COMP489: Distributed Computing - Assignment 3

Test Plan

Alex Perrin

Aug 31, 2025

Test Plan
- Assignment file directory
  - WEB-INF Directory Structure
  - client-stubs Directory
- Testing Environment
- Install Java 8
- Setup PostgreSQL Database Management System
- Download PostgreSQL JDBC Driver
- Setup WildFly Application Server
- 1. Compile the web service
- 2. Compile the client
- 2. Deploy Web Service:
- 3. Verify the FileShareService is deployed
- 4. Start Client(s):

Test Results
- Single Client Tests
- Multi-Client P2P Tests
- Input Validation Tests

Assignment file directory

All assignment program files are located in the comp489-a3.zip archive.

- `FileShareService.java` is the JAX-WS web service implementation.
- `FileShareClient.java` is the application P2P client implementation.
- `web.xml` is the web service deployment descriptor.
- `sun-jaxws.xml` is the JAX-WS configuration file.
- `database-schema.sql` is the database schema for the fileshare service postgres database.
- `postgresql-42.7.6.jar` is the JDBC driver for postgres.
- `wildfly-10.1.0.Final/` directory contains the Wildfly standalone install, which contains all of the JAX-WS dependencies.
- `WEB-INF/` directory contains the web application structure for the service.
- `client-stubs/` directory contains generated JAX-WS client stubs.
- `FileShareClient.jar` is the compiled executable for the client.
- `build-service.sh`, `build-client.sh` are scripts to complete the build steps.
- `testfile.txt` is the example file containing "Hello, World!"
- All java and compiled class files included for reference.

```
alex@alex-desktop:~$ cd comp489-a3
alex@alex-desktop:~/comp489-a3$ ls -1
'FileShareClient$1.class'
'FileShareClient$FileRequestHandler.class'
'FileShareClient$SocketServerRunnable.class'
FileShareClient.class
FileShareClient.jar
FileShareClient.java
FileShareService.java
FileShareService_Service.java
WEB-INF/
build-client.sh
build-service.sh
client-stubs/
database-schema.sql
postgresql-42.7.6.jar
sun-jaxws.xml
testfile.txt
web.xml
wildfly-10.1.0.Final/
```

WEB-INF Directory Structure

The `WEB-INF/` directory follows the standard Java web application structure required for WAR deployment:

- `WEB-INF/classes/` - Contains compiled Java classes (FileShareService.class)
- `WEB-INF/lib/` - Contains required JAR dependencies (postgresql-42.7.6.jar)
- `WEB-INF/web.xml` - Web application deployment descriptor
- `WEB-INF/sun-jaxws.xml` - JAX-WS service configuration

client-stubs Directory

The `client-stubs/` directory contains JAX-WS client stub classes generated by `wsimport`, contains the generated service interfaces.

Testing Environment

For my development environment, I'm using Ubuntu 24.04.2 LTS running in WSL on Windows 11.

```
alex@alex-desktop:~$ cat /etc/lsb_release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=24.04
DISTRIB_CODENAME=noble
DISTRIB_DESCRIPTION="Ubuntu 24.04.2 LTS"
```

Install Java 8

The `default-jre` and `default-jdk` ubuntu packages will install Java 21, this version of Java does not support the libraries for JAX-WS. We'll need to use Java 8.

```
$ sudo apt install openjdk-8-jre openjdk-8-jdk
```

Set `PATH` to the Java 8 installation.

```
$ export PATH=/usr/lib/jvm/java-8-openjdk-amd64/bin:$PATH
```

Verify versions of `java`, `javac`, and `wsimport`

```
$ java -version
openjdk version "1.8.0_462"
OpenJDK Runtime Environment (build 1.8.0_462-8u462-ga~us1-0ubuntu2~24.04.2-b08)
OpenJDK 64-Bit Server VM (build 25.462-b08, mixed mode)

$ javac -version
javac 1.8.0_462

$ wsimport -version
wsimport version "2.2.9"
```

Setup PostgreSQL Database Management System

Install the following ubuntu packages.

```
$ sudo apt install postgresql
```

Verify PostgreSQL installation:

```
$ psql --version
psql (PostgreSQL) 16.9 (Ubuntu 16.9-0ubuntu0.24.04.1)
```

Set the password for postgres user.

```
$ sudo -u postgres psql -c "ALTER USER postgres PASSWORD 'postgres';"
```

