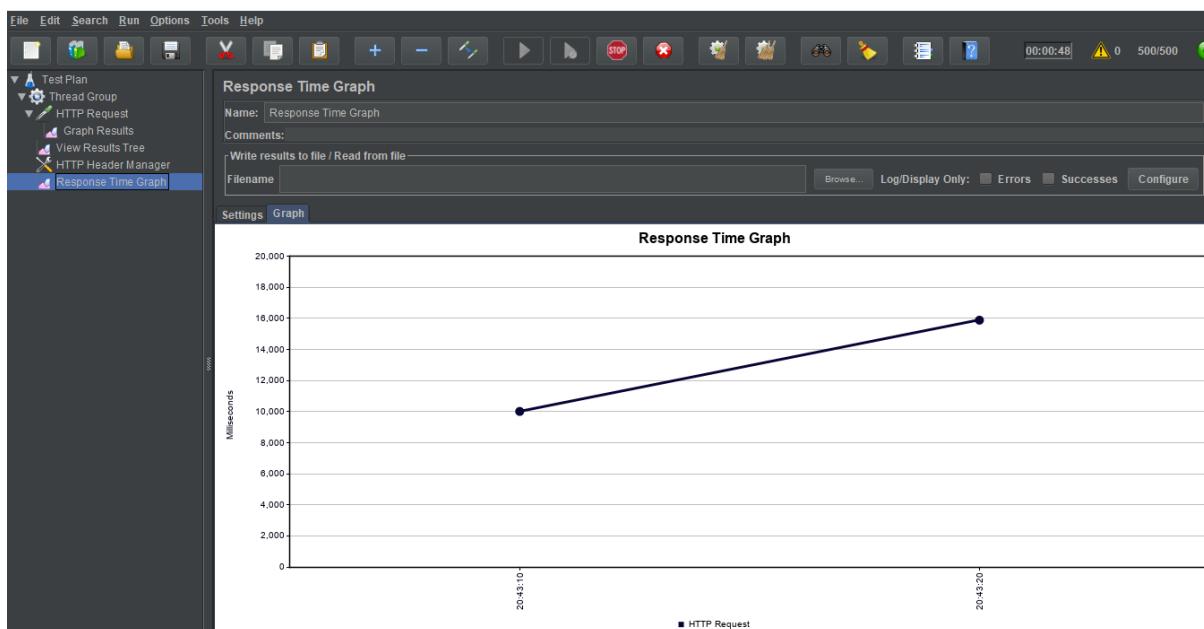
Result tree:



500 Concurrent users:



Response Time Graph:



Result Trees:

The screenshot shows the JMeter interface with the 'View Results Tree' listener selected in the left sidebar under the 'Test Plan' tree. The main panel displays the results of a single HTTP request. The request details include:

Thread Name	Value
Sample Start	2019-03-17 20:43:10 PDT
Load time	239
Connect Time	0
Latency	239
Size in bytes	290
Sent bytes	251

The response header section shows:

Response header	Value
HTTP/1.1 200 OK	
X-Powered-By	Express
Access-Control-Allow-Origin	*

The content section shows the parsed JSON response:

Additional field	Type Result	Contenttype	Value
Type Result	HTTPSampleResult		
Contenttype		application/json; charset=utf-8	

At the bottom, there are tabs for 'Raw' and 'Parsed'.

Questions and Answers:

1. Explain the encryption algorithm used in your application. Mention different encryption algorithms available and the reason for your selection of the algorithm used.

- I have used symmetric block cipher algorithm library Bcrypt which uses Blowfish encryption algorithm.
- It generates a secured Hash value by multiple rotation and symmetric key encryption, and it is free for use.

Other available Encryption algorithms:

1. AES (Advanced Encryption standard)
2. DES or D3ES (Data Encryption Standard)
3. Two FISH
4. IDEA
5. MD 5 (messaging Digest)
6. HMAC
7. RSA Security.

