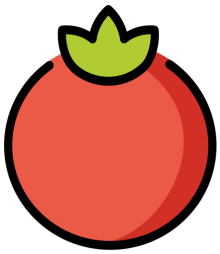
Atma Ram Sanatan Dharma College
University of Delhi

Software Engineeing Project

Paper Code:- (BHCS11)



Tomatoclock

Submitted By :
Chirag Wadhwa
College Roll No: 21/18080
B.Sc (Hons) Computer Science

Submitted To :
Ms Uma Ojha
Department of Computer Science

INDEX

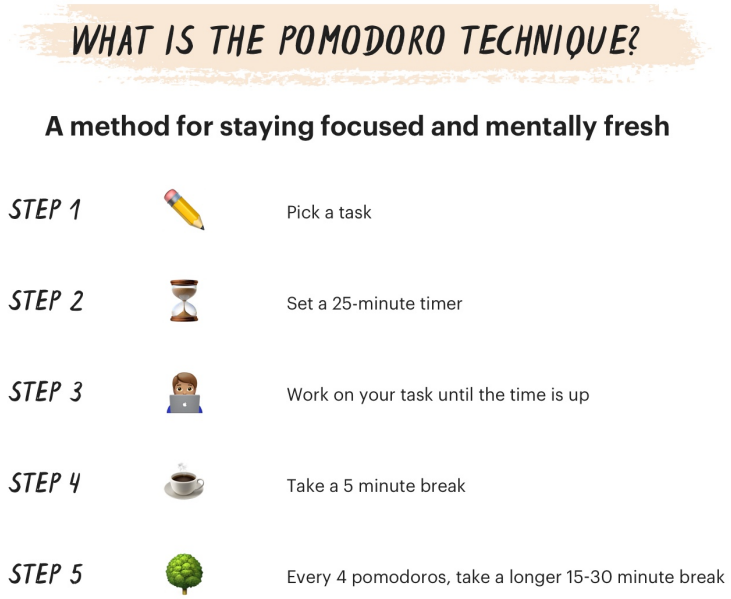
1. Problem Statement

How can a tomato help increase an individual's productivty? The answer lies in Pomodoro Technique, a time management strategy developed by Fracesco Cirillo in the 1980s as a student. Cirillo was struggling to focus on his studies and complete assignments. Feeling overwhelmed, he asked himself to commit to just 10 minutes of focused study time. Encouraged by the challenge, he found a tomato (pomodoro in Italian) shaped kitchen timer, and the Pomodoro Technique was born. The Pomodoro Technique is deceptively simple yet extremely powerful.

What is Pomodoro Technique?

The Pomodoro Technique is a popular time management method that involves breaking work into intervals of 25 minutes, followed by short breaks. Each interval is known as a pomodoro. Try the technique if you:

- Find little distractions often derail the whole workday.
- Consistently work past the point of optimal productivity.
- Have alots of open-ended work that could take unlimited amounts of time(eg., studying for an exam, researching a blogpost, etc.)
- Are overly optimistic when it comes to how much you can get done in a day.
- Enjoy gamefied goal-setting.
- Really like tomatoes



While this technique can help individuals be more productive and focused, there is need for a reliable time management application that incorporates the Pomodoro technique and addressed the following problems:

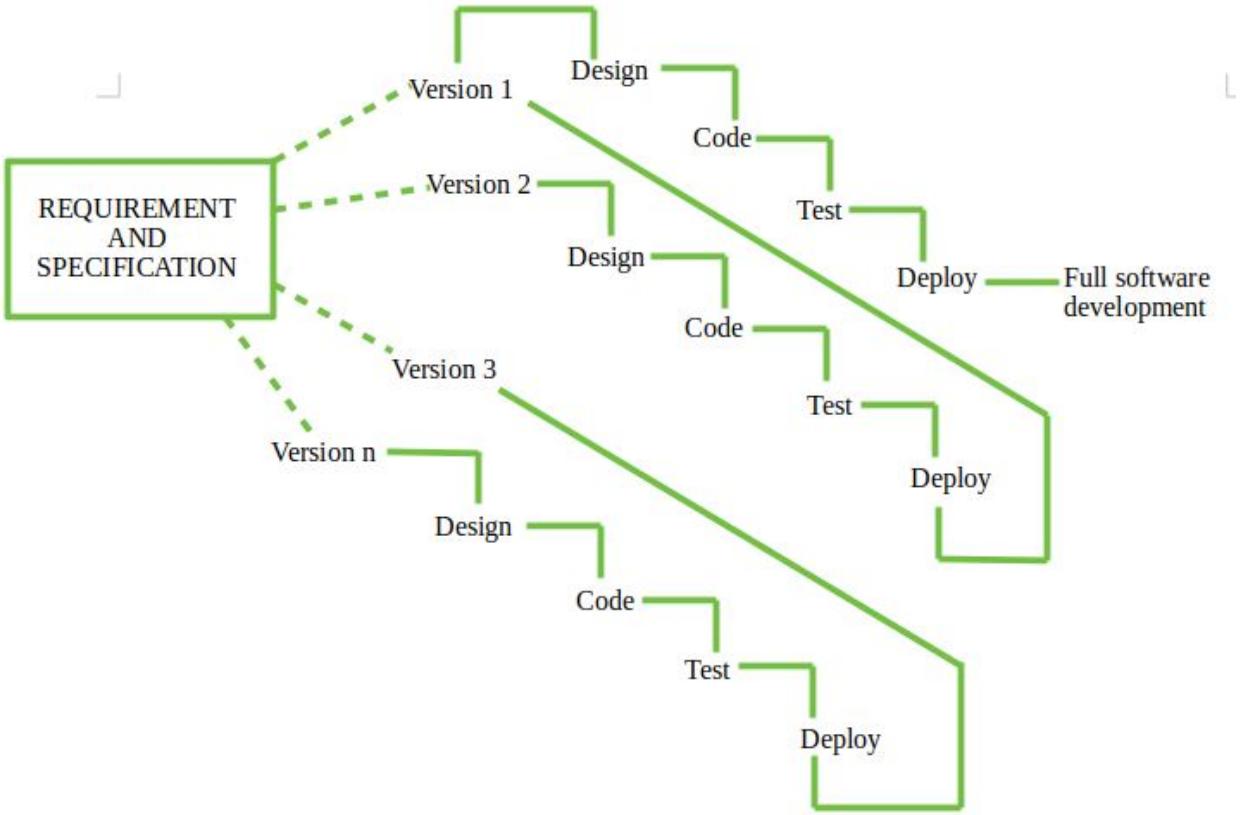
- Many individuals struggle with time management and find it difficult to stay focused on their tasks for extended periods, leading to procrastination, burnout and stress.
- It can be challenging to keep track of the number of Pomodoro sessions completed, track progress, and identify patterns of productivity.
- The task tracking feature should enable individuals to add and categorize their tasks and assign them to specific Pomodoro sessions.
- The data analysis feature should provide individuals with insights into their productivity patterns, identify distractions, and suggest ways to improve their work habits.
- Additionally, the application should have a customizable timer that allows individuals to set the duration of the Pomodoro session and the length of the break. The application should also have the ability to save tasks as routines.

Therefore, there is a need for a time management application that incorporates the Pomodoro Technique, provides a user-friendly interface, and includes features such as task tracking, progress monitoring, data analysis to help manage their time more effectively, improve productivity, and reduce stress.

2. Process Model

The application is developed using the Incremental process model.
In this model, a simple working system implementing only a few basic features is built and then that is delivered to the customer. then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.
Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered. At any time, the plan is made just for the next increment. Therefore, it is easier to modify the version as per the need of the customer.
The Development Team first undertakes to develop core features (these do not need services from other features) of the system.

Once the core features are fully developed, then these are refined to increase levels of capabilities by adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.
As each successive version of the software is constructed and delivered, now the feedback of the Customer is to be taken and these were then incorporated into the next version. Each version of the software has more additional features than the previous ones.



After Requirements gathering and specification, requirements are then split into several different versions starting with version 1, in each successive increment, the next version is constructed and then deployed at the customer site. After the last version (version n), it is now deployed at the client site.

This process model is used when :

- Requirements are known up-front.
- Software team is not very well skilled or trained.
- Requirements are prioritized.
- Customer demands quick release of the product.

