SRI KRISHNA COLLEGE OF TECHNOLOGY
(An Autonomous Institution)
Approved by AICTE | Affiliated to Anna University Chennai|
Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042

LIFE INSURANCE MANAGEMENT

A PROJECT REPORT

Submitted by

SANTHANA KUMAR A H	**727822TUCS205**
SHARON SWEETY V	**727822TUCS212**
SHIYAM R	**727822TUCS215**
SHREE HARINESH S	**727822TUCS216**

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

JULY 2024

BONAFIDE CERTIFICATE

Certified that this project report **LIFE INSURANCE MANAGEMENT** is the bonafide work **SANTHANA KUMAR A H 727822TUCS205 , SHARON SWEETY V 727822TUCS212 , SHIYAM R 727822TUCS215 , SHREE HARINESH S 727822TUCS216** who carried out the project work under my supervision.

SIGNATURE

Ms. A. GOMATHY

SUPERVISOR

Assistant Professor,

Department of Computer Science and Engineering,

Sri Krishna College of Technology,Coimbatore-641042

SIGNATURE

Dr. M. UDHAYAMOORTHI

HEAD OF THE DEPARTMENT

Associate Professor,

Department of Computer Science and Engineering,

Sri Krishna College of Technology,Coimbatore-641042

Certified that the candidate was examined by me in the Project Work Viva Voce examination held on_____at Sri Krishna College of Technology, Coimbatore-641042.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost we thank the **Almighty** for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, **Dr. M.G. Sumithra,** for providing all facilities.

We are grateful to our beloved Head, Computing Sciences **Dr.T. Senthilnathan,** for her tireless and relentless support.

With the grateful heart, our sincere thanks to our Head of the Department **Dr. M. Udhayamoorthi**, Department of Computer Science and Engineering for the motivation and all support to complete the project work.

We thank **Ms. A. Gomathy**, Assistant Professor, Department of Computer Science and Engineering, for his motivation and support.

We are thankful to all the **Teaching and Non-Teaching Staff** of Department of Computer Science and Engineering and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our **family members** and our beloved **friends**, who had been strongly supporting us in all our endeavour

ABSTRACT

Life Plus is an advanced life insurance website designed to simplify and enhance the process of purchasing and managing insurance policies. Utilizing modern web technologies such as React JS, Spring Boot, REST API, and MySQL, LifePlus offers a comprehensive platform for users to explore, apply for, and manage various insurance policies. The user-friendly interface ensures a seamless experience for both customers and administrators, with features tailored to meet their specific needs. Key functionalities include policy exploration, application processing, premium payment, and claim management. Customers can browse through a wide range of policies, compare benefits, and apply online with ease. The system supports secure payment processing for premium dues and facilitates quick and efficient claim submissions. Administrators have access to a powerful dashboard for policy management, user account oversight, and transaction tracking, ensuring smooth operations and enhanced customer service.Life Plus leverages React JS for a dynamic and responsive frontend experience, while Spring Boot's robust RESTful APIs ensure reliable backend support. MySQL's scalable database architecture provides a solid foundation for managing extensive policy data and user information. Integrated email and real-time push notifications keep customers informed about policy updates, payment reminders, and claim statuses. The platform's advanced reporting tools offer administrators valuable insights into customer behaviour, policy trends, and financial metrics, aiding in strategic planning and resource allocation. Scalability is a key feature, allowing Life Plus to grow alongside its expanding user base and policy offerings. In summary, Life Plus is a state-of-the-art solution tailored to the diverse needs of life insurance customers and administrators, promising a seamless, efficient, and user-centric experience. By automating and streamlining insurance processes, Life Plus significantly improves transparency, communication, and overall customer satisfaction within the insurance ecosystem.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Acronym
JSX	JAVASCRIPT XML
JPA	JPA PERSISTENCE API
REST	REPRESENTATIONAL STATE TRANSFER
SDLC	SOFTWARE DEVELOPMENT LIFE CYCLE
JWT	JSON WEB TOKEN

CHAPTER 1

INTRODUCTION

This project aims to offer a seamless and efficient solution for customers and administrators to manage life insurance policies through an online platform. In this chapter, we will explore the problem statement, provide an overview, and outline the main objectives of the Life Plus insurance management system.

1.1 PROBLEM STATEMENT

How can we develop a life insurance management system that allows customers to easily explore, apply for, and manage various insurance policies, while enabling administrators to efficiently oversee policy applications, premium payments, and claim processes, ensuring a user-friendly interface, secure transactions, and real-time notifications?

1.2 OVERVIEW

In the realm of life insurance management, customers and administrators often face challenges such as navigating complex policy options, managing premium payments, and efficiently processing claims. Traditional methods can be cumbersome, error-prone, and lead to customer dissatisfaction. To address these issues, we propose the creation of LifePlus, a comprehensive life insurance management system. This system will leverage modern web technologies to provide a robust, user-friendly platform that simplifies policy exploration, application, payment, and claim management. By incorporating secure transactions, real-time notifications, and advanced reporting tools, LifePlus aims to enhance operational efficiency and customer satisfaction.

1.3 OBJECTIVE

The primary objective of this project is to develop Life Plus, a life insurance management system that provides customers and administrators with a user-friendly and efficient platform for managing insurance policies. The system aims to streamline policy exploration, application, premium payments, and claim processing, while ensuring secure transactions, real-time notifications, and advanced reporting tools to enhance communication and operational efficiency.

CHAPTER 2
SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website. This chapter gives you a small description about the software used in the project.

2.1 VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of extensions that allow developers to quickly add features like code snippets, debugging tools, and linting supportto their projects.

2.2 LOCAL STORAGE

Local storage is a versatile tool in web development, offering a means to store data persistently on a user's browser without the need for server-side interaction. Unlike cookies, which are limited in size and automatically sent with every HTTP request, local storage allows developers to store up to 5MB of data in a more controlled and efficient manner. This makes it particularly useful for applications that require quick access to data, such as offline applications or features that enhance user experience by storing preferences and settings. Local storage operates on the same-origin policy, ensuring that data stored by one website cannot be accessed by another, which enhances security and privacy.

