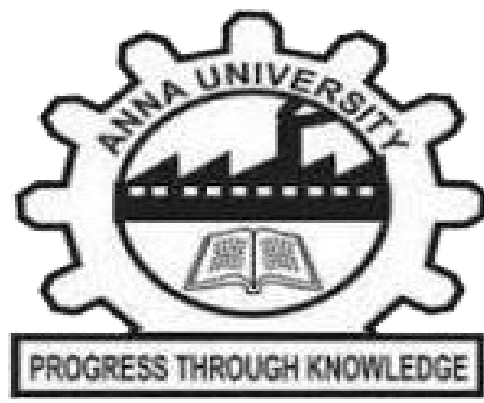


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING BHARATHIDASAN
INSTITUTE OF TECHNOLOGY CAMPUS ANNA UNIVERSITY
TIRUCHIRAPPALLI – 620 024



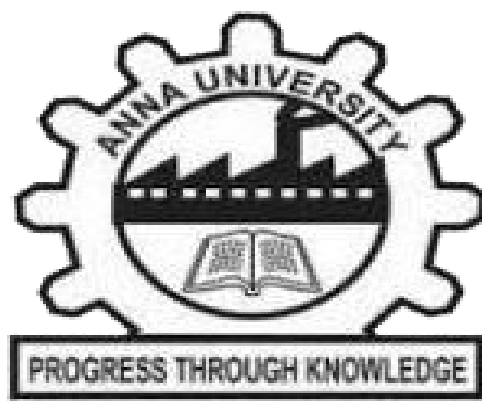
CCS332 - APP DEVELOPMENT LABORATORY

NAME :

REGISTER NUMBER :

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING BHARATHIDASAN
INSTITUTE OF TECHNOLOGY CAMPUS ANNA UNIVERSITY
TIRUCHIRAPPALLI – 620 024

Bonafide Certificate



This is to certify that Mr. /Ms. _____ bearing the
RegisterNo: _____ have satisfactorily completed the course of
practical examination of **CCS332-APP DEVELOPMENT LABORATORY** for **FIFTH**
semester ,**B.E** during the academic year
2023-2024.

Faculty in Charge

Head of the Department

Submitted for the practical examination held on _____ .

Internal Examiner

External Examiner

Ex No	Date	Experiment Title	Page no	Signature
1.		Using react native, build a cross platform application for a BMI calculator		
2.		Build a cross platform application for a simple expense manager which allows entering expenses and income on each day and displays category wise weekly income and expense.		
3.		Build a cross platform application for a simple expense manager which allows entering expenses and income on each day and displays category wise weekly income and expense.		
4.		Develop a cross platform application to convert units from imperial system to metric system (km to miles, kg to pounds etc.,)		
5.		Design an android application using Cordova for a user login screen with username, password, reset button and a submit button. Also, include header image and a label. Use layout managers.		
6.		Design and develop an android application using Apache Cordova to find and display the current location of the user.		
7.		Write programs using Java to create Android application having Databases for displaying books available, books lend, book reservation.		

BMI CALCULATOR

Aim:

To Build a cross platform application for a BMI calculator using react native.

Algorithm:

1. Initialize React Native Project: Set up a new React Native project using the CLI.
2. Design User Interface: Create UI components using React Native's built-in elements for input fields (height, weight), a calculation button, and a section to display the BMI result.
3. Manage User Input: Use React's state management (useState) to capture user input for height and weight.
4. Calculate BMI: Write a function to calculate BMI based on entered height and weight.
5. Display Results: Show the calculated BMI value and its category (e.g., Underweight, Normal, Overweight) based on standard BMI ranges.
6. Styling and Testing: Apply styles for UI enhancement and responsiveness. Test the app on different devices to ensure functionality.
7. Deployment: Prepare the app for iOS and Android platforms, following platform-specific guidelines.
8. Publishing: Publish the app on app stores or distribute it through appropriate channels

Program

App.js

```
import { StatusBar } from 'expo-status-bar';
import { SafeAreaView, StyleSheet, Text, View } from 'react-native';
import BmiForm from './components';

export default function App() {
  return (
    <SafeAreaView style={styles.container}>
      <BmiForm />
      <StatusBar style="auto" />
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
```

```

    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});

```

Components/index.js

```

import { useState } from "react";
import { Button, TextInput, View, StyleSheet, Text } from "react-native";
const BmiForm = () => {
  //Defined useStates to keep track of values
  const [height, setHeight] = useState("");
  const [weight, setWeight] = useState("");
  const [bmi, setBmi] = useState("");
  // the func calculates bmi and updates the bmi value
  function calculate_bmi() {
    let h = Number(height) / 100;
    let w = Number(weight);
    let r = (w / (h * h)).toFixed(1);
    if (r > 0) {
      setBmi("BMI: " + r);
    } else {
      setBmi('Invalid data');
    }
  }
  return (
    <View>
      <Text style={styles.label}>
        Height
      </Text>
      <TextInput
        style={styles.input}
        onChangeText={setHeight}
        value={height}
        keyboardType="number-pad"
        placeholder="in cms"
      />
      <Text style={styles.label}>Weight</Text>
      <TextInput
        style={styles.input}

