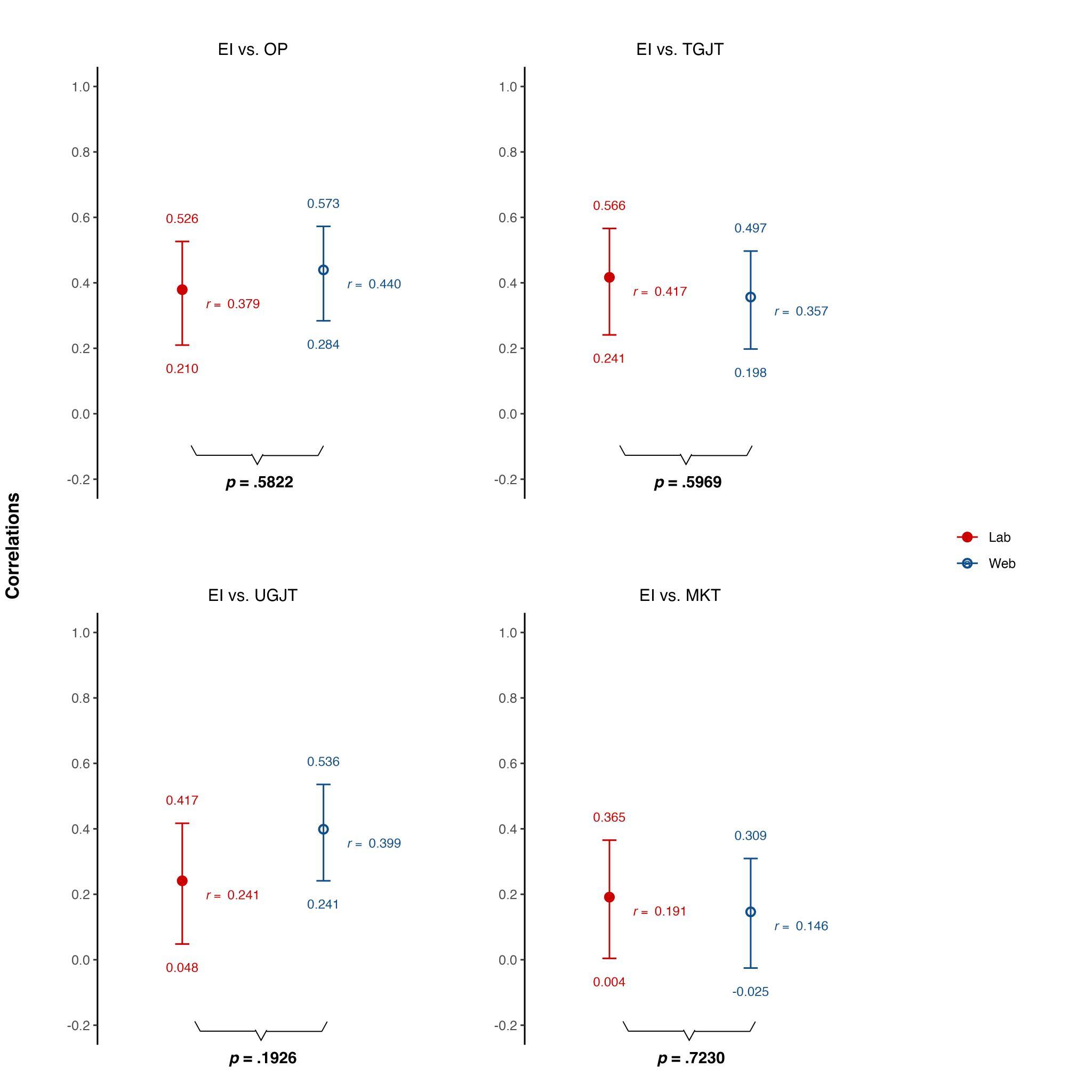
**Supplementary Materials**

**Figure S1**

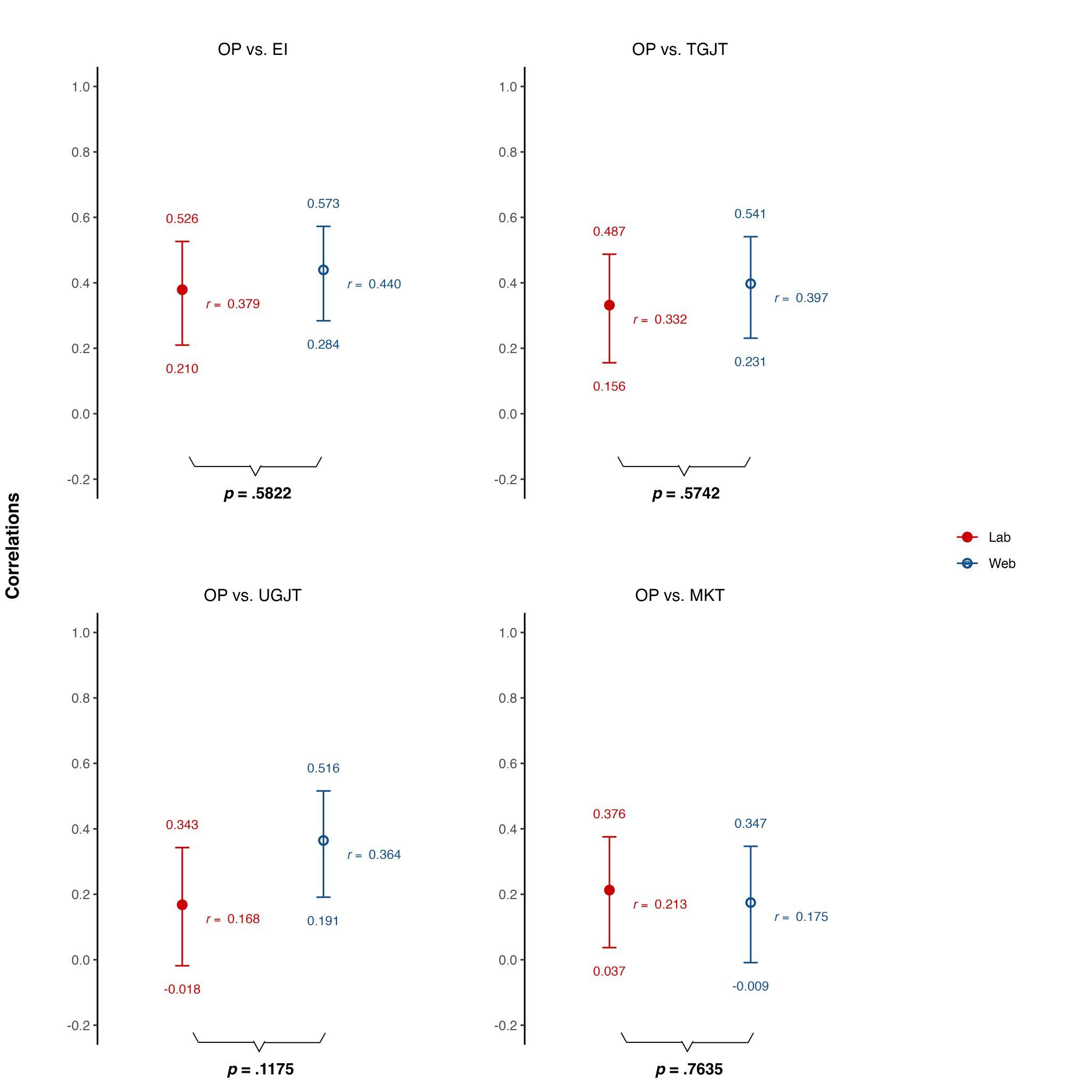
*Correlation Between EI and Other Linguistic Measures*



*Note.* Correlational values with 95% confidence interval illustrating the association between EI and Oral Production, Timed GJT, and Untimed GJT