Developers can use the localStorage object in JavaScript to easily set, retrieve, and remove data as needed. This data is stored in key-value pairs, where both the key and value must be strings. If necessary, complex data structures like objects and arrays can be stored by first converting them to strings using JSON.stringify() and then parsed back into their original form with JSON.parse() when retrieved. Local storage is asynchronous and remains intact even after the browser is closed or the device is restarted, providing a persistent storage solution for web applications. Its usage is supported across all major browsers, ensuring consistent behavior for users regardless of their browser choice. However, it's important for developers to manage local storage carefully, as it does not automatically expire and can accumulate over time, potentially leading to storage bloat if not properly maintained.

One of the key benefits of using local storage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storage is stored locally on the user's machine, and so is not affected by these issues. Another advantage of local storage is its speed. Because the data is stored locally, it is accessed quickly, as there is no need to send requests to a server. This makes it ideal for storing data that needs to be accessed quickly, such as user preferences or session data. Local storage is also secure, as the data is stored on the user's machine and not on a server. This means that the data is not accessible by anyone other than the user, making it a good choice for storing sensitive information.

CHAPTER 3

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

3.1 PROPOSED SYSTEM

The proposed Life Plus system offers a multitude of benefits from various perspectives. The online life insurance platform empowers customers to explore, apply for, and manage insurance policies efficiently, taking into account their specific needs and preferences. The system simplifies the entire insurance process, enabling customers to easily browse policies, submit applications, make premium payments, and manage claims online.

Once a policy application is submitted, it is directed to the designated personnel responsible for processing insurance applications. The administrative team ensures that applications are accurate and complete, with specialized staff members handling any necessary verifications or adjustments within a specified timeframe. Approved policies are updated in the system and are accessible to all relevant customers and administrators.

This system significantly reduces the administrative workload and streamlines operations in insurance management. Especially during peak periods or when dealing with complex policy applications, where manual processes may lead to delays or errors, the online platform mitigates such challenges. Customers can directly manage their policies and claims online, bypassing potential bottlenecks such as limited office hours or manual processing delays.

Moreover, the system enhances operational efficiency, enabling administrators to manage policy applications, payments, and claims promptly and effectively. By leveraging technology to automate data entry, verification, and notifications, the online insurance platform ensures swift and accurate processing, resulting in improved customer satisfaction and efficiency in policy management.

3.2 ADVANTAGES

- Efficiency: The staff scheduling system allows administrators to create, modify, and manage schedules quickly and easily from anywhere with internet access. This eliminates the need for time-consuming manual processes, reducing administrative workload and saving time for both administrators and staff members.
- Flexibility: Staff members can update their availability and view their schedules online at their convenience. This flexibility helps accommodate varying schedules and preferences, improving staff satisfaction and allowing for more accurate scheduling.
- Conflict Resolution: The system automatically detects and resolves scheduling conflicts, ensuring that all shifts and duties are covered without overlapping. This reduces the chances of errors and double-bookings, leading to smoother operations.
- Accessibility: Administrators and staff can access the scheduling system from any location with internet access. This accessibility ensures that schedules can be managed and viewed remotely, facilitating better coordination and planning.
- Transparency: The system provides clear and detailed information about staff schedules, roles, and assignments. This transparency helps administrators and staff stay informed about their duties and responsibilities, reducing confusion and enhancing communication.
- Real-Time Notifications: Automated notifications keep staff members updated on schedule changes, upcoming duties, and important reminders. This feature ensures that everyone is informed in real-time, improving coordination and reducing the risk of missed tasks.

CHAPTER 4

METHODOLOGIES

4.1 User FlowChart : -

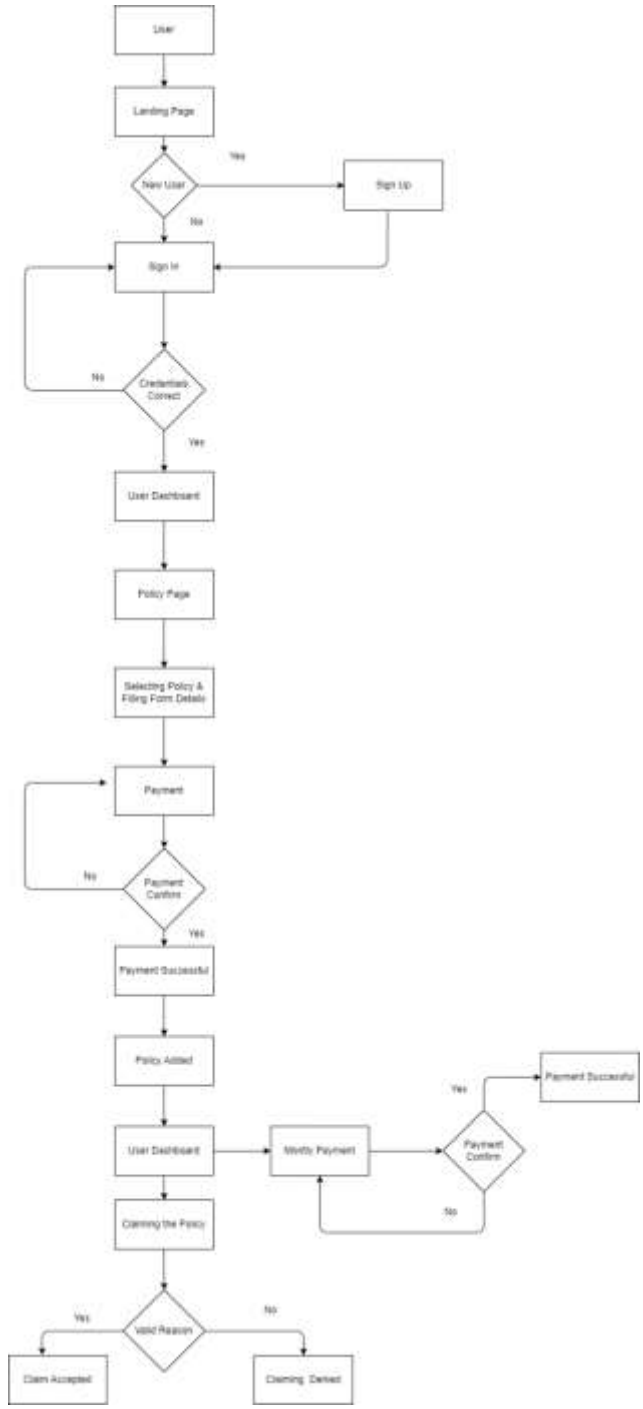
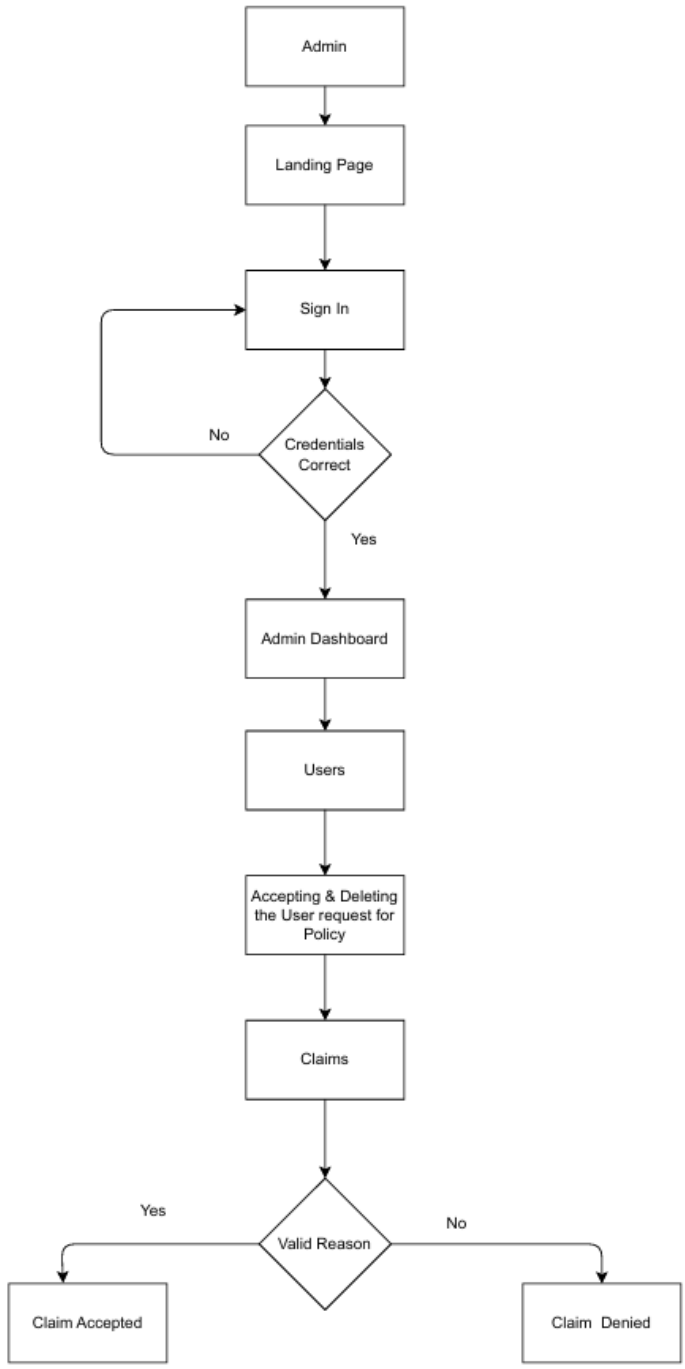