```

```

        onChangeText={setWeight}
        value={weight}
        keyboardType="number-pad"
        placeholder="in kgs"
      />
      <Button
        style={styles.button}
        title="Compute BMI"
        onPress={calculate_bmi}
      />
      <Text style={styles.bmiLabel}>
        {bmi}
      </Text>
    </View >
  );
};
const styles = StyleSheet.create({
  input: {
    flex: 0,
    borderColor: 'gray',
    borderWidth: 1,
    borderRadius: 20,
    margin: 8,
    padding: 8,
    width: 300,
  },
  button: {
    borderColor: 'blue',
  },
  label: {
    fontWeight: 'bold',
    fontSize: 15,
  },
  bmiLabel: {
    fontSize: 40,
    margin: 10,
    justifyContent: 'center'
  }
});
export default BmiForm;

```

Output:

11:53



Height

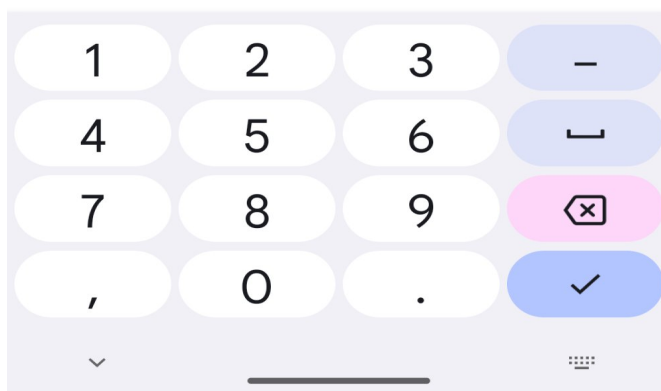
185

Weight

70

COMPUTE BMI

BMI: 20.5



Result:

Thus, the creation of a cross-platform application for a BMI calculator is done.

EXPENSE TRACKER

Aim:

To build a cross-platform application for a simple expense manager which allows entering expenses and income on each day and displays category-wise weekly income and expense.

Procedure:

1. Create an HTML file with a structure containing elements for input forms, tables, and display areas.
2. Write JavaScript logic to handle expense-related functionalities, utilizing local storage for data storage.
3. Initialize variables for form elements, expense list, and total amount. Retrieve expenses from localStorage or initialize an empty array.
4. Create a function (renderExpenses) to display expenses in the HTML table.
5. Calculate the total amount and update the corresponding display.
6. Create functions (addExpense and deleteExpense) to handle expense addition and deletion.
7. Validate input values, update the expenses array, and call renderExpenses for display updates.
8. Attach event listeners to the expense form submission and the expense list for delete actions.
9. Call the renderExpenses function initially to display existing expenses.
10. Ensure code adaptability by avoiding hardcoding specific IDs or values not likely to change.
11. Reference the JavaScript file in the HTML file using the <script> tag.

Program:

Index.html:

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Expense Tracker</title>
  <link rel="stylesheet" type="text/css" href="css/index.css" />
</head>
<body>
  <div class="container">
    <h1>Expense Tracker</h1>
    <div style="flex: auto; justify-content: space-between;">
```



```

    <form id="expense-form">
      <input type="text" id="expense-name" placeholder="Expense Name"
        required
    />

      <input type="number" id="expense-amount" placeholder="Amount"
        required
    />

      <button
        type="submit">
        Add Expense
      </button>
    </form>
  </div>
  <div class="expense-table">
    <table>
      <thead>
        <tr>
          <th>Expense Name</th>
          <th>Amount</th>
          <th>Action</th>
        </tr>
      </thead>
      <tbody id="expense-list"></tbody>
    </table>
    <div class="total-amount">
      <strong>Total:</strong>
      <span id="total-amount">0</span>
    </div>
  </div>
</div>
<script src="js/index.js"></script>
</body>
</html>

```

Index.js:

```

const expenseForm =
document.getElementById("expense-
form");

```

```

const expenseList =
document.getElementById("expense-
list");
const totalAmountElement =
document.getElementById("total-
amount");
let expenses =
JSON.parse(localStorage.getItem("expenses")) ||
[];
function renderExpenses() {
  expenseList.innerHTML = "";
  let totalAmount = 0;
  for (let i = 0; i < expenses.length; i++)
    { const expense = expenses[i];
    const expenseRow = document.createElement("tr");
    expenseRow.innerHTML =
      `
        <td>${expense.name}</td>
        <td>₹${expense.amount}</td>
        <td class="delete-btn" data-id="${i}">Delete</td>
      `;
    expenseList.appendChild(expenseRow);
    totalAmount += expense.amount;
  }
  totalAmountElement.textContent =
    totalAmount.toFixed(2);
  // Save expenses to
    localStorage
  localStorage.setItem("expens
    es",
    JSON.stringify(expenses));
}
function addExpense(event)
{ event.preventDefault();
  const expenseNameInput =
    document.getElementById("expense-
    name");

```

```

    const expenseAmountInput =
    document.getElementById("expense-
    amount");
    const expenseName =
    expenseNameInput.value;
    const expenseAmount =
    parseFloat(expenseAmountInput.val
    ue);
    expenseNameInput.value =
    "";
    expenseAmountInput.value
    = "";
    if (expenseName === "" ||
    isNaN(expenseAmount)) { alert("Please enter
    valid expense details."); return;
    }
    const expense =
    { name:
    expenseName,
    amount: expenseAmount,
    };
    expenses.push(expense);
    // Render expenses
    renderExpenses();
  }
  function deleteExpense(event) {
    if (event.target.classList.contains("delete-btn")) {
      const expenseIndex =
        parseInt(event.target.getAttribute("data-
        id"));
      expenses.splice(expenseIndex, 1);
      renderExpenses();
    }
  }
  expenseForm.addEventListener("submit",
  addExpense); expenseList.addEventListener("click",
  deleteExpense);
  renderExpenses();

```

Output:

Expense Tracker

Add Expense

Expense Name	Amount	Action
food	₹200	Delete
iv	₹3000	Delete
		Total: ₹3200.00

*

Result:

Thus, the creation of a cross-platform application for a simple expense manager is done.

UNIT CONVERTER

Aim :

To Develop a cross platform application to convert units from imperial system to metric system (km to miles, kg to pounds etc.,)

Algorithm:

1. Initialize Project : Set up a cross-platform application using a framework like React Native.
2. Design UI : Create components for entering daily expenses and income (amount,category, date).
3. Manage Data : Implement state management to store and organize user-entered expense and income data.
4. Calculate Weekly Summary : Write functions to aggregate and categorize daily expenses and income into a weekly view.
5. Display Category-wise Summary : Show category-wise totals for weekly income and expenses in an organized format.
6. Apply Styling : Apply styles for an intuitive and visually appealing interface.
7. Testing and Validation:Thoroughly test the app to ensure accurate calculations and functionality.
8. Deployment : Prepare the app for deployment on iOS and Android platforms.
9. Publishing : Publish the app on app stores or distribute it through suitable channels.

Program:

App.js

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';
import FormBox from './components'
export default function App() {
  return (
    <View style={styles.container}>
      <FormBox />
    </View>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    justifyContent: 'center',
  },
});
```