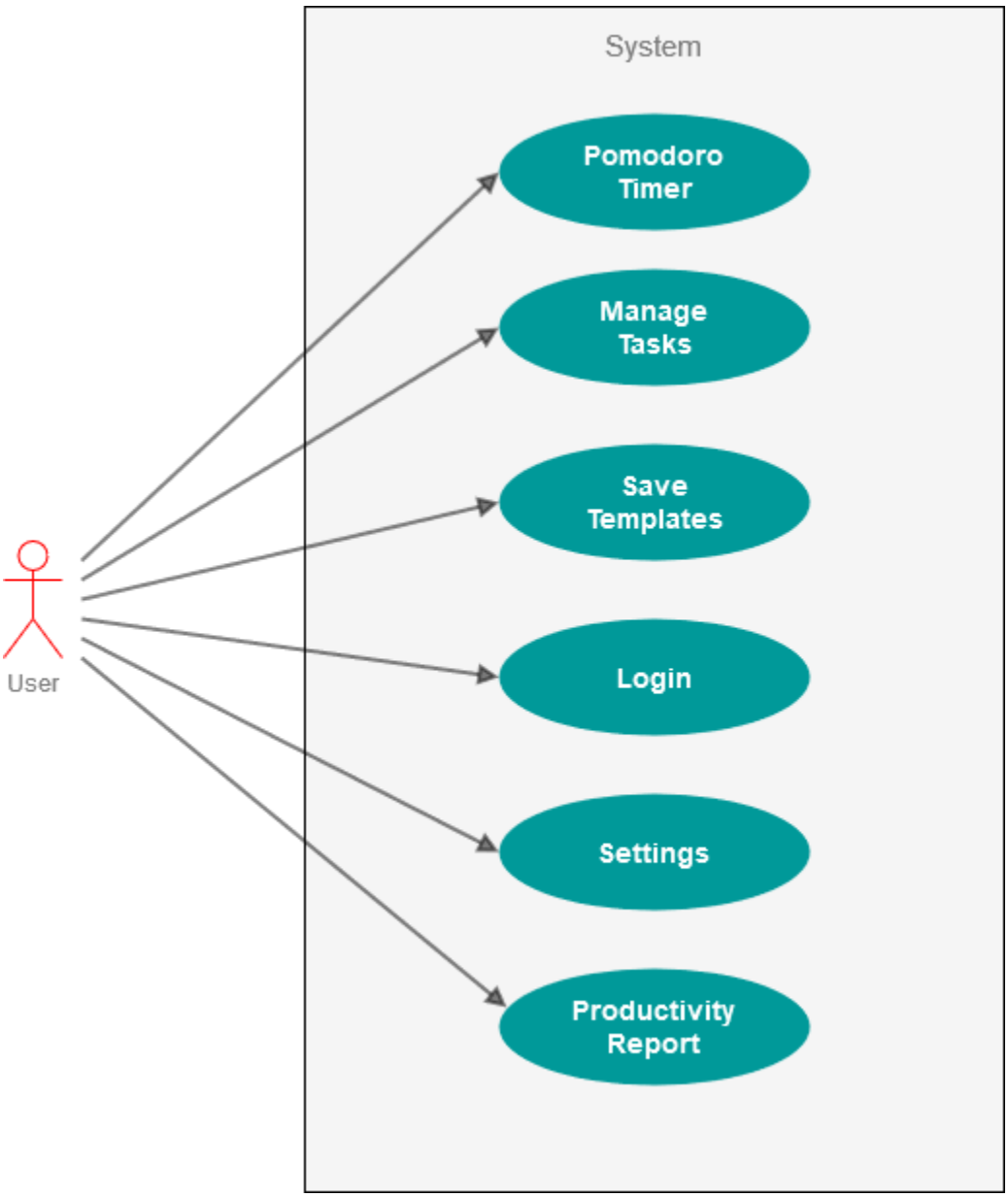
Its advantages are:

- Prepares software fast.
- Clients have a clear idea of the project.
- Changes are easy to implement.
- Provides risk handling support, because of its iterations.
- Errors are easy to recognise.
- More flexible.

Its disadvantages are:

- A good team and proper planned execution are required.
- Because of its continous iteration cost increases.
- Well defined module interfaces are needed.

3. Use Case Diagram



4. Use Cases

4.1. Pomodoro Timer

4.1.1. Brief Description

This use case enables users to use pomodoro technique(start a pomodoro, short break and long break) and keeps track of the number of pomodoros completed.

4.1.2. Actors

The following actor(s) interact or participate in this use case:

Individual, student, professional, anyone who wants to be more productive.

4.1.3. Flow of Events

4.1.3.1. Basic Flow

This use case starts when a user wishes to use pomodoro technique to complete a task.

1. The user presses the start button and a pomodoro is started.
2. User works on his single task until the timer rings.
3. When the session ends,the number of pomdoros is increased by one.
4. Then start a short break.
5. After four pomodoros, take a long break.
6. User repeats the above steps until the task is completed.

4.1.3.2. Alternative Flow

4.1.3.2.1. User switches to break before pomodoro is over

If in the basic flow, the actor switches to the break tab(short break or long break) before the timer rings, the system displays a confirmation message that the timer is running and asks if they want to switch. If the user denies, the timer continues. If the user confirms, the timer switches to the selected tab and the timer resets, at which point the use case ends.

4.1.4. Special Requirements

None.

4.1.5. Pre-Conditions

None.

4.1.6. Post-Conditions
If the pomodoro is completed, the timer rings. If not, timer does not ring.

4.1.7. Extension Points
None.

4.2. Manage Tasks
4.2.1. Brief Description
This use case enables users to create, remove and update tasks.

4.2.2. Actors
The following actor(s) interact or participate in this use case:

Individual, student, professional, anyone who wants to be more productive.

4.2.3. Flow of Events
4.2.3.1. Basic Flow
This use case starts when the user wishes to add a task.

1. User presses the add task button.
2. A dialog is opened that asks the user to enter the name of the task and the number of pomodoros they think it would take to complete the tasks.
3. The user presses save.
4. The task is added to the task list.
5. The task is selected as the current task and its name is displayed.
6. The number of completed pomodoros of the current task updates as the pomodoros are completed.

4.2.3.2. Alternate Flow
4.2.3.2.1. User presses the cance button
If the actor presses the cancel button instead of the save button, the task is not created.

4.2.3.2.2. Change current task
If the actor presses on another task from the tasklist, the task is selected as the current task and displayed.

4.2.3.2.3. Modify current task
If the actor presses the modify current task button from the kebab menu, a dialog box appears in which the actor can modify task

name, and and the number of estimated pomodoros.

4.2.3.2.4. Mark as completed
If the actor presses the mark as completed button from the kebab menu, the current task is strikethrough in the task list.

4.2.3.2.5. Mark is incomplete
If the actor presses the mark as incomplete button from the kebab menu, the current task is unstrikethrough in the task list, if the current task was marked as completed.

4.2.3.2.6. Clear current task
If the actor presses the clear current task button from the kebab menu, the current task is removed from the task list. The task that was added after the removed task is now the current task.

4.2.3.2.7. Clear all tasks
If the actor presses the mark as completed button from the kebab menu, a confirmation dialog is shown which tells the actor that this will remove all the tasks from the task list. If the actor confirms, then all the tasks are removed from the task list.

4.2.4. Special Requirements
None.

4.2.5. Pre-Conditions
None.

4.2.6. Post-Conditions
None.

4.2.7. Extension Points
None.

4.3. Save Templates
4.3.1. Brief Description
This use case enables users to save repetitive tasks as templates and add them with just one click.

4.3.2. Actors

The following actor(s) interact or participate in this use case:

Individual, student, professional, anyone who wants to be more productive.

4.3.3. Flow of Events

4.3.3.1. Basic Flow

This use case starts when the user wishes to save task(s) as template.

1. The user presses save as template button from the kebab menu.
2. A dialog is shown which asks user to enter name of the template.
3. All the tasks currently in the tasklist are saved as a template with that name.

4.3.3.2. Alternate Flow

4.3.3.2.1. Tasklist is empty

If the actor presses the save as template button, but the tasklist is empty, nothing will happen, at which point the case ends.

4.3.3.2.2. Add from template

If the actor presses on the add from template button from the kebab menu, a dialog box will appear with the spinner with the names of saved templates. The actor will select the template he/she wants to load and all the tasks from the templates will be entered into the task list. Note that if there were tasks present in the tasklist before importing, they will be cleared.

4.3.4. Special Requirements

None.

4.3.5. Pre-Conditions

None.

