

# **DATA STRUCTURE ALGORITHM WITH JAVA CipherSchools**

## **A Training Report**

Submitted in partial fulfilment of the requirements for the award of degree of

**B.Tech. (Computer Science Engineering)**

**Submitted to**

**LOVELY PROFESSIONAL UNIVERSITY PHAGWARA,  
PUNJAB**



**From June 4, 2025 to July 15, 2025**

**SUBMITTED BY**

**Name of student: Bablu Kumar**

**Registration Number: 12312283**

**Signature of the student: B.Kumar**

## **Annexure-II Student Declaration**

**To whom so ever it may concern**

I, **Bablu Kumar**, Registration Number [**12312283**], hereby declare that the work done by me on **“DATA STRUCTURE ALGORITHM WITH JAVA”** from **June, 2025** to **July, 2025**, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **B.Tech. in Computer Science Engineering**.

**Bablu Kumar** (12312283)

Dated: 12/08/2025



LOVELY  
PROFESSIONAL  
UNIVERSITY

## Training certificate from organization/ Company



### Certificate of Completion

This is to certify that

**Bablu Kumar**

studying at **Lovely Professional University**, has successfully completed training in **Data Structure and Algorithm using Java Programming** from CipherSchools during the period of **June 4, 2025 to July 15, 2025**.

The training comprised 70 hours of learning, and the participant's performance has been assessed as Satisfactory.

**ANURAG MISHRA**

Founder CipherSchools

Certificate ID : CSW2025-13023

CipherSchools, India



Scan to verify

## INTRIDUCTION OF THE PROJECT UNDERTAKEN

- **Objectives of the work undertaken:** The primary objective was to strengthen problem-solving and programming skills using Java by implementing various data structures and algorithms. The focus was on understanding the core concepts, their real-world applications, and optimizing solutions for efficiency.
- **Scope of the Work:** The scope included designing and implementing fundamental and advanced data structures (arrays, linked lists, stacks, queues, trees, graphs, hash tables) and algorithms (searching, sorting, recursion, greedy algorithms, dynamic programming) in Java. It also involved analyzing time and space complexities and applying these concepts to solve practical problems.
- **Importance and Applicability:** The project's importance lies in its role in building a strong foundation in algorithmic thinking and efficient coding practices. Mastery of DSA concepts is highly applicable in technical interviews, competitive programming, and the development of scalable, high-performance software systems.
- **Role and profile:** My role was that of a Java Developer trainee, responsible for designing, coding, testing, and optimizing various DSA implementations. I developed solutions from scratch, ensuring correctness and efficiency, under the guidance of the training program.

- **Company's Vision and Mission:** CipherSchools is a leading ed-tech platform with a mission to make high-quality, practical skill development accessible to students worldwide. Their vision is to bridge the gap between academic knowledge and industry requirements by providing comprehensive training in modern technologies, including programming, Data Structures, and Algorithms.
- **Origin and growth of company:** Founded to address the growing demand for skilled developers, CipherSchools has rapidly expanded its training programs, covering key areas like Java, DSA, and web development, while building a global community of learners.
- **Various departments and their functions:** CipherSchools operates with specialized teams dedicated to course content creation, technical mentorship, platform development, and student support, ensuring a seamless and high-quality learning experience for all enrolled learners.
- **Organization chart of the company:** As an online platform, the organization follows a structure with founders, content and curriculum leads, technical mentors, and support teams working together to deliver effective skill-based education.



## Brief Description of the work done

### **Position of Internship and roles:**

My position was a Summer Intern, focusing on Java programming and Data Structures & Algorithms (DSA). My primary role was to apply theoretical knowledge by implementing various data structures and algorithms in Java, demonstrating strong problem-solving and coding proficiency.

### **Activities/ equipment handled:**

The main activity was designing and implementing different data structures (arrays, linked lists, stacks, queues, trees, graphs, hash tables) and algorithms (sorting, searching, recursion, greedy algorithms, dynamic programming) in Java. This involved writing optimized code, analyzing time and space complexities, and testing solutions with various inputs. The primary "equipment" was my personal computer and the software environment for development, including IntelliJ IDEA/Eclipse, Java Development Kit (JDK), and Git for version control.

### **Challenges faced and how those were tackled:**

One key challenge was optimizing algorithms to achieve better time complexity while maintaining code readability. This was tackled by analyzing Big-O notations, refactoring code, and using efficient data structures. Another challenge was debugging complex recursion and dynamic programming solutions, which was solved by breaking down problems into smaller subproblems and using dry-run analysis.