components/index.js

```
import React, { useEffect, useState } from "react";
import { Button, TextInput, View, StyleSheet, Text } from "react-native";
import DropdownComponent from "../dropdown-input";
import cd from '../conversion-data.json'
```

```
const FormBox = () => {
  // console.log(cd["length"]);
  const [unit, setUnit] = useState(cd["unit_names"][0]["value"]);
  const [from_unit, setFrom_unit] = useState(null);
  const [to_unit, setTo_unit] = useState(null);
  const [inputData, setInputData] = useState(null);
  const [outputData, setOutputData] = useState('0.00');
  useEffect(() => {
    setFrom_unit(cd[unit][0]["value"]);
    setTo_unit(cd[unit][0]["value"]);
    setOutputData('0');
    setInputData(null);
  }, [unit])
```

```
const calculate_conversion = (value) => {
  setInputData(value);
  let to_litre = Number(value) / from_unit;
  let to_final = to_litre * to_unit;
  setOutputData(to_final);
}
return (
  <View style={styles.container}>
    <DropdownComponent
      dta={cd["unit_names"]}
      value={unit}
```

```

        setValue={setUnit}
      />
      <DropdownComponent
        dta={cd[unit]}
        value={from_unit}
        setValue={setFrom_unit}
      />
      <TextInput
        style={styles.numberInput}
        value={inputData}
        placeholder="0"
        onChangeText={calculate_conversion}
        inputMode="numeric"
      />
      <DropdownComponent
        dta={cd[unit]}
        value={to_unit}
        setValue={setTo_unit}
      />
      <Text style={styles.outputData}>
        {outputData}
      </Text>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    backgroundColor: 'white',
    padding: 16,
  },
  numberInput: {
    height: 50,
    borderColor: 'gray',
    borderWidth: 0.5,
    borderRadius: 8,
    paddingHorizontal: 8,
    margin: 10
  },
  outputData: {
    fontSize: 30,

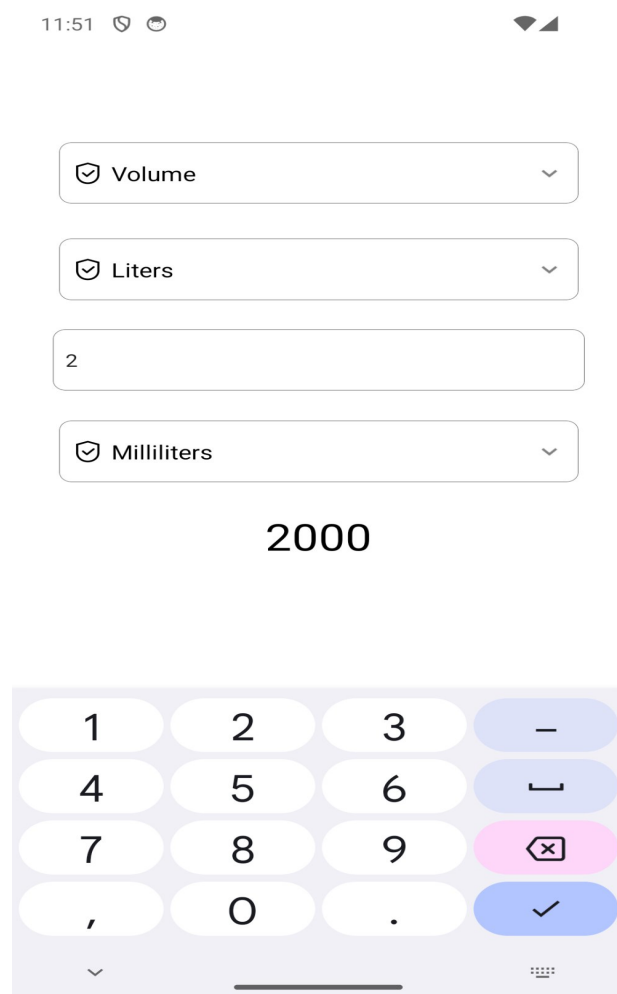
```

```

        margin: 10,
        textAlign: 'center'
    }
  })
export default FormBox;

```

Output:



11:51

Volume

Liters

2

Milliliters

2000

1 2 3 -

4 5 6 $\frac{\square}{\square}$

7 8 9 \times

, 0 . \checkmark

Result:

Thus, the creation of a cross-platform application for a unit converter is done.

TODO APP

Aim :

To Design and develop a cross platform application for day to day task (to-do) management.

Algorithm:

1. Initialize Project : Create a new cross-platform application using a framework like React Native or Flutter.
2. Design User Interface : Develop UI components using platform-specific or custom UI elements for tasks, categories, and task details.
3. Task Management Structure: Define a data structure to represent tasks with attributes such as title, description, deadline, priority, category, and completion status.
4. User Input and Interaction: Capture user input for creating and editing tasks through input fields.
5. Task Organization and Sorting: Allow users to categorize tasks into different lists or categories
6. Task Notifications and Reminders: Implement a feature for setting task reminders or notifications for upcoming deadlines or important tasks.
7. User Authentication and Data Persistence: Implement user authentication if needed for multiple users or data synchronization across devices.
8. Styling and UI Enhancements: Apply styling for a visually appealing and user-friendly interface.
9. Deployment and Publishing: Prepare the application for deployment on iOS and Android platforms.