Create the fileshare database.

```
$ sudo -u postgres createdb fileshare
```

Import the SQL Database Definition Language to establish the database

```
$ sudo -u postgres psql -U postgres -f database-schema.sql
```

Download PostgreSQL JDBC Driver

```
$ wget https://jdbc.postgresql.org/download/postgresql-42.7.6.jar
```

Setup WildFly Application Server

Download WildFly 10.1.0.Final for hosting the JAX-WS web service.

```
$ wget https://download.jboss.org/wildfly/10.1.0.Final/wildfly-10.1.0.Final.tar.gz
$ tar -xzf wildfly-10.1.0.Final.tar.gz
```

Make the WildFly directory executable:

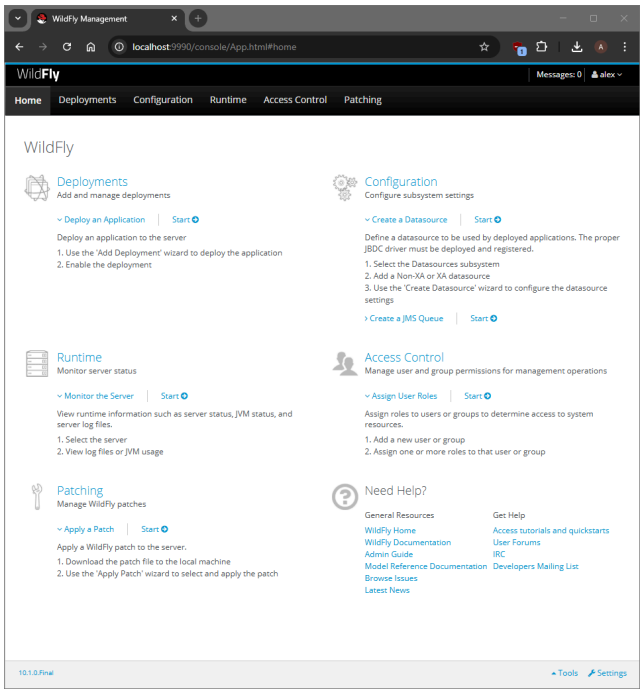
```
$ chmod +x -R wildfly-10.1.0.Final
```

Start WildFly in standalone mode (run this in its own terminal):

```
$ cd wildfly-10.1.0.Final/bin
$ ./standalone.sh
```

Verify WildFly installation by accessing the standalone service in a web browser at `localhost:8080`

and the management console is available at `localhost:9990/console/App.html#home` after running `add-user.sh` in the standalone folder.



1. Compile the web service

Compile the web service and create WAR file using the build script.

```
$ ./build-service.sh
```

You can also follow along with the build steps manually:

1. Create the `WEB-INF/` directory structure

```
$ mkdir -p WEB-INF/classes WEB-INF/lib
```

2. Compile the service java.

```
$ javac -cp "postgresql-42.7.6.jar" FileShareService.java
```

3. Copy files to `WEB-INF/` directory

```
cp FileShareService.class WEB-INF/classes/
cp postgresql-42.7.6.jar WEB-INF/lib/
```

```
cp web.xml WEB-INF/
cp sun-jaxws.xml WEB-INF/
```

4. Compile the service war file.

```
$ jar cf FileShareService.war WEB-INF/
```

5. Either manually drag and drop the `FileShareService.war` file into the `wildfly-10.1.0.Final/standalone/deployments/` folder, or

```
$ cp FileShareService.war wildfly-10.1.0.Final/standalone/deployments/
```

2. Compile the client

Compile the Fileshare client using the provided built script, or follow along with the instructions

```
$ ./build-client.sh
```

1. Create the `client-stubs/` directory

```
$ mkdir -p client-stubs
```

2. Generate the stubs with `wsimport`

```
$ wsimport -keep -d client-stubs http://localhost:8080/FileShareService/FileShareService?wsdl
```

3. Compile the client java with the generated stubs

```
$ javac -cp client-stubs FileShareClient.java
```

4. Create executable JAR with compiled java and generated stubs

```
jar cfe FileShareClient.jar FileShareClient \
    FileShareClient*.class \
    -C client-stubs .
```

2. Deploy Web Service:

After running `build-service.sh` or by moving the war file into `wildfly-10.1.0.Final/standalone/deployments/` , the WildFly standalone service will automatically create `FileShareService.war.deployed` .