### **Learning outcomes:**

I gained a solid understanding of core DSA concepts, improved my algorithmic thinking, and became proficient in implementing and optimizing solutions in Java. I also strengthened debugging and analytical skills, preparing for competitive programming and technical interviews.

### **Data analysis:**

Data analysis was performed on algorithm performance, measuring execution time and memory usage for different implementations to identify the most efficient approaches for given problem constraints.

## TABLE OF CONTENTS

S. No.	Title	Page
1	Declaration	2
2	Training Certificate	3
3	Acknowledgement	3
4	Introduction of Company/Work	4
5	Brief Description of work Done	5
6	<b>Chapter-1: INTRODUCTION</b>	7
	1.1 Problem Statement	7
	1.2 Project Overview	7
	1.3 Objectives	8
	1.4 Scope of the Project	8
	1.5 Relevance in Current Context	9
7	<b>Chapter-2: SYSTEM DESIGN AND ARCHITECTURE</b>	12
	2.1 System Architecture Overview	12
	2.2 Architectural Pattern: Model-View-Controller (MVC)	12
	2.3 Project and Package Structure Analysis	13
	2.4 Scene Management and Navigation	13
	2.5 UI Architecture with FXML and CSS	14
8	<b>Chapter-3: IMPLEMENTATION AND FEATURES</b>	15
	3.1 Core Technology: JavaFX	15
	3.2 Feature: Dashboard	15
	3.3 Feature: User Details	16
	3.4 Feature: Mood Tracker	17
	3.5 Feature: Breathing Exercise	18
	3.6 Feature: Gratitude Journal	19
9	<b>Chapter-4: DATA MANAGEMENT STRATEGY</b>	20
	4.1 Data Storage Approach: Volatile In-Memory	20
	4.2 Core Data Structure: Observable List	20
	4.3 Data Model Implementation	21
	4.4 Data Lifecycle	22
10	<b>Chapter-5: TESTING AND VALIDATION</b>	23

	5.1 Testing Strategy	23
	5.2 Test Case Matrix	24- 25
11	<b>Final Chapter: CONCLUSION AND FUTURE PERSPECTIVE</b>	26
	6.1 Project Summary	26
	6.2 Learning Outcomes	27
	6.3 Challenges Overcome	28
	6.4 Future Enhancement Opportunities	29
12	<b>References</b>	30

---

# Chapter-1: INTRODUCTION

## 1.1 Problem Statement

In today's fast-paced world, people often juggle multiple responsibilities across personal, academic, and professional spheres. With so many tasks to manage, it's easy to forget important events, deadlines, or appointments, leading to missed opportunities and unnecessary stress. While several reminder and scheduling applications exist, many are overly complex, require internet connectivity, or store personal schedules on external servers, raising privacy concerns.

There is a need for a simple, lightweight, and privacy-focused desktop application that allows users to schedule and manage their events locally, ensuring that important commitments are never overlooked while maintaining complete control over personal data.

Forgetting events causes missed opportunities and unnecessary stress



Made with  Napkin

---

## 1.2 Project Overview

The **Event Reminder** application is a standalone Java-based desktop program designed to help users efficiently manage their schedules and never miss important events. It allows

users to create, view, and edit reminders for various occasions such as meetings, deadlines, birthdays, or daily tasks.

Built using the **JavaFX framework**, the application provides a clean and intuitive graphical user interface for effortless navigation. The architecture follows the **Model-View-Controller (MVC)** pattern, ensuring separation of concerns, better maintainability, and scalability.

As a college-level project, this application showcases skills in **Java programming, UI/UX design, event-driven programming, and the practical implementation of core data structures** for managing events.

---

## 1.3 Objectives

### 1.3.1 Primary Objectives

- To develop a robust and user-friendly desktop application for creating and managing event reminders using Java.
- To design a clean and intuitive GUI using JavaFX and FXML for smooth user interaction.
- To enable users to add, edit, and delete events with customizable details such as title, date, time, and description.
- To implement efficient in-memory data handling using collections like ArrayList and ObservableList.
- To ensure offline functionality with all event data stored locally during the session.