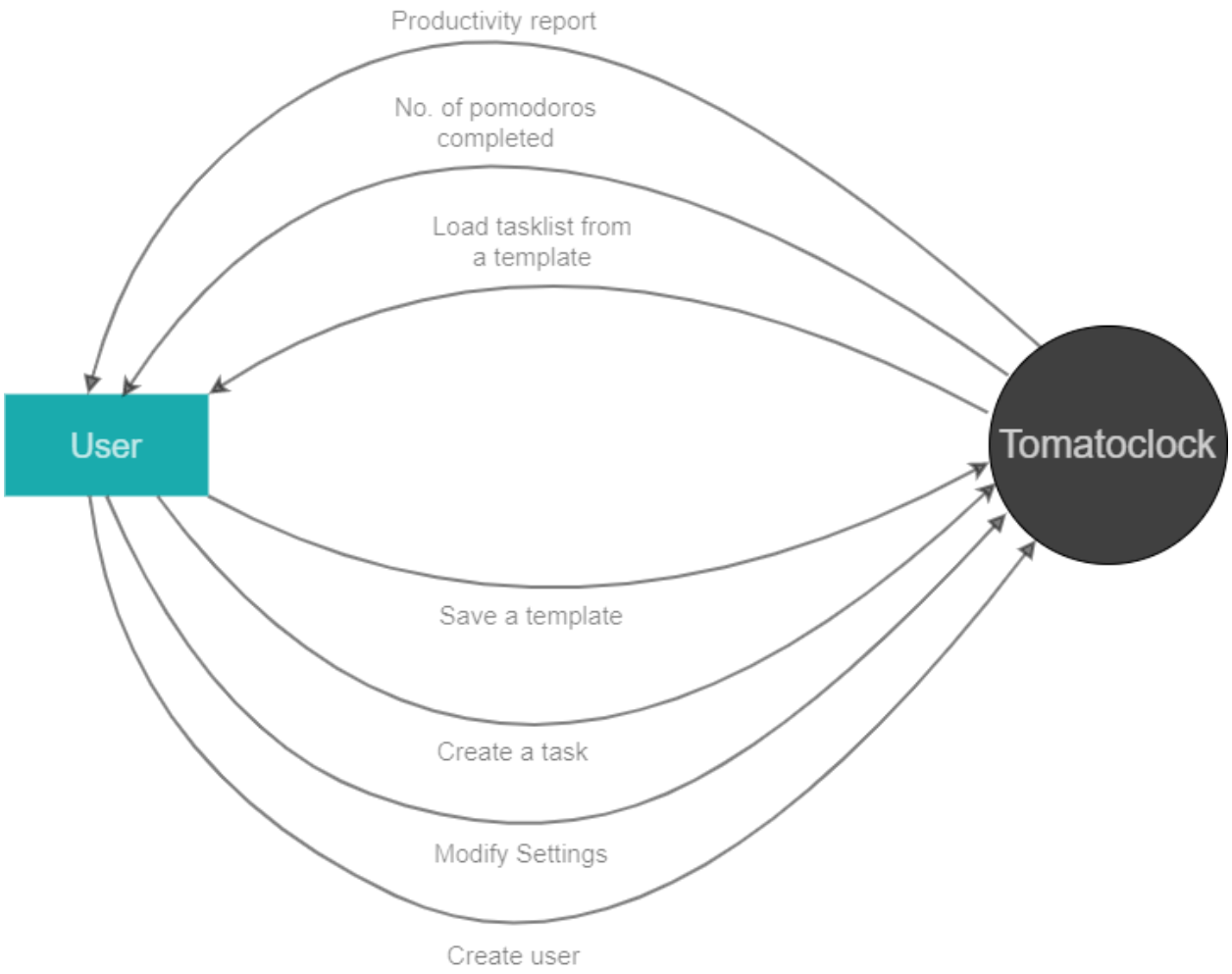
4.3.6. Post-Conditions

None.

4.3.7. Extension Points

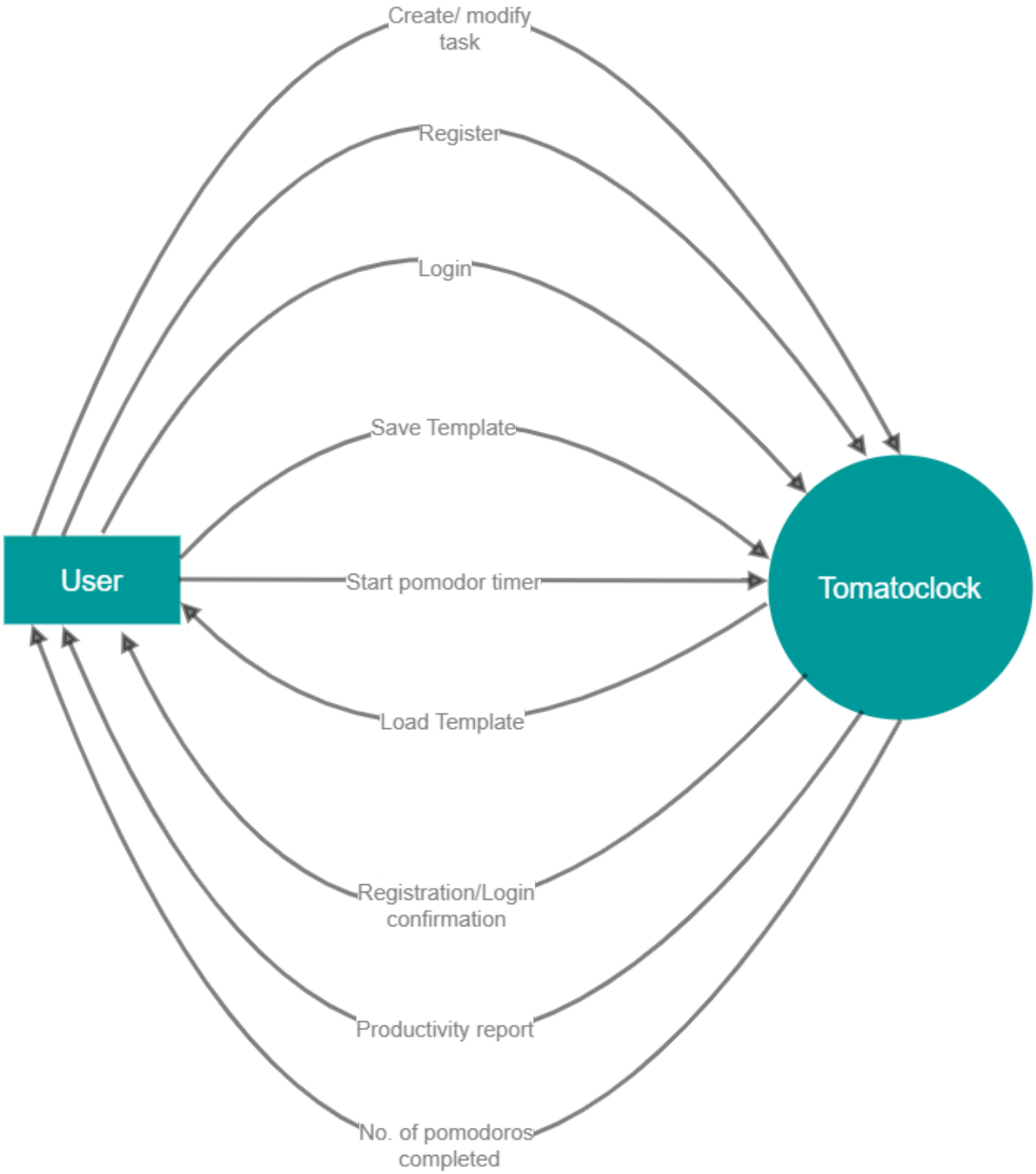
None.