Why Bcrypt?

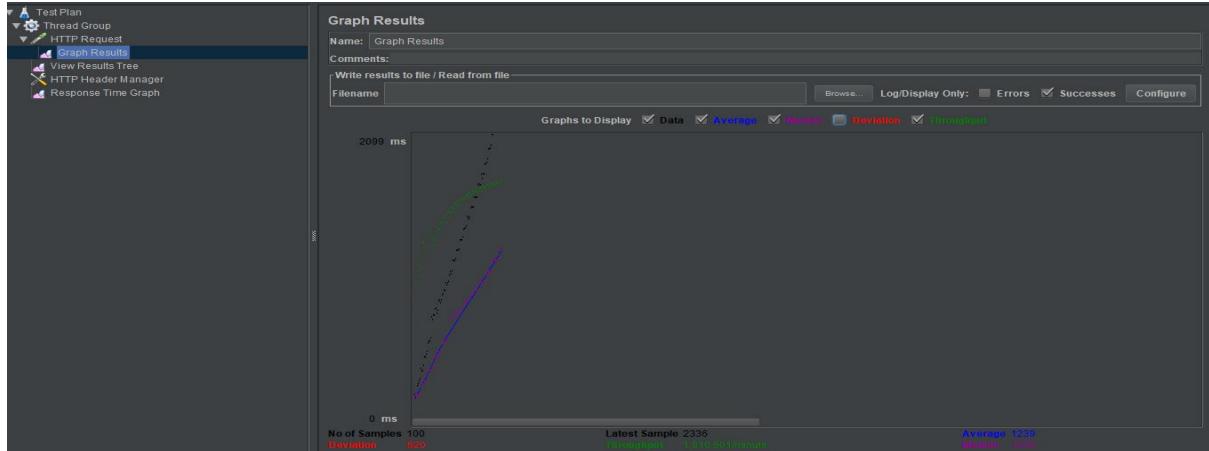
- It is an adaptive hashing algorithm that uses a blowfish symmetric cipher to encrypt.
- Its most important feature is that It uses a Key Factor that adjusts the cost of hashing. The ability to increase the cost of hashing i.e. increase the time and power for hashing is what makes Bcrypt more powerful. This process gets very slow.
- It can expand its Key Factor to compensate for increasingly more-powerful computers and effectively “slow down” its hashing speed. Changing the Key Factor also influences the hash output, so this makes Bcrypt extremely resistant to rainbow table-based attacks. Newer computers can attempt to guess the original input of the hash.
- Bcrypt is slower than SHA2 and it is adaptive whereas SHA2 is not adaptive. Speed is the key factor for using Bcrypt over other algorithm as in terms of encryption, the slower is better.

2. Compare the results of graphs with and without in-built MySql connection pooling of database. Explain the result in detail and describe the connection pooling algorithm if you need to implement connection pooling on your own.

