University of St Andrews

CS4103 Distributed Systems

Implementation of Paxos Synod algorithm

200012756

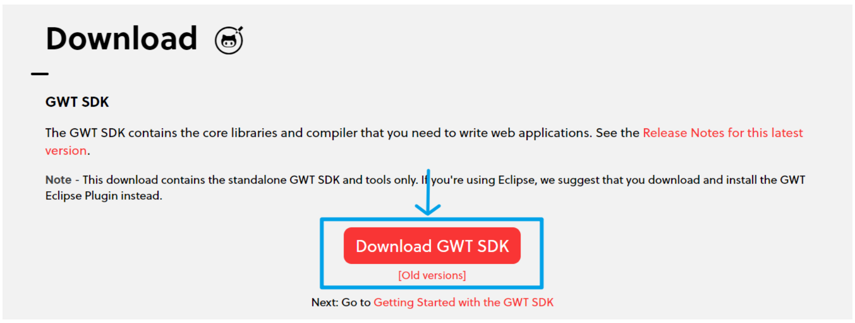
Word count: 1752

Table of contents

1. Installation and execution of a GWT project on a development mode server.

As explained on the GWT website, the library primarily supports development with the Eclipse IDE. Personally, it has been a long time since I used the Eclipse, so it was thought that it would take extra time to learn the usage again, in particular the shortcuts. Alternatively, Intellij was chosen since it is my preferable IDE and it was also found out that the GWT plugin could be installed in the IDE. Although it was a little hard to figure out how to set up the GWT environment in the IDE, Youtube was very helpful which clearly showed a way of setup (Уроки Java, 2019). Some of these conveniences made me select the IDE. The experience of starting the GWT project with the development environment is as follows.

First of all, the GWT SDK was downloaded and a basic project was created by using the webAppCreator command. At this time, Maven parameter was used to create the pom.xml file.



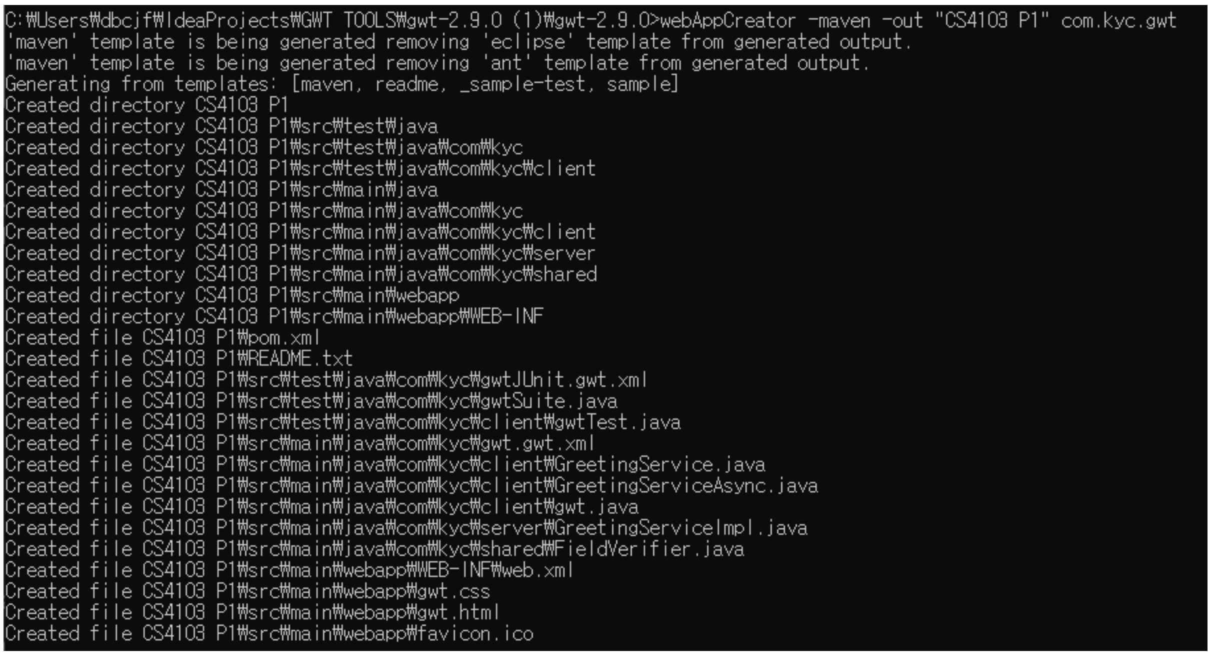
Download

GWT SDK

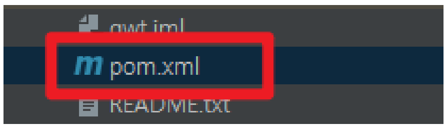
The GWT SDK contains the core libraries and compiler that you need to write web applications. See the Release Notes for this latest version.

Note - This download contains the standalone GWT SDK and tools only. If you're using Eclipse, we suggest that you download and install the GWT Eclipse Plugin instead.

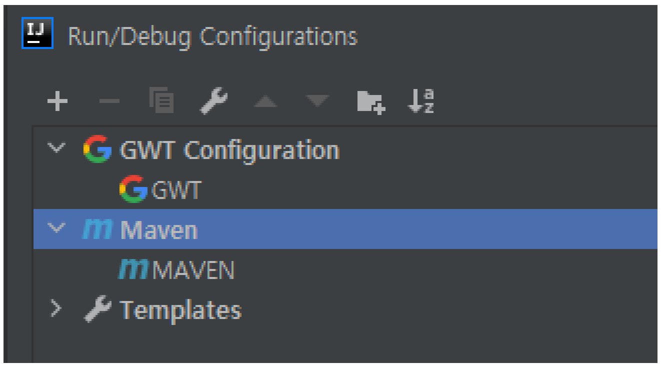
Next: Go to Getting Started with the GWT SDK