Fig 4.1 User FlowChart

4.2 Admin FlowChart : -



4.2 Admin FlowChart

User Registration: Customers register on the LifePlus platform by providing essential details such as name, email address, and password. Optional information like phone number and address may be included for better communication and policy management.

Role-Based Access: Upon registration/login, users are assigned roles such as customer or administrator, which determine their access permissions and functionalities within the system.

Policy Exploration: Users can explore available insurance policies by browsing through different categories or using search functionality to find policies that match their needs.

Policy Application: Customers can apply for chosen insurance policies by submitting necessary personal and financial information. Administrators review these applications and verify the provided details.

Premium Payment: Upon approval of the policy application, customers can make secure premium payments through the platform. The system processes these payments in real-time and updates the policy status.

Claim Request: Customers can submit claims by providing relevant details and documentation. Administrators review these claims and proceed to process them accordingly.

Notification: After processing, the system generates notifications about policy updates, payment confirmations, or claim statuses. Customers receive confirmation of new policies, successful payments, or claim resolutions.

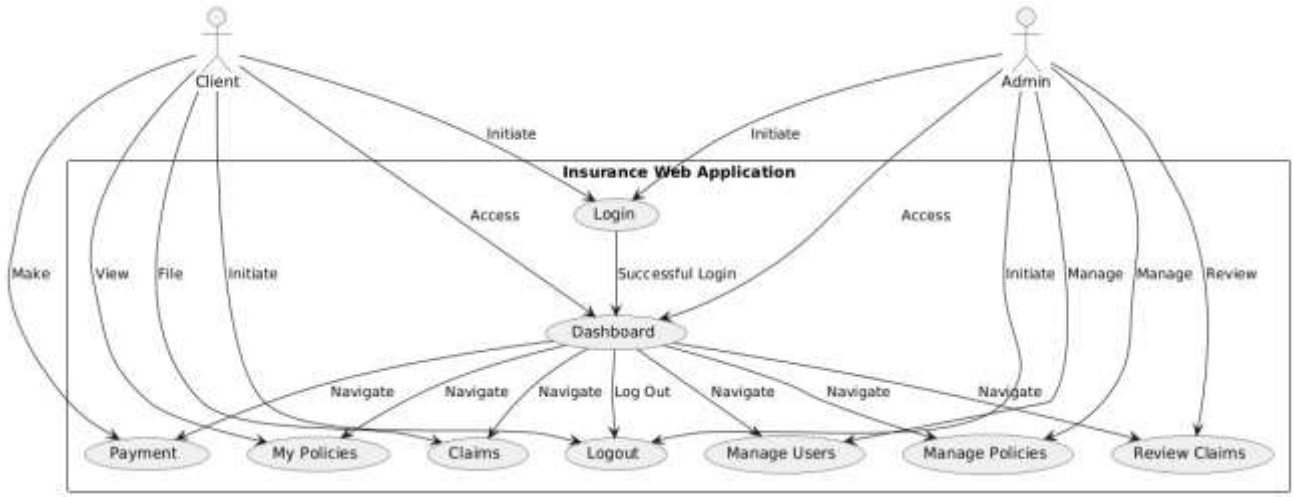
Policy Management: Administrators review and manage policy applications, premium payments, and claims. The system updates the policy status and notifies all relevant users about any changes.

Real-Time Updates: Customers receive real-time updates and reminders about premium due dates, policy renewals, and claim statuses.

Feedback and Adjustments: After interacting with the platform, customers can provide feedback on their experience. Administrators can make adjustments based on this feedback to improve the system.

