Google Summer of Code 2020

Chapel
Project - Libraries for Web Services

Introduction

I am Aniket Mathur, an Electronics and Communication Engineering undergraduate student at the **Indian Institute of Technology Roorkee**. I am pursuing Bachelor of Technology and currently in my third year of study. I got introduced to the world of programming languages and software in my first year, and since then I have made efforts in and explored various fields of Computer Science. The fields that capture my interest are Operating Systems, Computer Networks, Database Management, and Information Security. I am also proficient and have experience in the field of Software development (Android and Web development), with a handful of experience in Machine Learning as well.

Relevant course work for the project

- Computer Networks
- Object-Oriented Programming
- Data Structures
- Computer Architecture and Microprocessors
- Operating Systems

Motivation

Why you wish to participate in the Google Summer of Code?

Being a software enthusiast from my first year I always loved to think and develop ideas of my own. My interest in web development introduced me to Git and Github which, for me, ultimately paved a path towards open-source. It is really satisfying and worth it to contribute to big open source organizations, as your work directly affects the open-source community and a large number of people using that software.

Last year I was selected as a **Google Summer of Code student for Sugar Labs**. Over the summers it was a great learning experience for me, so I decided to apply this year as well. I think that Google Summer of Code is the best way to learn how to work in an open-source community where you have a number of people from different parts of the world working and building code collaboratively. The three-month duration of GSoC will also allow me to hone my current skills as well as acquire a new set of them under the guidance of quality

mentoring organizations and best mentors. The remote working policy of GSoC makes it the best program a college student can get involved in during their summer holidays for a great learning experience.

Why you wish to work with the Chapel project in particular?

The Chapel project is a great opportunity for me to contribute to the field of programming languages and Distributed Computing. I have always been keen about how a programming language is built and its libraries are developed and always wanted to contribute to one of them.

While contributing to Chapel, I have seen that the developer team of Chapel discusses issues very well and an equal weightage is given to every contributor's opinion. The contributors are given an equal opportunity to propose their ideas and discuss them with the developer team.

Also Chapel has great mentors, who are always willing to provide the best possible help. The mentors are very responsive, friendly and knowledgeable. This kept me motivated and enthusiastic towards contributing.

What do you hope to learn over the summer?

Over the summers I will get to learn the technical aspect of how Libraries for Web Services are developed in detail. The valuable experience of working on this project will get my skill set furnished with the help of my knowledgeable and experienced mentors. Also my last year GSoC project involved the majority of the changes in the existing code-base, this project will provide me with an experience of starting to code from scratch.

Language & Availability

How well can you comprehend and understand English? How strong is your written English?

My first language is Hindi but I am proficient in speaking, reading, writing, and understanding English as well. I have been taught English since my school time. I have also done a Communication Skills(English) course at my university in which I scored a grade point of 10/10.

Do you have any other commitments for the summer period? Do you have planned vacations?

My summer vacations are starting from 4th May to 19th July, and the official GSoC period is from 4th May to 31st August. In vacations, I can give about 45-50 hours per week and after college starts, I will be able to manage 40-45 hours a week. I am also free on weekends and will keep the community updated about my progress. **I have no other commitments for the summer vacations**, so I can devote most of my time to GSoC.

Contact

Name	Aniket Mathur
Github	Aniket21mathur
Email	amathur@ec.iitr.ac.in
Location	Roorkee, Uttarakhand
Timezone	IST(UTC+5:30)

The timezone will remain the same during the summers. The time I will be comfortable working-

- UTC 0430 - 0730 hrs (IST 1000 - 1300 hrs)
- UTC 0830 - 1330 hrs (IST 1400 - 1900 hrs)
- UTC 1430 - 2030 hrs (IST 2000 - 0200 hrs)

I can start/end my day a couple of hours earlier/late.

Coding experience

Languages- C, C++, Chapel, Python, JAVA, Ruby, Javascript, Php, SQL, HTML, CSS, Verilog, Matlab and Shell programming.

Language	Experience(years)	Level (1-10)
Python	2.5	10
C/C++	2.5	10
JAVA	2.5	8
Javascript	2	7
Php	1.5	6
Ruby	1	5

Chapel

I started coding in Chapel a couple of months ago. To become familiar with Chapel, I did the following ;

- Went through the Learn X in Y minutes docs for Chapel, and practiced the coding examples side by side.
- Went through the official online documentation.
- Followed some Youtube lectures on Chapel.
- I made some pull requests to the Chapel project(listed below).
- Worked with some modules of Chapel like **ZMQ**, **LinearAlgebra**, **Random**, **Map**, **Math**, **DistributedBag** and **DistributedDeque**.

As a result, I am quite comfortable in writing and understanding code in Chapel now. I will continue to learn Chapel for the next 1 month as well.

Frameworks- Django, Django REST, Flask, Flask RESTful API, React, Node.js, Node.js RESTful API.