5. Context Diagram



6. Level-n Data Flow Diagrams

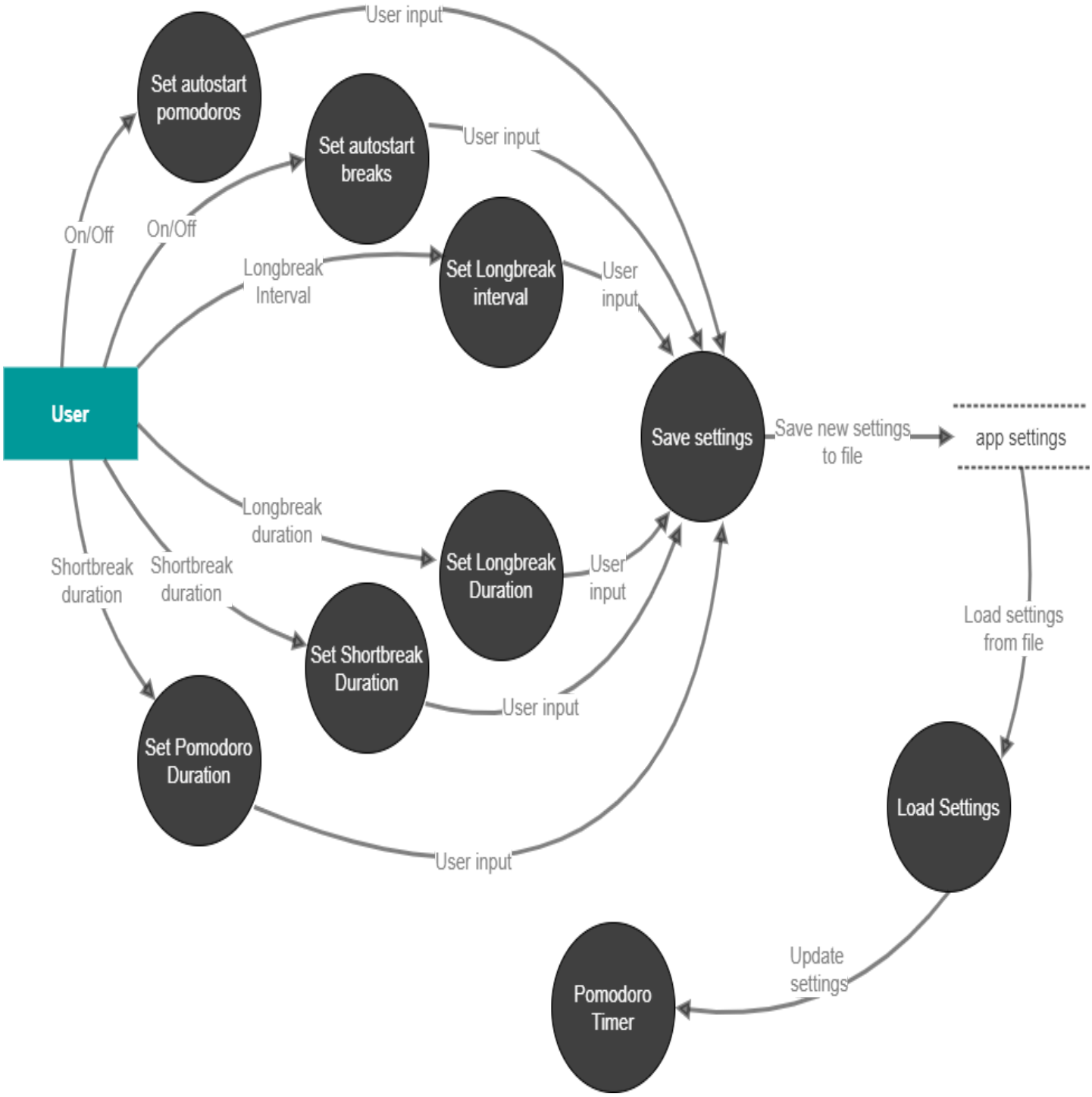
6.1. Level-0 Data Flow Diagram



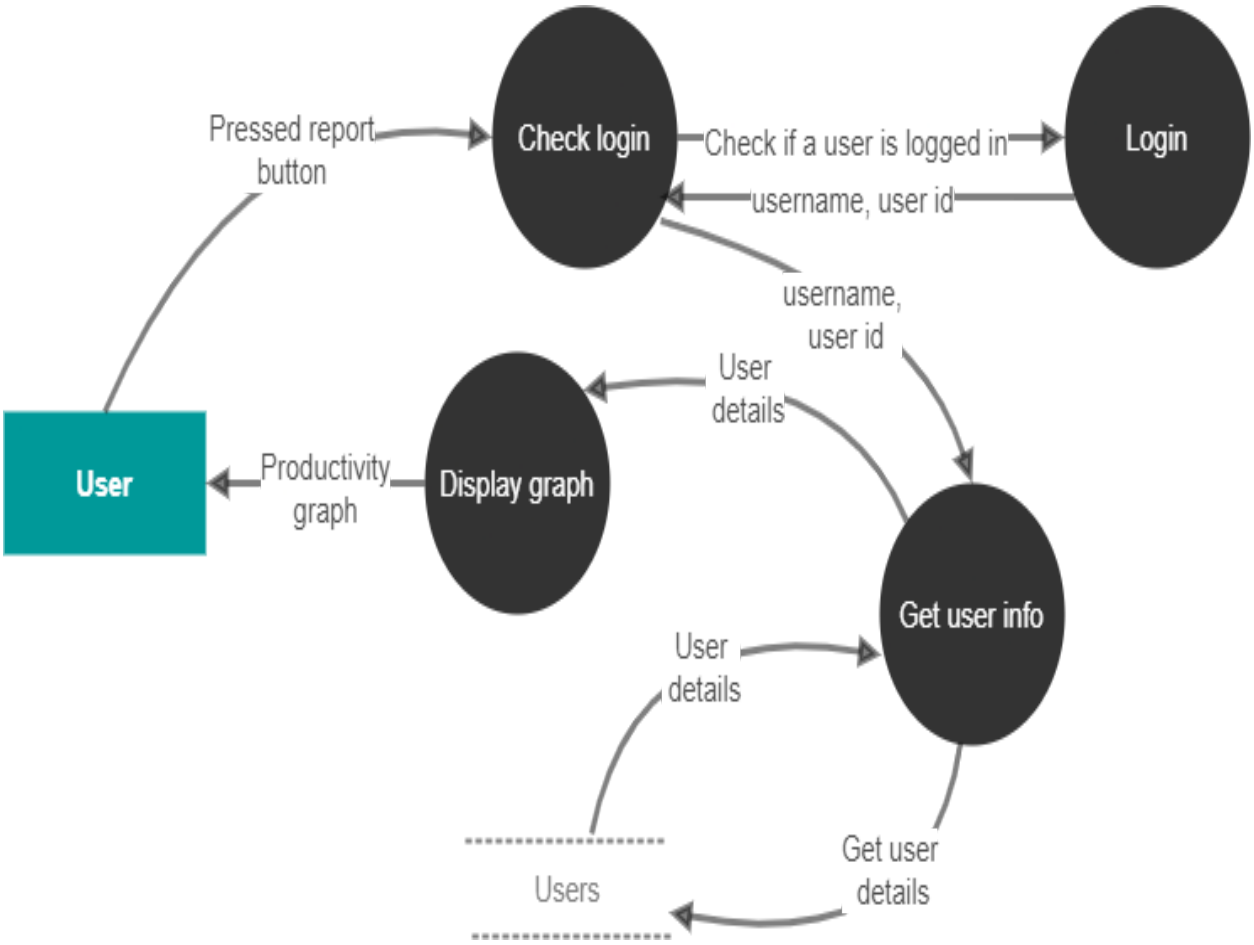
6.2. Level-1 Data Flow Diagram

6.3. Level-2 Data Flow Diagram

6.3.1. Settings



6.3.2. Report



7. Data Dictionary

The data dictionary provides an organized approach for representing the characteristics of each data object and control item. It has been proposed for describing the content of objects defined during structured analysis. A data dictionary is very important in the software development process because:

- A data dictionary lists standard terminology for use by an engineer working on a project.
- The dictionary provides the analyst with means to determine the definition of different data structures in the terms of their component elements.

The format of data dictionary is as follows:

- Name the primary name of the data or control item, the data store or an external entity.
- Aliases-other names used for the first entity.
- Description of a notion for representing content.
- Type of the data.

Data	Description
LoginDetails	Username + Password
UserDetails	Username + PasswordHash + id + totalHoursFocused + hoursFocusedInthisWeek + keepLoggedIn
Templates	Tasklist[taskname | est. number of pomodoros]
AppSettings	PomodoroDuration + ShortBreakDuration + LongBreakDuration + LongBreakInterval + autoStartPomodoros + autoStartBreaks
Task	taskName + est. number of pomodoros + number of pomodorosCompleted + taskCompleted
PomodoroCount	total number of pomodoros completed in the session

8. SRS Document

8.1. Introduction

This document aims at defining the overall software requirements for Tomatoclock: a pomodoro timer. Efforts have been made to define the requirements exhaustively and accurately. The final product will be having onlyfeatures/ functionalities mentioned in this document and assumptions for any additional feature/functionality should not be made by any of the parties involved in developing/testing/implementing/using this product. In case it is required to have additional features, a formal change request will need to be raised and subsequently a new release of this document and/or the product shall be produced.

8.1.1. Purpose

This specification document describes the capabilities that will be provided by the software application "Tomatoclock: a pomodoro timer". It also states the various required constraints by which the system will abide by. The intended audience for this document are the members of the software development team, testing team and the end usersof the product.

8.1.2. Scope
Tomatoclock is a time management application that incorporates the Pomodoro Technique to help users manage their time more effectively, improve productivity, and reduce stress. It includes a user-friendly interface, customizable pomodoro and break durations, task tracking, progress monitoring, data analysis, and productivity graph to track user's productivity in a week. It is designed for anyone who wants to stay focused on their tasks, especially those who struggle with time management and find it difficult to stay focused for extended periods.
The application is available on Windows 10 PC and requires JDK 17 and above. Although the application lacks some productivity tracking features, has minor bugs, and works only on desktop, it aims to help users be more productive and track their progress. The success of the application will be determined by its usefulness to its users.

8.1.3. Definitions, Acronyms, and Abbrevations
Following definitions and abbreviations have been used throughout this document:
JDK: Java Development Kit
Pomodoro Technique: A time management technique developed by Francesco Cirillo.
Pomodoro: A 25 minute time interval.(Generally)

8.1.4. References
(i) "The Pomodoro Technique: The Acclaimed Time Management System That Has Transformed How We Work", by Francesco Cirillo.