4.3 Use Case Diagram:

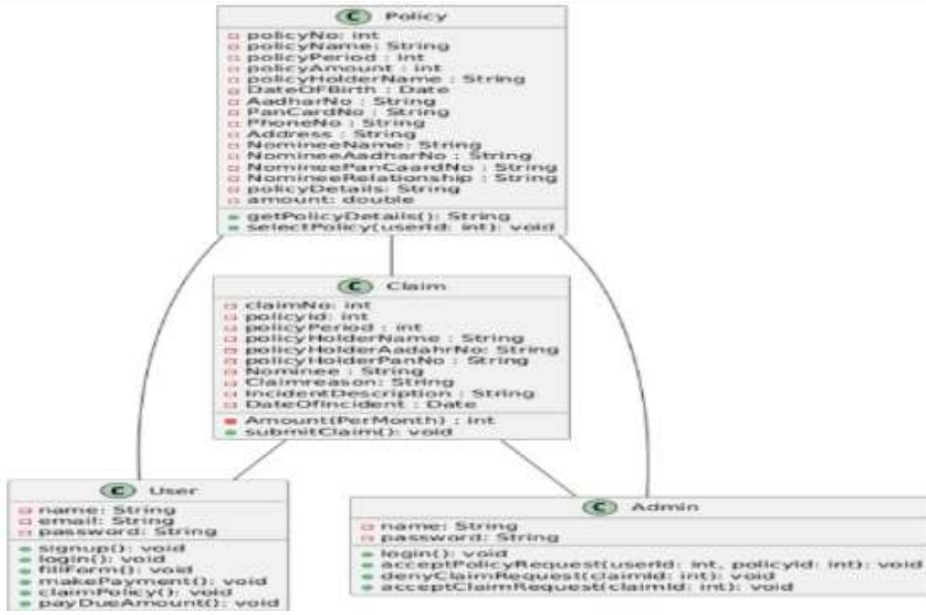
The use case diagram illustrates a Life Insurance Management System, depicting various user roles including Clients and Administrators. Each role has specific functionalities to efficiently manage insurance policies, payments, and claims. The diagram emphasizes interaction points for roles to perform tasks like policy management, claim processing, and user account management, showcasing the workflow and the interplay between different users and system features for seamless insurance operations.



4.3 Use Case Diagram

4.4 Class Diagram:

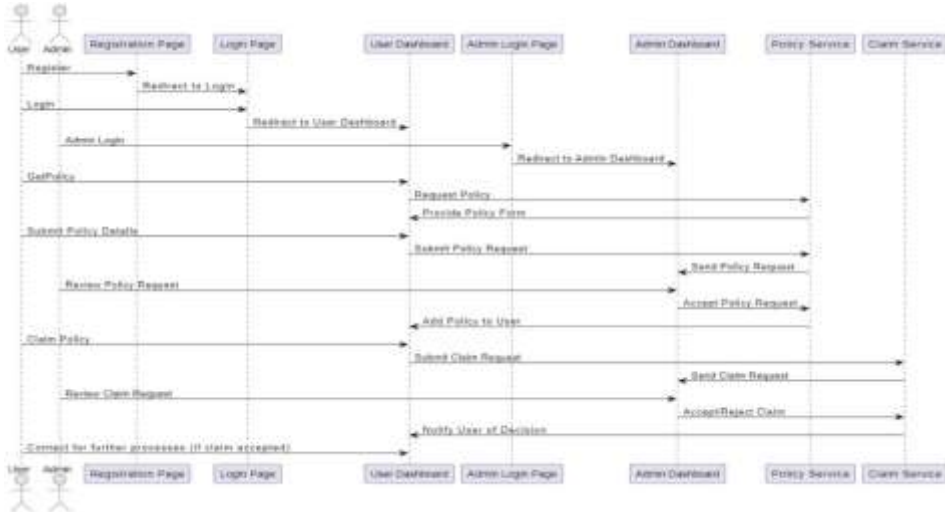
The class diagram outlines the architecture of the **Insurance Management System**, detailing the roles and their specific functionalities. Key classes include **Policy**, **Claim**, **User**, and **Admin**, each interacting with various attributes and methods related to policy management, claims processing, and user actions. This diagram highlights methods for managing policies, claims, user authentication, and administrative tasks, illustrating how each role contributes to the system's operations.



4.4 Class Diagram

4.5 Sequence Diagram:

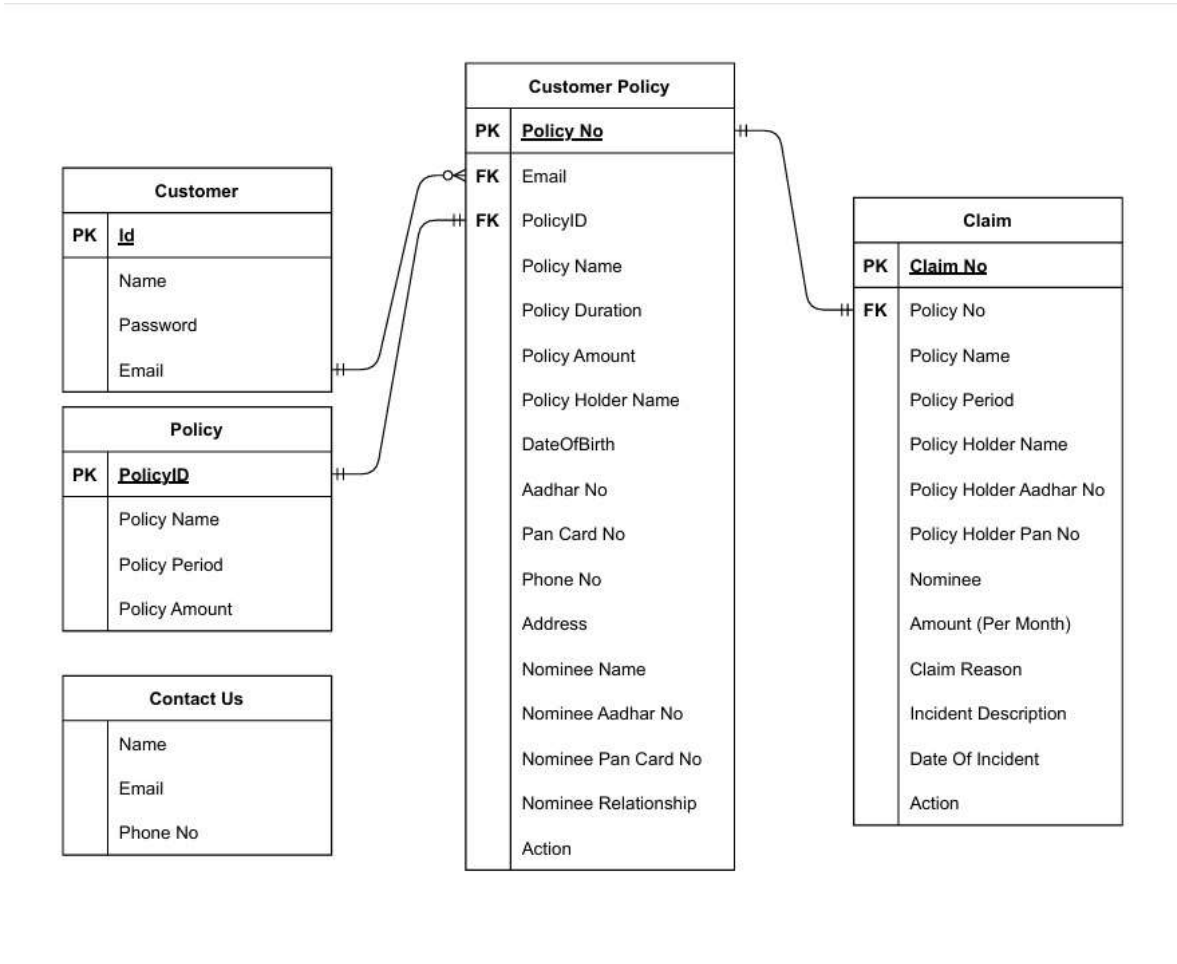
The sequence diagram illustrates the process for users and administrators to manage policies and claims in the Insurance Management System. The diagram shows interactions between users (both regular users and admins) and various system components such as the registration page, login page, user dashboard, admin dashboard, policy service, and claim service. It highlights key activities including user registration, policy requests, and claims processing.