Framework	Experience(years)	Level(1-10)
Django/Django REST	1.5	9
Node.js/Node.js RESTful	1.5	9
Flask/Flask RESTful	0.75	7
React	1	6

Projects

I am well versed in writing and understanding code in C/C++. Being a member of **InfoSecIITR**(Information Security Group), I am also familiar with **reversing** and **pwning** which includes understanding compiled C/C++ code. This provides me a better understanding of C/C++ structure and libraries. I also write code for competitive programming in C++, I have experience working with both advanced and basic data structures in C++.

Projects in **C/C++**

- A simple **Search Engine for Library**, implemented in C++.
- A wrapper taking data from a **Verilog** code and interacting with **Bash** to perform some operations.
- Implemented **Round Robin Scheduling** in C.
- Implemented **First Come First Serve(FCFS) Scheduling** in C.

Projects on **Parallel/Distributed** Computing

- A basic implementation of **Secure Data Transfer** using Image Steganography.
- A **Web System** with DNS server, Web server, and client program(Used **sockets in C**).
- A **Storage service** using GET, PUT and DELETE operations.

Projects on **Compiler Development**

- Implemented a **Lexical Analyser**, a **Parser** and a **Transpiler** based on Javascript following this tutorial.
- Implemented **Recursive Descent Top-Down** parsing.

I also have experience in working with REST frameworks and socket programming through various internships and projects. This provides me a better understanding of how a web service library should be designed and developed, what functions should be included in the library to make it easier for a web service developer.

Projects/Internships involving the use of **REST API** and **Socket programming**

- **MoodCafe App-**
  An internship with Moodcafe in which I worked as a backend developer. Improved existing APIs and added new features like Group Chat, switch between counselor and user APIs, counselor test APIs, etc.
  Techstack- Node.js RESTful, MongoDB and socket.io

- **IIT Roorkee Alumni Portal-**
  The new official alumni portal of IIT Roorkee. I worked as a frontend and backend developer in this project. Developed the database structure and API functions. Also coded some part of the portal frontend.
  Techstack- Django REST and React

- **Credify App-**
  An internship with Fredit Financial Tech Pvt Ltd. in which I worked as a backend developer and started the project almost from scratch. Added new user APIs, also integrated OTP, Bank, and Payment service APIs. I wrote the loan distribution algorithm of the app.
  Techstack- Flask RESTful and MongoDB

- **D3 Connect-**
  An application developed during a 24-hour hackathon organized by UST Global, based on connecting people with similar interests and resume. Also had one-to-one and group chat services.
  Techstack- Flask RESTful, Postgresql and Sockets in Python

- **Chat Server-**
  A personal mini-project, in which I implemented a one to one RSA encrypted Chat application.
  Techstack- socket.io

Familiarity

- **git/make-** I am using these tools for almost 2 years in my development projects. I am quite comfortable using them.
- **gdb/valgrind-** Being an information security enthusiast, I use these debuggers for reversing compiled files and finding out any memory leaks.
- **gcc-** I use the gcc compiler system for compiling the C++ code that I write, and is quite familiar with it.

Experience

I am exploring the field of software development for 2.5 years now. In that span of time, I have worked with a number of developers having different skill sets and expertise. My major involvements are-

Sugar Labs (Member)
I am an active member of the Sugar Labs organization. Sugar Labs is a community-run software project whose mission is to produce, distribute, and support the use of Sugar, an open-source desktop environment and learning platform. Sugar Labs is a member of the Software Freedom Conservancy, an umbrella organization for free software projects. I am contributing to Sugar Labs since December 2018. I was also a student developer under the **Google Summer of Code 2019** program for Sugar Labs. I was also a mentor in **Google Code-in 2019** from Sugar Labs. Being a member of this great organization, I have learned open-source ethics, opening and reviewing pull requests, discussing on threads and submitting issues.

SDS InfoSec (Member)
I am also a member of SDS(Software Development Section) InfoSec, a group of information security enthusiasts from IIT Roorkee. We actively participate in CTF(Capture the flag) competitions and organize talks related to information security. I have also designed various CTF challenges for https://backdoor.sdslabs.co. Being a part of this group I learned that quality software can only be built if you are well versed in exploiting and finding bugs in it. **My knowledge of Web security will help me in building a secure web service library for this project.**

Institute Alumni Relations Cell (Joint Secretary)
I am the Joint Secretary of the technical team of the cell working on maintaining the existing and developing new official portal for alumni of the Indian Institute of Technology, Roorkee. Being a part of this team, along with **web development skills**, I have learned **how to lead** and make a group of people work together as a team.

The biggest project that I have worked on is **Port to Python 3** as a student developer under **Google Summer of Code 2019** for **Sugar Labs**. The aim of this project was to extend the compatibility of the sugar environment to Python 3. The major parts of the project were to port all static telepathy bindings to TelepathyGLib, port core sugar (sugar-toolkit-gtk3, sugar-desktop, sugar-datastore, and sugar-artwork), sugar activities and **gwebsockets**(A websocket server written in python. It uses GIO for network communication and hence it

easily integrates with the GLib mainloop) to Python 3. The project involved working with Python and its libraries like telepathy, TelepathyGLib, dbus, six and tools like gdb/pdb, dbus-monitor for debugging and **Wireshark for analyzing packets emitted from web sockets**.