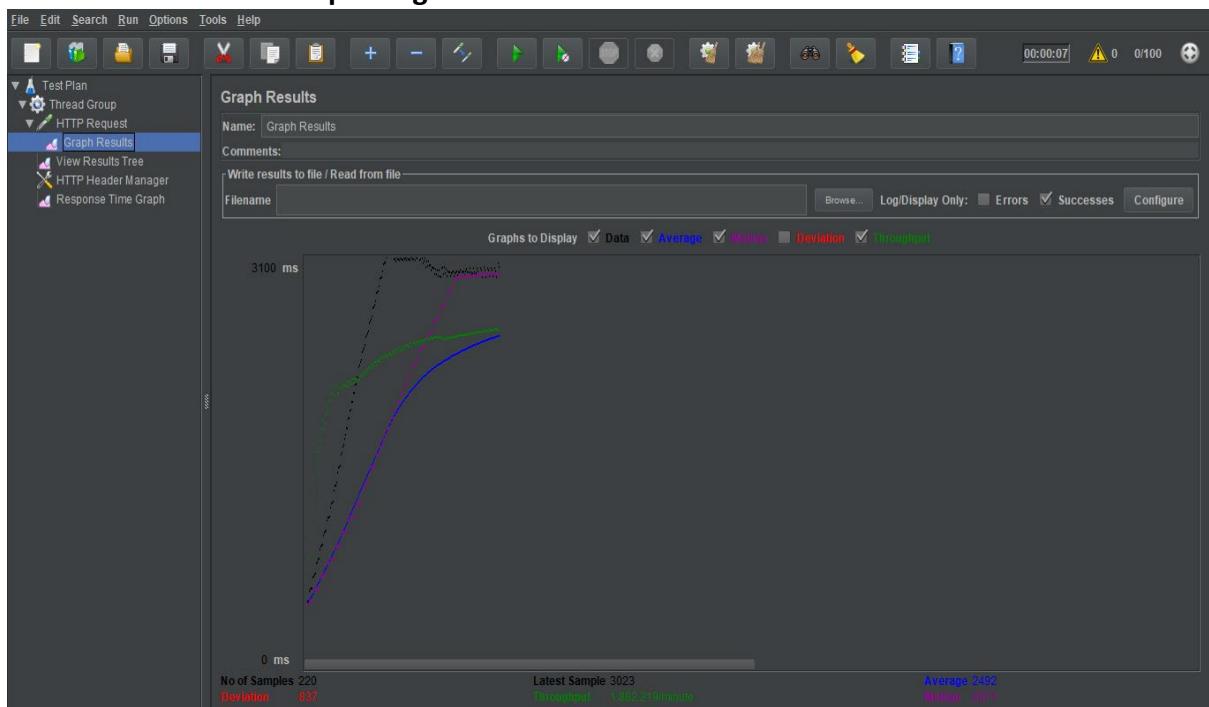
Answer:

Results of graphs with and without connection pooling:(100 users)

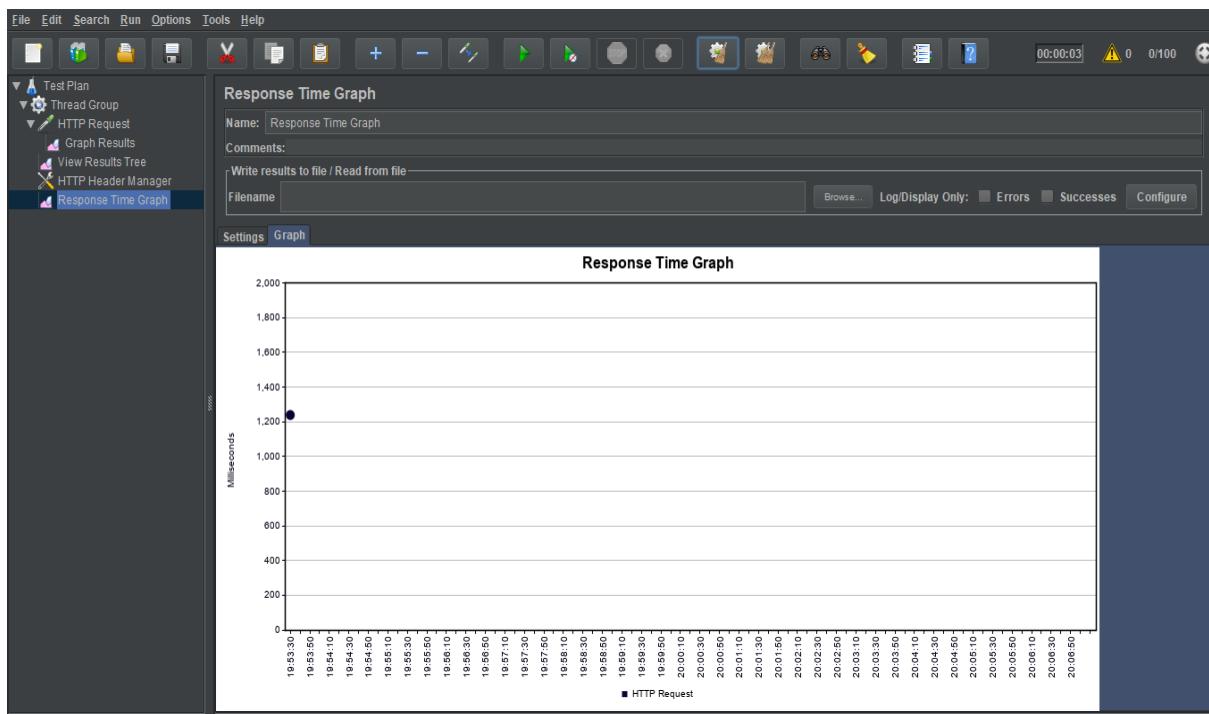
1. With connection pooling(graph results):