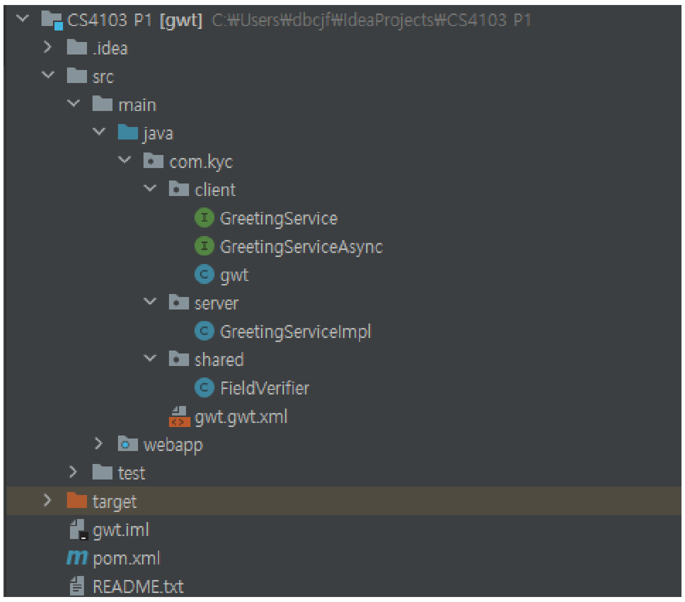
```
C:₩Users₩dbcjf₩IdeaProjects₩GWT TOOLS₩gwt-2.9.0 (1)₩gwt-2.9.0>webAppCreator -maven -out "CS4103 P1" com.kyc.gwt
'maven' template is being generated removing 'eclipse' template from generated output.
'maven' template is being generated removing 'ant' template from generated output.
Generating from templates: [maven, readme, _sample-test, sample]
Created directory CS4103 P1
Created directory CS4103 P1₩src₩test₩java
Created directory CS4103 P1₩src₩test₩java₩com₩kyc
Created directory CS4103 P1₩src₩test₩java₩com₩kyc₩client
Created directory CS4103 P1₩src₩main₩java
Created directory CS4103 P1₩src₩main₩java₩com₩kyc
Created directory CS4103 P1₩src₩main₩java₩com₩kyc₩client
Created directory CS4103 P1₩src₩main₩java₩com₩kyc₩server
Created directory CS4103 P1₩src₩main₩java₩com₩kyc₩shared
Created directory CS4103 P1₩src₩main₩webapp
Created directory CS4103 P1₩src₩main₩webapp₩WEB-INF
Created file CS4103 P1₩pom.xml
Created file CS4103 P1₩README.txt
Created file CS4103 P1₩src₩test₩java₩com₩kyc₩gwtJUnit.gwt.xml
Created file CS4103 P1₩src₩test₩java₩com₩kyc₩gwtSuite.java
Created file CS4103 P1₩src₩test₩java₩com₩kyc₩client₩gwtTest.java
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩gwt.gwt.xml
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩client₩GreetingService.java
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩client₩GreetingServiceAsync.java
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩client₩gwt.java
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩server₩GreetingServiceImpl.java
Created file CS4103 P1₩src₩main₩java₩com₩kyc₩shared₩FieldVerifier.java
Created file CS4103 P1₩src₩main₩webapp₩WEB-INF₩web.xml
Created file CS4103 P1₩src₩main₩webapp₩gwt.css
Created file CS4103 P1₩src₩main₩webapp₩gwt.html
Created file CS4103 P1₩src₩main₩webapp₩favicon.ico
```



Additionally, Run Configuration was added in the IDE to run the project.



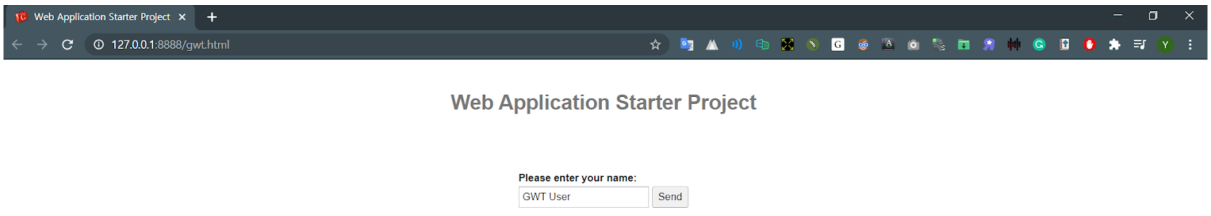
Through this process, the project could be built by the Maven and could be run with the GWT plugin. The picture below shows the creation of the basic project containing RPC service.



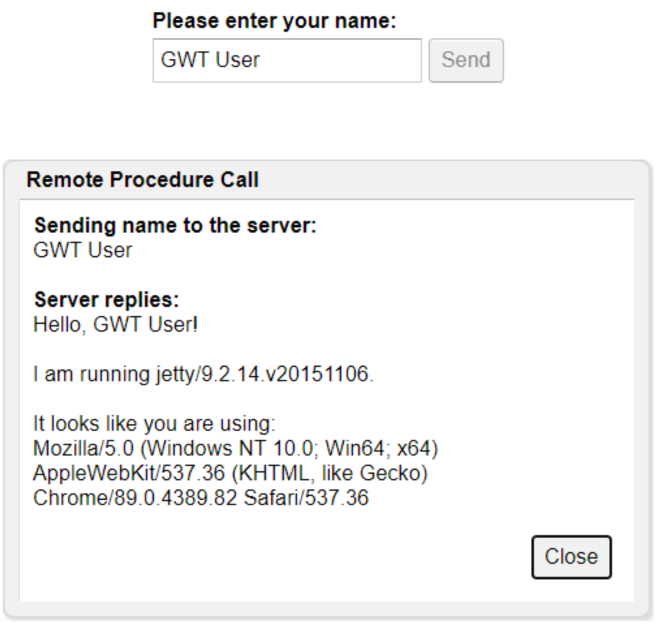
```
CS4103 P1 [gwt]  C:₩Users₩dbcjf₩IdeaProjects₩CS4103 P1
  .idea
  src
    main
      java
        com.kyc
          client
            GreetingService
            GreetingServiceAsync
            gwt
          server
            GreetingServiceImpl
          shared
            FieldVerifier
          gwt.gwt.xml
      webapp
    test
  target
  gwt.iml
  pom.xml
  README.txt
```

Looking at these packages, it can be clearly recognised that the client and shared directories are compiled to Javascript. As one of the directories is shared between server and client, it is compiled to JVM as well.

The picture below is the outcome of executing the basic project above. This showed the run-time communication between the server and client.


Web Application Starter Project
127.0.0.1:8888/gwt.html
Web Application Starter Project
Please enter your name:
GWT User
Send

Talking about the communication, once the send button is clicked, the FieldVerifier in the shared package checks the number of letters in the text box. If the number is greater than a certain number, async call with the letters is sent to the server. The server makes HTML with the sent letters to create a dialog and delivers it to the client as below.



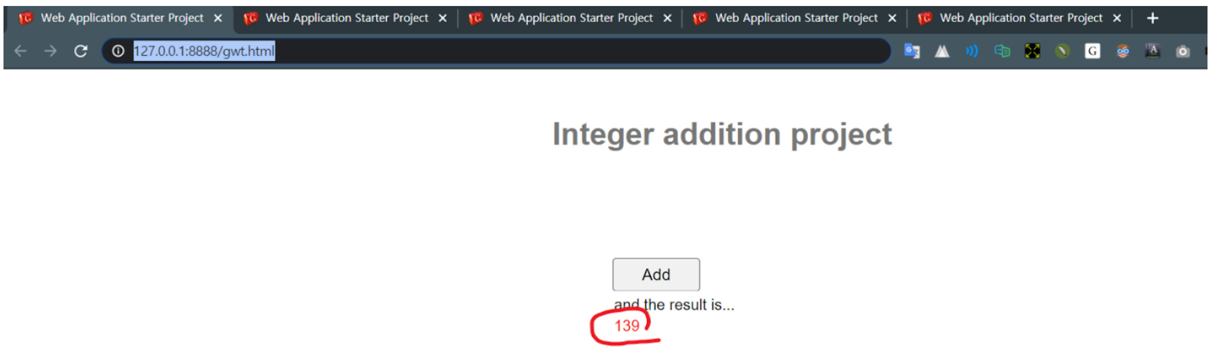
Sending name to the server:
GWT User

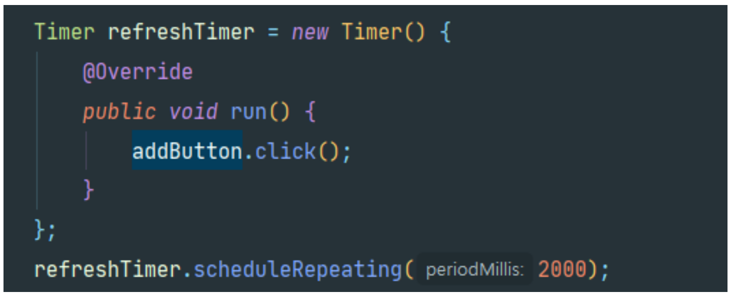
Server replies:
Hello, GWT User!