Program:

```
import { KeyboardAvoidingView, StyleSheet, Text, TextInput, TouchableOpacity,
Keyboard, View, ScrollView } from 'react-native';
import Task from './components/task';
import { useState } from 'react';
export default function App() {
  const [task, setTask] = useState();
  const [taskItems, setTaskItems] = useState([]);
  const handleAddTask = () => {
    Keyboard.dismiss();
    setTaskItems([...taskItems, task])
```

```

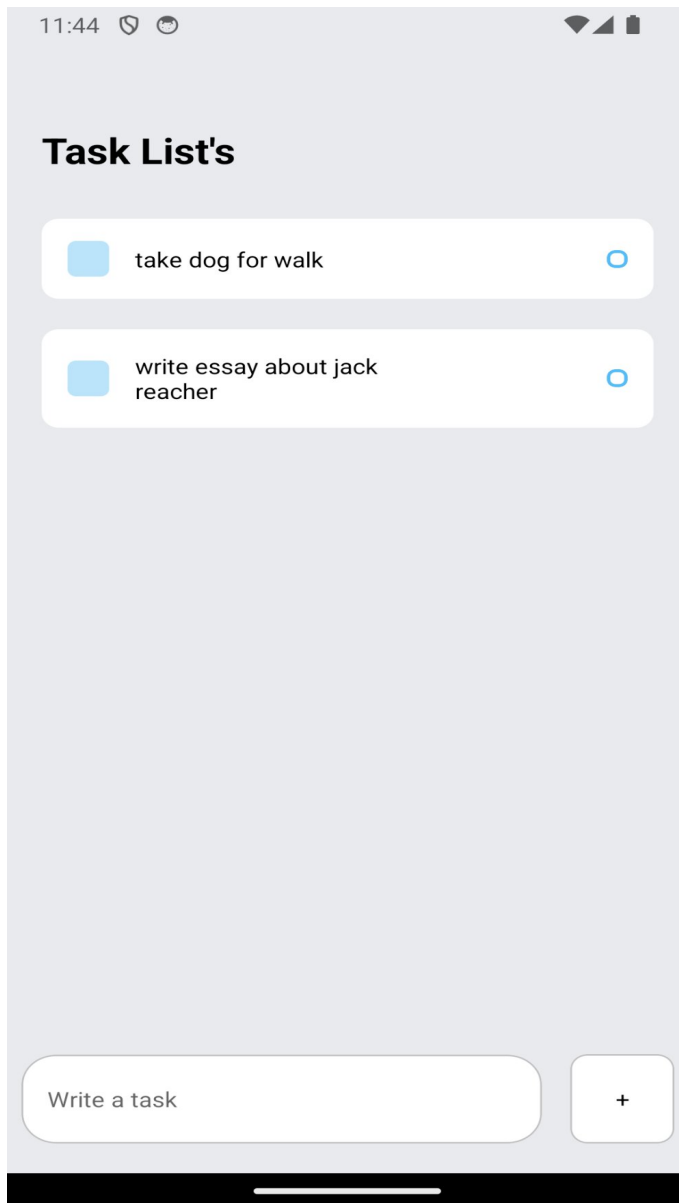
    setTask(null);
  }
  const completeTask = (index) => {
    let itemsCopy = [...taskItems];
    itemsCopy.splice(index, 1);
    setTaskItems(itemsCopy)
  }
  return (
    <View style={styles.container}>
      <ScrollView contentContainerStyle={{
        flexGrow: 1,
      }}
        keyboardShouldPersistTaps='handled'>
        <View style={styles.wrapper}>
          <Text style={styles.heading}>
            Task List's
          </Text>
          <View style={styles.tasklist}>
            { /* tasks here... */ }
            { /* This is where the tasks will go! */ }
            {
              taskItems.map((item, index) => {
                return (
                  <TouchableOpacity key={index} onPress={() => completeTask(index)}>
                    <Task text={item} />
                  </TouchableOpacity>
                )
              })
            }
          </View>
        </View >
      </ScrollView>
      <KeyboardAvoidingView
        behavior={Platform.OS === "ios" ? "padding" : "height"}
        style={styles.writeTaskWrapper}>
        <TextInput style={styles.input} placeholder={'Write a task'} value={task}
onChangeText={text => setTask(text)} />
        <TouchableOpacity onPress={() => handleAddTask()}>
          <View style={styles.addWrapper}>
            <Text style={styles.addText}>+</Text>

```

```
        </View>
      </TouchableOpacity>
    </KeyboardAvoidingView>
  </View>
);
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#E8EAED'
  },
  wrapper: {
    paddingTop: 80,
    paddingHorizontal: 20,
  },
  heading: {
    fontSize: 24,
    fontWeight: 'bold'
  },
  tasklist: {
    marginTop: 30,
  },
  writeTaskWrapper: {
    position: 'absolute',
    bottom: 20,
    width: '100%',
    flexDirection: 'row',
    justifyContent: 'space-around',
    alignItems: 'center'
  },
  input: {
    paddingVertical: 15,
    paddingHorizontal: 15,
    backgroundColor: '#FFF',
    borderRadius: 20,
    borderColor: '#C0C0C0',
    borderWidth: 1,
    width: 300,
  },
});
```

Output:



Result:

Thus, the creation of a cross-platform application for a todo app is done.

LOGIN SCREEN USING APACHE CORDOVA

Aim :

To Design an android application using Cordova for a user login screen with username, password, reset button and a submit button. Also, include header image and a label. Use layout managers.

Algorithm:

1. Setup Cordova Project : Create a new Cordova project using the Cordova CLI.
2. Design User Interface : Use HTML, CSS, and JavaScript to design the login screen. Include HTML elements for username input, password input, reset button, submit button, header image, and label.
3. Reset Button Functionality : Write JavaScript logic to reset the input fields when the reset button is clicked.
4. Submit Button Functionality : Implement JavaScript validation for the entered username and password.
5. Include Header Image and Label : Add an HTML element for the header image, ensuring its proper placement and styling.
6. Responsive Design : Ensure the layout and components are responsive and compatible with various screen sizes and orientations using appropriate CSS and layout management techniques.
7. Testing and Validation : Test the login screen functionality on different devices and screen sizes to ensure proper behavior.
8. Integration with Cordova : Integrate the HTML, CSS, and JavaScript files into the Cordova project structure.
9. Build and Deployment : Build the Cordova application for the Android platform. Deploy the application to an Android device or distribute it through appropriate channels.

Program:

```
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="Content-Security-Policy" content="default-src 'self' data:
https://ssl.gstatic.com 'unsafe-eval'; style-src 'self' 'unsafe-inline'; media-src *; img-src
'self' data: content;;">
    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
```

```

    <meta name="viewport" content="initial-scale=1, width=device-width, viewport-
fit=cover">
    <meta name="color-scheme" content="light dark">
    <link rel="stylesheet" href="css/index.css">
    <title>Hello World</title>
    <style>
body {
  padding-top: 20px;
}

.header {
  text-align: center;
}

.login-form {
  max-width: 400px;
  margin: 0 auto;
}
</style>
</head>
<body>
  <div class="app">
    <div class="container">
      <div class="header">
        <h2>Login</h2>
      </div>

      <div class="login-form">
        <form id="loginForm">
          <div class="form-group">
            <label for="username">Username:</label>
            <input type="text" class="form-control" id="username" placeholder="Enter
your username" required>
          </div>
          <div class="form-group">
            <label for="password">Password:</label>
            <input type="password" class="form-control" id="password"
placeholder="Enter your password" required>
          </div>
          <button type="button" class="btn btn-link" id="resetButton">Reset</button>

```

```

        <button type="submit" class="btn btn-primary"
id="submitButton">Submit</button>
    </form>
</div>
</div>
</div>
<script>
document.addEventListener('deviceready', onDeviceReady, false);

function onDeviceReady() {
    // Add your device ready code here if needed.
    // For simplicity, we are not adding any device ready code in this example.
}

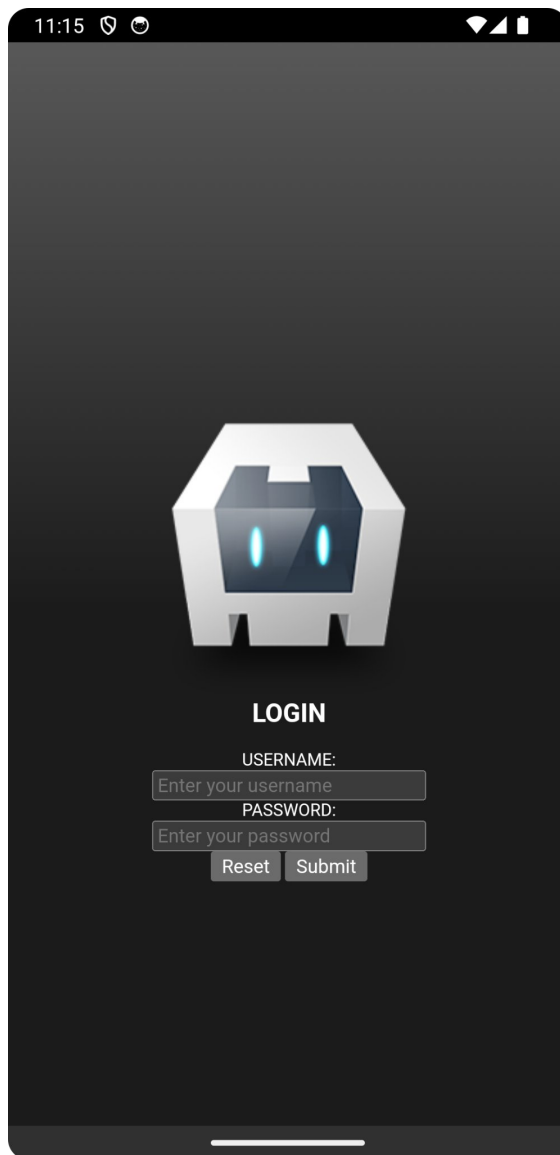
document.getElementById('resetButton').addEventListener('click', function () {
    document.getElementById('username').value = "";
    document.getElementById('password').value = "";
});

document.getElementById('loginForm').addEventListener('submit', function (e) {
    e.preventDefault();

    // Add your login logic here.
    var username = document.getElementById('username').value;
    var password = document.getElementById('password').value;

    // Example: Check if the username and password are not empty
    if (username.trim() === "" || password.trim() === "") {
        alert('Please enter both username and password.');
```

Output:



Result:

Thus, the creation of a cross-platform application for a android application using Cordova for a user login screen with username, password, reset button and a submit button is done.

LOCATION FINDER USING APACHE CORDOVA

Aim :

To Design and develop an android application using Apache Cordova to find and display the current location of the user.

Algorithm:

1. Initialize Cordova Project : Create a new Cordova project using the Cordova CLI.
2. Add Geolocation Plugin : Use the Cordova CLI or npm to add the Geolocation plugin to the project.Design.
3. User Interface : Create UI elements to display the user's current location information (latitude, longitude, address).
4. Request Location Permission : Implement Cordova's Geolocation API to request permission from the user to access their device location.
5. Fetch User's Current Location : Write JavaScript code to retrieve the user's current geolocation using the Geolocation plugin.
6. Display Location Information : Update the UI elements with the retrieved location information, displaying the latitude, longitude, and address (if available) to the user.
7. Build and Deployment : Build the Cordova application specifically for the Android platform.Deploy the application to an Android device or distribute it through suitable channels.

Program:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1, user-scalable=no">
  <title>Location App</title>
  <link rel="stylesheet" href="css/bootstrap.min.css" />
  <link rel="stylesheet" href="css/location.css" />
  <style>
body {
  padding-top: 20px;
}

.header {
```

```

    text-align: center;
}
.location-info {
    text-align: center;
    margin-top: 20px;
}
</style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h2>Current Location</h2>
        </div>

        <div class="location-info">
            <p id="latitude">Latitude: </p>
            <p id="longitude">Longitude: </p>
            <button class="btn btn-primary" id="getLocationButton">Get Location</button>
        </div>
    </div>
    <script>
document.addEventListener('deviceready', onDeviceReady, false);
document.getElementById('getLocationButton').addEventListener('click', function () {
    navigator.geolocation.getCurrentPosition(onSuccess, onError);
});
function onSuccess(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    document.getElementById('latitude').textContent = 'Latitude: ' + latitude;
    document.getElementById('longitude').textContent = 'Longitude: ' + longitude;
}

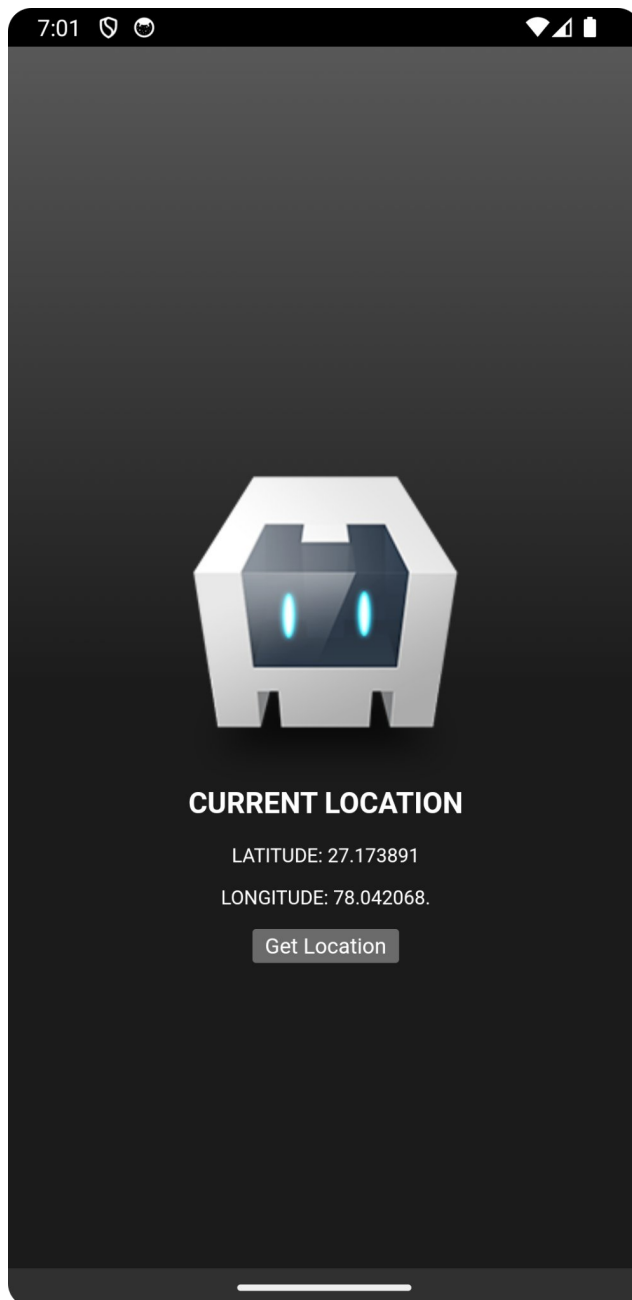
function onError(error) {
    alert('Error getting location: ' + error.message);
}