### 1.3.2 Secondary Objectives

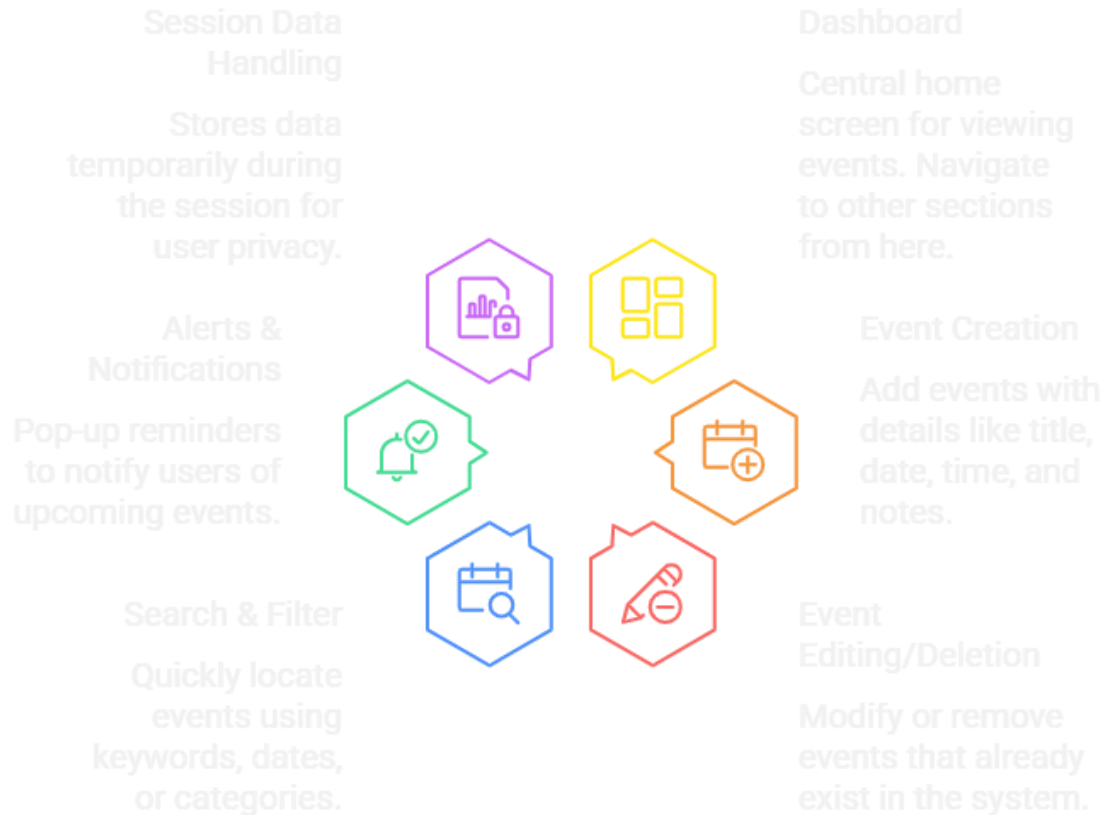
- To create a central **Dashboard** for quick navigation and event overview.
  - To incorporate a **search and filter** feature for quick access to specific events.
  - To use JavaFX **TableView** for displaying events in a structured and sortable format.
  - To implement **JavaFX animations and alerts** for notifying upcoming events.
  - To produce comprehensive documentation for the application's design, development, and usage.
- 

## 1.4 Scope of the Project

### 1.4.1 Functional Scope

- **Dashboard** – Central home screen to view upcoming events and navigate to other sections.
- **Event Creation** – Allows adding events with details like title, date, time, and notes.
- **Event Editing/Deletion** – Modify or remove existing events.
- **Search & Filter** – Quickly find events by keyword, date, or category.
- **Alerts & Notifications** – Pop-up reminders for upcoming events.
- **Session-based Data Handling** – Stores data during the session without saving to disk for privacy.