You will also so the console log in the terminal where `standalone.sh` is ran

```
19:41:00,902 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 2) WFLYSRV0010: Deployed "FileShareService.war" (runtime-name : "FileShareService.war")
```

3. Verify the FileShareService is deployed

```
$ curl http://localhost:8080/FileShareService/FileShareService?wsdl
```


```
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://service/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="FileShareService" targetNamespace="http://service/">
<wsdl:message name="getFileOwnerResponse">
<wsdl:part name="return" type="xsd:string">
</wsdl:part>
</wsdl:message>
<wsdl:message name="searchFilesResponse">
<wsdl:part name="return" type="xsd:string">
</wsdl:part>
</wsdl:message>
<wsdl:message name="unregisterFile">
<wsdl:part name="filename" type="xsd:string">
</wsdl:part>
<wsdl:part name="clientAddress" type="xsd:string">
</wsdl:part>
<wsdl:part name="clientPort" type="xsd:int">
</wsdl:part>
</wsdl:message>
<wsdl:message name="registerFileResponse">
<wsdl:part name="return" type="xsd:boolean">
</wsdl:part>
</wsdl:message>
<wsdl:message name="unregisterFileResponse">
<wsdl:part name="return" type="xsd:boolean">
</wsdl:part>
</wsdl:message>
<wsdl:message name="registerFile">
<wsdl:part name="filename" type="xsd:string">
</wsdl:part>
<wsdl:part name="clientAddress" type="xsd:string">
</wsdl:part>
```

```
<wsdl:part name="clientPort" type="xsd:int">
</wsdl:part>
</wsdl:message>
<wsdl:message name="getFileOwner">
<wsdl:part name="filename" type="xsd:string">
</wsdl:part>
</wsdl:message>
<wsdl:message name="searchFiles">
<wsdl:part name="searchQuery" type="xsd:string">
</wsdl:part>
</wsdl:message>
<wsdl:portType name="FileShareService">
<wsdl:operation name="getFileOwner">
<wsdl:input message="tns:getFileOwner" name="getFileOwner">
</wsdl:input>
<wsdl:output message="tns:getFileOwnerResponse" name="getFileOwnerResponse">
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="searchFiles">
<wsdl:input message="tns:searchFiles" name="searchFiles">
</wsdl:input>
<wsdl:output message="tns:searchFilesResponse" name="searchFilesResponse">
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="unregisterFile">
<wsdl:input message="tns:unregisterFile" name="unregisterFile">
</wsdl:input>
<wsdl:output message="tns:unregisterFileResponse" name="unregisterFileResponse">
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="registerFile">
<wsdl:input message="tns:registerFile" name="registerFile">
</wsdl:input>
<wsdl:output message="tns:registerFileResponse" name="registerFileResponse">
</wsdl:output>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="FileShareServiceSoapBinding" type="tns:FileShareService">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="getFileOwner">
<soap:operation soapAction="" style="rpc"/>
<wsdl:input name="getFileOwner">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:input>
<wsdl:output name="getFileOwnerResponse">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="searchFiles">
<soap:operation soapAction="" style="rpc"/>
<wsdl:input name="searchFiles">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:input>
<wsdl:output name="searchFilesResponse">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="unregisterFile">
<soap:operation soapAction="" style="rpc"/>
<wsdl:input name="unregisterFile">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:input>
<wsdl:output name="unregisterFileResponse">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="registerFile">
<soap:operation soapAction="" style="rpc"/>
<wsdl:input name="registerFile">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:input>
<wsdl:output name="registerFileResponse">
<soap:body namespace="http://service/" use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="FileShareService">
<wsdl:port binding="tns:FileShareServiceSoapBinding" name="FileShareServicePort">
<soap:address location="http://localhost:8080/FileShareService/FileShareService"/>
</wsdl:port>
```

```
</wsdl:service>
</wsdl:definitions>
```

4. Start Client(s):

Start FileShareClient instance connected to the web service:

```
java -jar FileShareClient.jar
```

Client startup

```
alex@alex-desktop:~/comp489-a3$ java -jar FileShareClient.jar
Connected to web service at: http://localhost:8080/FileShareService/FileShareService
Client socket server listening on port 32825
Commands:
  register <filename> - Register a file for sharing
  unregister <filename> - Stop sharing a file
  search <query> - Search for files
  download <filename> [output_filename] - Download a file
  quit - Exit
```