</script>
<script src="js/jquery.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/location.js"></script>

```

```
</body>  
</html>
```

OUTPUT:



Result:

Thus, the creation of a cross-platform application for an android application using Apache Cordova to find and display the current location of the user is done.

LIBRARY APPLICATION USING JAVA

Aim :

To Write programs using Java to create Android application having Databases for displaying books available, books lend, book reservation.

Algorithm :

1. Set Up Android Project : Create a new Android project in Android Studio for the application.
2. Design User Interface : Design the UI layout using XML in Android Studio, incorporating RecyclerViews, TextViews, and Buttons for displaying book information.
3. Database Connection Setup : Establish database connectivity using SQLite, Room, or connect to an external database server by configuring connection parameters.
4. Retrieve Book Data : Write Java code to query the database server, executing SQL queries to fetch details about available, lent, and reserved books.
5. Display Book Information : Populate UI components with the retrieved book data, structuring the display using adapters and layouts.
6. Implement Book Reservation and Lending : Enable users to reserve and lend books via UI components, updating database records accordingly.
7. Handle User Interactions : Implement event listeners or click handlers to manage user interactions like viewing book details or performing book reservations.
8. Error Handling and Validation : Implement error handling for database connectivity issues and validate user actions to prevent invalid operations.
9. Deployment : Build the Android application APK, test it on emulators or physical devices, and deploy it on Android devices or distribution channels.

Program:

```
public class Book {  
    private int id;  
    private String title;  
    private String author;
```

```

    private boolean isAvailable;
    private boolean isLent;
    private boolean isReserved;
}

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "LibraryDB";
    private static final int DATABASE_VERSION = 1;

    // Define table and column names

    private static final String TABLE_BOOKS = "books";
    private static final String COLUMN_ID = "id";
    private static final String COLUMN_TITLE = "title";
    private static final String COLUMN_AUTHOR = "author";
    private static final String COLUMN_AVAILABLE = "available";
    private static final String COLUMN_LENT = "lent";
    private static final String COLUMN_RESERVED = "reserved";

    private static final String CREATE_TABLE_BOOKS = "CREATE TABLE " + TABLE_BOOKS
+ "(" +
        COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
        COLUMN_TITLE + " TEXT," +
        COLUMN_AUTHOR + " TEXT," +
        COLUMN_AVAILABLE + " INTEGER," +
        COLUMN_LENT + " INTEGER," +
        COLUMN_RESERVED + " INTEGER)";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_TABLE_BOOKS);
    }
}

```

```

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_BOOKS);
    onCreate(db);
}
}

```

ManiActivity

```

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Example usage
        BookRepository bookRepository = new BookRepository(this);

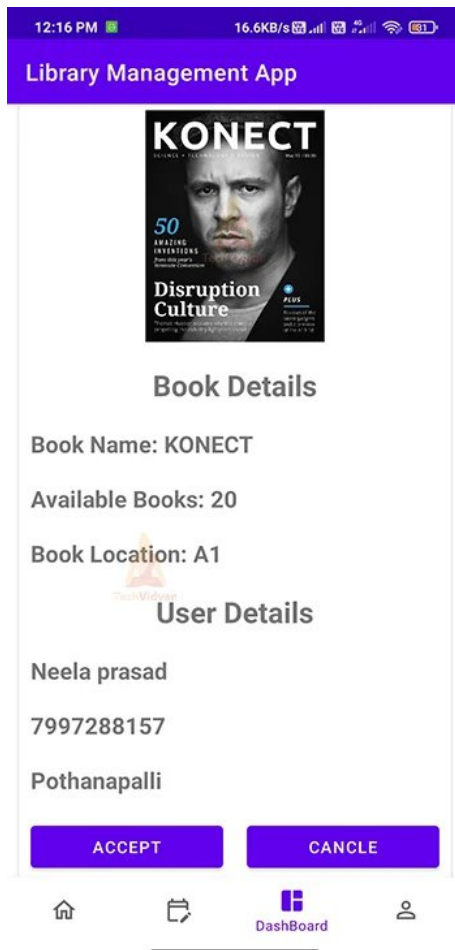
        // Add a book
        Book newBook = new Book();
        newBook.setTitle("Sample Book");
        newBook.setAuthor("John Doe");
        newBook.setAvailable(true);
        newBook.setLent(false);
        newBook.setReserved(false);
        bookRepository.addBook(newBook);

        // Get all books
        List<Book> allBooks = bookRepository.getAllBooks();

        // Now you can use the list of books as needed
    }
}

```

OUTPUT:



Result:

Thus the programs using Java to create Android application having Databases for displaying books available, books lend, book reservation was successfully verified.