4.5 Sequence Diagram

4.6 ER Diagram:

The ER diagram illustrates a life insurance database with entities including Customer, Policy, Customer Policy, Claim, and Contact Us. The Customer entity stores customer details such as Id, Name, Password, and Email, while the Policy entity contains information like PolicyID, Policy Name, Policy Period, and Policy Amount. The Customer Policy entity links customers to their policies, capturing additional details like Policy No, Policy Holder Name, and nominee information. The Claim entity is associated with policies and stores claim details, including Claim No, Amount, and Incident Description. Lastly, the Contact Us entity holds basic contact information for user inquiries. The relationships indicate that a customer can hold multiple policies, and each policy can be associated with multiple claims.



4.6 ER Diagram

CHAPTER 5

IMPLEMENTATION AND RESULT

This chapter gives a description about the output that we produced by developing the website of our idea.

5.1 LOGIN

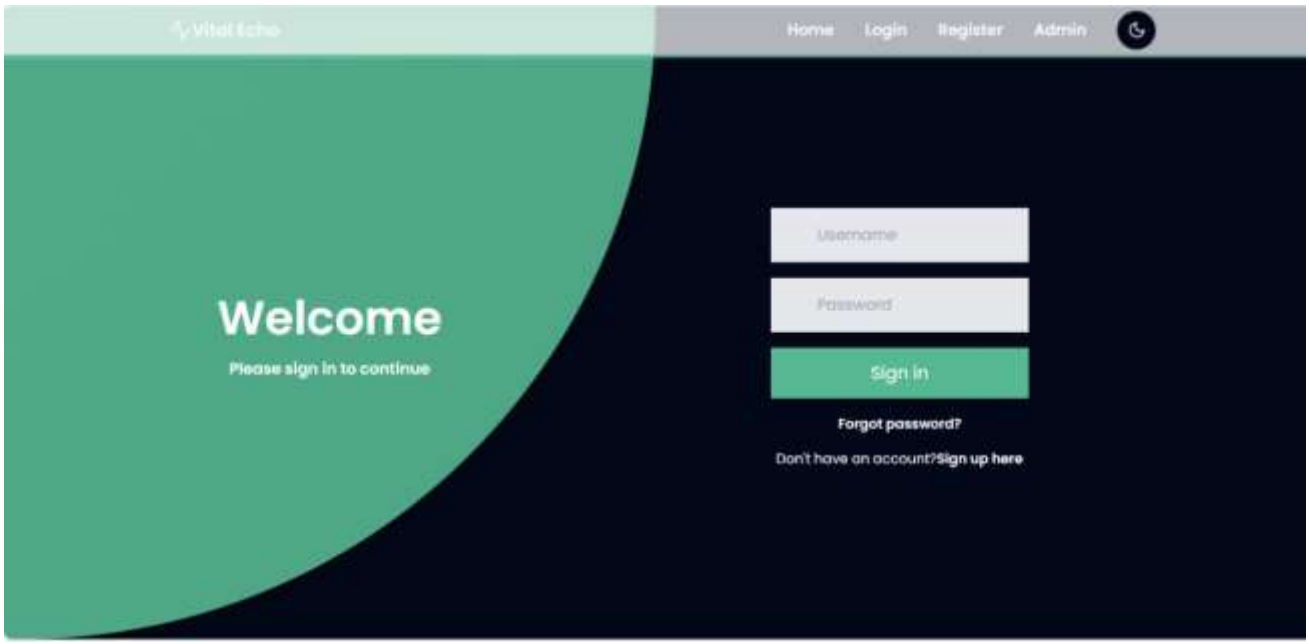


Fig 5.1 LOGIN PAGE

5.2 USER REGISTER

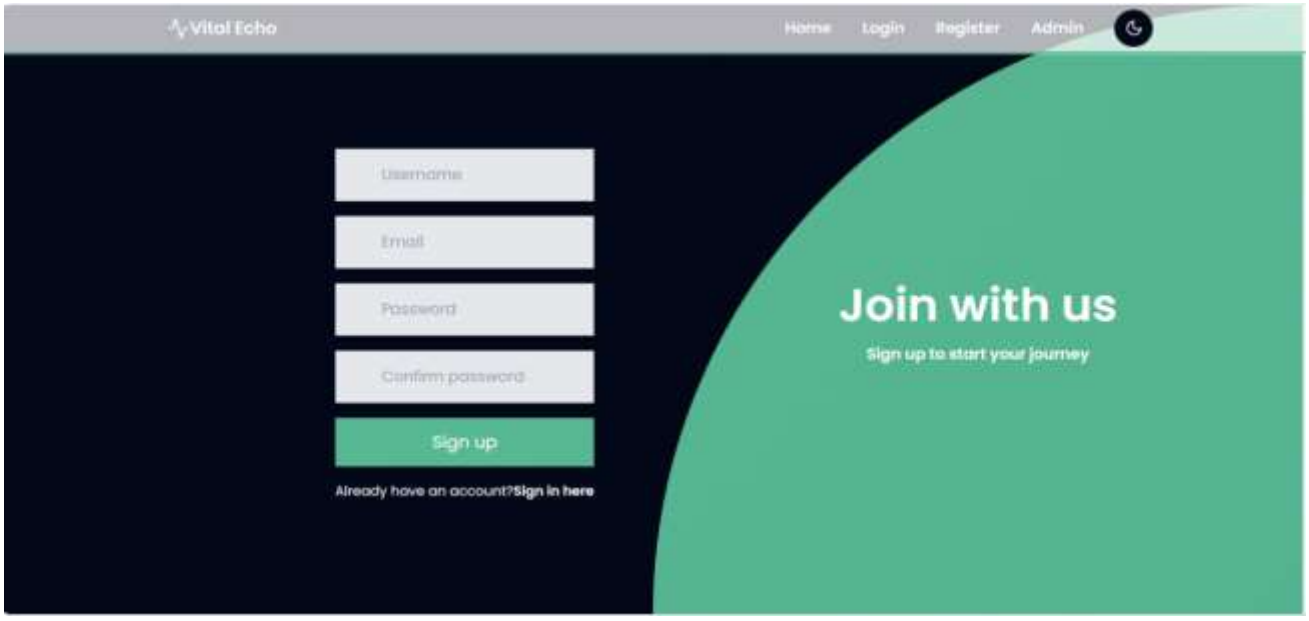


Fig 5.2 USER REGISTER

5.3 ADMIN DASHBOARD

Fig 5.3 Admin Dashboard

5.4 ADMIN DASHBOARD Users

Fig 5.4 Admin Dashboard Users Page

5.5 USER DASHBOARD

Fig 5.5 User Dashboard

5.6 USER DASHBOARD POLICY

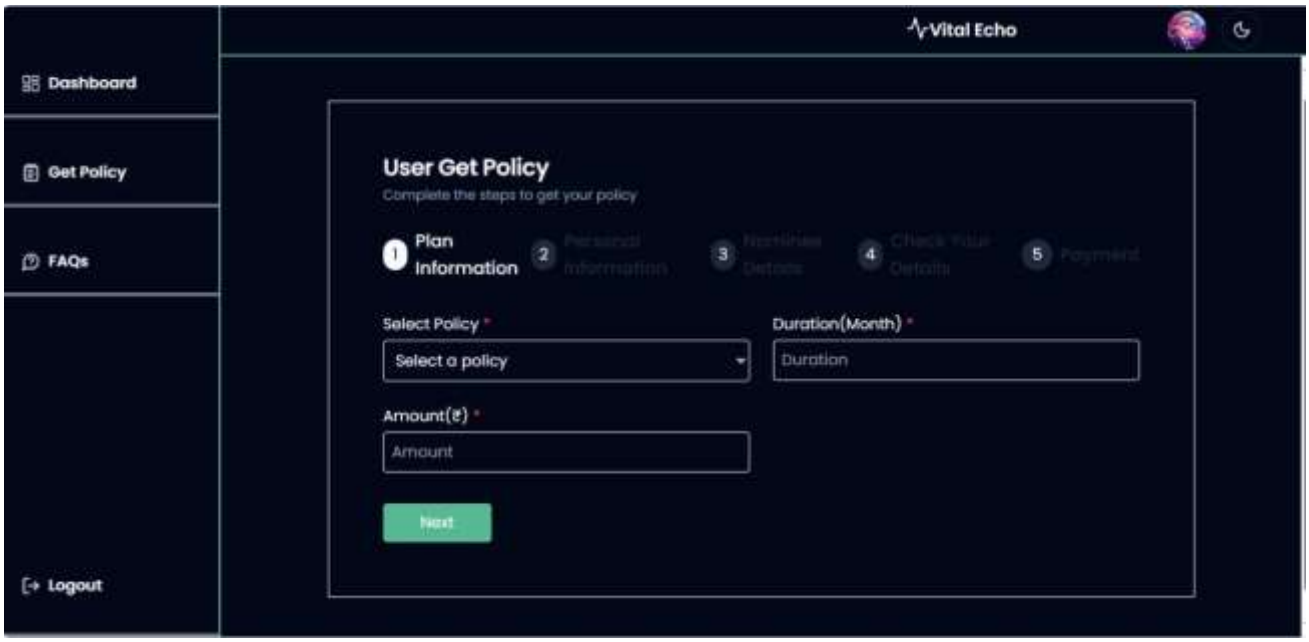


Fig 5.6 User Dashboard Policy Form

5.7 FAQ PAGE

Fig 5.7 FAQ Page

5.8 CODING

Config:

```
@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {

    private final UserRepo userRepo;

    @Bean
    public UserDetailsService userDetailsService(){
        return  username -> userRepo.findByEmail(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not Found"));
    }
```

```
    @Bean
    public AuthenticationProvider authenticationProvider(){

        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
throws Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}


@Configuration
public class CorsConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
```

```
        public void addCorsMappings(CorsRegistry registry) {
            registry.addMapping("/**")
                    .allowedOrigins("localhost:5173")
                    .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                    .allowedHeaders("*")
                    .allowCredentials(true);
        }
    };
  }
}
```

```
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {
   private final JwtService jwtService;
   private final UserDetailsService userDetailsService;

   @Override
   protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain) throws ServletException, IOException {
      final String authHeader =request.getHeader("Authorization");
      final String jwtToken;
      final String userEmail;

      if(authHeader == null || !authHeader.startsWith("Bearer "))
      {
```

```
            filterChain.doFilter(request,response);
            return;
        }

        jwtToken = authHeader.substring(7);
        userEmail = jwtService.extractUserName(jwtToken);