I am running jetty/9.2.14.v20151106.

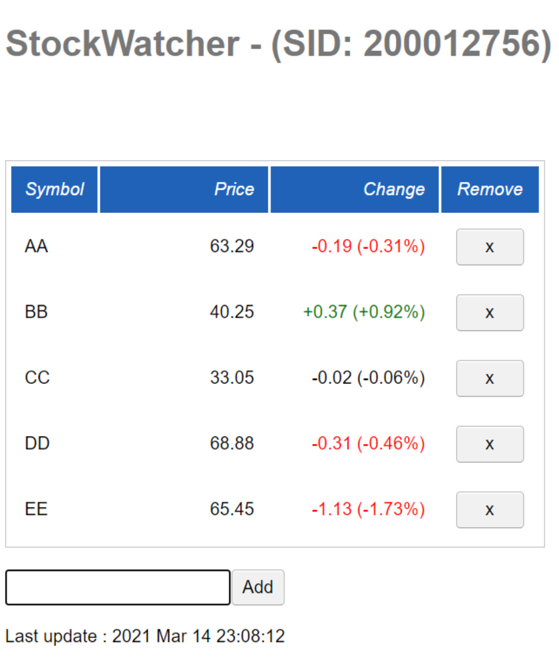
It looks like you are using:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36

This below is the exercise of a concurrency test in the phase 1. On the client side, the add button was set to be pressed once every specific second (used Timer), and on the server side, it returns an incremented integer which is thread-safe int (scalar type).


Web Application Starter Project
Web Application Starter Project
Web Application Starter Project
Web Application Starter Project
Web Application Starter Project
127.0.0.1:8888/gwt.html
Integer addition project
Add
and the result is...
139


```
Timer refreshTimer = new Timer() {
    @Override
    public void run() {
        addButton.click();
    }
};
refreshTimer.scheduleRepeating( periodMillis: 2000);
```

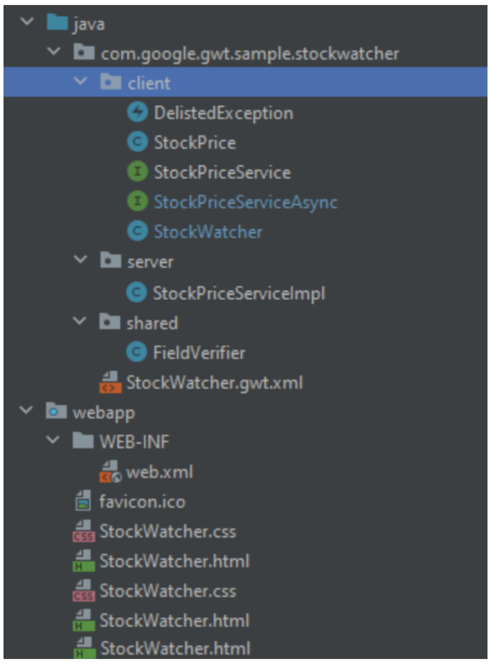
From here on, it is the extended practice which the tutorials on the GWT website were followed to learn more about it. This picture below is an outcome of the tutorial.


StockWatcher - (SID: 200012756)

Symbol	Price	Change	Remove
AA	63.29	-0.19 (-0.31%)	x
BB	40.25	+0.37 (+0.92%)	x
CC	33.05	-0.02 (-0.06%)	x
DD	68.88	-0.31 (-0.46%)	x
EE	65.45	-1.13 (-1.73%)	x

Add

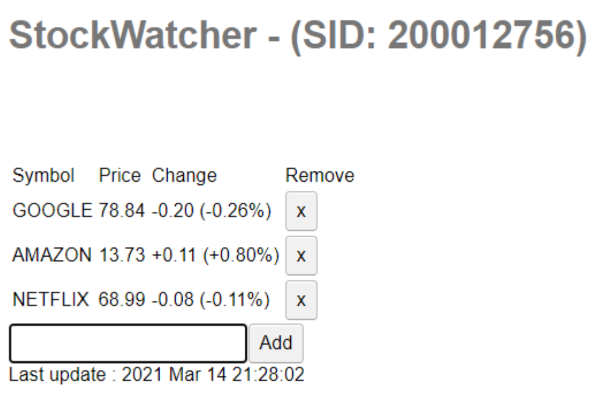
Last update : 2021 Mar 14 23:08:12


```
java
  com.google.gwt.sample.stockwatcher
    client
      DelistedException
      StockPrice
      StockPriceService
      StockPriceServiceAsync
      StockWatcher
    server
      StockPriceServiceImpl
    shared
      FieldVerifier
    StockWatcher.gwt.xml
webapp
  WEB-INF
    web.xml
  favicon.ico
  StockWatcher.css
  StockWatcher.html
  StockWatcher.css
  StockWatcher.html
  StockWatcher.html
```

The purpose of the tutorial was to teach how to manipulate HTML or CSS elements through Java code and communicate with the server through RPC.

A GWT UI element such as FlexTable, for instance, was used to create the table to include the stock information and a GWT method, addStyleName(), was used to apply CSS which changed the colour in the Change field depending on increment and decrement of the stocks (The Timer was used to poll the Price and Change regularly).

During this practice, it was found out that there were two more ways to communicate with a server aside from the RPC. Despiste of those existence, RPC was only used throughout all this practical since it enables the Java objects to be passed efficiently over standard HTTP - “You can use the GWT RPC framework to transparently make calls to Java servlets and let GWT take care of low-level details like object serialization” (GWT, n.d.). These pictures below are a sequence of developing the system.