Test Results

Single Client Tests

Client Input (STDIN)	Client Output (STDOUT)	WildFly Service Output (STDOUT)	Result
> search test	No files found matching: test	19:58:04,570 INFO [stdout] (default task-31) SEARCH REQUEST: test 19:58:04,770 INFO [stdout] (default task-31) SEARCH SUCCESS: Found 0 files	No files found initially
> register testfile.txt	Registered: testfile.txt	19:58:10,161 INFO [stdout] (default task-32) REGISTER REQUEST: testfile.txt from 127.0.0.1:32825 19:58:10,194 INFO [stdout] (default task-32) REGISTER SUCCESS: Registered testfile.txt from 127.0.0.1:32825	File registered successfully
> search test	Search results for 'test': testfile.txt	19:58:13,669 INFO [stdout] (default task-33) SEARCH REQUEST: test 19:58:13,684 INFO [stdout] (default task-33) SEARCH SUCCESS: Found 1 files	Find registered file
> download testfile.txt output.txt	Served file: testfile.txt (13 bytes) Downloaded: testfile.txt → output.txt (13 bytes)	19:58:26,511 INFO [stdout] (default task-35) OWNER REQUEST: testfile.txt 19:58:26,524 INFO [stdout] (default task-35) OWNER SUCCESS: testfile.txt → 127.0.0.1:32825	Download from self successful
> unregister testfile.txt	Unregistered: testfile.txt	9:58:38,932 INFO [stdout] (default task-36) UNREGISTER REQUEST: testfile.txt from 127.0.0.1:32825 19:58:38,945 INFO [stdout] (default task-36) UNREGISTER RESULT: Removed testfile.txt from 127.0.0.1:32825 (rows affected: 1)	File unregistered successfully
> search test	No files found matching: test	19:58:43,330 INFO [stdout] (default task-37) SEARCH REQUEST: test 19:58:43,343 INFO [stdout] (default task-37) SEARCH SUCCESS: Found 0 files	No files found after unregister

Multi-Client P2P Tests

Client A Actions	Client B Actions	Client A Output	Client B Output	Server Output	Result
Start client A	Start client B	Connected to web service at: http://localhost:8080/FileShareService/FileShareService Client socket server listening on port 32825	Connected to web service at: http://localhost:8080/FileShareService/FileShareService Client socket server listening on port 46447	Server accepts both connections	Two clients connected
-	register testfile.txt	-	Registered: testfile.txt	19:58:10,161 INFO [stdout] (default task-32) REGISTER REQUEST: testfile.txt from	Client B registers file

Client A Actions	Client B Actions	Client A Output	Client B Output	Server Output	Result
				127.0.0.1:32825 19:58:10,194 INFO [stdout] (default task-32) REGISTER SUCCESS: Registered testfile.txt from 127.0.0.1:32825	
> search test	-	Search results for 'test': testfile.txt	-	19:58:13,669 INFO [stdout] (default task-33) SEARCH REQUEST: test 19:58:13,684 INFO [stdout] (default task-33) SEARCH SUCCESS: Found 1 files	Client A finds B's file
> download testfile.txt output1.txt	-	Downloaded: testfile.txt → output1.txt (13 bytes)	Served file: testfile.txt (13 bytes)	19:58:26,511 INFO [stdout] (default task-35) OWNER REQUEST: testfile.txt 19:58:26,524 INFO [stdout] (default task-35) OWNER SUCCESS: testfile.txt → 127.0.0.1:32825	Client A downloads from B
-	> download testfile.txt output2.txt	-	Downloaded: testfile.txt → output2.txt (13 bytes) Served file: testfile.txt (13 bytes)	19:58:26,511 INFO [stdout] (default task-35) OWNER REQUEST: testfile.txt 19:58:26,524 INFO [stdout] (default task-35) OWNER SUCCESS: testfile.txt → 127.0.0.1:46447	Client B downloads from self

Input Validation Tests

Client Input (STDIN)	Client Output (STDOUT)	Result
register	Usage: register <filename>	Invalid command format rejected
unregister	Usage: unregister <filename>	Invalid command format rejected
search	Usage: search <query>	Invalid command format rejected
download	Usage: download <filename> [output_filename]	Invalid command format rejected
register nonexistent.txt	File not found: nonexistent.txt	Cannot register nonexistent file
download nonexistent.txt	File not found: nonexistent.txt	Cannot download nonexistent file
invalid_command	Unknown command: invalid_command Type 'quit' to exit or use one of the available commands.	Invalid commands rejected
quit	Goodbye!	Application exits gracefully