        if(userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null){
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);
            if(jwtService.isTokenValid(jwtToken, userDetails)){
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request)
                );
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request,response);


    }
}

@Service
public class JwtService {
```

```
    private static final String SECRET_KEY =
"EbeEsh7VhXpHMAkLz7Xb3TYm7a4KLMlYn0Kr1NJEhTIOeU9HJsv3t2bMa5OjoiaD";

    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver){
        final Claims claims  = extractAllClaims(token);
        return  claimsResolver.apply(claims);
    }

    public String generateToken(UserDetails userDetails){
        return generateToken(new HashMap<>(), userDetails);
    }

    public String generateToken(
            Map<String, Object> extraClaims,
            UserDetails userDetails
    ){
        return Jwts
                .builder()
                .claims(extraClaims).
                subject(userDetails.getUsername())
                .issuedAt(new Date(System.currentTimeMillis()))
                .expiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();

    }
```

```
    public Boolean isTokenValid(String token ,UserDetails userDetails)
    {
        final  String username = extractUserName(token);
        return  (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    private Claims extractAllClaims(String token)
    {
        return Jwts
                .parser()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyByte = Decoders.BASE64.decode(SECRET_KEY);
        return Keys.hmacShaKeyFor(keyByte);
    }
}
```

```
@Configuration
public class LogoutConfiguration {

   @Bean
   public CustomLogoutHandler logoutHandler(TokenRepo tokenRepo, JwtService jwtService)
{
      return new CustomLogoutHandler(tokenRepo, jwtService);
   }

   @Bean
   public LogoutSuccessHandler logoutSuccessHandler() {
      return new CustomLogoutSuccessHandler();
   }
}
```

Home page:

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Builder
public class Customer implements UserDetails {

   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   @Id
   private Long id;
   private String name;
```

```
    private String email;
    private String password;

    @JsonBackReference
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Token> tokens;

    @Enumerated(EnumType.STRING)
    private Role role;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
```

```
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

```
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Entity
public class Schedule {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;

    private String title;
    private LocalDateTime date;
```

```
    private LocalDateTime time;
    private String assignedTo;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = “user_id”)
    @JsonIgnore
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = “cutomer_mail”)
    @JsonIgnore
    private Customer customer;
}
```

UserDashboard.jsx : -

```
const UserDashboard = () => {
  const [open, setOpen] = useState(false);
  const [selectedPolicy, setSelectedPolicy] = useState(null);
  const [action, setAction] = useState(null);
  const [formValues, setFormValues] = useState({});
  const [formErrors, setFormErrors] = useState({});
  const [isFormValid, setIsFormValid] = useState(false);

  const policies = [
    {
      policyNo: "POL001",
```

```
  policyName: "LifePlus Term Insurance",
  planMonths: 12,
  planYears: 1,
  policyHolderName: "John Doe",
  policyHolderAadhar: "1234 5678 9012",
  policyHolderPAN: "ABCDE1234F",
  nominee: "Jane Doe",
  amount: "₹750",
  status: "",
},
{
  policyNo: "POL002",
  policyName: "LifePlus Whole Life Insurance",
  planMonths: 24,
  planYears: 2,
  policyHolderName: "Alice Smith",
  policyHolderAadhar: "9876 5432 1098",
  policyHolderPAN: "XYZAB5678C",
  nominee: "Bob Smith",
  amount: "₹1000",
  status: "",
},
{
  policyNo: "POL003",
  policyName: "LifePlus Universal Insurance",
  planMonths: 6,
  planYears: 0,
  policyHolderName: "Charlie Brown",
  policyHolderAadhar: "1357 2468 9023",
  policyHolderPAN: "LMNOP1234D",
```

```
    nominee: "Lucy Brown",
    amount: "₹900",
    status: "",
  },
];

useEffect(() => {
  validateForm();
}, [formValues]);

const openSheetWithPolicy = (policy, actionType) => {
  setSelectedPolicy(policy);
  setAction(actionType);
  setOpen(true);
  setFormValues({});
  setFormErrors({});
};

const handleInputChange = (e) => {
  const { id, value } = e.target;
  setFormValues({ ...formValues, [id]: value });
};

const validateForm = () => {
  const errors = {};
  const requiredFields = action === 'Claim'
    ? ['claimReason', 'additionalDetails', 'dateOfIncident']
    : ['cardNumber', 'cname', 'expirydate', 'cvv'];

  requiredFields.forEach(field => {
```

```
    if (!formValues[field]) {
      errors[field] = 'This field is required';
    }
  });

  setFormErrors(errors);
  setIsFormValid(Object.keys(errors).length === 0);
};

const handleSubmit = () => {
  if (action === 'Claim') {
    toast.info("Your claim is being processed. You will be notified shortly.");
  } else {
    toast.success("Payment successful", {
      icon: " ✔ ",
    });
  }
  setOpen(false);
};
return (
  <>
    <Card>
      <CardHeader className='w-full flex flex-row justify-between items-center'>
        <CardTitle> My Policies</CardTitle>
      </CardHeader>
      <CardContent>
        <Table>
          <TableHeader>
            <TableRow>
              <TableHead className="w-[100px]">Policy No</TableHead>
```

```
            <TableHead>Policy Name</TableHead>
            <TableHead>Duration (Months/Years)</TableHead>
            <TableHead>Policy Holder Name</TableHead>
            <TableHead>Policy Holder Aadhar No</TableHead>
            <TableHead>Policy Holder PAN No</TableHead>
            <TableHead>Nominee</TableHead>
            <TableHead>Amount (per month)</TableHead>
            <TableHead className="text-center w-[200px]">Actions</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {policies.map((policy) => (
            <TableRow key={policy.policyNo}>
              <TableCell className="font-medium">{policy.policyNo}</TableCell>
              <TableCell>{policy.policyName}</TableCell>
              <TableCell>{policy.planMonths} Months / {policy.planYears}
Years</TableCell>
              <TableCell>{policy.policyHolderName}</TableCell>
              <TableCell>{policy.policyHolderAadhar}</TableCell>
              <TableCell>{policy.policyHolderPAN}</TableCell>
              <TableCell>{policy.nominee}</TableCell>
              <TableCell className="text-right">{policy.amount}</TableCell>
              <TableCell className="text-right">
                <Button variant="ghost" size="sm" className="mr-2 bg-customGreen
rounded-3xl" onClick={() => openSheetWithPolicy(policy, 'Claim')}>
                  Claim
                </Button>
                <Button variant="ghost" size="sm" className="bg-customGreen rounded-
3xl" onClick={() => openSheetWithPolicy(policy, 'Pay')}>
                  Pay Now
```

```
                </Button>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </CardContent>
  </Card>

  <Sheet open={open} onOpenChange={() => setOpen(false)}>
    <SheetContent>
      <SheetHeader>
        <SheetTitle>{action === 'Claim' ? 'Claim Policy' : 'Pay Policy'}</SheetTitle>
        <SheetDescription>
          {action === 'Claim' ? 'Provide details for your claim.' : 'Enter payment details.'}
        </SheetDescription>
      </SheetHeader>
      <div className="grid gap-4 py-4">
        {selectedPolicy && (
          <>
            <div className="grid grid-cols-4 items-center gap-4">
              <Label htmlFor="policyNo" className="text-right">
                Policy No
              </Label>
              <Input id="policyNo" value={selectedPolicy.policyNo} disabled
className="col-span-3" />
            </div>
            <div className="grid grid-cols-4 items-center gap-4">
              <Label htmlFor="policyName" className="text-right">
                Policy Name
```

```
            </Label>
            <Input id="policyName" value={selectedPolicy.policyName} disabled
className="col-span-3" />
          </div>

          {action === 'Claim' && (
            <>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="claimReason" className="text-right">
                  Claim Reason
                </Label>
                <Input
                  id="claimReason"
                  className={`col-span-3 ${formErrors.claimReason ? 'border-red-500' :
"}`}