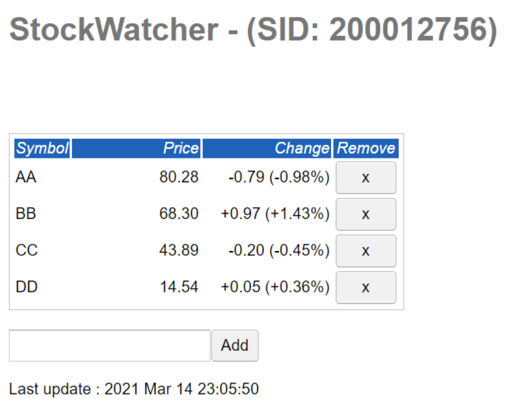


StockWatcher - (SID: 200012756)

Symbol	Price	Change	Remove
GOOGLE	78.84	-0.20 (-0.26%)	x
AMAZON	13.73	+0.11 (+0.80%)	x
NETFLIX	68.99	-0.08 (-0.11%)	x

Add

Last update : 2021 Mar 14 21:28:02

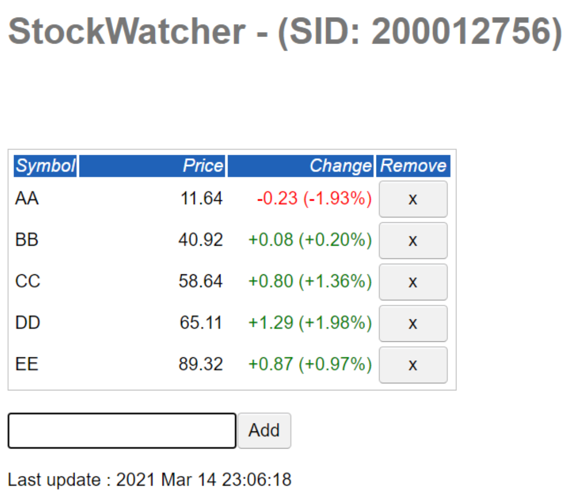


StockWatcher - (SID: 200012756)

Symbol	Price	Change	Remove
AA	80.28	-0.79 (-0.98%)	x
BB	68.30	+0.97 (+1.43%)	x
CC	43.89	-0.20 (-0.45%)	x
DD	14.54	+0.05 (+0.36%)	x

Add

Last update : 2021 Mar 14 23:05:50

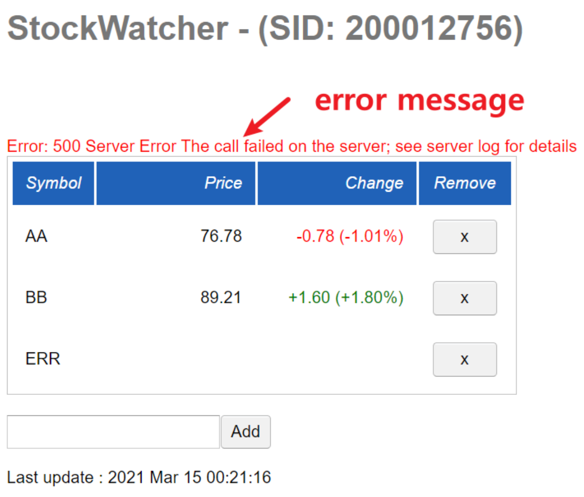


StockWatcher - (SID: 200012756)

Symbol	Price	Change	Remove
AA	11.64	-0.23 (-1.93%)	x
BB	40.92	+0.08 (+0.20%)	x
CC	58.64	+0.80 (+1.36%)	x
DD	65.11	+1.29 (+1.98%)	x
EE	89.32	+0.87 (+0.97%)	x

Add

Last update : 2021 Mar 14 23:06:18



StockWatcher - (SID: 200012756)

error message

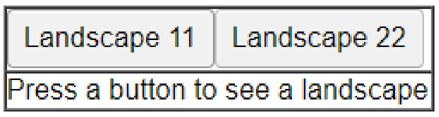
Error: 500 Server Error The call failed on the server; see server log for details

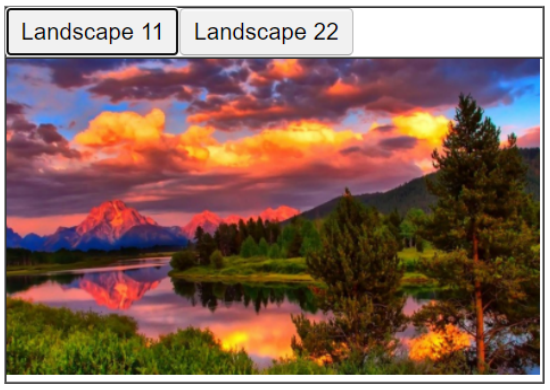
Symbol	Price	Change	Remove
AA	76.78	-0.78 (-1.01%)	x
BB	89.21	+1.60 (+1.80%)	x
ERR			x

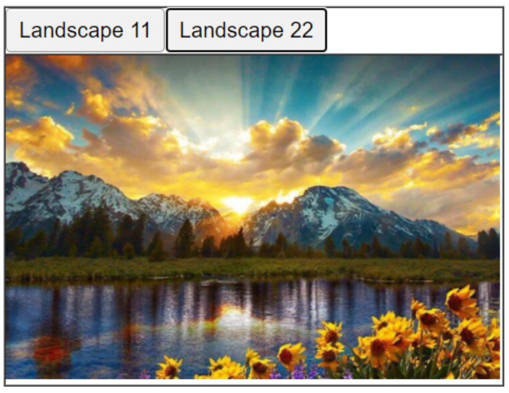
Add

Last update : 2021 Mar 15 00:21:16

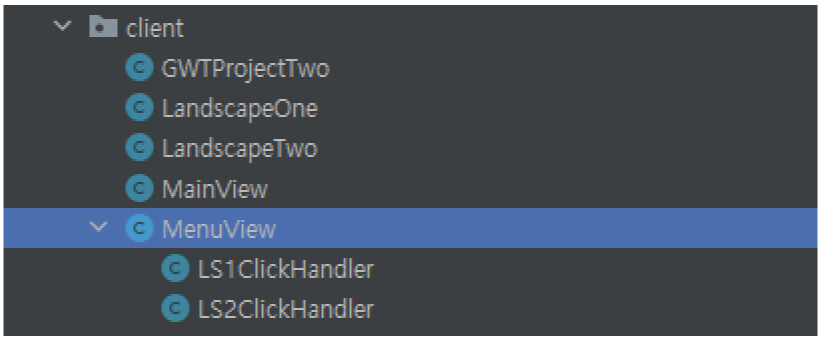
Aside from the exercises above, more learning was conducted with another lecture on YouTube (GWTLecturer, 2012). These pictures below are the outcome.

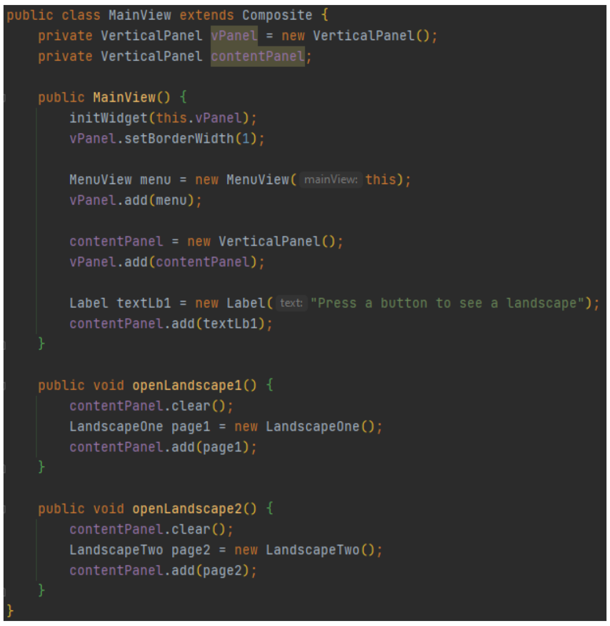




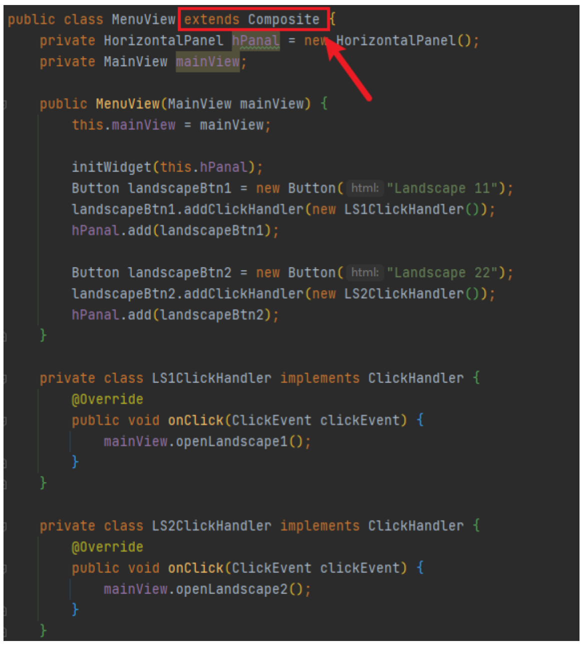


The key point of this lecture was making proper GWT architecture using Composite class to attach a sub view onto the main view. The MenuView, for instance, which extends the Composite and has two buttons, were put into MainView as below.





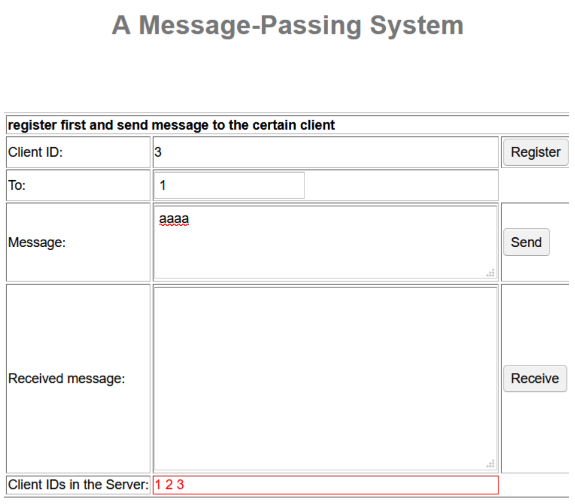
```java
public class MainView extends Composite {
    private VerticalPanel vPanel = new VerticalPanel();
    private VerticalPanel contentPanel;

    public MainView() {
        initWidget(this.vPanel);
        vPanel.setBorderWidth(1);

        MenuView menu = new MenuView( mainView: this);
        vPanel.add(menu);

        contentPanel = new VerticalPanel();
        vPanel.add(contentPanel);

        Label textLb1 = new Label( text: "Press a button to see a landscape");
        contentPanel.add(textLb1);
    }

    public void openLandscape1() {
        contentPanel.clear();
        LandscapeOne page1 = new LandscapeOne();
        contentPanel.add(page1);
    }

    public void openLandscape2() {
        contentPanel.clear();
        LandscapeTwo page2 = new LandscapeTwo();
        contentPanel.add(page2);
    }
}
```



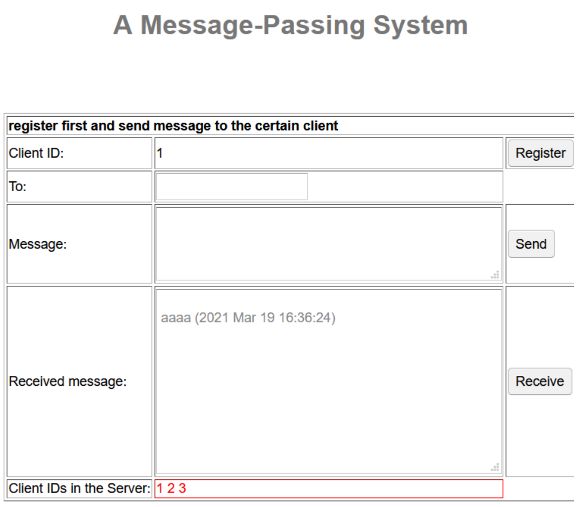
```java
public class MenuView extends Composite {
    private HorizontalPanel hPanal = new HorizontalPanel();
    private MainView mainView;

    public MenuView(MainView mainView) {
        this.mainView = mainView;

        initWidget(this.hPanal);
        Button landscapeBtn1 = new Button( html: "Landscape 11");
        landscapeBtn1.addClickHandler(new LS1ClickHandler());
        hPanal.add(landscapeBtn1);

        Button landscapeBtn2 = new Button( html: "Landscape 22");
        landscapeBtn2.addClickHandler(new LS2ClickHandler());
        hPanal.add(landscapeBtn2);
    }

    private class LS1ClickHandler implements ClickHandler {
        @Override
        public void onClick(ClickEvent clickEvent) {
            mainView.openLandscape1();
        }
    }

    private class LS2ClickHandler implements ClickHandler {
        @Override
        public void onClick(ClickEvent clickEvent) {
            mainView.openLandscape2();
        }
    }
```

2. Implementation of a message-passing system among browser processes.

The pictures below are the outcome of the executed program. Multiple browsers were opened, and each browser was registered. Then, To and Message fields were filled to send a message to a specific browser. The sent message was able to be seen in the destination browser.


A Message-Passing System

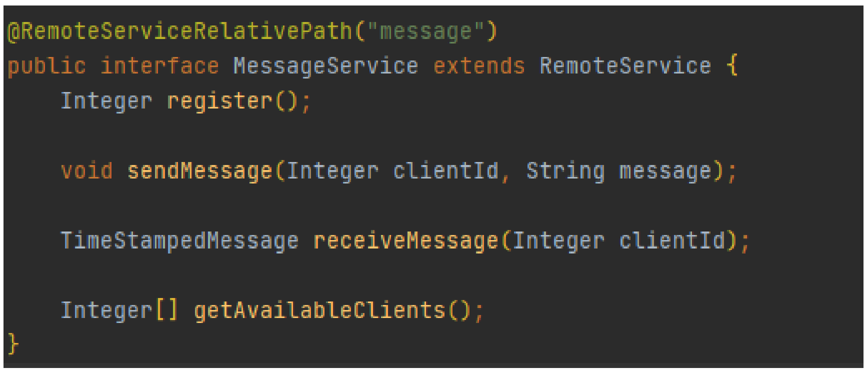
register first and send message to the certain client		
Client ID:	3	Register
To:	1	
Message:	aaaa	Send
Received message:		Receive
Client IDs in the Server:	1 2 3	


A Message-Passing System

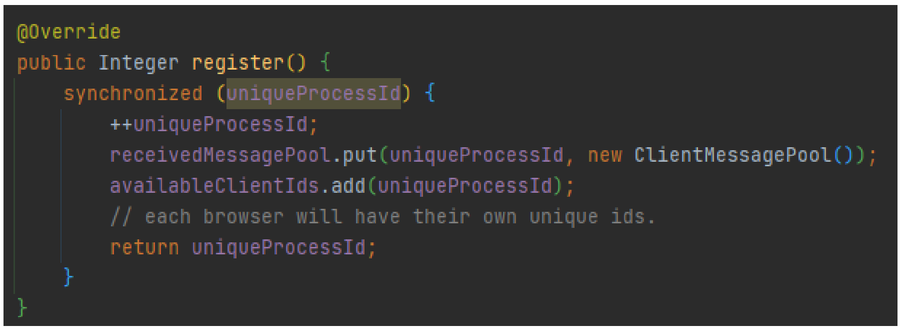
register first and send message to the certain client		
Client ID:	1	Register
To:		
Message:		Send
Received message:	aaaa (2021 Mar 19 16:36:24)	Receive
Client IDs in the Server:	1 2 3	

To enable a message exchange system, each client had to have an address to communicate with each other. If it had operated a peer-to-peer technique without going through a server, it would have been possible to use an IP address. Since the server had to be utilised, a strategy of assigning an id to each client was selected.