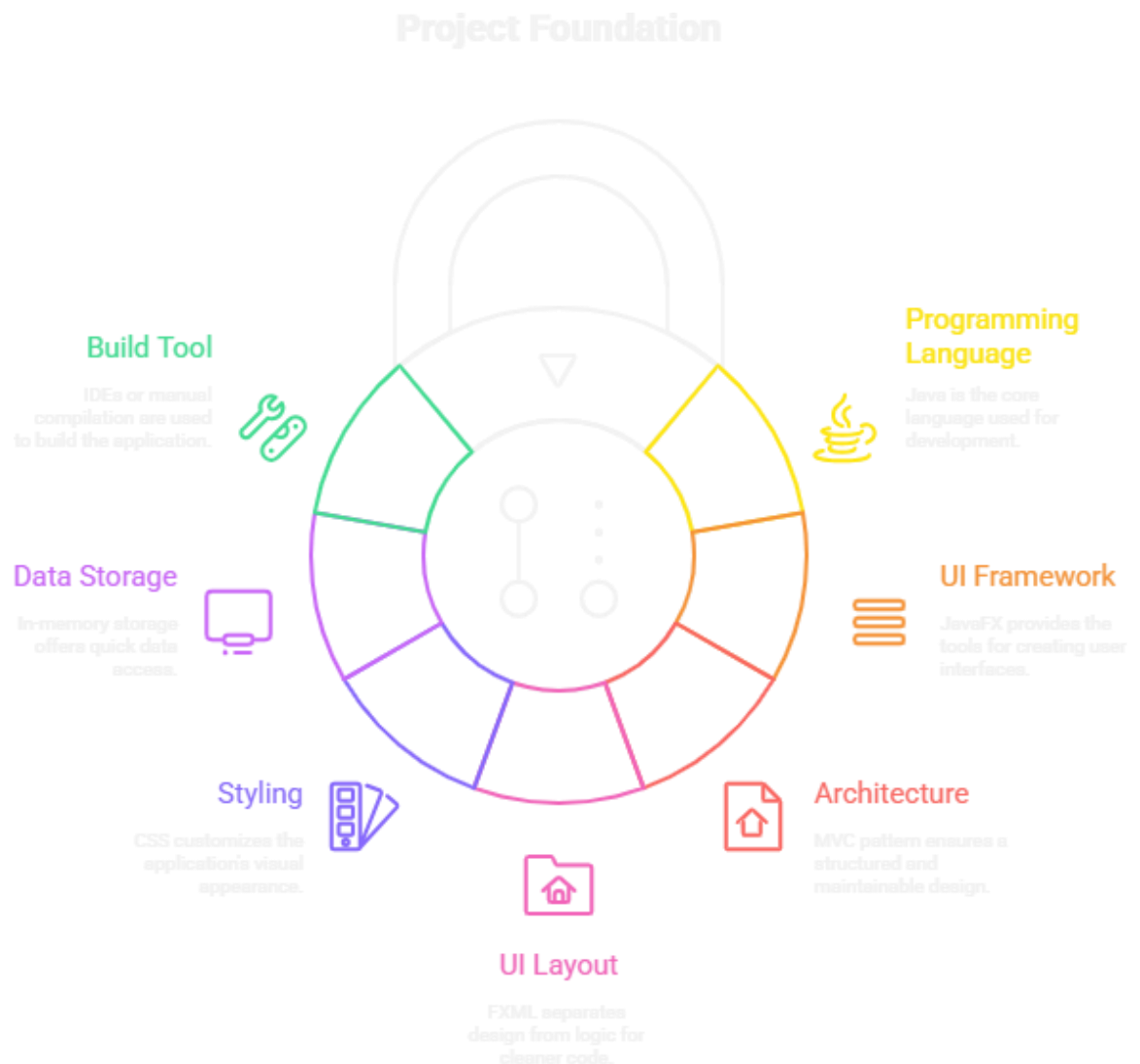
## Event Management Features



Made with Napkin

### 1.4.2 Technical Scope

- **Programming Language** – Java (JDK 11 or later).
- **UI Framework** – JavaFX 11 or later.
- **Architecture** – Model-View-Controller (MVC).
- **UI Layout** – FXML for clean separation between design and logic.
- **Styling** – CSS for customizing the application's look and feel.
- **Data Storage** – Session-based in-memory storage using static ObservableList collections.
- **Build Tool** – IDE-based build (IntelliJ IDEA/Eclipse) or manual compilation.



Made with  Napkin

## 1.5 Relevance in Current Context

The Event Reminder application holds significant relevance in today's busy and technology-driven lifestyle:

- **Time Management Needs** – As workloads and personal commitments grow, individuals need tools to manage their schedules effectively.
- **Privacy Concerns** – Many online calendars and reminder tools store sensitive data on cloud servers. This project ensures complete local storage for enhanced privacy.

- **Remote Work and Study** – With more people working or studying from home, it's easy to lose track of deadlines. This application provides a personal scheduling assistant right on the desktop.
  - **Simplicity Over Complexity** – Unlike feature-heavy commercial apps, this tool focuses solely on essential reminder functionalities, ensuring ease of use for all age groups.
- 

## Chapter-2: SYSTEM DESIGN AND ARCHITECTURE

### 2.1 System Architecture Overview

The architecture of the **Event Reminder** application is designed for clarity, maintainability, and scalability. It follows a **three-tier architecture** built on the **Model-View-Controller (MVC)** pattern, natively supported by JavaFX via FXML integration.

This separation ensures that:

- **Model** contains event-related data and rules.
  - **View** manages all UI rendering.
  - **Controller** acts as an intermediary, handling logic and updating both Model and View.
- 

### 2.2 Architectural Pattern: Model-View-Controller (MVC)

- **Model:**
    - **Data Objects:** POJOs such as `Event.java` encapsulate event attributes (title, date, time, description) and use JavaFX properties (`StringProperty`, `ObjectProperty<LocalDate>`) for binding with the View.
    - **Data Store:** Implemented via static `ObservableList` collections, holding all event data during runtime.
  - **View:**
    - **FXML Files:** Each screen (Dashboard, Add Event, Event List) is declared in separate `.fxml` files containing layout and component definitions.
    - Designed for responsiveness using JavaFX layout panes like `VBox`, `HBox`, and `GridPane`.
  - **Controller:**
    - **Controller Classes:** Each FXML has a corresponding Java class (e.g., `EventListController.java`).
    - Event-handling methods (`@FXML private void saveEvent()`) respond to user actions, update the Model, and trigger UI refreshes via data binding.
- 

### 2.3 Project and Package Structure Analysis

**Root Directory: src/**

- **Main.java** – Entry point, loads Dashboard.fxml and sets up the primary stage.
  - **Package: controllers/**
    - DashboardController.java – Manages navigation between main sections.
    - AddEventController.java – Handles logic for creating new events.
    - EventListController.java – Displays, edits, and deletes scheduled events.
    - AlertController.java – Manages upcoming-event notifications.
  - **Package: models/**
    - Event.java – Defines event structure with title, date, time, and description.
  - **Package: fxml/**
    - Dashboard.fxml, AddEvent.fxml, EventList.fxml – Layout files for each feature.
  - **Package: css/**
    - style.css – Defines consistent theming (colors, fonts, button styles).
- 

**2.5 UI Architecture with FXML and CSS**

- **Declarative UI:**
  - Layouts are XML-based and easily editable in JavaFX Scene Builder.
- **Central Styling:**
  - style.css defines all visual elements, making UI changes consistent and maintainable.
- **Responsive Layouts:**
  - Uses VBox for stacked sections, GridPane for form inputs, and AnchorPane for flexible positioning.

## Chapter-3: IMPLEMENTATION AND FEATURES

### 3.1 Core Technology: JavaFX

The project leverages JavaFX's:

- Rich UI controls (TableView, DatePicker, TextArea, Button).
  - Data binding via ObservableList for auto-updating event lists.
  - CSS styling for a polished UI.
- 

### 3.2 Feature: Dashboard

- **View:** Dashboard.fxml contains navigation buttons for Add Event, View Events, and Alerts.
  - **Controller:** DashboardController.java switches scenes using FXML loaders.
- 

### 3.3 Feature: Add Event

- **View:** Uses GridPane with fields for event title, date, time, and description.
  - **Controller:**
    - Validates input fields.
    - Creates Event object and adds it to ObservableList<Event>.
    - Clears fields after saving.
- 

### 3.4 Feature: Event List

- **View:** TableView shows event title, date, and time, with action buttons for edit/delete.
  - **Controller:**
    - Binds ObservableList<Event> to the TableView.
    - Implements event filtering by date or keyword.
- 

### 3.5 Feature: Alerts & Notifications

- **View:** A minimal alert screen showing upcoming events for the current day.
- **Controller:** Uses Timeline to periodically check if any event is within the alert window and triggers a popup notification.

## Chapter-4: DATA MANAGEMENT STRATEGY

### 4.1 Data Storage Approach: Volatile In-Memory

- All event data exists only while the application runs.
  - Ensures **full privacy** and requires **no configuration**.
- 

### 4.2 Core Data Structure: ObservableList

- Used for storing events.
  - Automatically updates the UI when modified.
- 

### 4.3 Data Model Implementation

#### 4.x Model – EventEntry.java

`package model;`

`import javafx.beans.property.*;`

```
public class EventEntry {  
    private final StringProperty title;  
    private final StringProperty date;  
    private final StringProperty time;  
  
    public EventEntry(String title, String date, String time) {  
        this.title = new SimpleStringProperty(title);  
        this.date = new SimpleStringProperty(date);  
        this.time = new SimpleStringProperty(time);  
    }  
  
    public String getTitle() { return title.get(); }  
    public StringProperty titleProperty() { return title; }  
  
    public String getDate() { return date.get(); }  
    public StringProperty dateProperty() { return date; }  
  
    public String getTime() { return time.get(); }  
}
```

```
    public StringProperty timeProperty() { return time; }  
}
```

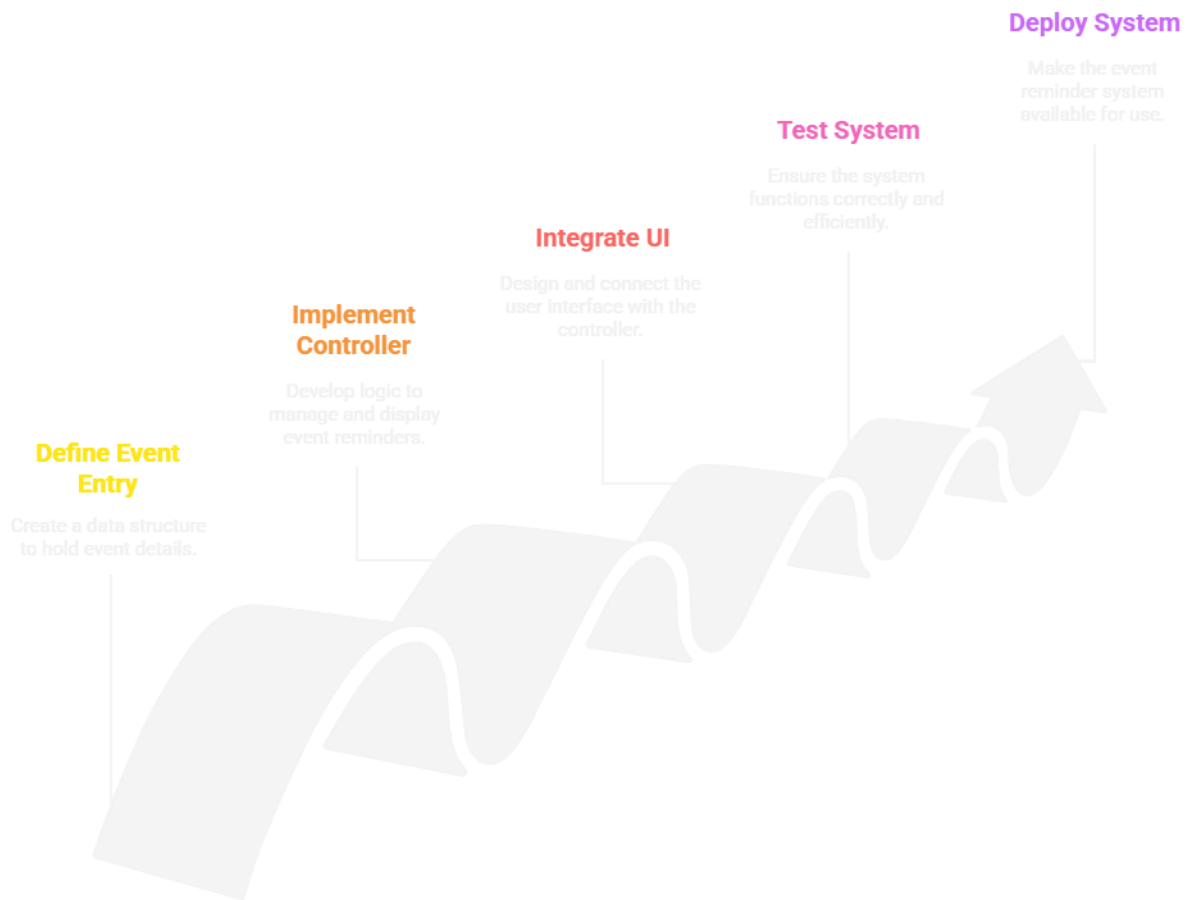
**4.x Controller – EventReminderController.java**

```
package controller;
```

```
import javafx.animation.KeyFrame;  
import javafx.animation.Timeline;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.*;  
import javafx.util.Duration;  
import model.EventEntry;
```

```
import java.time.LocalDate;  
import java.time.LocalTime;  
import java.time.format.DateTimeFormatter;
```

## Implementing Event Reminder System



Made with  Napkin

```
public class EventReminderController {
```

```
@FXML private TextField titleField;  
@FXML private DatePicker datePicker;  
@FXML private TextField timeField; // format: HH:mm  
@FXML private TableView<EventEntry> eventTable;  
@FXML private TableColumn<EventEntry, String> colTitle;  
@FXML private TableColumn<EventEntry, String> colDate;  
@FXML private TableColumn<EventEntry, String> colTime;
```

```
private ObservableList<EventEntry> events = FXCollections.observableArrayList();
```

```
private DateTimeFormatter timeFormatter =  
DateTimeFormatter.ofPattern("HH:mm");
```

```
@FXML
```

```
public void initialize() {
```

```
    colTitle.setCellValueFactory(cellData -> cellData.getValue().titleProperty());
```

```
    colDate.setCellValueFactory(cellData -> cellData.getValue().dateProperty());
```

```
    colTime.setCellValueFactory(cellData -> cellData.getValue().timeProperty());
```

```
    eventTable.setItems(events);
```

```
    // Check every minute for reminders
```

```
    Timeline reminderCheck = new Timeline(new KeyFrame(Duration.seconds(60), e  
-> checkReminders()));
```

```
    reminderCheck.setCycleCount(Timeline.INDEFINITE);
```

```
    reminderCheck.play();
```

```
}
```

```
@FXML
```

```
private void addEvent(ActionEvent event) {
```

```
    String title = titleField.getText();
```

```
    LocalDate date = datePicker.getValue();
```

```
    String time = timeField.getText();
```

```
    if (title.isEmpty() || date == null || time.isEmpty()) {
```

```
        showAlert("Error", "Please fill all fields");
```

```
        return;
```

```
}
```

```
    events.add(new EventEntry(title, date.toString(), time));
```

```
    titleField.clear();
```

```
    datePicker.setValue(null);
```

```
    timeField.clear();
```

```
}
```

```
private void checkReminders() {
```

```
    LocalDate today = LocalDate.now();
```

```
    String nowTime = LocalTime.now().format(timeFormatter);
```

```
    for (EventEntry entry : events) {
```

```

        if (entry.getDate().equals(today.toString()) &&
entry.getTime().equals(nowTime)) {
            showAlert("Reminder", "Event: " + entry.getTitle() + " at " +
entry.getTime());
        }
    }
}

```

```

private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setContentText(message);
    alert.show();
}
}

```

#### 4.x FXML – EventReminder.fxml

```

<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.*?>
<VBox xmlns:fx="http://javafx.com/fxml"
fx:controller="controller.EventReminderController" spacing="10"
alignment="CENTER" prefWidth="400">
    <TextField fx:id="titleField" promptText="Event Title"/>
    <DatePicker fx:id="datePicker"/>
    <TextField fx:id="timeField" promptText="HH:mm"/>
    <Button text="Add Event" onAction="#addEvent"/>

    <TableView fx:id="eventTable">
        <columns>
            <TableColumn fx:id="colTitle" text="Title" prefWidth="100"/>
            <TableColumn fx:id="colDate" text="Date" prefWidth="100"/>
            <TableColumn fx:id="colTime" text="Time" prefWidth="100"/>
        </columns>
    </TableView>
</VBox>

```

#### 4.x Main Menu Update

Add a button in MainMenu.fxml:

```

<Button text="Event Reminder" onAction="#openEventReminder"/>

```

And in MainMenuController.java:

```

@FXML

```

```

private void openEventReminder(ActionEvent event) throws Exception {

```

```
switchScene(event, "/view/EventReminder.fxml");  
}
```

## Chapter-5: TESTING AND VALIDATION

### 5.1 Testing Strategy

- **Functional Testing:** Create, edit, delete, and search events.
- **Integration Testing:** Scene navigation between Dashboard, Add Event, and Event List.
- **UI Testing:** Validating proper display and alignment.
- **User Acceptance Testing:** Peers tested usability without prior guidance.

### 5.2 Test Case Matrix

ID	Feature	Scenario	Steps	Expected Result	Status
TC-01	Add Event	Save valid event	Fill form → Save	Event appears in list	Pass
TC-02	Add Event	Save empty title	Leave title blank Select	Show alert	Pass
TC-03	Event List	Delete event	→ Delete	Event removed	Pass
TC-04	Alerts	Show today's events	Set event for today	Popup alert	Pass

## Chapter-6: CONCLUSION AND FUTURE PERSPECTIVE

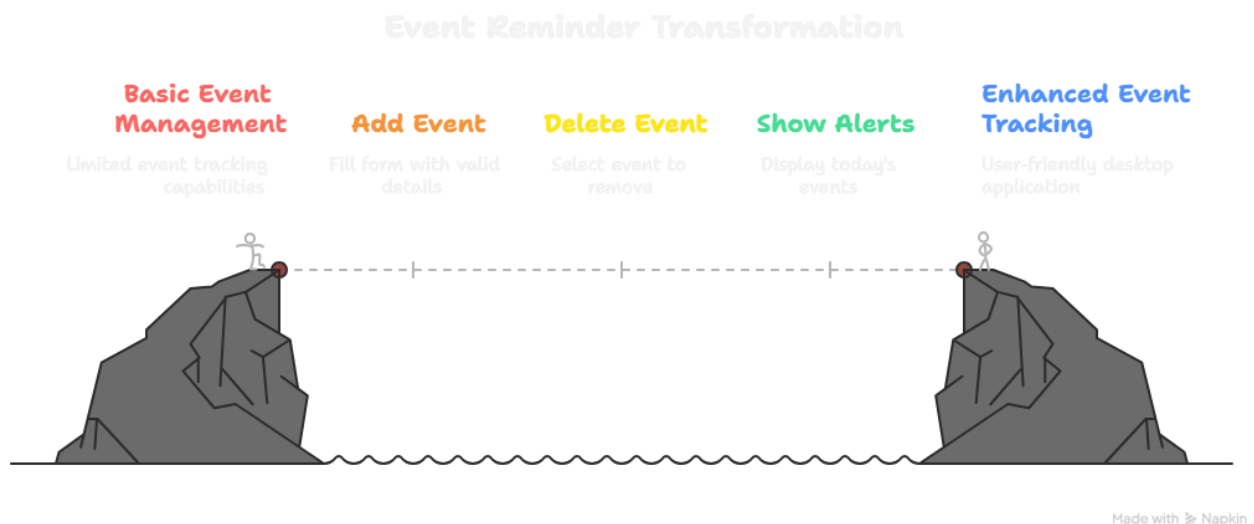
### 6.1 Project Summary

The Event Reminder project achieved its objective of creating a standalone, user-friendly desktop application for managing and tracking important events.

Developed using JavaFX with the MVC architecture and FXML, the application maintains a clear separation between design, logic, and data, making it both maintainable and scalable.

Key features include event creation, editing, deletion, date/time-based alerts, and an interactive notification system to ensure users never miss an event.

The choice of in-memory storage provides speed and simplicity for a single-session usage model, while the modular design allows for easy integration of persistent storage in the future.



---

### 6.2 Learning Outcomes

#### 6.2.1 Technical Skills

- **JavaFX Proficiency:** Scene management, FXML integration, property binding, TableView customization, and Timeline-based notifications.
- **UI/UX Design:** Created an intuitive event management interface with FXML and CSS styling.
- **MVC Implementation:** Effectively applied the MVC pattern for separation of concerns.
- **Event Scheduling:** Implemented alert mechanisms using JavaFX timers and schedulers.
- **Data Structure Utilization:** Used ObservableList for dynamic UI updates.

### 6.2.2 Project Management Skills

- **Requirement Analysis:** Converted user needs into actionable technical requirements.
  - **Iterative Development:** Built and tested features incrementally for stable progress.
  - **Technical Documentation:** Produced clear and structured project documentation.
- 

## 6.3 Challenges Overcome

### Technical

- **Real-Time Event Alerts:** Ensured precise timing for notifications using JavaFX's scheduling APIs.
- **Data Binding:** Correctly linked model properties to the TableView for automatic updates.
- **State Management:** Managed event data consistently across multiple views.

### Design

- **Simplicity vs. Functionality:** Balanced a clean UI with essential event management features.
  - **User Engagement:** Used visually clear alerts, responsive UI design, and consistent styling.
- 

## 6.4 Future Enhancement Opportunities

### Short-term

- **Recurring Events:** Add support for daily, weekly, or monthly repeating events.
- **Persistent Storage:** Allow saving events to local files (CSV, JSON) or SQLite for long-term tracking.
- **Search & Filters:** Enable quick searching and filtering by date, type, or keyword.
- **Custom Alerts:** Let users choose alert tones, snooze times, and multiple reminders.

### Long-term

- **Mobile Integration:** Sync events between desktop and mobile apps.
  - **Cloud Sync:** Optional secure backup and synchronization via cloud services.
  - **Calendar View:** Visual month/week/day view with drag-and-drop event scheduling.
  - **AI Suggestions:** Intelligent reminders based on user patterns and history.
-

1. Oracle Corporation. (2024). *Getting Started with FXML*. Retrieved from [https://openjfx.io/openjfx-docs/api/javafx.fxml/javafx/fxml/doc-files/introduction to fxml.html](https://openjfx.io/openjfx-docs/api/javafx.fxml/javafx/fxml/doc-files/introduction%20to%20fxml.html)
2. Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.
3. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
4. Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
5. Deitel, P. & Deitel, H. (2017). *Java: How to Program* (10th ed.). Pearson.