COMP1815 Group Report

Section 1: Checklist of features

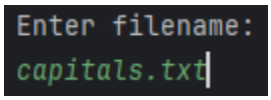
Feature	Implemented
File Loading and Processing	Yes
User-friendly Command Line Interface	Yes
Option to view graph appropriately	Yes
Domain and entity classes – representing graph and edges	Yes
Minimum Spanning Tree algorithm implementation	Yes
Option to view Minimum Spanning Tree	Yes
Option to view 'cable length' (MST total)	Yes
Appropriate error handling	Yes

Repository link: https://dev.azure.com/mv5742c/_git/Grp%2034%20-%20COMP1815%20Coursework

Section 2: Features demonstration

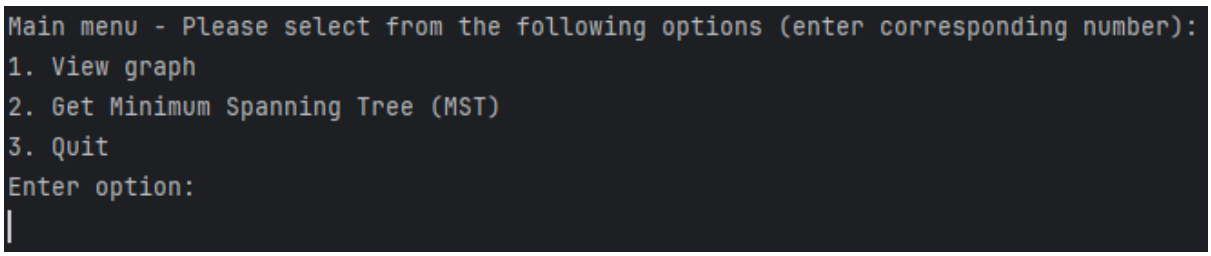
File Loading and Processing

This shows the application prompting the user to enter the filename of the data for processing.



User-friendly Command Line Interface

This shows the application providing the user with a straightforward menu to interact with to navigate around the program. The menu shows each functionality of the program and how to reach it.



Option to view graph appropriately

This shows the user entering the number '1' to view the graph.

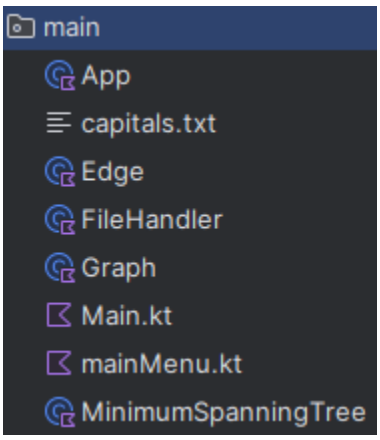
```
Main menu - Please select from the following options (enter corresponding number):
1. View graph
2. Get Minimum Spanning Tree (MST)
3. Quit
Enter option:
1
```

This shows a snippet of the output provided by the application, showing each unique capital city in the data file and providing an easy-to-read list of the distances between the two capitals.

```
Amsterdam → [(Andorra la Vella, 1161), (Athens, 2161), (Belgrade, 1374), (Berlin, 577), (
Andorra la Vella → [(Amsterdam, 1161), (Athens, 1725), (Belgrade, 1404), (Berlin, 1383),
Athens → [(Amsterdam, 2161), (Andorra la Vella, 1725), (Belgrade, 807), (Berlin, 1806), (
Belgrade → [(Amsterdam, 1374), (Andorra la Vella, 1404), (Athens, 807), (Berlin, 1003), (
Berlin → [(Amsterdam, 577), (Andorra la Vella, 1383), (Athens, 1806), (Belgrade, 1003), (
Bern → [(Amsterdam, 660), (Andorra la Vella, 682), (Athens, 1612), (Belgrade, 1083), (Ber
Bratislava → [(Amsterdam, 982), (Andorra la Vella, 1292), (Athens, 1286), (Belgrade, 349)
Brussels → [(Amsterdam, 173), (Andorra la Vella, 1073), (Athens, 2101), (Belgrade, 1394),
```

Domain and entity classes – representing graph and edges

This shows the project structure, showing the separate class files for both the Graph and Edges, as well as source files for the main application loop and menu system.