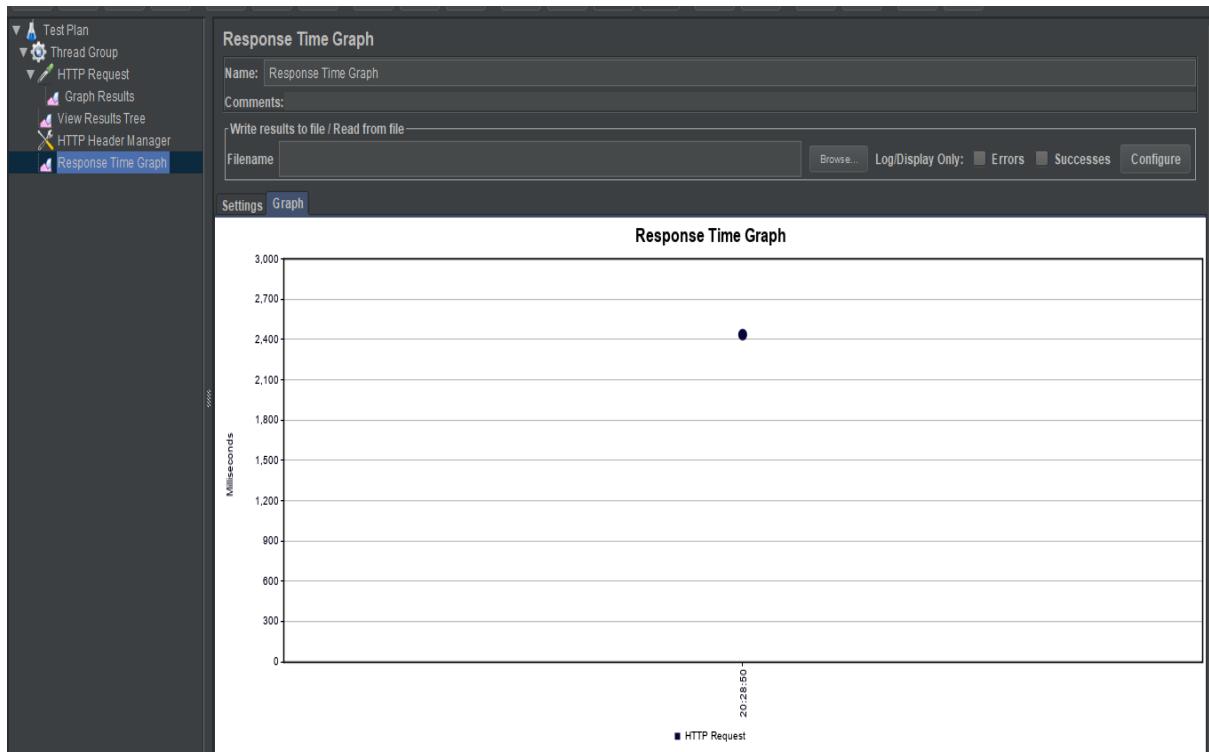
2. Without connection pooling.