Along with the various technologies mentioned above, from this project I learned how to work in a disciplined manner. Submitting weekly reports, giving daily updates, group discussions and deadlines were a part of it. This project helped me to improve my code quality and developing standards. My major role in this project was to maintain smooth working and to have good progress over time by keeping a great interaction with my mentors. The entire report of my work can be found here.

Open-Source

I have been contributing to open-source since 2018. The projects I have worked upon are public on my Github page. My major open source contributions are to **Sugar Labs**, which can be found here. I have also contributed to **mushorg(snare/tanner)**, **DefectDojo, syslog-ng, IARC** and **InfoSecIITR** organizations.
I started contributing to **Chapel** from January 2020, below is the list of contribution I made to the organization-

Pull Requests(PRs)

Pull Request	Fixes	Description
#14828(Merged)	**#14725**	Extend the LinearAlgebra.eig function to support complex-valued matrices as well.
#14857& #15156 (Merged)		Grammar fixes in the chapel documentation.
#15154(Open)	**#14865**	Implement a version of Random.choice function that returns indices instead of elements.
#15145(Open)	**#5753**	Improve _expBySquaring Algorithm in LinearAlgebra module.
#15135&#14872 (Merged)	**#11417**	Add throws tags to functions in Random, Path and Spawn modules.
#14881(Merged)	**#13336**	Modifies the "Updating mason-registry" message to specify the registry name
#15140(Open)	**#14388**	Implements parallel merge shuffle for Random.shuffle.
#14869(Merged)	**#13257**	Add RVF pragma to DistributedDeque and DistributedBag records.
#14863(Closed)	**#11587**	Change language_level to default.
#15221(Open)	**#15218**	Standardize string/bytes buffer allocations in ByteBufferHelper.
#15222(Open)	**#15217**	Change implicit type to explicit type in ByteBufferHelper.

#14964(Open)	**#14571**	Redesign getCurrentTime function.

I am also working on building a **gzip** library for Chapel.

Issues

Issue	Description
#15159	The array reindex function fails for a domain with negative stride.
#14963 (Duplicate)	Zero-based indexing for tuples and other cases.
#15227 (Duplicate)	Support type coercion within the array literals.

Survey

Had you heard about Chapel before the Summer of Code? If so, where? If not, where would you advise us to advertise?

Yes, I heard about Chapel before Summer of Code. One of my friends has an intern at Australian National University, his project includes the use of Chapel for memory management. He told me about Chapel and its benefits which motivated me to start contributing to Chapel and thus pursue to apply for GSoC in Chapel.

Though, the biggest way to advertise is to increase the use case of the language. After the successful completion of this project, Chapel will have a web service library and its use case will surely expand, and so its number of users.

What was the first question concerning Chapel that you could not find an answer to quickly?

The first question that I could not find an answer to quickly was how to install and use LAPACK/BLAS packages for compiling LinearAlgebra. I found the answer to this question by discussing it with Ben on Github thread and self-exploring through Google search.

What will keep you actively engaged with the Chapel community after this summer is over?

I will keep myself engaged with Chapel after this summer is over. Great response and guidance from quality developers here will keep me motivated. I would keep working on and improving the web service library, after the summer. I will also work on improving the user documentation for the same.

Are you applying to any other organizations for this year's Google Summer of Code? If so, what is the order of your preference, in case you are accepted to multiple organizations?

No, I am not applying to any other organization for this year's Google Summer of Code.

Prerequisites

What operating system(s) do you work with?
I work with Ubuntu 18.04 and Kali Linux.

Are you able to install software on the computer you plan to use?
I am planning to use Ubuntu 18.04 for this project. I am using Ubuntu for almost 2 years now. I am comfortable in installing any software on my system. **I was also able to successfully set up Chapel in my system**.

Will you have access to a computer with an internet connection for your development?
Yes, I will have access to a computer with a good internet connection for the development period.

Self-assessment

What does useful criticism look like from your point of view as a committing student?
As a committing student, I think one should accept useful criticism, and should use it as a guide for improvement and better implementation. There is no harm in accepting if you are wrong.

What techniques do you use to give constructive advice? How do you best like to receive constructive feedback?
I like to give any kind of advice by side by side reviewing the current work and pointing out the advantages and disadvantages of the current implementation, also highlighting what benefits we can achieve by doing it the other way. I like to receive constructive feedback in the same manner, it helps in having healthy and productive communication.

What is your development style? Do you prefer to figure out/discuss changes before you start coding? Or do you prefer to code a proof-of-concept to see how it turns out?
I prefer to code a proof-of-concept and then submit it for review, making further changes as suggested. I have also demonstrated this in my pull requests to the Chapel project. Though I have learned a valuable thing from the Chapel developers that a design issue should be discussed before start coding it.

The task

The task here is to create a library to support writing web services in Chapel, which will allow Chapel to fit into more use cases. It will also involve creating libraries to create a web server in Chapel and libraries to support socket programming in Chapel.