**Figure S2**

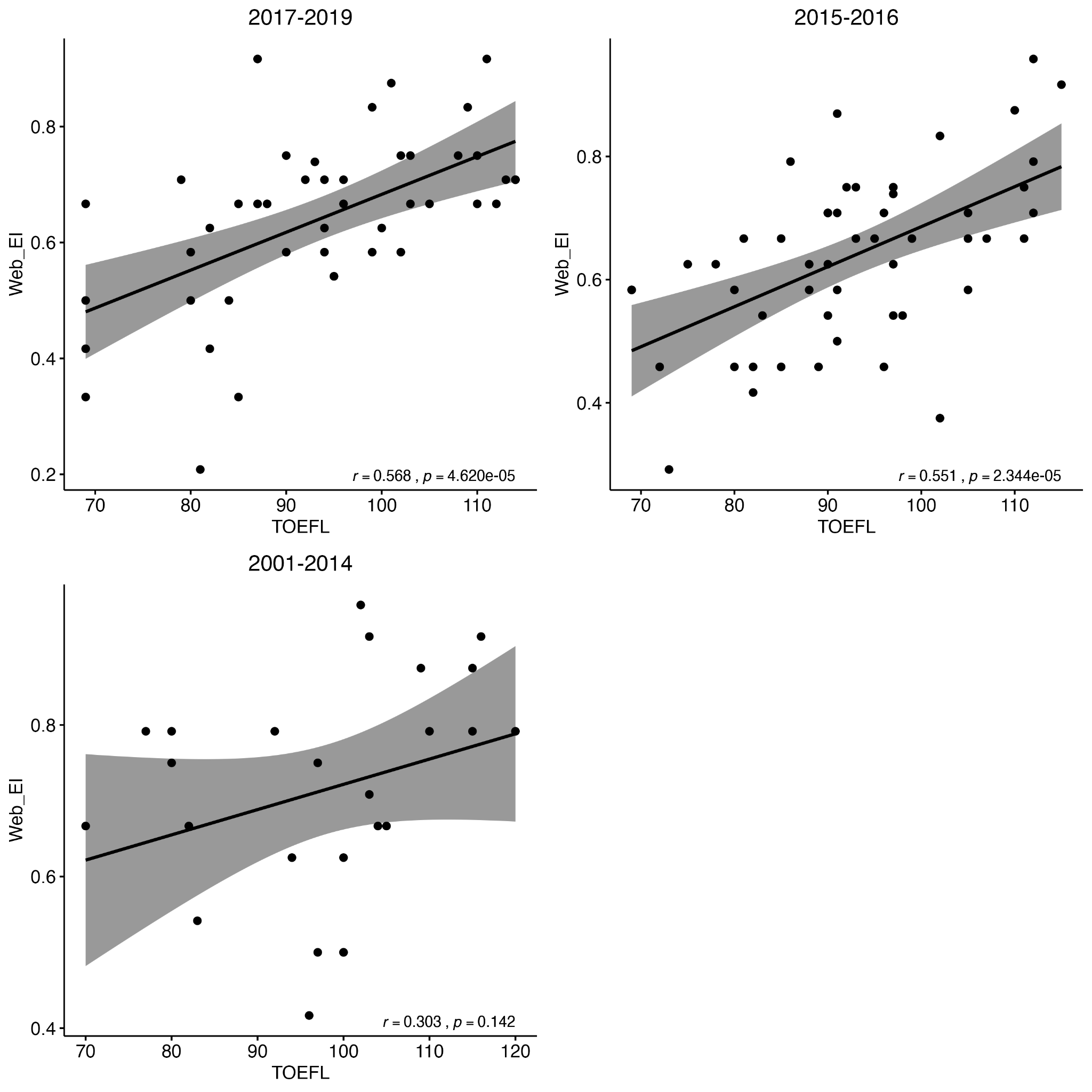
*Correlation Between OP and Other Linguistic Measures*



*Note.* Correlational values with 95% confidence interval illustrating the association between OP and Elicited Imitation, Timed GJT, and Untimed GJT

**Figure S3**

*Correlation Between TOEFL Score Groups and Web-Based EI*



*Note.* Correlation between TOEFL scores and web-based EI across three time periods. The regression lines with 95% confidence intervals demonstrate the strength and direction of associations for Group 1 (2017-2019), Group 2 (2015-2016), and Group 3 (2001-2014). This analysis was carried out only with Kim's data since the test-taking year was collected.

**Table S1**

*Correlation Between TOEFL Taken in 2017-2019 and Web-Based EI Measures*

|  |  |  |
| --- | --- | --- |
| Variable | *r* | 95% CI |
| EI | .57 | [.33, .74] |
| EI\_G | .60 | [.37, .76] |
| EI\_UG | .42 | [.15, .64] |

*Note*. This analysis was carried out only with Kim's data since the test-taking year was collected.