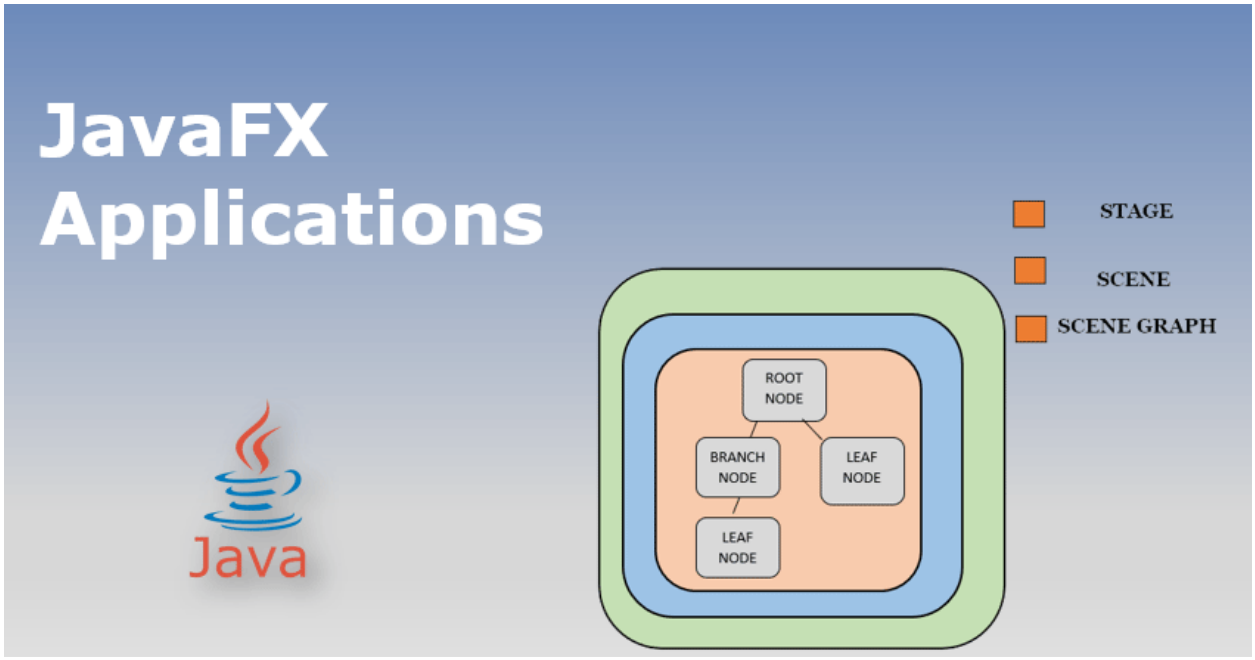
8.1.2. Overview
The rest of this SRS document describes the various system requirements, features, interfaces and functionalities in detail. The second chapter of the SRS deals with and introduces the reader to the system with an overview of the system functionality and system interaction with other systems. The third chapter provides the requirements specification in detailed terms and a description of the different system features.

8.2. Overall Description
Pomodoro technique is a popular time management technique which is used by many people to help them manage their time more effectively, improve productivity, and reduce stress. But, Humans are fallible. No matter how motivated you are at the start of the day, it's really hard to

actually stick to your pomodoros. So hold yourself accountable with a break reminder app. There is a need for a time management application that incorporates the Pomodoro Technique, provides a user-friendly interface, and includes features such as task tracking, progress monitoring, data analysis to help manage their time more effectively, improve productivity, and reduce stress. Tomatoclock provides a user-friendly interface, customizable pomodoro and break durations, task tracking, progress monitoring, data analysis, and productivity graph to track user's productivity in a week. It is designed for anyone who wants to stay focused on their tasks, especially those who struggle with time management and find it difficult to stay focused for extended periods.

8.2.1. Product Perspective

The application will be windows based, self-contained and independent.



8.2.1.1. System Interfaces

None

8.2.1.2. User Interfaces

The front-end application shall have responsive, accessible interfaces. The following screens shall be provided:

1. The main screen has three tabs, namely, pomodoro tab, shortbreak tab, and longscreen tab. In these tabs, users can see the timer and a

start/stop button.The screen changes color based on the tab which is opened. On the top there is the app name and logo on the left side and three buttons(report, settings, login) on the right. On the bottom there is a task list where the added tasks will be shown. On the top right of the task lsit, there is a kebab menu which lists more options.

2. A login/ registration screen for entering username and password. This screen will be used by individuals to create a user.
3. Settings screen for changing settings for the application. The user can change pomodoro duration, shortbread duration, longbreak duration, longbreak interval, autobreaks and autopomodoro from this screen. This screen also contains version information and ABOUT information.
4. Report screen displays information about the time the user has spent in the past week through a line chart.

8.2.1.3. Hardware Interfaces

The system shall run as a windows native application offering and interfacing with the application shall need the following hardware interfaces:

1. **Internet**: No need for an internet connection.
2. **Processor**: 1 gigaherts(GHz) or faster or SoC.
3. **RAM**: 2GB.
4. **Hard disk space**: 100 MB.
5. **Graphics Card**: DirectX or later with WDDM 1.0 driver.
6. **Display**: 800 x 600

8.2.1.4. Software Interfaces

The following software interfaces will be needed to deploy the said system:

1. Java Development Kit 17.
2. Windows 10(64 bit)operating system.

8.2.1.5. Communication Interfaces

None

8.2.1.6. Memory Interfaces

2 GB of ram and 100 MB of hard disk space.

8.2.1.7. Operations
None

8.2.1.8. Site Adaptation Requirements
None

8.2.2. Product Perspective
The system shall allow access to any individual who wants to use the application, although to access certain features a user has to login. Depending upon whether the user is logged in or not, they will be able to access specific modules of the system and utilise the folloowingg functions:

1. A Login function for users who have already registered and want to login .
2. A Registration function for users who want to register themselves in the application.
3. Pomodoro timer for any user who wants to use pomodoro technique.
4. Settings can be changed by any user who wishes to change application prefrences.
5. Tasks can be created, modified and deleted by any user who uses the application.
6. Templates(stored task lists) can be accessed, modified and deleted by any user who uses the application.
7. To view a user's Productivity Report, the user has to be logged in with his/ her credentials.

8.2.3. User Characterstics
1. General knowledge: Should know what pomodoro technique is and how to use it effectively.
2. Technical expertise: Should be able to install JDK dependency and be comfortable using a desktop application.
3. Education level: Should be able to read and write English language.

8.2.4. Constraints
1. The application might contain some minor bugs.
2. The productivity report lacks some useful features.
3. The windows are not resizable.
4. Is not fully self-contained, as it requires JDK to run.

8.2.5. Assumptions and Dependencies

1. This application requires JDK 17 or above to run.

8.2.6. Assumptions and Dependencies

This project is still in beta testing, so some minor bugs can be removed in next iterations. Also some features are not fully fleshed out due to time constraints, and incremental improvements will be made in subsequent iterations of the application.

8.3. Specific Requirements

This section contains the software requirements to a level of detail sufficient to enable system designers to design the system, and testers to test the system.

8.3.1. External interface Requirements

8.3.1.1. User interfaces

Pomodoro Screen

It is the main screen of the application. It implements the pomodoro technique. Various elements to be included on this screen are:

1. Pomodoro tab
2. Short break tab
3. Long break tab
4. Timer
5. Start/stop button
6. Add task button
7. Task list
8. Kebab menu for more options
9. Report button
10. Settings button
11. Login button
12. Name and logo of the application.

Tasklist Screen

The tasklist is a sub-screen present inside the Pomodoro screen. It should contain following elements:

1. List of tasks added by the user.
2. Each task should mention its name, (pomodoros completed / estimated no of pomodoros), and completion status.

Login Screen

Login screen allows individuals to enter their credentials and login as a user. It should contain following elements:

1. Username field
2. Password field
3. Kepp me logged in checkbox.
4. Registration button, if the individual is not registered.
5. Login button.

Registration Screen

Registration screen allows individuals to register themselves as a user in the application. It should contain following elements:

1. Username
2. Password
3. Repeat password
4. Go to Login button, to go to login screen
5. Register button.

Settings Screen

Settings screen allows individuals to change application preferences to their liking, e.g , if they want to change short break duration or they want to auto start pomodoros, etc. It should contain following elements:

1. Pomodoro duration field.
2. Short break duration field.
3. Long break duration field.
4. Auto start breaks check box.
5. Auto start pomodoros check box.
6. Long break interval field.
7. ABOUT information.
8. Reset all settings to the default values button.

Report Screen

This screen allows a logged in user to track his/ her productivity in the past week. It contains following elements:

1. A line chart of the number of hours the user spent working per day in the last seven days.

8.3.1.2. Hardware interfaces

As stated in Section 8.2.1.3.

8.3.1.3. Software interfaces

As stated in Section 8.2.1.4.

8.3.1.4. Communication interfaces

As stated in Section 8.2.1.5.