With Connection pooling: (time Graph):



Without Connection pooling:



Comparison:

It is clearly seen from the graphs that average response time decreases when we use the connection pooling where as without connection pooling, the connection time increases and average response time increases.

- To connect with a database is a very time-consuming task. Every time when a user's wants to connect with a database, it creates a new thread and try to connect with database. In such case the response can only be given if the queued requests are entertained first. In such cases connection pooling creates and maintain multiple threads which are reused to make connections.
- With the use of connection pooling, one need not have to establish new connections every time when it wants to connect with database. Connection pooler maintains a live set of active connections for each given connection configuration.
- Creating your own connection pool:
- When a user requests to connect for the first time, the pool is created, and multiple threads are maintained. And when it closes the connection it goes to active thread pools and frees the thread to user. Now when another request comes then a new connection is not created but the previous connection is reused.
- If 2 requests come at a time and pool has only one connection object, then one connection is reused by one request and new connection is created for the 2nd request.
- The connection pooler satisfies requests by reallocating connections as they are release back into the pool.
- There can be multiple connection pools available at the same time. Connections are separated into pools by connection string.
- **Pseudo code for connection pooling algorithm:**
 - A. create a thread pool.
 - B. Set the size of thread which will be a pool size say 1000. then the pool will have 1000 concurrent thread running.
 - C. Set connection pool using connection string.
 - D. On connection request connect to the database by creating connection.
 - E. When concurrent requests occur at the same time then,
 - F. Firstly, check the thread pool,
 - G. If (active threads in pull) then (use that thread) and connect to database.
 - H. Else (create new thread and add to the queue).
 - I. when use leaves the connection,
 - J. Free the thread and add back to pool.
 - K. Set thread time out and use async and wait for maintaining threads.
 - L. Throw an exception when connection is timed out.
 - M. If (connection State== idle && Timed Out) then Remove connection();

3. What is SQL caching? What all types of SQL caching are available, and which suits your code the most. You don't need to implement the caching, write pseudocode or explain in detail .

- When it's not possible to remove SQL statements that are unnecessary and the rate of change of common data is relatively low, caching SQL results can provide a significant performance boost to your application and enable additional scalability of your database server.

Types of Caching:

1. Cache-Aside (for reading heavy workloads)
 2. Read-Through Cache(read heavy workload when same data is used for many times)
 3. Write-Through Cache (paired and read through caches)
 4. Write-Around (data is written once and read less frequently)
 5. Write-Back (for writing heavy workloads)
- The impact of the MySQL query cache can degrade the performance for environments with heavy write to read ratios. In an environment with a high number of reads and writes, this can result in significant invalidations of SELECT statements that do not gain the benefit of ever being used.
 - Specifically for SQL Server, the following are cached:
 - Raw data (table data and index structures), in the buffer pool. Subsequent queries that need to read the same data do not need to wait until the data is retrieved from disk. Data modifications are written to the buffer pool only and later hardened to disk in an asynchronous process. (The transaction log is used to prevent data loss in case of loss of power)
 - Execution plans, in the procedure cache (also called plan cache). This avoids expensive recompilations for queries that are executed repeatedly.
 - Query results are not cached in SQL Server. That would save even more performance if the exact same query is repeated with the exact same parameters, but in a realistic workload later executions of the same query usually are for a different parameter. And the logic to detect cache invalidation for query results is very complex, so it would involve probably more overhead than it would save.

The caching strategy that is most suitable for this application is Cache aside.

- Here there will be so many read operations for the similar kinds of data and hence cache aside will improve the system performance.
- Here cache will sit aside, and system will interact with both cache and data source. The application logic will hit the cache before hitting the data source.