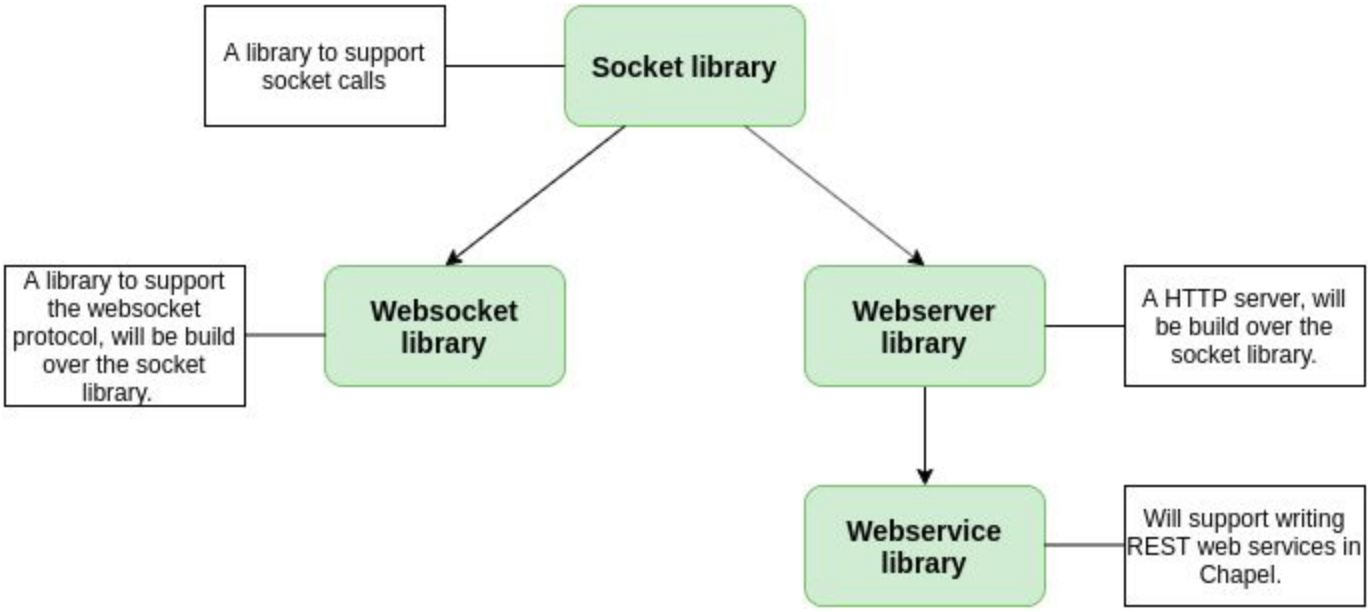
The key elements that I will focus on during this project include-

- Library to support socket programming in Chapel.
- Building a library to create a web server in Chapel.

- Creating a library to support writing REST web services in Chapel.
- Creating a library to support websockets in Chapel.

Description

After going through various documentation and articles on sockets, web services, web server and websockets, I came up with the following flow for the project-



Sockets

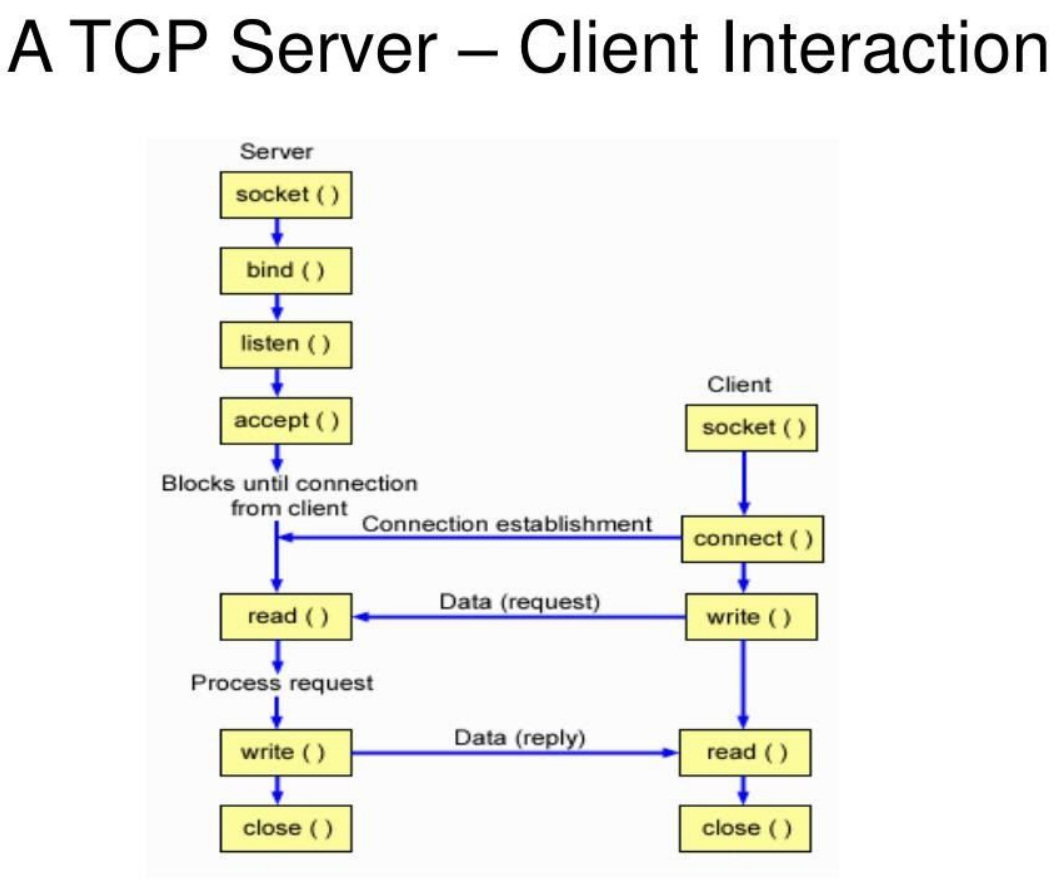
To revise the concepts of sockets programming in C, I followed Youtube tutorials and a simple client-server interaction tutorial. I also looked at the sources of socketmodule in C.