Minimum Spanning Tree Algorithm Implementation

This shows a snippet of the resulting minimum spanning tree, showing the connections between the capitals and the distances between them.

```
Minimum Spanning Tree: [(Vienna, 55), (Tallinn, 82), (Tirana, 105), (Zagreb, 117), (Vaduz, 135), (Tirana, 142), (Sarajevo, 148), (Vilnius, 159),
```

Option to view Minimum Spanning Tree

This shows the user entering ‘2’ into the menu system to reach the Minimum Spanning Tree feature.

```
Main menu - Please select from the following options (enter corresponding number):
1. View graph
2. Get Minimum Spanning Tree (MST)
3. Quit
Enter option:
2
```

Option to view 'cable length' (MST total)

The user enters '2' into the menu system to reach the Minimum Spanning Tree feature which contains the cable length result.

```
Main menu - Please select from the following options (enter corresponding number):
1. View graph
2. Get Minimum Spanning Tree (MST)
3. Quit
Enter option:
2
```

The result of the application calculating the minimum required cable length is then shown:

```
Minimum Cable Length: 14562
```

Appropriate error handling

This shows the user entering an invalid filename into the program.

```
Enter filename:
not-a-real-file.txt
```

The application deals with the invalid file by using a backup to still showcase the features.

```
File not found, using backup file: src/main/capitals.txt
Main menu - Please select from the following options (enter corresponding number):
1. View graph
2. Get Minimum Spanning Tree (MST)
3. Quit
Enter option:
```

Section 3: Code listing

Main.kt | **Kotlin**

```kotlin
package main

fun main() {

    // Create and run app
    val app = App()
    app.run()

    return
}
```

App.kt | **Kotlin**

```kotlin
package main
import kotlin.system.exitProcess

class App {
    fun run() {
        val defaultFile = "src/main/capitals.txt"
        val path = "src/main/"
        var createdGraph = false
        var cityGraph = Graph(mutableListOf())

        // Ask user for filename input and create graph from data
        while(!createdGraph) {
            println("Enter filename: ")
            val file = readlnOrNull()
            val fileHandler = FileHandler(path + file, defaultFile)
            cityGraph = Graph(fileHandler.readFile())

            // Check graph created successfully
            if (cityGraph.toString() == "") {
                println("Graph not created")
            } else {
                createdGraph = true
            }
        }

        // View main menu
        val mainMenu = MainMenu()
        while (true) {
            // Open menu
            val selectedOption = mainMenu.run()

            // Handle menu options
            if (selectedOption == 1) {
                // Print graph
                println(cityGraph)
            } else if (selectedOption == 2) {
```

```
                // Create MST class
                val minimumSpanningTree = MinimumSpanningTree(cityGraph)

                // Create MST and print
                val mst = minimumSpanningTree.getMST()
                val cableLength = minimumSpanningTree.getTotalCableLength()
                println("Minimum Spanning Tree: $mst")
                println("Minimum Cable Length: $cableLength")

                continue
            } else if (selectedOption == 3) {
                // Terminate program normally
                println("Goodbye...")
                exitProcess(0)
            }
        }
    }
}
```

FileHandler.kt | **Kotlin**

```
package main

import java.io.File
import java.io.FileNotFoundException

class FileHandler(private val file: String, private val backupFile: String) {
    fun readFile(): MutableList<Triple<String, String, Int>> {
        // Data structure for lines in the file
        val data = mutableListOf<Triple<String, String, Int>>()

        // Read the file
        try {
            File(file).forEachLine {
                // Split the line by spaces
                val line = it.split(", ")

                // Add the line to the data structure
                data.add(Triple(line[0], line[1], line[2].toInt()))
            }
        }
        catch (e: FileNotFoundException) {
            println("File not found, using backup file: $backupFile")
            File(backupFile).forEachLine {
                // Split the line by spaces
                val line = it.split(", ")

                // Add the line to the data structure
                data.add(Triple(line[0], line[1], line[2].toInt()))
            }
        }

        return data
```

```
    }
}
```

MainMenu.kt | **Kotlin**

```kotlin
package main

import java.io.File
import java.io.FileNotFoundException

class FileHandler(private val file: String, private val backupFile: String) {
    fun readFile(): MutableList<Triple<String, String, Int>> {
        // Data structure for lines in the file
        val data = mutableListOf<Triple<String, String, Int>>()

        // Read the file
        try {
            File(file).forEachLine {
                // Split the line by spaces
                val line = it.split(", ")

                // Add the line to the data structure
                data.add(Triple(line[0], line[1], line[2].toInt()))
            }
        }
        catch (e: FileNotFoundException) {
            println("File not found, using backup file: $backupFile")
            File(backupFile).forEachLine {
                // Split the line by spaces
                val line = it.split(", ")

                // Add the line to the data structure
                data.add(Triple(line[0], line[1], line[2].toInt()))
            }
        }

        return data
    }
}
```