❖ Pseudo code for writing SQL catch:

1. SET GLOBAL query_cache_size=1024*1024*16;
2. SET GLOBAL query_cache_type=1;
3. SET PROFILING=1;
4. Write select Queries here
5. Call all the queries.

Working:

Application makes a call of query-> cache

Request goes to Cache control-> check whether cached results exists->

```
If available(check Result)
{
    Return (Result)
}
```

```
Else{
```

Request query -> Database

Return result -> client

```
Save Query result -> cache
}
```

Q. 4 Is your session strategy horizontally scalable? If **YES**, explain your session handling strategy. If **NO**, then explain how you can achieve it.

Answer:

YES.

For the sake of the small platform, I have used JWT (JSON web tokens) for authentication which is a strategy to maintain session on client side in his local storage. As we are using RESTful APIs every request becomes stateless and there will be no need to store session data on the server side.

But when creating a distributed system there is a possibility to read or alter token values if someone is able to read the token strings. One way to make it possible is encrypting the tokens as well.

A consideration when choosing a distributed cache for session management is determining how many nodes may be needed in order to manage the user sessions. In a distributed session cache, the sessions are divided by the number of nodes in the cache cluster. In the event of a failure, only the sessions that are stored on the failed node are affected. ElastiCache offerings for In-Memory key/value stores include ElastiCache for Redis, which can support replication.

There are a number of ways to store sessions in Key/Value stores. Many application frameworks provide libraries which can abstract some of the integration plumbing required to GET/SET those sessions in memory. In other cases, you can write your own session handler to persist the sessions directly.

So, the conclusion is to use Token based authentication that will reduce the server load and make the sessions and application horizontally scalable.

Here after generation of token, every request will contain the token values which will be matched on server side. If the token is not expired and is matched on the server then it is authenticated and will respond to client.

Commit History:

The screenshot shows a GitHub commit history for the repository 'Hariae / CMPE273-SP19-63'. The branch is set to 'master'. The commits are organized by date:

- Commits on Mar 17, 2019:**
 - committing react application chnages and adding tests final (5db42ed) - Rohan shah committed 12 minutes ago
 - file quiz enrollment grade (4790b89) - Rohan shah committed 14 hours ago
- Commits on Mar 16, 2019:**
 - asigments and permission numbers (81fdc9c) - Rohan shah committed 2 days ago
- Commits on Mar 14, 2019:**
 - assignments and quizzes functionalities (694e8ca) - Rohan shah committed 3 days ago
- Commits on Mar 12, 2019:**
 - update profile and login (b8c0a1a) - Rohan shah committed 6 days ago
 - final authentication and backend db 03/11 (e129f52) - Rohan shah committed 6 days ago
- Commits on Mar 11, 2019:**
 - pushing root directory changes (39fea40) - Rohan shah committed 6 days ago
 - multiple changes added node backend , back to my sql, handling querie... (d966529) - Rohan shah committed 6 days ago
- Commits on Mar 8, 2019:**
 - trying to push changes in react component (e079bd7) - Rohan shah committed 9 days ago
 - object is not a function error ! (c98c4ca) - Rohan shah committed 9 days ago

↳ Commits on Mar 5, 2019

react log-in-signup

 Rohan shah committed 12 days ago

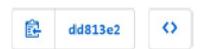
↳ Commits on Mar 4, 2019

3/17/2019, 11:20 PM

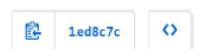
· Hariae/CMPE273-SP19-63

<https://github.com/Hariae/CMPE273-SP19-63/commits/master>

server side signup login and db connectivity

 Rohan shah committed 13 days ago

adding react-router-dom

 Rohan shah committed 13 days ago

server db set up and frontend component set up

 Rohan shah committed 14 days ago

↳ Commits on Mar 3, 2019

git ignore

 Rohan shah committed 14 days ago

backend index.js

 Rohan shah committed 14 days ago

first commit

 Rohan shah committed 14 days ago