Sockets allow communication between two different processes on the same or different machines. A client server application framework uses **Unix Sockets**. The system calls that are involved in establishing a connection between a client and a server are-

- Socket() - Creates an endpoint for communication.
- Bind() - It assigns an address to the created socket(IP address and Port address).
- Listen() - Marks the socket ready to be used for accepting incoming message requests.
- Connect() - Connects the socket to a specified address.
- Accept() - Used to accept a connection request.
- Send(), Recv() - Used for exchanging data.
- Close() - Closing a socket connection.

There are two widely used socket types: stream sockets and datagram sockets. The stream sockets use the TCP(Transmission control Protocol, **SOCK_STEAM**), while the datagram sockets use UDP(User datagram Protocol, **SOCK_DGRAM**).

A TCP client-server model is shown below-



Chapel needs to have a socket module with **non-blocking calls and with the ability to use multiple threads**. The module can be built by taking Python's socket(wrapper module for the C extension) as a reference.

A pseudo script in Chapel to **connect to Google using socket,** after the socket module is implemented is shown below-

```
use socket;

// Make a new instance of the socket class defined in the socket module.
var sock = new Socket();

// Create a new socket instance with IPV4 address and TCP protocol.
var s = sock.socket(socket.AF_INET, socket.SOCK_STREAM);
```

```
// Specify port address for the socket.
const port = 80;

// Get host ip address.
// gethostbyname will be implemented in the socket module.
var host_ip = sock.gethostbyname('www.google.com');

// Create a socket connection.
s.connect((host_ip, port));
```

Web server(HTTP server)

A web server is a piece of software designed to serve web pages/web sites/web services. Examples are IIS, Apache and many more. The basic objective of the web server is to store, process and deliver web pages to the client. This intercommunication is done using **Hypertext Transfer Protocol** (HTTP). A web server uses HTTP over **TCP as a transport layer**.

An HTTP server must return a response in the same format as of the request from the HTTP client.

Ordinary server

```
char* response = "Hello world!";
```

HTTP server

```
char *response = "HTTP/1.1 200 OK\nContent-Type: text/plain\nContent-Length:
12\n\nHello world!";
```

In HTTP the request is always initiated by the client, so it's **unidirectional** and it supports only **half-duplex** channels. HTTP supports 9 methods-

- GET - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- HEAD - The HEAD method asks for a response identical to that of a GET request, but without the response body.
- POST - The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- DELETE - The DELETE method deletes the specified resource.
- PUT - The PUT method replaces all current representations of the target resource with the request payload.

- CONNECT - The CONNECT method establishes a tunnel to the server identified by the target resource.
- OPTIONS - The OPTIONS method is used to describe the communication options for the target resource.
- TRACE - The TRACE method performs a message loop-back test along the path to the target resource.
- PATH - The PATCH method is used to apply partial modifications to a resource.

The web **server module for chapel can be written over the socket module** designed above for Chapel. For writing a server module we can take **Python3's server and socketserver modules as a reference.** I have also gone through the guide for implementing a **multithreaded web server in Rust** and the sources of **libhttp C library**. We can also follow this guide for a start.

I have also found some example servers like chapel-http and chearch written in Chapel over Github.