When it comes to the sequence of the communication process, there were several tasks that the server should handle. It needed to register the client, and enable them to send and receive messages. As the practical exercise asked to implement the function of retrieving the client ids, it had to be in place as well. As a result, the following methods were created.


```
@RemoteServiceRelativePath("message")
public interface MessageService extends RemoteService {
    Integer register();

    void sendMessage(Integer clientId, String message);

    TimeStampedMessage receiveMessage(Integer clientId);

    Integer[] getAvailableClients();
}
```

Since an Integer was used for the client id, synchronization was needed while registering in order to be thread safe. The code is as follows.



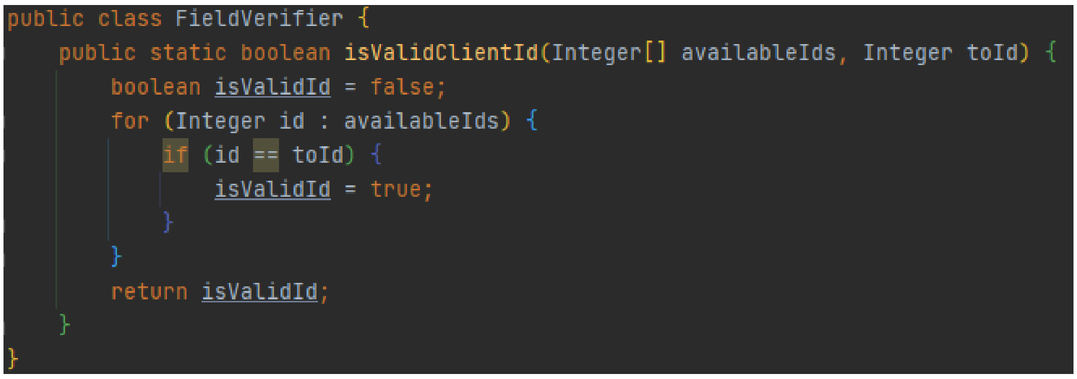
```java
@Override
public Integer register() {
    synchronized (uniqueProcessId) {
        ++uniqueProcessId;
        receivedMessagePool.put(uniqueProcessId, new ClientMessagePool());
        availableClientIds.add(uniqueProcessId);
        // each browser will have their own unique ids.
        return uniqueProcessId;
    }
}
```

The received message pool is initialized for each client during the registration. Clients can send and receive messages via the pool. Map was selected for the message pool in that the key of the map is unique so each client id could be assigned to the key. To be thread safe, Collections.synchronizedMap(~); was used. The map is synchronized every time when the pool is accessed by each client.

The reason why thread safe code is needed is probably to avoid risk of the program's unsafe ending while the program is running. A transpirable error case can be the occurrence of NullPointerException or having the same id in different browsers. This would make the system unusable.

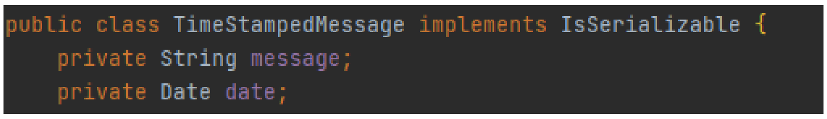
As stated in the requirements ("not the Java Thread"), GWT Timer was used appropriately in this phase. With this, the aim of retrieving messages and available ids from the server was achieved.

The advantage of GWT is that it allows the server and the client to share resources. The FieldVerifier in the shared directory, for instance, was created to check if the input is valid on both sides. When a client sends a message, the client side verifies whether the destination ID is valid. Besides, since the destination id can be falsified in the middle (MITM attack), the server side utilises the validation inspection once more. The verifier is as below.



```java
public class FieldVerifier {
    public static boolean isValidClientId(Integer[] availableIds, Integer toId) {
        boolean isValidId = false;
        for (Integer id : availableIds) {
            if (id == toId) {
                isValidId = true;
            }
        }
        return isValidId;
    }
}
```

The TimeStampedMessage below is another shared resource by both sides. This implements IsSerializable to allow the data to be exchanged between them.

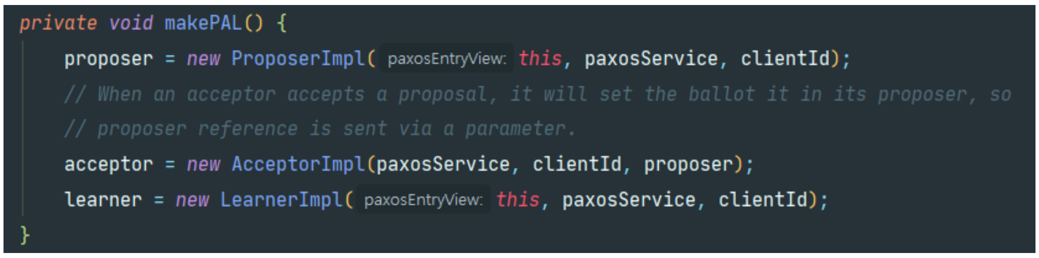


```
public class TimeStampedMessage implements IsSerializable {
    private String message;
    private Date date;
```

A difficulty was faced while transmitting a specific type, List<Integer>, to the client. Client fetched the available client ids from the server and server had to send them, so the List type was sent to client. This brought about an error. To solve this, the code was changed to return an Integer[] rather than List<Integer> and it worked fine. Perhaps this is because List does not implement Serializable, whereas Integer inherits Number which implements Serializable.

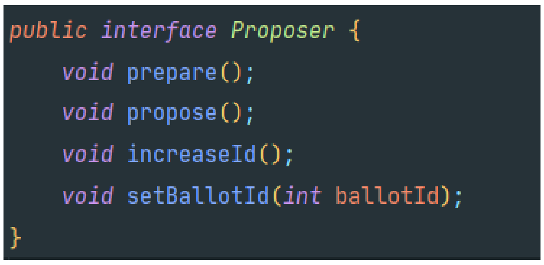
3. Implementation of the Paxos synod algorithm.

In order to apply Paxos to the GWT, the interaction between browsers had to be considered. As we learned in class, in reality the three roles (proposer, acceptor, and learner) would not be completely separated. For this task, it was decided to create three classes for the clear decomposition to enhance the understandability of the logic. Consequently, the creation of the three classes can be seen in the EntryPoint as below.

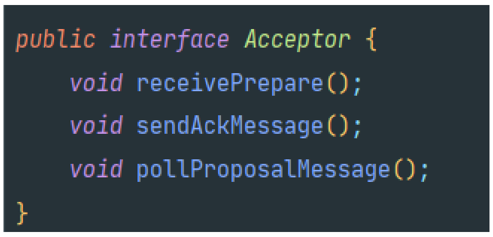


```java
private void makePAL() {
    proposer = new ProposerImpl( paxosEntryView: this, paxosService, clientId);
    // When an acceptor accepts a proposal, it will set the ballot it in its proposer, so
    // proposer reference is sent via a parameter.
    acceptor = new AcceptorImpl(paxosService, clientId, proposer);
    learner = new LearnerImpl( paxosEntryView: this, paxosService, clientId);
}
```

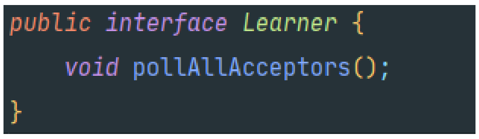
The three classes implement each of its interfaces as below.



```java
public interface Proposer {
    void prepare();
    void propose();
    void increaseId();
    void setBallotId(int ballotId);
}
```

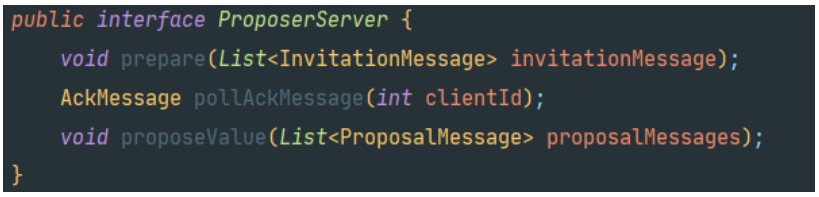


```java
public interface Acceptor {
    void receivePrepare();
    void sendAckMessage();
    void pollProposalMessage();
}
```

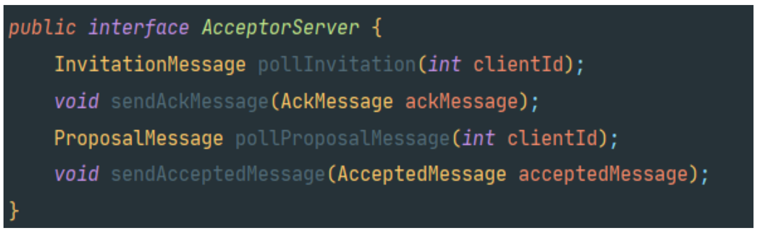


```java
public interface Learner {
    void pollAllAcceptors();
}
```

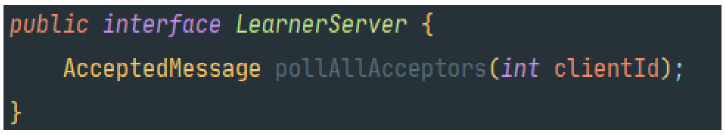
If each agent communicates directly to other agents, IP addresses would be used, but due to the purpose of this GWT project, an utilisation of the server should be considered, i.e., how the server would deliver the messages to the browsers. These pictures below indicate the interfaces with necessary methods for the server.



```java
public interface ProposerServer {
    void prepare(List<InvitationMessage> invitationMessage);
    AckMessage pollAckMessage(int clientId);
    void proposeValue(List<ProposalMessage> proposalMessages);
}
```

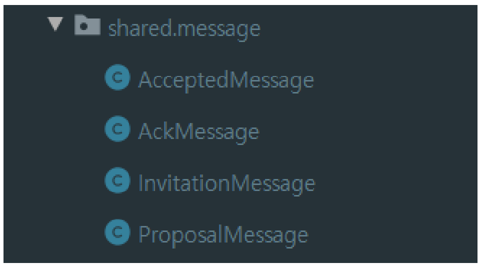


```java
public interface AcceptorServer {
    InvitationMessage pollInvitation(int clientId);
    void sendAckMessage(AckMessage ackMessage);
    ProposalMessage pollProposalMessage(int clientId);
    void sendAcceptedMessage(AcceptedMessage acceptedMessage);
}
```



```java
public interface LearnerServer {
    AcceptedMessage pollAllAcceptors(int clientId);
}
```

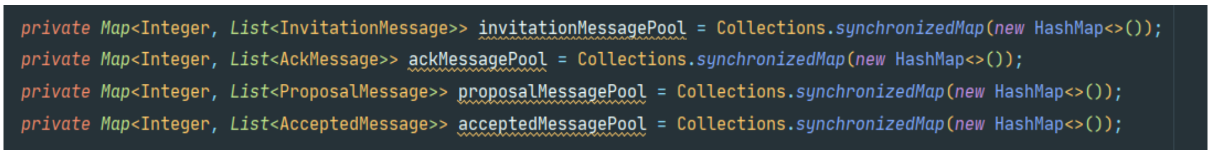
As can be seen above, the server returns various message types and retrieves parameters of different message types: InvitationMessage, AckMessage, ProposalMessage, and AcceptedMessage. At the beginning, a single message class, which combines all the functions of them, was considered, yet it turned out that there were some unshared variables. As a result, some message classes were derived as below.



```
shared.message
    AcceptedMessage
    AckMessage
    InvitationMessage
    ProposalMessage
```

Talking about the messages, as the AcceptedMessage is broadcasted to all learners, the destination id (toId) attribute is not required, whereas AckMessage is sent to the proposer who gave the invitation and Proposal is sent only to the acceptors who expressed positive in the ack message, so the destination id is needed. In terms of the InvitationMessage, it is not broadcasted since the sender (proposer) should not be included (the implementation can be changed depending on an algorithm).

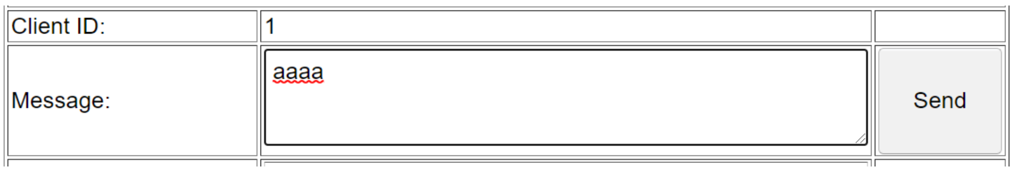
Like the previous section (Phase 2), the synchronizedMap, which is a static method of Collections, was used to create thread-safe code. The key of each map becomes the client ID, so agents can poll their message from the pools.



```java
private Map<Integer, List<InvitationMessage>> invitationMessagePool = Collections.synchronizedMap(new HashMap<>());
private Map<Integer, List<AckMessage>> ackMessagePool = Collections.synchronizedMap(new HashMap<>());
private Map<Integer, List<ProposalMessage>> proposalMessagePool = Collections.synchronizedMap(new HashMap<>());
private Map<Integer, List<AcceptedMessage>> acceptedMessagePool = Collections.synchronizedMap(new HashMap<>());
```