**Table S2**

*Correlation Between TOEFL Taken in 2015-2016 and Web-Based EI Measures*

|  |  |  |
| --- | --- | --- |
| Variable | *r* | 95% CI |
| EI | .55 | [.33, .72] |
| EI\_G | .58 | [.37, .74] |
| EI\_UG | .39 | [.13, .60] |

*Note*. This analysis was carried out only with Kim's data since the test-taking year was collected.

**Table S3**

*Correlation Between TOEFL Taken in 2001-2014 and Web-Based EI Measures*

|  |  |  |
| --- | --- | --- |
| Variable | *r* | 95% CI |
| EI | .30 | [-.11, .62] |
| EI\_G | .35 | [-.05, .65] |
| EI\_UG | .34 | [-.06, .65] |

*Note*. This analysis was carried out only with Kim's data since the test-taking year was collected.

**Appendix S1**

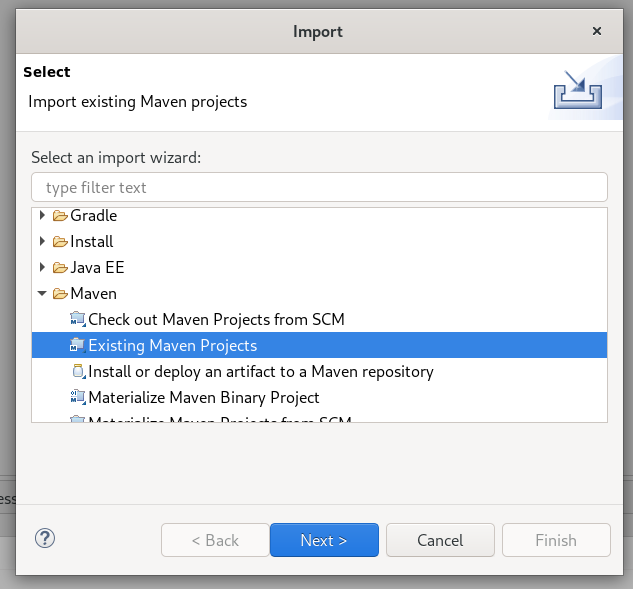
Web-based Task Instruction

**Instructions on the use of the AIED Web-based testing program**

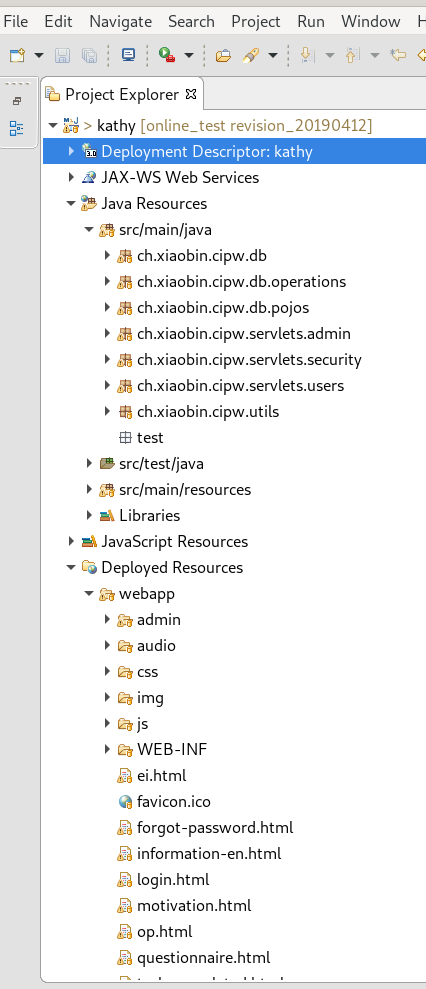
The AIED program has two major components: a front-end component consisting of HTML files with embedded Javascript code, and a back-end component written in Java, which offers an API to store and retrieve data from the server. The both components are organized into a single Webapp project, which is openable with the development tool Eclipse. I will first show you how to import the project and how to make changes to the setup.

# **Basic setup**

First fire up Eclipse, and choose from **File → Import**. In the Import Wizard, select **Maven/Existing Maven Projects** and navigate to the folder containing the code (you will find a **pom.xml** file at the project root) to import the project.



Once imported, you will see all the files the project requires in the Project Explorer window as shown below. The back-end code which runs on the server is under **Java Resources**, while the front-end code is under **Deployed Resources/webapp**.



Back-end code

Front-end code

If you want to change the presentation of the test, you need to change the html files in the “webapp” folder. If you want to change anything related to the back-end, including changing the type of data, number of test items and how they are stored on the server, you would need to change the back-end code.

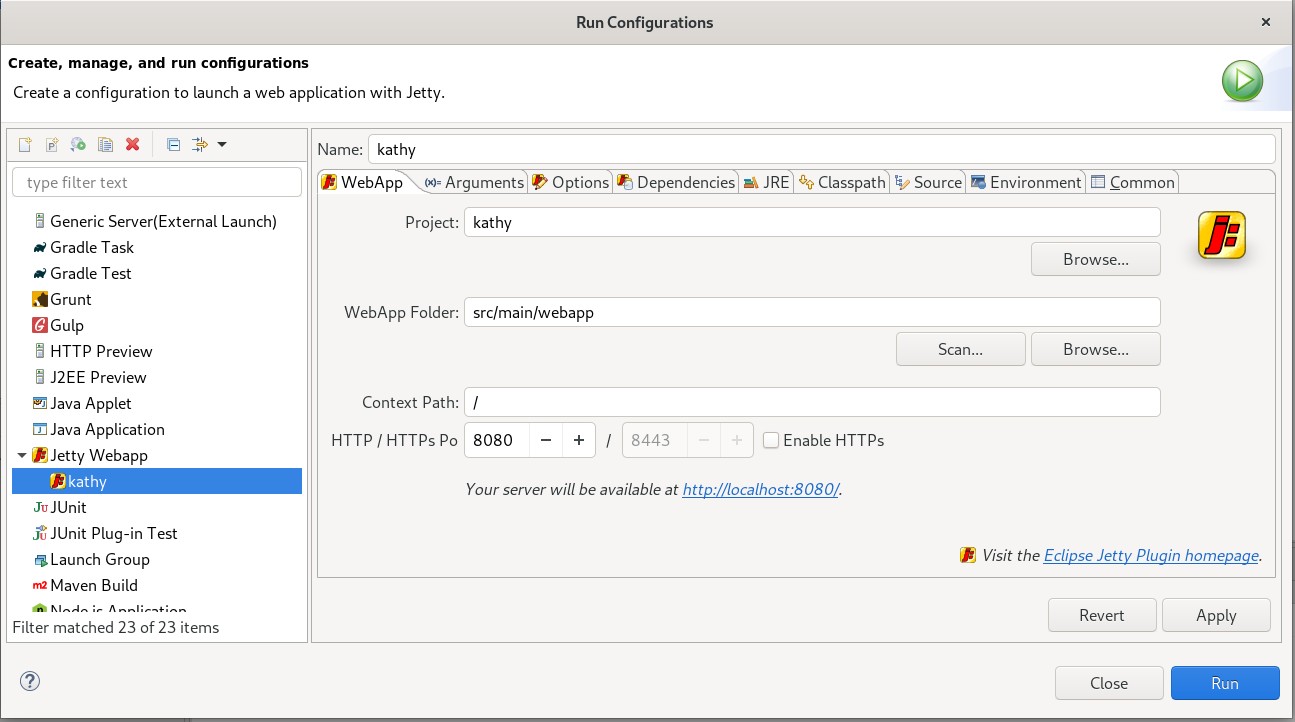
Before you make any changes, it is better to test if the project already runs in your environment. As said before, the project is a Java Webapp project. As a result, it requires something called a Web Container to run. A Web container is basically an environment to run Java projects for the web. You can think of it as a Web server with Java support. Popular Web containers include Tomcat, Jetty, and so on. We are going to use Jetty to run the project because it is very easy to set up, and Jetty has very good documentation.

Before we can run the project under Apache Jetty, we need to install the Jetty plugin for Eclipse. Go to Eclipse’s menu **Help → Eclipse Marketplace**. Search for “Jetty” in the Marketplace to install it.

A screenshot of a browser window

Description automatically generatedAfter installing the Jetty plugin, you can now run the project in Eclipse.

Go to **Run → Run Configuration** (if you don’t see the menu item, open a random Java file from the Java Resources folder, then select Run → Run Configuration). In the run configuration window, select the Jetty Webapp project, then click the New configuration icon on the top-right corner of the window. Leave everything else by default, and click **Run** to run the project.



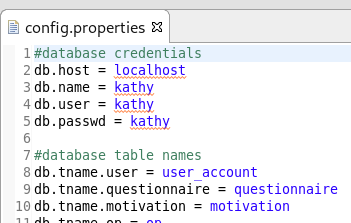
At this point you should already be able to access the test environment from your Web browser. Try entering the address http://localhost:8080 in your Web browser. If everything goes well, you will see a login interface to the test.

# 

# **Database**

In order for the program to run properly, you would also need a database on your server so that the participants’ responses to the tests can be stored. The current code uses Postgresql as the database management system. To set up Postgresql server, you can follow the instructions on <https://www.postgresql.org/docs/12/index.html>or ask your system administrator to set up one for you. What you need to get from your system administration is the host address of the database server, the name of the database they created for you, the user and password you can use to access that database.

This information will need to be entered in the **src/main/resources/config.properties** file in the project. In the file, under the “#database credentials” section, enter and replace the values for **db.host**, **db.name**, **db.user**, **db.passwd** with the database credentials you obtained. In the setup shown below, I am accessing my locally installed database, but it can also be a database in another machine or host.



# **Mail server**

Another thing to set up for the program to run properly is a mail server with which the program sends the participants credentials for logging into the system to do the tests. Because we want to control who has access to the testing environment, we don’t want the system to be open for new-user sign-ups. The researcher in charge of the experiment will need to gather participants’ emails and enter them manually into the system to create accounts for the participants. As a result, the system needs a mail server to be able to send notification emails to the participants when an account is set up for them in the system.

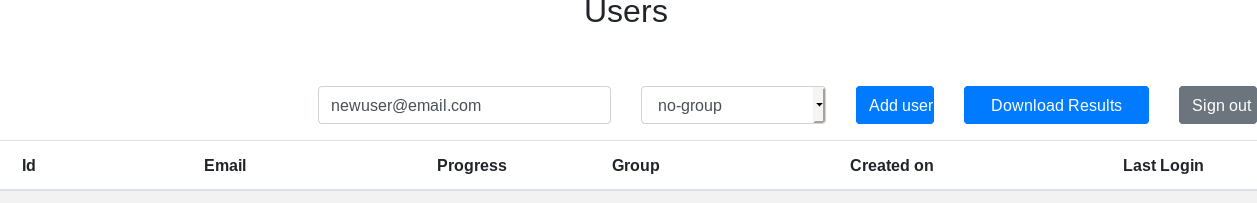
The mail server setup is located in the same file as the database setup: **src/main/resources/config.properties.** Under the “#mail server settings”, enter the SMTP server information as you normally would when setting up a mail client. You can find this information from your email service provider.

# **Creating participant account**

Once everything is set up, restart the application from Eclipse. You may need to stop the previous running instance of the program. Otherwise, the internet port would have already been occupied, and the new instance would fail to run.

When the program is rebooted, go to http://localhost:8080/admin/users.html in your browser. The program would ask you for the admin username and password to access the admin functions. The admin credentials are also set in the **src/main/resources/config.properties** file under the section “#Admin credentials”. In the admin page, enter the credentials listed here. Use this admin page to invite participants and later retrieve data for the tests.

To add a new participant, simply enter the participant’s email and click “Add user”. The new user will be recorded in the database and an email will be sent to the participant to inform them about how to participate in the experiment.



The email sent to the participant is customizable. Just search for the file **AdminApiUsersServlet.java** and go to Line 179. This is where the subject and message of the email are set.

The email guides the participants to log in to the system with the credential and address provided.

# 

# **Advanced: Changing the questionnaire and tests**

Changing the questionnaire and tests requires changing both the front-end file and the back-end database structure and code to access the database.

For changes to the front-end presentation, open the HTML file you need to change from the **webapp/** folder. Edit the texts or information fields you want to change. This requires knowledge on how to work with HTML and CSS. The pages use the Bootstrap framework for presentation. So, some knowledge on how Bootstrap works would be helpful. If you have never heard of or worked with these technologies, it is recommended that you ask someone with Web design experience for help. Otherwise, you would need to learn at least HTML, CSS, and Bootstrap to be able to make changes to the front-end code. Knowledge about Javascript would also be needed.

The back-end program is responsible for receiving data the participants submit. It is implemented as RESTful APIs under the **servlets/users/** folder. Each test or questionnaire has a corresponding servlet in charge of data recording from and retrieval for the front-end. Change the servlets if you need to change the structure of the data you need to collect.

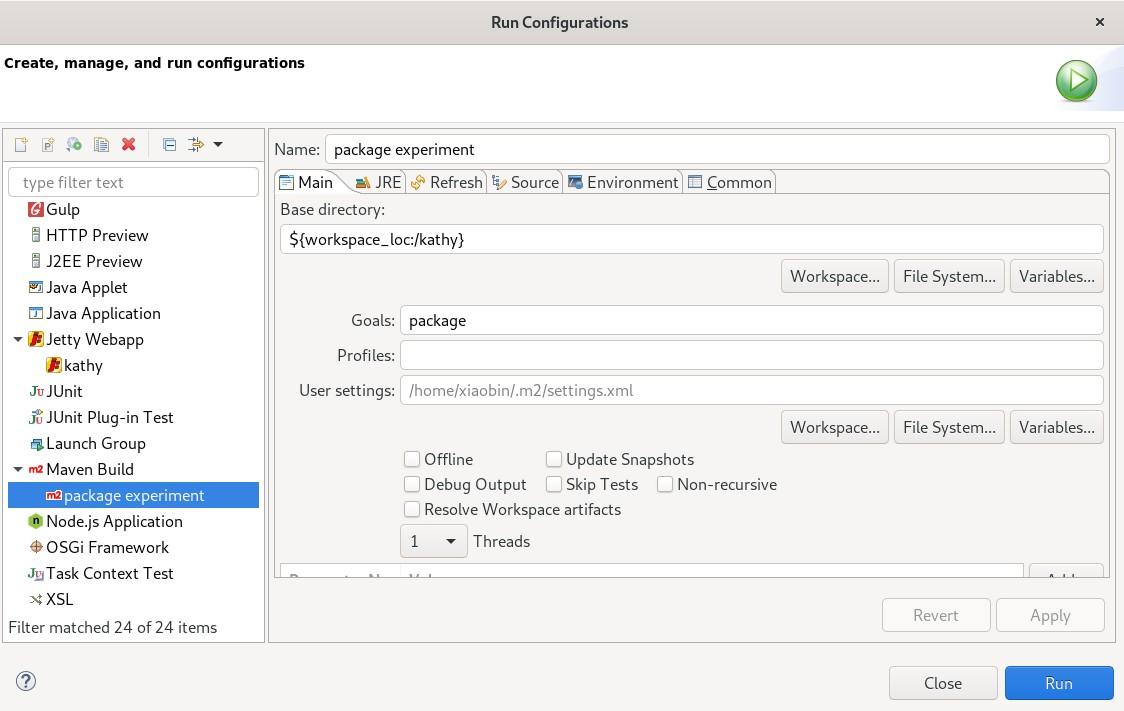
Data is transferred to and from the servlets in JSON format. The POJOs or data models for each test are located under **db/pojos**. These POJOs are used by the servlets to parse the JSON data passed from the front end. The **db/operations** folder contains the actual code to handle data transmission between our program and the underlying database management system, namely Postgresql in this case. The operation on each table is implemented as a Java class in the **db/operations** folder. Modifying the operations requires knowledge on SQL. The code in these files is quite self-explanatory and well-documented. Make changes to these files based on your needs.

Again, if any of the technologies sounds unfamiliar to you, ask a Java programmer for help, or learn Java, RESTful API, Servlet, and SQL technologies before you make any changes.

**6. Deploying the tests**

The procedures described above are mostly for testing the program on your own machine. Once you are done making changes and are ready to deploy the tests to your participants, you would need a production server that runs 24\*7 and is publicly accessible on the Internet so that your participants can do the tests anywhere and anytime. Ask your system administrator for the deployment environment if you don’t have access to a production server. Once you gain access to such a server, package the program as a WAR file and deploy it in a Web container (usually Tomcat or Jetty).

To package the program, run in a terminal under the root folder of the project files **mvn clean package,** or create an Eclipse **Run Configuration** of Maven Build with the Goals set to “package”. The root folder of the project is where you can find a **pom.xml** file and an **src/** folder.



After running the package command, you get a new folder called **target** in the project root folder under which you will see the packaged WAR file whose filename is suffixed with **.war**. Send this file to your system administrator for deployment or deploy it yourself on the Web container in the production server. This usually is as simple as copying the war file into a folder where the Web container searches for Web applications.

Good luck with your experiments!