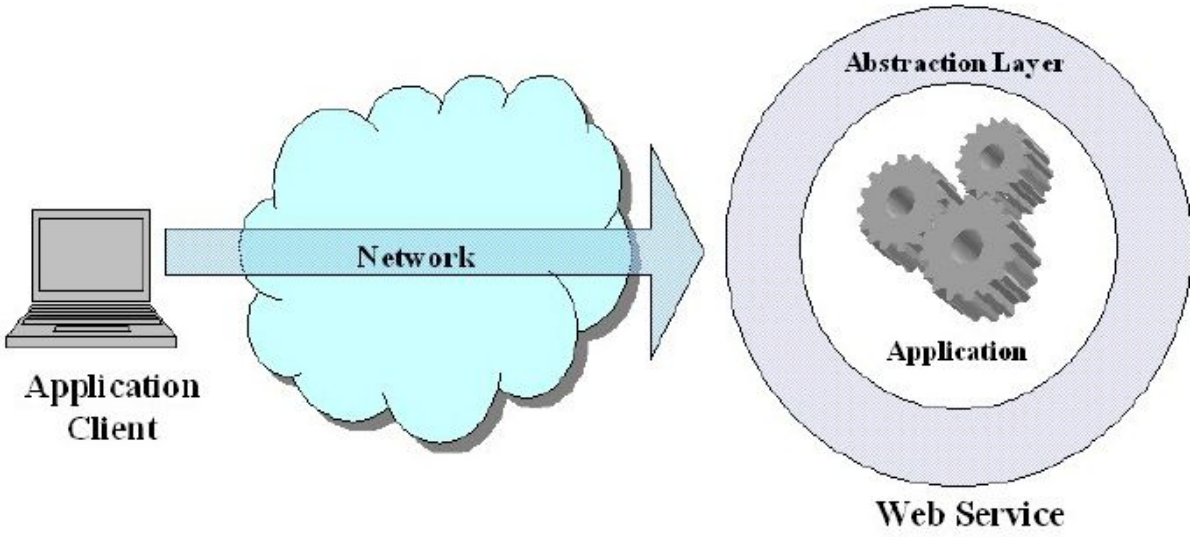
I have implemented a basic interface(pseudocode) of **HTTPRequestHandler** class in Chapel. It is just a general structure and not the complete working code. During implementation, the class might include more functions.

```
use socket;

module server {

  // Class to handle HTTP requests
  class HTTPRequestHandler {

      proc parseRequest(req) {
        /*
        Logic to parse the request.
        Examining headers and determine protocol version
        */
        if req.method == "GET" {
          handleGet(obj);
        }

        if req.method == "HEAD" {
          handleHead(obj);
        }

        if req.method == "POST" {
          handlePost(obj);
        }
```

```
        if req.method == "DELETE" {
          handleDelete(obj);
        }

      }

      proc readFile(obj) {
        //Logic to read file requested by the client.
        returnResponse(resp);
      }

      proc modifyFile(obj) {
        //Logic to modify file requested by the client.
        returnResponse(resp);
      }

      proc handleGet(obj) {
        //Logic to handle get request.
        readFile(file);
      }

      proc handleHead(obj) {
        //Logic to handle head request.
        readFile(file);
      }

      proc handlePOST(obj) {
        //Logic to handle post request.
        modifyFile(file);
      }

      proc handleDelete(obj) {
        //Logic to handle delete request.
        modifyFile(file);
      }

      proc returnResponse(resp) {
        // Logic to generate response
        return resp;
      }

}

}
```

Web services

A *web service* is a network-accessible interface to application functionality, built using standard Internet technologies. It acts as an abstraction layer, separating the platform and programming-language-specific details of how the application code is actually invoked. This standardized layer means that any language that supports web service can access the application's functionality.



Libraries for web services provide the infrastructure that allows us to process and create responses to the request that the web-server received. Once we have processed and generated a response, we hand it off to the web server and it sends it back to the client.

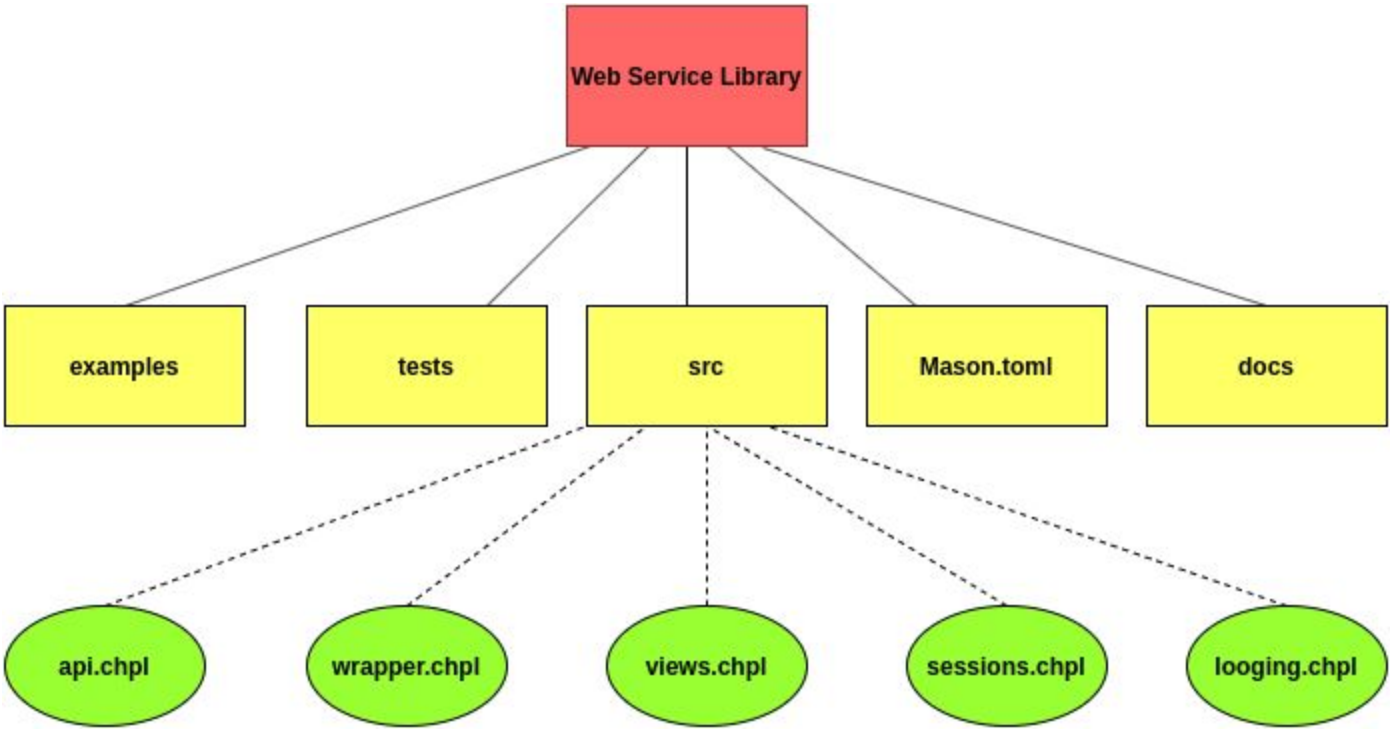
The architectural style, **REST** (REpresentational State Transfer) is by far the most standardized way of structuring the web APIs for requests. REST is purely an architectural style based on several principles. The APIs adhering to REST principles are called RESTful APIs.