8.3.2. System Features

8.3.2.1. Pomodoro Timer

Description

The system will allow users to use pomodoro technique to focus on the task he/ she wants to complete. It stores the number of pomodoros completed in the session. It also plays an alarm when the timer is over.

Validity Checks

1. If the user switches tabs and the timer is running, the user will be shown an alert dialog telling that the timer will be reset on switching tabs and pomodoro will not be counted.

Sequencing Information

The pomodoro duration, short break duration, long break duration and long break interval will be set to their default values, if the user has not changed them. So there is no strict need to add prior information to use this module.

Error Handling / Response to Abnormal Situations

If any of the above validations / sequencing flow does not hold true, appropriate error and warning messages should be displayed to the user and instruct them to do the needful.

8.3.2.2. Task Management

Description

This system allows users to create, update and delete tasks. Each task shows a task name, estimated number of pomodoros, completed pomodoros and completion status.

Validity Checks

1. User needs to provide a valid name for the task before the task is created.
2. The task name should be less than 30 characters.

Sequencing Information

A valid task name has to be provided before a task is added to the tasklist.

Error Handling / Response to Abnormal Situations

1. If the user does not provide a valid name for the task, the user wil not be able to create the task.
2. If the task name is longer than 30 characters, it will be truncated.

8.3.2.3. Templates

Description

This system allows users to save repetitive tasks as templates. A template can be loaded later, and all the tasks in the template will be added to the tasklist.

Validity Checks

1. User needs to provide a valid name for the template(should not contain special characters like *, \, etc).
2. There should be atleast one task present in the tasklist to store it as a template.

Sequencing Information

User needs to provide a valid name for the template and there should be atleast one task present in the tasklist.

Error Handling / Response to Abnormal Situations

1. If the template name contains invalid characters the template will not be created.
2. If the task list is empty, the user won't get the promt to enter template name, hence won't be able to create the template.

8.3.2.4. Settings

Description

This system allows users to change various application preferences such as pomodoro duration, short break duration , long break duration, long break interval, auto start pomodoros and auto start breaks. it also allows users to reset all the preferences to their default value.

Validity Checks

1. Users need to provide valid values for pomodoro duration, short break duration , long break duration, long break interval fields.

Sequencing Information

If the user changes the default values, he/she needs to provide valid values for pomodoro duration, short break duration , long break duration, long break interval fields.

Error Handling / Response to Abnormal Situations

1. If the user enters invalid values for the pomodoro duration, short break duration , long break duration, long break interval fields, the user will not be able to sve the changed settings.

8.3.2.5. Productivity Report

Description

This system allows a logged in user to track his/ her productivity in the past week. It shows a line chart of the number of hours the user spent working per day in the last seven days.

Validity Checks

None

Sequencing Information

The user should have worked for atleast one hour in the past seven days for the data to be shown in the line chart.

Error Handling / Response to Abnormal Situations

None

8.3.2.6. Registration

Description

This system allows individuals to register as a user in the application. This allows users to access certain functionality in the application(i.e. reports).

Validity Checks

1. The password ad the repeat password fields must match
2. The password must have atleast 8 symbols, a lowercase letter, an uppercase letter and a number.

Sequencing Information

The user needs to provide a valid username, password.

Error Handling / Response to Abnormal Situations

If any of the above validations / sequencing flow does not hold true, appropriate error and warning messages should be displayed to the user and instruct them to do the needful.

8.3.2.7. Login

Description

This allows individuals to login as users by entering their credentials.

Validity Checks

The username should be present in users list, and the password should match the respective passordHash.

Sequencing Information

The user needs to enter a valid username and password to login.

Error Handling / Response to Abnormal Situations

If any of the above validations / sequencing flow does not hold true, appropriate error and warning messages should be displayed to the user and instruct them to do the needful.

8.3.3. Performance Requirements

1. The application should reliably work most of the time.
2. The application should be able to handle multiple users.

8.3.4. Design Constraints

None

8.3.5. Software System Atributes

8.3.5.1. Availability

1. The system should have very high uptime.
2. User data is stored on users' devices locally, so there should be very low latency and high availability.

8.3.5.2. Security

1. The user productivity data is secured by username and password.

8.3.5.3. Maintainability

The application will be designed in a maintainable manner. It will be easy to incorporate new requirements in the individual modules (i.e., pomodoro timer, login, registration, task management, templates, report, settings).

8.3.5.4. Portability

The application is currently only compiled for Windows 10 operating system.

8.3.5.5. Logical Database Requirements
None.

8.3.5.6. Other Requirements
None.

9. Project Management
9.1. Project Estimation
9.1.1. Function points Analysis
This section strives to estimate the size of the software delivered independent of the programming languages, design patterns and toolchains used.

We tabulate the direct measures of software's information domain as follows:

Info. Domain Values	Count	Complexity	Weight Factor	Count Total
Pomodoro Timer				
External Inputs(EIs)	1	Simple	3	3
External Outputs(EOs)	2	Simple	4	8
External Inquiries(EQs)	0	Simple	3	0
Internal Logical Files(ILFs)	1	Complex	15	15
External Interface Files(EIFs)	0	Simple	5	0
Settings				
External Inputs(EIs)	6	Simple	3	12
External Outputs(EOs)	0	Simple	4	0
External Inquiries(EQs)	0	Simple	3	0
Internal Logical Files(ILFs)	1	Average	10	10
External Interface Files(EIFs)	1	Simple	5	5

Report				
External Inputs(EIs)	0	Simple	3	0
External Outputs(EOs)	1	Average	5	5
External Inquiries(EQs)	0	Simple	4	0
Internal Logical Files(ILFs)	1	Simple	7	7
External Interface Files(EIFs)	1	Average	7	7
Manage Tasks				
External Inputs(EIs)	2	Simple	3	6
External Outputs(EOs)	1	Simple	4	4
External Inquiries(EQs)	0	Simple	4	0
Internal Logical Files(ILFs)	1	Average	10	10
External Interface Files(EIFs)	0	Simple	5	0
Templates				
External Inputs(EIs)	1	Simple	3	3
External Outputs(EOs)	1	Average	5	5
External Inquiries(EQs)	0	Simple	4	0
Internal Logical Files(ILFs)	1	Average	10	10
External Interface Files(EIFs)	1	Average	7	7
Registration				
External Inputs(EIs)	3	Simple	3	9
External Outputs(EOs)	1	Simple	4	4
External Inquiries(EQs)	0	Simple	4	0
Internal Logical Files(ILFs)	1	Average	10	10
External Interface Files(EIFs)	0	Simple	5	0

Login				
External Inputs(EIs)	2	Simple	3	6
External Outputs(EOs)	1	Simple	4	4
External Inquiries(EQs)	2	Simple	4	8
Internal Logical Files(ILFs)	1	Average	10	10
External Interface Files(EIFs)	1	Average	7	7

Table 1 Calculation of FP

We answer the following questions pertaining to the General System Characteristics on a scale of 0-5 as follows, where a 0-pointer implies the characteristic has no influence, 1-pointers denote an incidental influence, 2-pointers denote moderate influence, 3-pointers denote average influence, 4- pointers denote significant influence, and 5-pointers denote that the aspect is essential to the system:

S. NO	Questions	Points
1.	Are specialized data communications required to transfer information to or from the application?	0
2.	Are there distributed processing functions?	0
3.	Is performance critical?	2
4.	Will the system run in an existing, heavily utilized operational environment?	3
5.	Does the online data entry require the input transaction to be built over multiple screens or operations?	0
6.	Does the system require online data entry?	0
7.	Does the system require reliable backup and recovery?	2
8.	Are the ILFs updated online?	0
9.	Is the internal processing complex?	3
10.	Is the code designed to be reusable?	3

11.	Are conversion and installation included in the design?	0
12.	Are the inputs, outputs, files, or inquiries complex?	3
13.	Is the system designed for multiple installations in different organizations?	1
14.	Is the application designed to facilitate change and ease of use by the user?	4

Table 2 Value Adjustment Factor Questionnaire

Now, the Function Points (FP) for the project can be calculated, taking the Cost Adjustment Factor(CAF) or Value Adjustment Factor (VAF) in account, as follows:

$$FP = Count\ Total \cdot VAF = Count\ Total \cdot \left(0.65 + (0.01 \cdot \sum Fi)\right)$$

The Count Total had been computed in Table 1 as 175. The sum $\sum Fi$ can be computed from the values in Table 2 as 21. Using the given information, the Function Points (FP) for the project is computed as,

$$FP = Count\ Total \cdot \left(0.65 + \left(0.01 \cdot \sum Fi\right)\right)$$

or, FP = 175• (0.65 + (0.01 • 21))
or, FP = 175• 0.86
or, FP = 150.2 ≈ 151

Therefore, the Function Points (FP) for the project is 151.
Assuming organizational average productivity for systems of this type with low developer experience and environment maturity is 6.5 FP / person - month. Based on a burdened labour rate of $8000 per month, we estimate the cost and effort needed to do the project in this section.