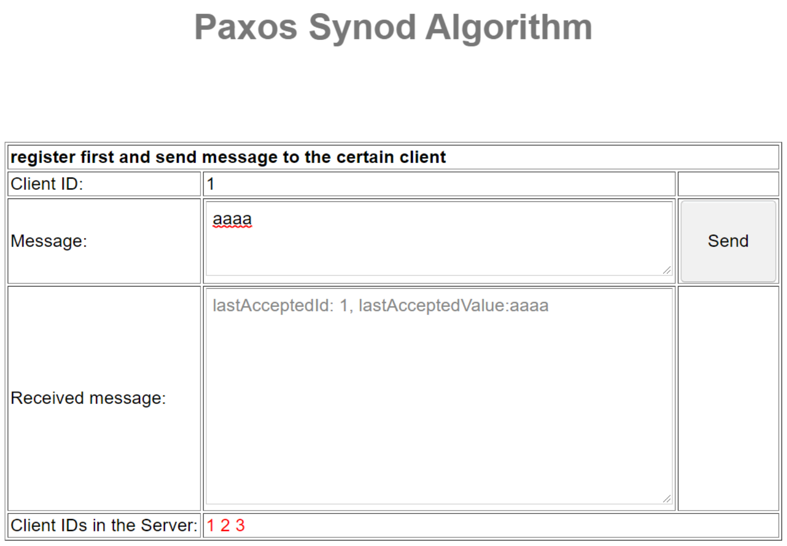
Brief explanation about the process of the Paxos algorithm to be implemented is as follows. In the prepare phase, a proposer increases the ballot id (The ballot ID must be greater than the number used in the previous prepare message), and makes the invitation messages and stores them to the server. Server sends the messages to all acceptors (actually each acceptor polls them by itself). Acceptors make the ack message depending on the comparison with their minIdToAccept. They may put a decree value if they have received a value previously and the proposer will use the value. If the ack messages indicate that the majority of acceptors have agreed, the propose phase starts. The acceptors, who sent a positive ack message, will have the proposal and the value will be accepted if some condition is matched. Finally, the accepted value is broadcasted to all learners. Some omitted detail can be found in the code. Javadocs were also created to supplement them.

To check its operation, three browsers were opened. By adding the text (decree value) and clicking the send button, the message from client id one was sent to all acceptors (opened browsers).



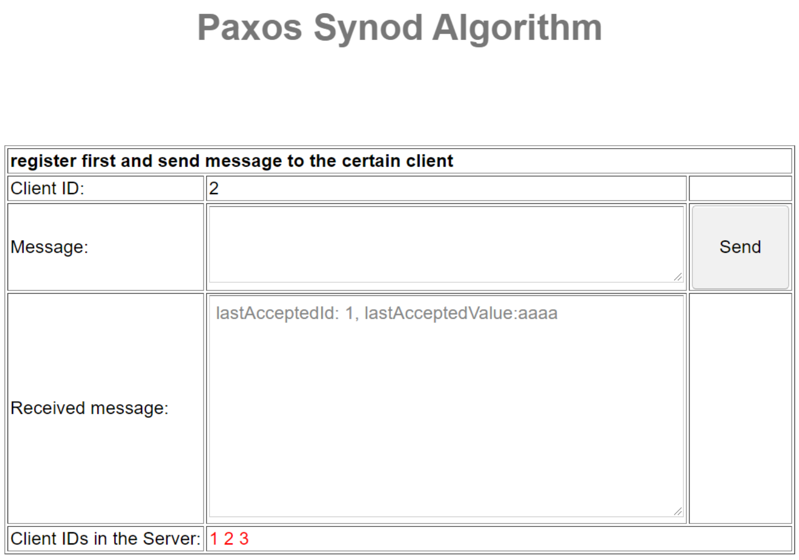
Client ID:	1	
Message:	aaaa	Send

Since all other agents have not received any invitation before the event above, their minIdToAccept is zero, so the message is accepted in all browsers as below (It was confirmed that it was worked well even with many more browsers).



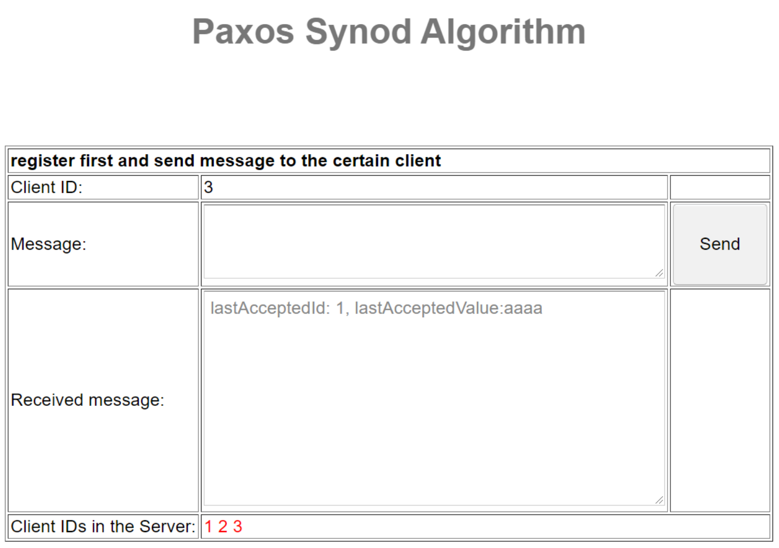
Paxos Synod Algorithm

register first and send message to the certain client		
Client ID:	1	
Message:	aaaa	Send
Received message:	lastAcceptedId: 1, lastAcceptedValue:aaaa	
Client IDs in the Server:	1 2 3	



Paxos Synod Algorithm

register first and send message to the certain client		
Client ID:	2	
Message:		Send
Received message:	lastAcceptedId: 1, lastAcceptedValue:aaaa	
Client IDs in the Server:	1 2 3	



Paxos Synod Algorithm

register first and send message to the certain client		
Client ID:	3	
Message:		Send
Received message:	lastAcceptedId: 1, lastAcceptedValue:aaaa	
Client IDs in the Server:	1 2 3	

Through this assignment, a huge experience was gained such as learning the interesting technology, GWT which converts Java into JavaScript, and Paxos algorithm which is the basis of all distributed systems. As a person who did not know the existence of the GWT library, it was a pleasure to implement the algorithm with the tool.

Reference List

GWT (n.d.). *GWT Project*. [online] www.gwtproject.org. Available at: http://www.gwtproject.org/doc/latest/tutorial/clientserver.html [Accessed 22 Mar. 2021].

GWTLecturer (2012). *GWT Tutorial 1.1 - Proper Architecture for GWT GUI Building*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=AapyOMOoQNM&list=PLoWne5q-c9E92-rrra5eiztpq1ACrCWNm&index=6&ab_channel=GWTLecturer [Accessed 22 Mar. 2021].

Уроки Java (2019). *GWT 1: Что такое GWT, создание и запуск первого приложения*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=0-HjCHh09H4&ab_channel=%D0%A3%D1%80%D0%BE%D0%BA%D0%B8Java [Accessed 22 Mar. 2021].