REST APIs use a request/response model where every message from the server is the response to a message from the client. In general, RESTful APIs use HTTP as its transport protocol. For such cases, **lookups** should use **GET** requests. **PUT**, **POST**, and **DELETE** requests should be used for **mutation, creation**, and **deletion** respectively**.**

Proposed Web service Library Structure

The web service library for Chapel can be **built over the web server module**. I went through a blog series on how to build a web framework for Python. I also studied the source code and structure of **Django**, **flask and Rocket** frameworks and then designed a proposed structure for the Chapel web service library.

(**This is a proposed structure, since it is a design issue I would like to discuss it with my mentors before finalizing things.**)



Components

- **examples** - code example for the library.
- **tests** - tests for the implemented functions.
- **Mason.toml** - A manifest file, all the additional metadata about the packages as well as the dependencies are listed in this file.
- **docs** - documentation of the functions implemented in the library.
- **src** - key source files of the library.
    - *api.chpl* - The module that will be the core module of the library. This module will have all types of **request handlers** and support for **simple and parameterized routes**.
    - *wrapper.chpl* - A wrapper module for the Chapel web server, it will handle the requests coming from the web server and will return the response in a format the web server expects. This will ensure the **compatibility of the library with the web server**.
    - *views.chpl* - This module will have support for **Django like class based handlers**. The api.chpl file will have function-based request handlers

.

- *sessions.chpl* - This module will provide support for **cookie-based sessions** for the Chapel library.

- *logging.chpl*- This module will provide **support for logging**.

WebSockets

WebSockets allow both the server and the client to push messages at any time without any relation to a previous request.

To originate a WebSocket connection a client sends an HTTP request to the server and the server sends a response to indicate that the protocol is changing from HTTP to WebSocket.

WebSocket solves a few issues with HTTP:

- Bi-directional protocol — either client/server can send a message to the other party
- Full-duplex communication — client and server can talk to each other independently at the same time.
- Single TCP connection — After upgrading the HTTP connection in the beginning, the client and server communicate over that same TCP connection throughout the lifecycle of WebSocket connection.