Graph.kt | **Kotlin**

```kotlin
package main

class Graph(private val data: MutableList<Triple<String, String, Int>>) {
    private val graph = generate()

    // Go over data, add vertices, then add edges
    private fun generate(): MutableMap<String, MutableSet<Edge>> {
        val graph = mutableMapOf<String, MutableSet<Edge>>()

        // First go over the graph and add the vertices
        for (line in data) {
```

```
            val source = line.first // u
            val destination = line.second // v

            if (!graph.containsKey(source))
                graph[source] = mutableSetOf()

            if (!graph.containsKey(destination))
                graph[destination] = mutableSetOf()
        }

        // Add edges to both vertices
        for (line in data) {
            val source = line.first // u
            val destination = line.second // v
            val weight = line.third // w

            // Add sources and destination and weights to the graph, only if the do not
already exist
            if (graph[source]?.none { it.destination == destination } == true) {
                graph[source]?.add(Edge(source, destination, weight))
            }
            if (graph[destination]?.none { it.destination == source } == true) {
                graph[destination]?.add(Edge(destination, source, weight))
            }
        }

        return graph
    }

    // Method to print the graph in a readable format
    override fun toString(): String {
        var str: String = ""

        for ((u, edges) in graph) {
            str += "$u -> $edges\n"
        }

        return str
    }

    // Get edges
    fun getEdges(): MutableList<Edge> {
        val edges = mutableListOf<Edge>()

        for ((_, edgeSet) in graph) {
            for (edge in edgeSet) {
                edges.add(edge)
            }
        }

        return edges
    }

    // Get vertices
```

```
    fun getVertices(): MutableList<String> {
        return graph.keys.toMutableList()
    }
}
```

Edge.kt	Kotlin

```
package main

class Edge(val source: String, val destination: String, private val weight: Int) {
    // Returns the weight of the edge
    fun getWeight(): Int {
        return weight
    }

    // Method to print the edge
    override fun toString(): String {
        return "($destination, $weight)"
    }
}
```

MinimumSpanningTree.kt	Kotlin

```
package main

// Minimum Spanning Tree - Kruskal's Algorithm
class MinimumSpanningTree(graph: Graph) {
    private val sortedEdges = graph.getEdges().sortedBy { it.getWeight() } // Sort edges
by weight
    private val vertices = graph.getVertices()
    private val parent = mutableMapOf<String, String>()
    private val rank = mutableMapOf<String, Int>()

    // Find the root of the vertex using path compression
    private fun find(vertex: String): String {
        if (parent[vertex] != vertex) {
            parent[vertex] = find(parent[vertex]!!)
        }
        return parent[vertex]!!
    }

    // Union of two vertices - merge two vertices by linking their roots and update
ranks
    private fun union(vertex: String, vertex2: String) {
        val root = find(vertex)
        val root2 = find(vertex2)

        if (rank[root]!! > rank[root2]!!) {
            parent[root2] = root
        } else if (rank[root]!! < rank[root2]!!) {
            parent[root] = root2
        } else {
            parent[root2] = root
```

```
            rank[root] = rank[root]!! + 1
        }
    }

    // Get the minimum spanning tree
    fun getMST(): List<Edge> {
        // Initialise parent and rank
        vertices.forEach { vertex ->
            parent[vertex] = vertex
            rank[vertex] = 0
        }

        val mst = mutableListOf<Edge>()
        // Iterate over sorted edges and add to the MST if they don't create a cycle
        for (edge in sortedEdges) {
            if (find(edge.source) != find(edge.destination)) {
                mst.add(edge)
                // Merge the two vertices (perform union)
                union(edge.source, edge.destination)
            }
        }

        // Return the list of edges forming the overall MST
        return mst
    }

    // Get the total cable length (return the sum of all weights)
    fun getTotalCableLength(): Int {
        return getMST().sumOf { it.getWeight() }
    }

}
```

Section 4: Coursework Contribution Form

In percentage, please indicate the work contribution of each member. This should be agreed by all group members.
The total of all members work must add up to 100%

Team member name	Student ID	Individual overall work contribution	Note
Deanna White	001208356	25%	Accepted EC
Deeya Patel	001230057	25%	Accepted EC
Jake Brown	001239595	25%	Accepted EC
Matthew Vallance	001221832	25%	Accepted EC
		Total 100%	

COMP1815 Individual Report

Section 1: Object-oriented and functional programming

During this module, Kotlin and Scala have been taught. They are excellent examples of the Object-Oriented (Kotlin) and Functional Programming (Scala) paradigms. This discussion explains the key differences between these paradigms and languages, compares them, and analyses their strengths and weaknesses.

The object-oriented programming (OOP) paradigm is based on three main concepts: classes and instances, inheritance, and encapsulation (MDN Web Docs Contributors, 2024). Classes are a template for creating objects of that type; an instance of a class is created using a constructor, in which values are passed, and an instance is initialized. Inheritance is when a subclass inherits properties and methods (like functions) from the superclass (W3Schools, n.d.). Encapsulation hides 'sensitive data' that should be protected from unauthorized access. This is done by declaring variables/attributes as private (W3Schools, n.d.).

In comparison, the functional programming (FP) paradigm is based on key concepts such as pure functions, recursion, immutability and higher-order functions. Pure functions are deterministic; they always return the same result for the same values that are passed to them (GeekforGeeks, 2023). Pure functions have no 'side-effects' such as modifying external variables or outputting something; the only output is a return value. Recursion is when a function can call itself to solve a problem. Each recursion works on a smaller part of the same problem until the base case (the condition that stops the recursion, preventing infinite recursions) stops the process (Tuteja, 2024). Immutability is when data cannot be changed after it is created, which keeps it consistent, eliminates unexpected changes (Zhirnov, 2024) and makes code easier to maintain. Higher-order functions (HOFs) take functions as an argument and can return other functions. HOFs allow code reusability, abstracting common patterns into reusable functions (Ayebola, 2024); this reduces code duplication and helps with maintenance. HOFs also help break down complex problems into smaller chunks, leading to modularity.

Kotlin and Scala can both be used in OOP and FP styles (Scala, 2024) (JetBrains, 2024). However, we have used Kotlin for OOP and Scala for FP within this module. In the past, I have worked with Java (among other OOP languages such as PHP and JavaScript) and adjusting these skills to working with Kotlin was not a huge ask; I felt comfortable working with Kotlin immediately and found it an intuitive, easy-to-use language. Compared to pure Java, Kotlin feels like a significant improvement in speed and efficiency in development, meaning that code is much quicker to implement and readability is much improved. When using Kotlin to implement a larger software project such as the 'European Telecoms Network' application developed within this coursework,

I found that implementing the logic of a concept (such as Graphs and MSTs) was easy, with little need for researching syntax using the internet.

Both languages and their accompanying paradigms are suited for different problems and have different strengths and weaknesses. Firstly, to complete mathematical computations, Scala is certainly the approach. Its emphasis on immutability, which disallows data to be changed after it is created along with pure functions, prevents side effects, which makes it easier to understand and verify the logical steps of complex problems. Scala is also well suited to data transformations, which is where raw data is converted to a unified format, which enhances data quality and usability (Hayes & Downie, 2024); this is because of its built-in higher-order functions such as map, reduce and filter (Baeldung, 2024) which are powerful tools for transforming data.

On the other hand, Kotlin is better for large-scale applications. Kotlin's modern language features, such as its concise syntax, allow it to contain less 'boilerplate' code, which makes the codebase easy to maintain in bigger projects. Kotlin is also better for UI development, especially in terms of Android development, as it is officially endorsed by Google for these purposes. Kotlin is also usable cross-platform across desktop, mobile, and web, utilizing the same codebase, which significantly reduces development time (JetBrains, 2024).

From my first introduction to these, I found that the learning curve is Scala's biggest weakness. Scala's syntax is more complex to understand, especially coming from a background in OOP languages. Scala 3+ allows you to use indentation or braces (Scala, 2023); while this isn't inherently a flaw, if these are not used consistently, it can make for very confusing code!

Kotlin is much easier to use when coming from an OOP background. However, it also has weaknesses. Debugging in Kotlin is more challenging than in Scala; side-effects can be hidden due to methods changing an object's state, add inheritance and polymorphism into this, and it makes debugging much more difficult than in Scala.

In summary, both Kotlin and Scala have been valuable to learn within this module. Kotlin excels in large-scale, cross-platform projects, and Scala is ideal for data transformations and mathematical computations. Understanding their strengths and weaknesses will allow me to choose the best tool for a project's specific needs in the future.

Section 2: Evaluation of Application Evolution

We started the project by gathering the group and discussing the approach to the project, splitting up tasks based upon each individuals' strengths and weaknesses, within this meeting, we also setup an online chat and created the Azure repository so we could keep track of what each person was doing, and check on the progress of each person.

The FileHandler class was the first part to be implemented, this is where the capitals file is imported into the program. From the start, we made sure that a file could always be read, including a default file, which from a development standpoint meant that we could skip inputting a file every time the code was run and from a user perspective, this means that any graph file can be imported into the program.

The graph classes were then created. Firstly, the Edge() class was added in order to store each edge of the graph, it also included a method to print the edge as needed. The Graph() class was next, and the first iteration of this was functional, but variables were not named well. This meant that it wasn't very clear to read and should have been a consideration from the start, but it wasn't until later on in the lifecycle that this was made readable with better naming and commenting.

At the start, there wasn't an easy way to navigate the system. At this point, it was simply a "set and forget" system of typing in a filename, and the software would print out the graph. So, we opted to create a menu system, which neatly tied together all the functionality, meaning that the user had an option to print out the graph by inputting a number, or they could exit cleanly. Unfortunately, this did not have any error handling initially, causing a NumberFormatException if anything other than a number was inputted, this was fixed after testing down the line.

Next, it was onto creating an algorithm to calculate the Minimum Spanning Tree (MST). We used pair programming here to efficiently complete this task, landing on Kruskal's MST algorithm which we implemented successfully after research on how the algorithm completes its job. The algorithm creates a list of edges forming the MST which is returned and displayed alongside a total count of the weights.

At this point, every part of the application was working together well. All that remained was to test the program which revealed that we needed to add exception handling along with better commenting. Overall, the application successfully meets all requirements of the coursework and completes all desired tasks quickly and efficiently.

Our group worked well together; we kept a conversation open via an online chatroom for the whole project and met several times to complete the code together. I feel that tasks were split up well and we all contributed evenly to the code and supplementary group report.

One area I feel could have been improved upon was our communication initially. We were a little slow in getting the project up and running, especially when another group member and I started working on the first task without consulting each other or the wider group, despite already having a chatroom in place. The lesson learnt here is that if we had communicated effectively, this wouldn't have happened and that time would have been better spent working on the code together, meaning we could have ramped up development quicker.

Due to my experience working in a software development role, my primary role (other than programming) within this team was to manage the Azure DevOps project used for version control with Git – to manage pull requests and merge code when needed. In code development, I created the Minimum Spanning Tree (MST) algorithm, which was developed using pair programming, and we decided the best approach was to use Kruskal's MST algorithm. I also refactored some of the code, particularly in relation to making sure variable and class names were consistently using Camel Case along with adding and updating comments within all files and implemented error handling within the codebase.

There are a number of Legal, Social, Ethical and Professional issues related to software development in general. The main legal issues are intellectual property rights, licensing agreements, and data protection. Intellectual property rights provide legal protection for software. Copyright protects code written for a software platform, patents protect inventions and new methods with trademarks protecting symbols, names and slogans used to identify software such as logos and app names (Pulsion.co.uk, 2024). Licensing agreements outline the terms and conditions of using, distributing and modifying software. Data protection involves appropriate security controls implemented into software (Sencode, 2023) so that user data is protected from unauthorized access.

Social issues consist of the digital divide and cultural sensitivity. The digital divide indicates the gap between those with and without access to technology and therefor the internet. As of 2024, 32% of the planet's population does not have access to the internet (ITU, 2024), limiting their opportunities for education, employment, and social participation (IEEE, 2023).

The most important ethical issues are preventing algorithmic bias, unethical data collection and weak cyber security. Algorithmic bias is when algorithms produce unfair/discriminatory outcomes, which can include racial, gender and socioeconomic biases (Jonker & Rogers, 2024). Unethical data collection is when user data is collected without consent and/or transparency. Weak cyber security goes back to data protection, all user data must be protected from cyber-attacks and breaches.

The IEEE Computer Society's Code of Ethics (IEEE-CS/ACM, 1999) gives a good overview of professional issues within software development such as maintaining professional integrity, ethical decision-making, and continuous learning. Professional integrity encompasses maintaining ethical behavior and displays honesty, dependability, and trustworthiness (Indeed Editorial Team, 2022). Ethical decision making is effectively choosing actions which are morally right, considering the impact of decisions on stakeholders and consistency with ethical standards. Software engineers should adhere to a code of ethics, which highlights public, client and employer interests, product quality, and professional integrity such as the IEEE Code of Ethics discussed above.

References

Ayebola, J., 2024. *What are Higher Order Functions in JavaScript? Explained With Examples.* [Online]
Available at: https://www.freecodecamp.org/news/higher-order-functions-explained/
[Accessed 2 December 2024].
Baeldung, 2024. *Higher-Order Functions in Scala.* [Online]
Available at: https://www.baeldung.com/scala/higher-order-functions
[Accessed 6 December 2024].
GeekforGeeks, 2023. *Pure Functions.* [Online]
Available at: https://www.geeksforgeeks.org/pure-functions/
[Accessed 1 December 2024].
Hayes, M. & Downie, A., 2024. *What is data transformation?.* [Online]
Available at: https://www.ibm.com/think/topics/data-transformation
[Accessed 6 December 2024].
IEEE, 2023. *Impact of the Digital Divide: Economic, Social, and Educational Consequences.* [Online]
Available at: https://ctu.ieee.org/blog/2023/02/27/impact-of-the-digital-divide-economic-social-and-educational-consequences/
[Accessed 12 December 2024].
IEEE-CS/ACM, 1999. *Code of Ethics.* [Online]
Available at: https://www.computer.org/education/code-of-ethics
[Accessed 12 December 2024].
Indeed Editorial Team, 2022. *How to Maintain Professional Integrity in the Workplace.* [Online]
Available at: https://www.indeed.com/career-advice/career-development/maintaining-professional-integrity
[Accessed 12 December 2024].
ITU, 2024. *Facts and Figures 2024 - Internet Use.* [Online]
Available at: https://www.itu.int/itu-d/reports/statistics/2024/11/10/ff24-internet-use/
[Accessed 12 December 2024].
JetBrains, 2024. *FAQ | Kotlin Documentation.* [Online]
Available at: https://kotlinlang.org/docs/faq.html#what-is-kotlin
[Accessed 3 December 2024].
JetBrains, 2024. *Introduction to Kotlin Multiplatform.* [Online]
Available at: https://kotlinlang.org/docs/multiplatform-intro.html
[Accessed 6 December 2024].
Jonker, A. & Rogers, J., 2024. *What is algorithmic bias?.* [Online]
Available at: https://www.ibm.com/think/topics/algorithmic-bias
[Accessed 8 December 2024].
MDN Web Docs Contributors, 2024. *Object-oriented programming.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming
[Accessed 14 November 2024].

Pulsion.co.uk, 2024. *Ten legal issues in software development.* [Online]
Available at: https://www.lawdonut.co.uk/business/blog/24/01/ten-legal-issues-software-development
[Accessed 8 December 2024].
Scala, 2023. *Optional Braces.* [Online]
Available at: https://docs.scala-lang.org/scala3/reference/other-new-features/indentation.html
[Accessed 6 December 2024].
Scala, 2024. *The Scala Programming Language.* [Online]
Available at: https://www.scala-lang.org/
[Accessed 3 December 2024].
Sencode, 2023. *What is Data Protection By Design?.* [Online]
Available at: https://sencode.co.uk/achieve-data-protection-by-design/
[Accessed 8 December 2024].
Tuteja, S., 2024. *Introduction to Recursion.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-recursion-2/
[Accessed 1 December 2024].
W3Schools, n.d. *Java Encapsulation.* [Online]
Available at: https://www.w3schools.com/java/java_encapsulation.asp
[Accessed 14 November 2024].
W3Schools, n.d. *Java Inheritance (Subclass and Superclass).* [Online]
Available at: https://www.w3schools.com/java/java_inheritance.asp
[Accessed 14 November 2024].
Zhirnov, M., 2024. *The Power of Immutability in Functional Programming.* [Online]
Available at: https://hemaks.org/posts/the-power-of-immutability-in-functional-programming/
[Accessed 1 December 2024].