1 TENXWebDev

Hello and welcome to University Malaya! You're at the beginning of an exciting Computer Science journey. And must remember, the self-esteem of a CS grad is directly proportionally to how well they can code (even if it's not the main point of the degree xD). Also, a large portion of you will end up working on user-facing servers and websites. So why not make something that seems complex and blackbox to improve your coding skills and also learn about the web more deeply. To give a overview we will be developing a webserver that does server side rendering with templetes, (similar to PHP, django, etc).

Whether you're coming in with prerequisite knowledge or not, remember that the best programmers are always learning. This assignment is designed to challenge you and help you grow. It'll be a bit harder than other projects, so any prerequisites like having developed a website before or being familiar with concepts such as routing, templates, etc. in websites, will be super helpful but not required if you can learn efficiently. These aren't just Java skills, mind you - any aspiring 10x dev (click me) in front-end, backend, or whatever role will eventually need to learn these.

We've provided you with a starter template to give you a head start and do some heavy lifting as some of you are still beginner. I believe it'll allow you to play with it, make it more lucid, and have a simple baseline to work on. Your task? Extend and improve upon it, creating a project that any Special UM student (each of 700 of you xD) would be proud of.

Scared? This project won't be that tedious, so you won't have to write billions of lines to get JavaFx to work (ask your seniors about that nightmare). But the code you do write needs to be well thought out. It's important because while ChatGPT and other coding assistants can handle the heavy repetitive coding, they can't do the planning for you. So it's okay to take a long time to understand and plan and then write just a few lines that get the job done in a couple of evenings. Focus on improving only few things, but take you time to understand what is happening.

Check out how much functionality is packed into the simple, small codebase provided in the GitHub repo. You'll be surprised! So you can also make cool things if you implement them nicely. Read the comments in the code to understand how they work.

2 Prerequisites and Implemented Features

The starter code implements a basic HTTP server with the following features:

- Basic request parsing and handing a POST and GET
- Simple routing system
- Response generation
- A rudimentary templating system

- Simple context and state management (by connection/session)

It can be used as such:

- Git Clone from https://github.com/soda4fries/Webserver_starter_FOP.git
- Open the project in your favourite IDE (Netbeans or IntelliJ is usually good)
- I created a simple application that uses the webserver to create a simple website. To run the application goto /src/Application/ and run the file. Look at the configuration and you should see it running on port 8080, so visit in your browser localhost:8080/

Now what happened? As you can see you are running a no frills basic (but dynamic mind you) web-server that even has basic session management (set the email and open from incognito and another tab. see the difference). This is more advanced than the first HTTP server that started the internet. We have only used basic sockets and utilities provided in java collections.

3 Question (12 marks)

Your task is to improve different aspects of the web server. It is very open-ended and you can develop a lot based on your skills and how much you understand and learn. If you look at the code they are implemented as interface and this means that for different modules you can try different techniques and implementation and improve them modularly. so issues will be isolated and you can develop choose the part of the server you want to improve and develop them. They will also expose why OOP and code structure is important otherwise the complexity will be hard to manage.Read the code and comment to understand what is being asked to improved

3.1 Basic Features (8 marks)

Implement the following features:

Feature	Marks	Hint
(a) Request Parsing	2	Look for the "?" character in the URL and split the string that follows into key-value pairs.
(b) Static File Serving	2	Check if the requested path corresponds to a file in your static directory. If it does, read and send the file content instead of processing it as a dynamic route.
(c) Deploy a website	2	You can get free hosting by using Azure student credits and use one of their sub-domains.
(d) More Templating Feature	1	Understand the code-base and see how is the rendering actually accomplished. This is the hardest part most likely and others should be much simpler once you start.
(e) Persistent storage	1	Modify the context manager

Basic Features Implementation

3.2 Advanced Features (4 marks)

Choose two of the following advanced features to implement (2 marks each):

Feature	Description
(a) Improved Code base and API	Implement more modular design or better patterns, e.g., filter chains for request handling. Improve GET/POST handling in the webserver.
(b) Enhanced Context Manager	Provide object storage instead of just key-value strings. Implement IoC (Inversion of Control).
(c) Session Management	Implement cookie-based session management with user-specific data access and route restrictions.
(d) RESTful Routing	Support different HTTP methods (GET, POST, PUT, DELETE) for the same route, implementing a basic RESTful API.
(e) Request Logging and Analytics	Create a logging system for incoming requests and implement a simple analytics page.
(f) MIME Type Handling	Implement proper MIME type detection, compression, and TLS for static file serving.
(g) Advanced Templating	Implement additional templating features inspired by other templating engines and server-side renderers.

Advanced Features Implementation Options

Students are encouraged to be creative and expand upon these ideas. The

implementation should focus on moving functionality to the Web server and providing a pleasant API for application developers to use.

For the advanced feature, explain your approach, any challenges you faced, and how your implementation could be further improved or expanded.

You will demonstrate (show-off) the features by actually making a website in the application and templete that demonstrates your feature.

4 But How Does the Starter Implementation Work?

1. Whenever you make a request from your browser, it establishes a connection with the webserver. In the project's webserver class, you can see that it listens on a port and creates a new thread for each connection to process the input.

2. After the connection is established, the server reads the incoming stream. Since HTTP is a plaintext protocol, we use a reader (which you will learn about in File I/O) to read and parse the request.

3. We also extract header information. The browser assists by sending required headers and any cookies previously set by the server.

4. Crucially, we parse the URL and check if it matches any of our predefined routes. If a match is found, we use the corresponding handler to process the request, passing along the body and necessary information. On your PC, when you request `c://hello.txt`, it is sent to your application as a stream. Similarly, our webserver examines the URL to determine what needs to be sent. However, this data is generated on the fly based on the route, rather than being read from disk.

5. The handler then loads the webpage template from `Template_dir`, which is configured when initializing the webserver class in the application. It processes the template and returns the response, adding any required headers. Anything enclosed in {{}} in `index.html` and other templates are directives that the template renderer will process accordingly. These can be simple substitutions, as shown in the example, or more advanced. The substitution is actually quite powerful, supporting features like #FOREACH loops, giving you an idea of what's possible.

The files under SimpleWebServer will be where you will develop the webserver. The Application and Templetes folder is just a example website, similar to how anyone can use PHP, Spring to make their own website. So be thoughtful of your API so that it is general case and not just for building your website.

5 Additional Guidance

As you work on this assignment, here are some concepts you may need to learn or refresh and some reason why i am mentioning them:

- URL parsing and query string manipulation, headers: This will basically determine how your request will be handled when it hits your server. For example, as you can see we have **index.html** and another **all_email.html**. When you make a request, they are routed to different functions based on your URL. **localhost:8080/** or `/` is directed to process `index.html` in the add route function. You can also imagine if headers do not have auth cookies, they can be sent to the login page instead of receiving the requested resource.

- Concept of streams (file streams, network (TCP/IP) streams, etc.) : This are important concepts that you will always need to build any realworld application.

- File I/O for serving static files : instead of handling with a templete renderer you may make a magic /static/xxx.pdf route that will serve static files directly. Beware of directory traversal attack. if they request /../xxx.pdf they can access file that are outside the folder if not protected against.

- Base64 encoding/decoding (HTTP is a text-based protocol) if you are using HTTP to serve images and other resources binary data is usually converted to this. If you implement mime types and other function you can make it efficient since base64 encoding usually inflates package by 33 percent

- Regular expressions for advanced string manipulation: if you choose to improve the templating part of your webserver. If you are interesting you can read about parsers, templeting engines, etc. if you choose to improve this part focus less on other part since it is more complex but satisfing. Also regular expression cannot process nested expression (learn more why, hardcore automata theory. Google to start the rabbit hole : Parsing HTML using Regex) so you may need to use other tools.

- HTTP headers and their meanings : As mentioned above. Also see what are the conventions, since browsers handle some part of this. If you find this interesting read (RFCs for example HTTPS RFC)

- Cookies and session management: Servers need to store this to know how much information the user can access. Understand authentication, authorization and how they are different. I have seperated the context class to make it more apparant. so make it more advanced to allow more advanced security features. Manipulate how much data is provided by checking the headers and params etc.

Tips for success:

- **This is for FOP so you should not care about runtime complexity or performance or appropiate data structure or algorithms, rather use clever code to solve the problem and try to get comfortable with libraries and abstractions. Some OOP concepts such as polymorphims, interfaces, etc have already been used and see how they are making the coding much more modular. So try to use them**
- Ensuring your code is thread-safe. While not required for a basic implementation, consider using concurrent data structures from the `java.util.concurrent` package. Think about why these might be necessary in a web server environment. Even if you don't fully understand how they work internally, using them can make your server more robust when handling multiple simultaneous requests. Search the terms (Thread-safety) and try to analyse which data can be accessed from different threads. No need to optimize them rather use blocking things like synchronized methods or non blocking concurrent collection to prevent bugs. Focus on why you are using them not how they work. Most collection datastructure tools have a thread-safe counterpart (Java is actually very praised for the good implementation of this) .
- Balancing between code simplicity and feature richness. Remember you cannot do everything but you can do a lot. So choose which part you want to improve judiciously
- Break down complex tasks into smaller, manageable pieces, **get comfortable with the debugger**
- Test your code frequently, especially after implementing each feature
- Use version control (like Git) to track your changes and experiment safely. Fork my Repo
- Don't hesitate to research and learn new Java libraries that might help you.
- Learn about a web framework if possible. I was inpired by watching vidoes in this channel :Youtube - Spring IO

Remember, the goal is not just to complete the assignment, but to learn and grow as a programmer. Good luck, and enjoy the process of building your web server!

Contact Me

If you are confused or need any help or talk about life contact. Dont ask me about Web dev I actually never made a website due to avoiding JavaScipt:

- Email: rahmanmahinur@gmail.com
- Phone: +60123563426 (Whatsapp)