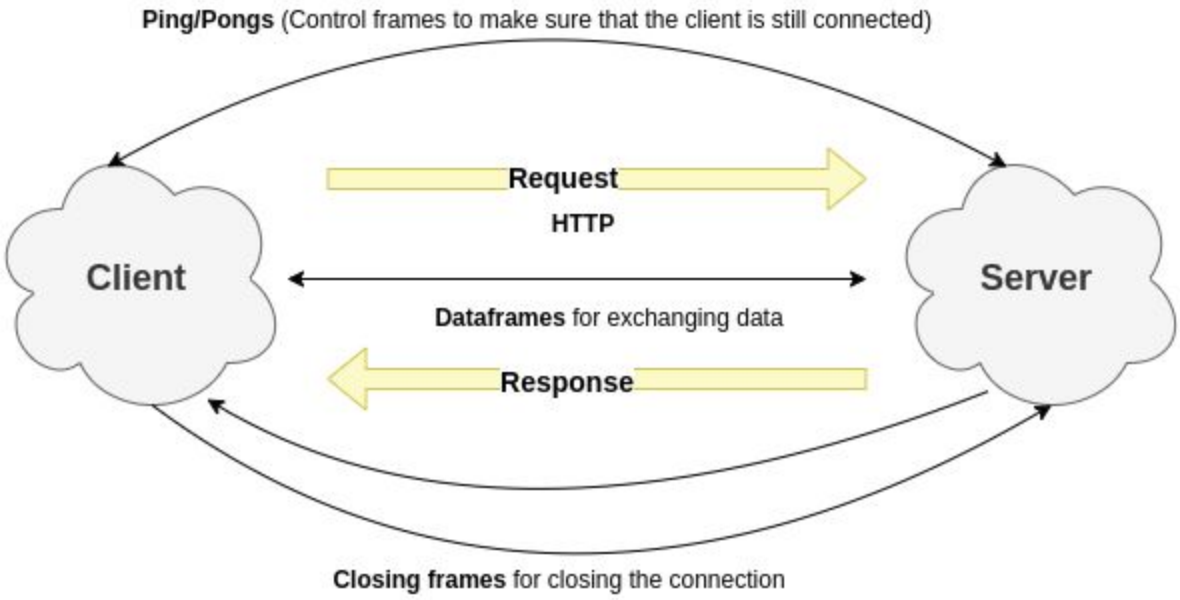
Request headers sent by the client

```
GET /chat HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

Response headers sent by the server

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

A simple representation of client-server interaction over WebSockets-



The WebSocket library for Chapel can be **built over the socket module**. I have experience working with the gwebsockets module for Sugar Labs. I read about **libwebsockets** in C. I also went through this article of building web sockets from scratch in ruby. The Python websocket can be taken as a reference to build the WebSocket library.

Motivation for the task

I have been working and exploring the fields of web development and web exploitation/security for a considerable time now. I have experienced both the development and exploitation of API and web services. This task particularly excites me because it will provide exposure to both sides of my skill sets. This task is a great opportunity for me to **use my complementary skill sets together** in a single project.

I have chosen this particular task, as this task will provide me with an amazing learning experience, it matches with the field of my interest, at the same time it also **seems like an important task and a step ahead for the Chapel project**, as this task will greatly increase the use case of the Chapel language.

This task will entitle me with the knowledge of how exactly the popular web frameworks like Django, flask, and Node.js are developed. By the end of the summers, I will have a good understanding of how sockets and web servers work, I will get to know about their internal functioning. I will also get the opportunity to work with experienced and knowledgeable Chapel developers and other fellow students and I will, for sure, get to learn a lot of things from them. This project would be a major advancement for me in the open-source community.

Timeline

Community bonding period	
4 May, 2020 to 31 May, 2020	● Continue contributing to the Chapel project. ● Continue to learn about sockets, web servers, WebSockets and web service libraries. Discuss project design problems with mentors and get their feedback. ● Get involved with the Community. ● Get prepared for the coding period. Get mock structures and resources ready for the libraries to be implemented.
Coding Period	
1 June, 2020 to 15 June, 2020	● Discuss with mentors and finalize the design of the socket library. ● Work on building the socket library. ● Add tests and examples for the library. ● Make bug fixes and code refactoring.
16 June, 2020 to 26 June, 2020	● Discuss with mentors and finalize the design of the web server library. ● Work on building the web server library. ● Add tests and examples for the library. ● Make bug fixes and code refactoring.
27 June, 2020 to 30 June, 2020	● Fix issues in the implemented libraries. ● Discuss with mentors and finalize the design of the web service library.
Phase One Evaluation	
1 July, 2020 to 11 July, 2020	● Implement the **wrapper** module for the web service. ● Implement the **api** module having **request handlers** and support for **routes**. ● Add tests and examples for the modules.
12 July, 2020 to 15 July, 2020	● Implement **views** module to support **class-based handlers**. ● Add tests and examples for the modules.
16 July, 2020 to 25 July, 2020	● Implement a module to support **cookie-based sessions**. ● Add tests and examples for the modules.
26 July, 2020 to 30 July, 2020	● Discuss with mentors and finalize the design of the web socket library.

	● Start working on building the web socket library.

Phase two evaluation	
31 July, 2020 to 10 Aug, 2020	● Complete the implementation of the WebSocket library. ● Add examples and tests for the library.
11 Aug, 2020 to 15 Aug, 2020	● Document the complete project. ● Make bug fixes and code refactoring in the implemented libraries.
16 Aug, 2020 to 24 Aug, 2020	● Buffer period to complete any pending tasks. ● Implement **logging** module for the web service library. ● Start work on adding **https** support to the web server library. ● Make bug fixes and code refactoring in the implemented libraries
25 Aug, 2020 to 30 Aug, 2020	● Buffer period to complete any pending tasks ● Improve **error handling** in the implemented libraries, though I will try to take care of error handling during the course of development. ● Make bug fixes and code refactoring in the implemented libraries.
Post GSoC period	
Continue to work on the project, complete any remaining work from the summers. Work on additional features and maintain the project.	

Contributor Agreement

I have already submitted a CLA. Here is a link for the same.

References

- https://www.oreilly.com/library/view/programming-web-services/0596000952/ch01.html
- https://journals.ala.org/index.php/ltr/article/view/4457/5193
- https://www.tutorialspoint.com/restful/index.htm
- https://www.geeksforgeeks.org/socket-programming-python
- https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers
- https://medium.com/platform-engineer/web-api-design-35df8167460
- https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa