**Software Implementation and Testing Document**

**For**

**“Resistor Calulator”**

Version 1.0

**Authors**:

Vinzce Yadao

Ronald Nazaire

Jorge Atencio

Jacob Hobson

Schmidt Jean

# Programming Languages (5 points)

*List the programming languages use in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).*

For our project we used object-oriented programming.

The programming languages that we used were Java, PHP, HTML, JavaScript, and CSS. PHP was used for log in and log out features. Java was used for the gui and the registration of the application. HTML was used to create the final html file for the application.

# Platforms, APIs, Databases, and other technologies used (5 points)

*List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).*

We used Github, SQL, IntelliJ IDEA, command prompt, Notepad++.

Github is used to upload all the work that we did. SQL is used for all login related features. IntelliJ IDEA will be used to write the java code. The command prompt and Notepad++ will be backups just in case some members won’t be able to get IntelliJ IDEA to download or to work. Notepad++ might have some issues but it has worked before for a group member. Xamp as a local webserver with PHP.

# Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the* ***functional requirements*** *listed in your RD).*

Resistor Calculation – tested to make sure that the calculations were accurate. Bands were able to be selected. Value and tolerance of the resistors can be calculated.

Multi-band implementation – Supports 4 and 5 band resistors. Both calculations were accurate according to the number of bands that the user picks.

Updating the Resistor Visually – The visual does update when changing bands, and updates accordingly.

User Log in – Tested by making several “accounts” to make sure that we were able to log to their individual account.

# Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the* ***non-functional requirements*** *listed in your RD).*

Passwords should never be viewable – tested by typing in a random password to make sure that it was not able to be seen while typing.

Display – The display is understandable; everything is clear and readable.

High-speed calculations – calculations are fast and displays the answer accurately to the nearest whole number.

Calculations stored under user account – tested by logging into an account, doing some calculations, logging out, and then logging back in to make sure that those calculations stayed under the account.

Non-crashing – we tested by using all the features that we managed to implement and seeing to see if the application would crash

# Non-Execution-based Testing (10 points)

*Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).*

We have not done any non-execution-based testing but have been testing every implementation that we managed to successfully put into our application continuously.