Effort Estimation

The Estimated Effort in person-months using Function Point Analysis can be calculated as,

$$Effort\ Estimate = FP\ /\ PROD$$

or, $Effort\ Esimate = 151\ FP\ /\ 6.5\ FP\ per\ pm$
or, $Effort\ Estimate = 23.23\ pm \approx 23.2\ pm \approx 24\ pm$

Therefore, using FPA, the Estimated Effort for the project is 24 person-months.

Cost Estimation

As the burdened labour rate is $8000 per month, the Cost per Function Point roughly translates to $8000 per month / 6.5 FP per person-months or $1230.77 per FP. As the Function Points for the project are 151 , the Estimated Cost of the project can be calculated as,

$Cost\ Estimate = FP \bullet Cost\ per\ FP$
or, $Cost\ Estimate = \$\ (151 \bullet 1230.77) \approx \$\ 185847$
Therefore, using FPA, the Estimated Cost for the project is $185847.

9.1.2. COCOMO-II

The Base Equation to estimate effort in the Early Design Stage of the project, which uses KLOC as a measure of size, using COCOMO-II is as follows:

$$PMnominal = A \bullet Size^{B}$$

where $PM_{nominal}$ is the estimated effort for the project in person months, A is constant representing the nominal productivity, B is an exponent that depends on scaling factors and Size denotes the size of the software project.

The Application Composition Model assumes value of B equal to 1, however, the Early Design Model assumes the value of B to be greater than 1. The exponent B is defined as,

$$B = 0.91 + (0.01 \cdot \sum_{i=1}^{5} SF_i)$$

Each scaling factor is assigned values corresponding to their relevance in this project according to assumptions are as follows:

S. No.	Scaling Factor	Remarks	Value
1.	Precedentness	High	2.48
2.	Development Flexibility	Average	3.04
3.	Architecture / Risk Resolution	High	2.83
4.	Team Cohesion	Extra High	0.00
5.	Process Maturity	High	3.12
		Total	12.46

Table 3 Scaling Factors for COCOMO-II

Effort Estimation

For initial calibration, we take A = 2.5 and we have B = 1.025 from Table 3 and the definition of B. Taking Java as the primary programming language this project will be developed in, the Mean LOC/FP for the language is approximately 54, which implies that for a count total of 476, The KLOC metric is 25.693. Substituting these values in the COCOMO-II Base Equation, we get,

$$PMnominal = 2.5 \cdot 25.6931.025\ pm \approx 70\ pm$$

Therefore, according to COCOMO-II, the Effort Estimate is 70 person-months.

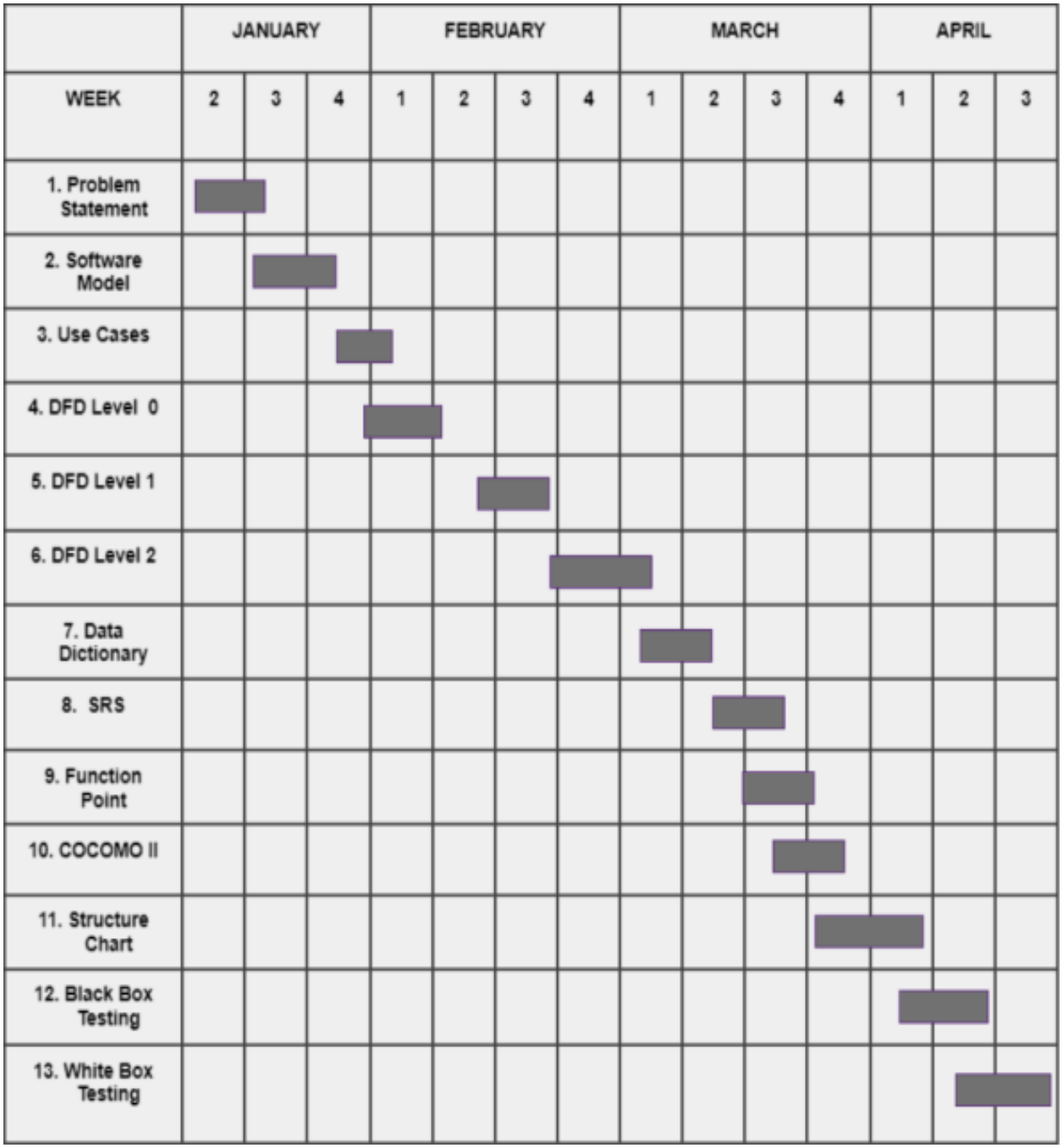
Cost Estimation

Using a burdened labour rate of $8000 per month and assuming Estimated LOC per person to be 351 LOC per person-month, we obtain a cost of $22.79 per LOC taking into account language preference and productivity in Section 9.1.1. We have the Cost Estimate as,

$$Cost\ Estimate = LOC \cdot Cost\ per\ LOC = \$\ (25693 \cdot 22.79) \approx \$\ 585543$$

Therefore, the Cost Estimate using COCOMO-II is $585543.

9.2. Gantt Chart



10. Software Testing
10.1. Black Box Testing
10.1.1 Test Cases
Experience shows that test cases that are close to boundary conditions have a higher chance of detecting an error. Here, boundary conditions means an input value may be on the boundary, just below the boundary (upper side) or just above the boundary (lower side).
We, here, perform Black Box Testing on the LOGIN module. The username and password both should be strictly alphabets and in order to login the username and password will be checked with the database.

Best Case
As we know, with the single fault assumption theory, 4n + 1 test cases can be designed and which are here in this case equal to 9. The boundary value test cases are:
n=2(number of inputs)

Test Case	Username	Password	Expected output
1.	manna	manna	Login successful
2.	manna	home	Incorrect password
3.	kiya	kite	Login successful
4.	aditi	blue	Incorrect password
5.	aditi	aditi	Login successful
6.	harry	miyan	User does not exist
7.	blue	moon	User does not exist
8.	blue	jackal	User does not exist
9.	mona	mona	Login successful

Worst Case
If we reject the “single fault” assumption theory of reliability and would like to see what happens when more than one variable has an extreme value. Worst case testing for a function of n variables generates 5n test cases. Our login module takes 2 variables input and will have 5^2 cases.

