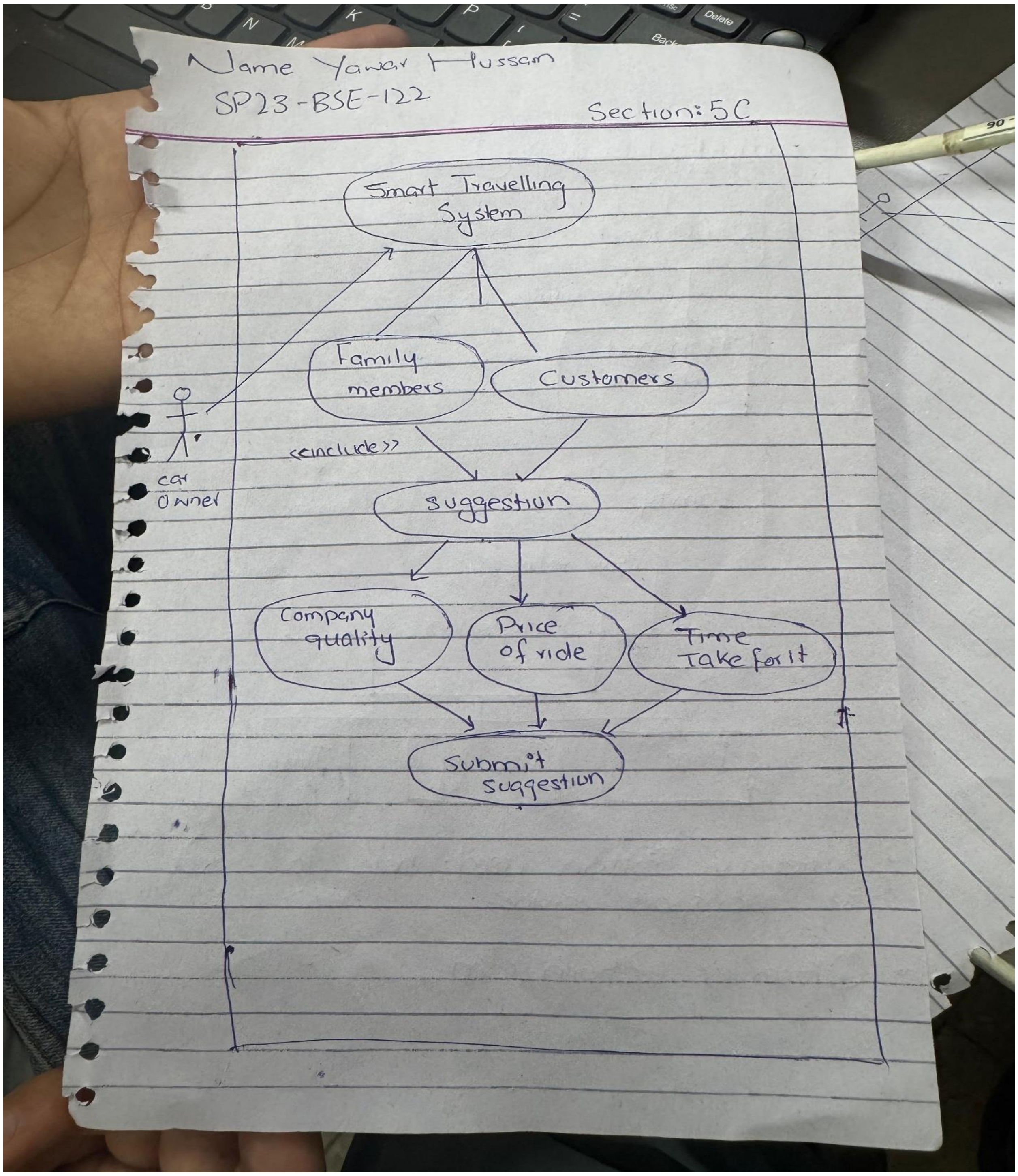
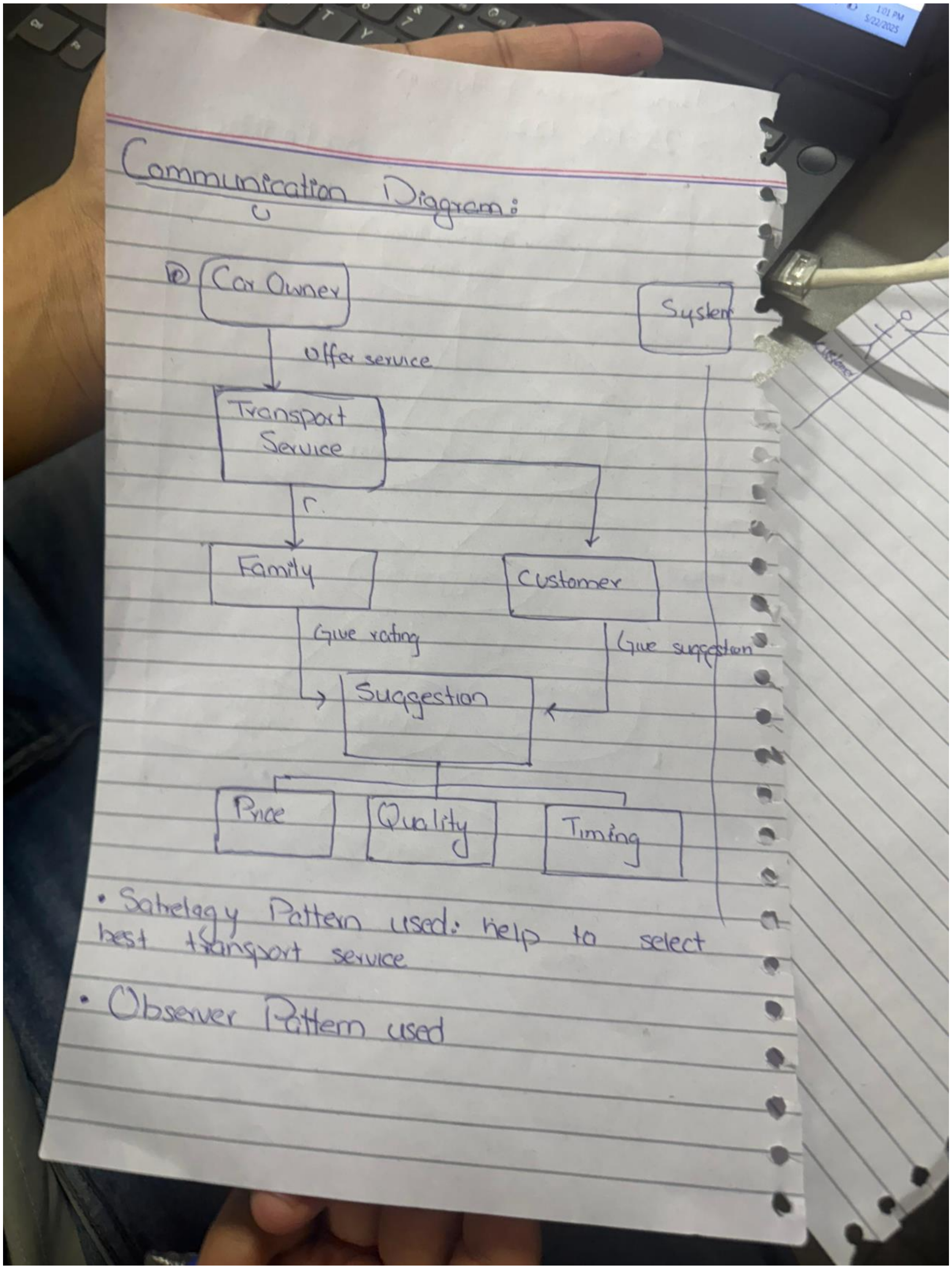
Name:Yawar Hussain **SP23-BSE-122** **Section: BSE 5C**

USECASE:



COMMUNICATION DIAGRAM:



Communication Diagram:

- Satrelogy Pattern used: help to select best transport service
- Observer Pattern used

Principle and Pattern Use:

Strategy Pattern: it provides the dynamic and interchangeable strategies for selection the best transportation option. The system has different strategies based on cost, time and quality

Observer Pattern: For real-time update where family member or commuters themselves can request for ride to get real time updates when there is a change in their journey

CODE:

HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Traveling System</title>
</head>
<body>
  <h1>Smart Traveling System</h1>

  <label for="destination">Enter your destination:</label>
  <input type="text" id="destination" placeholder="Destination">
  <button id="getOptionsButton">Get Travel Options</button>

  <div id="travelOptions"></div>

  <h2>Family Member</h2>
  <label for="familyName">Enter your name to subscribe:</label>
  <input type="text" id="familyName" placeholder="Family Member's Name">
  <button id="subscribeButton">Subscribe for Updates</button>

  <script src="script.js"></script>
</body>
</html>
```

JavaScript:

```javascript
// Strategy Pattern: Transport options
class TransportStrategy {
  getTravelDetails(destination) {
    throw new Error("Method must be implemented");
  }
}

class Taxi extends TransportStrategy {
  getTravelDetails(destination) {
    return {
      transport: 'Taxi',
      time: '30 minutes',
      cost: '$20',
      quality: 'High',
    };
  }
}

class Public_Transport extends TransportStrategy {
  getTravelDetails(destination) {
    return {
      transport: 'Public Transport',
      time: '45 minutes',
      cost: '$5',
      quality: 'Low',
    };
  }
}

// Observer Pattern: Family member who wants updates
class FamilyMemberObserver {
  constructor(name) {
    this.name = name;
  }

  update(message) {
    console.log(`${this.name} received an update: ${message}`);
  }
}

// TravelBookingSystem that combines Strategy and Observer patterns
```

```javascript
class TravelBookingSystem {
  constructor() {
    this.observers = []; // List of observers (family members)
  }

  addObserver(observer) {
    this.observers.push(observer);
  }
notifyObservers(message) {
  for (let i = 0; i < this.observers.length; i++) {
    this.observers[i].update(message);
  }
}

  // Book travel using a strategy
  bookTravel(strategy, destination) {
    const details = strategy.getTravelDetails(destination);
    this.notifyObservers(`Booking Confirmed: ${details.transport} to
${destination}`);
    return details;
  }
}

// Simple Frontend logic for UI interaction
const travelSystem = new TravelBookingSystem();

// Get travel options and display them
document.getElementById("getOptionsButton").addEventListener("click", function()
{
  const destination = document.getElementById("destination").value;
  if (destination) {
    const Taxi = new Taxi();
    const Public_Transport = new Public_Transport();

    const taxiDetails = travelSystem.bookTravel(Taxi, destination);
    const publicTransportDetails = travelSystem.bookTravel(Public_Transport,
destination);

    displayOption(taxiDetails);
    displayOption(publicTransportDetails);
  } else {
    alert("Please enter a destination.");
  }
});
```

```
// Display travel options
function displayOption(details) {
  const travelOptionsDiv = document.getElementById("travelOptions");
  const optionDiv = document.createElement("div");
  optionDiv.innerHTML = `
    <h3>${details.transport}</h3>
    <p>Time: ${details.time}</p>
    <p>Cost: ${details.cost}</p>
    <p>Quality: ${details.quality}</p>
    <button onclick="bookTravel('${details.transport}')">Book
${details.transport}</button>
  `;
  travelOptionsDiv.appendChild(optionDiv);
}

// Handle booking when user clicks "Book"
function bookTravel(transportType) {
  alert(`${transportType} booked successfully!`);
}

// Handle subscription of family member for updates
document.getElementById("subscribeButton").addEventListener("click", function() {
  const familyName = document.getElementById("familyName").value;
  if (familyName) {
    const familyMember = new FamilyMemberObserver(familyName);
    travelSystem.addObserver(familyMember);
    alert(`${familyName} is now subscribed for updates in DataBase`);
  } else {
    alert("Please enter a family member's name.");
  }
});
```