Test Case	Username	Password	Expected output
1.	manna	grapes	Incorrect password
2.	manna	home	Incorrect password
3.	manna	white	Incorrect password
4.	manna	morgan	Incorrect password
5.	kiya	kite	Login successful
6.	manna	manna	Login successful
7.	manna	tiara	Incorrect password
8.	kiya	snake	Incorrect password
9.	kiya	hope	Incorrect password
10.	kiya	red	Incorrect password
11.	kiya	jute	Incorrect password
12.	maala	maala	User does not exist
13.	zaan	begum	User does not exist
14.	yello	line	User does not exist
15.	manna	umbrella	Incorrect password
16.	aditi	aditi	Incorrect password
17.	aditi	ball	Incorrect password
18.	aditi	Green	Incorrect password
19.	aditi	LIGHTS	Incorrect password
20.	aditi	camera	Incorrect password
21.	mona	MONA	Incorrect password
22.	mona	mona	Login successful
23.	mona	final	Incorrect password
24.	mona	staet	Incorrect password
25.	mona	kon	Incorrect password

Robustness Testing

It is an extension to the boundary value analysis. Here, we would like to see what happens when the extreme values are exceeded with a value slightly greater than the maximum and value slightly less than minimum. It means, we want to go outside the legitimate boundary of the input domain. There are four additional test cases which are outside the
legitimate input domains. Hence, total test cases in robustness testing is 6n + 1, where n is the number of input variables. So, here:
cases n=2
Total cases 6(2)+1=13

Test Case	Username	Password	Expected output
1.	@qrut	1234	User does not exist
2.	#wet	123@gt	Incorrect password
3.	00987	b124	Incorrect password
4.	2345	678	Incorrect password
5.	manna	manna	Login successful
6.	mona	mona	Login successful
7.	aditi	aditi	Login successful
8.	kiya	kite	Login successful
9.	gaana	gaana	Login successful
10.	green	3por	Incorrect password
11.	white	QUH@3	User does not exist
12.	black	9775	Incorrect password
13.	blue	rewyu	User does not exist

10.2. White Box Testing

For the login module

10.2.1. Code

```
package application;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.Window;
public class LoginController implements Initializable {
	// Buttons
	@FXML
	private Button loginButton, registerButton;
	// Labels
	@FXML
	private Label errorMessage;
	// Other elements
	@FXML
	private CheckBox keepLoggedInCheckbox;
	@FXML
	private TextField usernameField;
	@FXML
	private PasswordField passwordField;
	@FXML
	private ImageView closeButton;
	@Override
```

```
	public void initialize(URL location, ResourceBundle resources) {
		closeButton.setImage(new
Image(System.getProperty("user.dir") + "/resources/close_icon.png"));
		loginButton.setOnMouseEntered(e -> loginButton.setStyle(
				"-fx-background-color: #bd93f9; -fx-text-fill:
#f8f8f2; -fx-font-size:18.0px; -fx-font-family:
Calibri;-fx-background-radius: 10"));
		loginButton.setOnMouseExited(e -> loginButton.setStyle(
				"-fx-background-color: #44475a; -fx-text-fill:
#f8f8f2; -fx-font-size:18.0px; -fx-font-family:
Calibri;-fx-background-radius: 10"));
		registerButton.setOnMouseEntered(e ->
registerButton.setStyle(
				"-fx-background-color: #bd93f9; -fx-text-fill:
#f8f8f2; -fx-font-size:18.0px; -fx-font-family:
Calibri;-fx-background-radius: 10"));
		registerButton.setOnMouseExited(e ->
registerButton.setStyle(
				"-fx-background-color: #44475a; -fx-text-fill:
#f8f8f2; -fx-font-size:18.0px; -fx-font-family:
Calibri;-fx-background-radius: 10"));
	}
	@FXML
	private void loginClicked() {

		String username = usernameField.getText();
		String password = passwordField.getText();
		User currentUser = null;
		// Resets error message text if there was any
		errorMessage.setTextFill(Color.rgb(255, 85, 85)); // red
		errorMessage.setText("");

		if (Controller.umap == null) {
			errorMessage.setText("No such user exists");
			return;
		}
		// If no username or no password inserted, display error
		if (username.isEmpty() || password.isEmpty())
			errorMessage.setText("Please fill all fields.");
		// Tries to construct user with given username and password
```

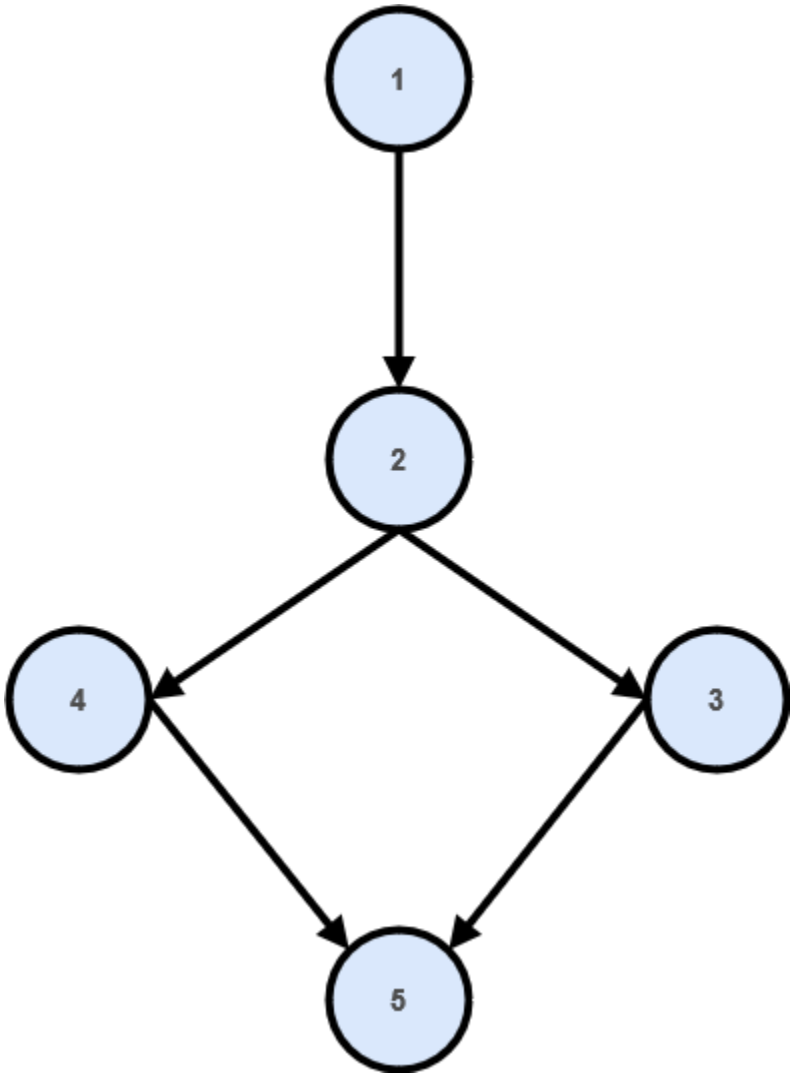
```
            else {
                  try {
                        currentUser = new User(username);
                  } catch (Exception e) {
                        errorMessage.setText("Such user does not
exist.");
                        currentUser = null;
                  }
            }
            // If user construction was successful & password is correct
logs in the user
            if (currentUser != null) {
                  if (!currentUser.passwordsMatch(password)) {
                        errorMessage.setText("Wrong password");
                  }
                  else {
                        // Saves user in the session
                        Session.beginSession(currentUser);
                        // Updates keepLoggedIn variable

currentUser.setKeepLoggedIn(keepLoggedInCheckbox.isSelected());
                        // display login message
                        errorMessage.setTextFill(Color.rgb(80, 250,
123)); // green
                        errorMessage.setText("Login Successful");
                  }
            }
      }
      @FXML
      private void registerClicked() {
            try {
                  FXMLLoader loader = new
FXMLLoader(getClass().getResource("Registration.fxml"));
                  Parent root = loader.load();
                  Stage stage = new Stage();
                  Scene scene = new Scene(root);

                  stage.initOwner(((Stage)
loginButton.getScene().getWindow()).getOwner());
                  stage.setTitle("Registration");
```

```
                stage.setResizable(false);
                stage.initModality(Modality.APPLICATION_MODAL); //
dialog always remains on top
                stage.initStyle(StageStyle.UNDECORATED); // to make
the dialog unmovable
                stage.setOnShown(event -> {
                      Window window =
loginButton.getScene().getWindow();
                      ((Stage) window).close();
                });
                stage.setScene(scene);
                stage.centerOnScreen();
                stage.show();
          } catch (Exception e) {
                System.out.println("Exception whilst changing scene
from Login to Register by registerButton.");
          }
     }
     @FXML
     private void keyPressed(KeyEvent event) {
          // Enter pressed
          if (event.getCode().toString().equals("ENTER"))
                loginClicked();
     }
     @FXML
     private void closeClicked(MouseEvent event) {
          Window window = registerButton.getScene().getWindow();
          ((Stage) window).close();
     }
}
```

11. Context Flow Graph



11.1. Cyclomatic complexity

It is a software metric used to indicate the complexity of a program. It was computed using the context flow graph of the program. It is calculated by the following formula:

$$V(G) = E\text{-}N+2*p$$

where
E = number of edges
N = number of nodes

ρ = number of connected components
also:

$$V(G) = P+1$$

where
P = number of predicate nodes
V(G)= number of regions

So, in our case:
V(G)= 1+1= 2, i. e Cyclomatic complexity is 2.

11.2. Independent Paths

R1 : 1 → 2 → 3
R2 : 1 → 2 → 4
Number of regions is 2.