                  placeholder="Enter reason for claim"
                  value={formValues.claimReason || ''}
                  onChange={handleInputChange}
                />
                {formErrors.claimReason && <span className="col-span-3 text-red-
500">{formErrors.claimReason}</span>}
              </div>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="additionalDetails" className="text-right">
                  Incident Description
                </Label>
                <Input
                  id="additionalDetails"
                  className={`col-span-3 ${formErrors.additionalDetails ? 'border-red-
500' : ''}`}
```

```
                placeholder="Enter Incident description"
                value={formValues.additionalDetails || ''}
                onChange={handleInputChange}
              />
              {formErrors.additionalDetails && <span className="col-span-3 text-red-
500">{formErrors.additionalDetails}</span>}
            </div>
            <div className="grid grid-cols-4 items-center gap-4">
              <Label htmlFor="dateOfIncident" className="text-right">
                Date of incident
              </Label>
              <Input
                id="dateOfIncident"
                type="date"
                className={`col-span-3 ${formErrors.dateOfIncident ? 'border-red-500' :
''}`}
                value={formValues.dateOfIncident || ''}
                onChange={handleInputChange}
              />
              {formErrors.dateOfIncident && <span className="col-span-3 text-red-
500">{formErrors.dateOfIncident}</span>}
            </div>
          </>
        )}

        {action === 'Pay' && (
          <>
            <div className="grid grid-cols-4 items-center gap-4">
              <Label htmlFor="cardNumber" className="text-right">
                Card Number
```

```
            </Label>
            <Input
              id="cardNumber"
              className={`col-span-3 ${formErrors.cardNumber ? 'border-red-500' :
"}`}
              placeholder="Enter Card Number"
              value={formValues.cardNumber || ''}
              onChange={handleInputChange}
            />
            {formErrors.cardNumber && <span className="col-span-3 text-red-
500">{formErrors.cardNumber}</span>}
          </div>
          <div className="grid grid-cols-4 items-center gap-4">
            <Label htmlFor="cname" className="text-right">
              Name on Card
            </Label>
            <Input
              id="cname"
              className={`col-span-3 ${formErrors.cname ? 'border-red-500' : ''}`}
              placeholder="Enter Card Holder Name"
              value={formValues.cname || ''}
              onChange={handleInputChange}
            />
            {formErrors.cname && <span className="col-span-3 text-red-
500">{formErrors.cname}</span>}
          </div>
          <div className="grid grid-cols-4 items-center gap-4">
            <Label htmlFor="expirydate" className="text-right">
              Expiry date
            </Label>
```

```
                <Input
                  id="expirydate"
                  type="text"
                  placeholder="MM/YY"
                  className={`col-span-3 ${formErrors.expirydate ? 'border-red-500' : ''}`}
                  value={formValues.expirydate || ''}
                  onChange={handleInputChange}
                />
                {formErrors.expirydate && <span className="col-span-3 text-red-
500">{formErrors.expirydate}</span>}
              </div>
              <div className="grid grid-cols-4 items-center gap-4">
                <Label htmlFor="cvv" className="text-right">
                  CVV
                </Label>
                <Input
                  id="cvv"
                  type="password"
                  className={`col-span-3 ${formErrors.cvv ? 'border-red-500' : ''}`}
                  placeholder="Enter CVV"
                  value={formValues.cvv || ''}
                  onChange={handleInputChange}
                />
                {formErrors.cvv && <span className="col-span-3 text-red-
500">{formErrors.cvv}</span>}
              </div>
            </>
          )}
        </>
      )}
```

```
        </div>
        <SheetFooter className='flex flex-col flex-1'>
          <Button className='w-1/2 bg-destructive hover:bg-destructive/80' onClick={()
=> setOpen(false)}>Cancel</Button>
          <Button
            type="button"
            className='w-1/2'
            onClick={handleSubmit}
            disabled={!isFormValid}
          >
            {action === 'Claim' ? 'Submit Claim' : 'Pay Now'}
          </Button>
        </SheetFooter>
      </SheetContent>
    </Sheet>

    <ToastContainer />
  </>
 );
};

export default UserDashboard;
```

Admin Dashboard.jsx : -

```
const AdminDashboard = () => {
  const [open, setOpen] = useState(false);
  const [editMode, setEditMode] = useState(false);
  const [selectedPolicy, setSelectedPolicy] = useState(null);
  const [newPolicy, setNewPolicy] = useState({
    policyName: '',
```

```
    duration: '',
    amount: '',
  });
  const [policies, setPolicies] = useState([
    { policyNo: 'POL001', policyName: 'LifePlus Term Insurance', duration: '240', amount: '₹500,000' },
    { policyNo: 'POL002', policyName: 'LifePlus Whole Life Insurance', duration: '360', amount: '₹750,000' },
    { policyNo: 'POL003', policyName: 'LifePlus Universal Insurance', duration: '180', amount: '₹400,000' },
    { policyNo: 'POL004', policyName: 'LifePlus Variable Life Insurance', duration: '300', amount: '₹600,000' },
    { policyNo: 'POL005', policyName: 'LifePlus Final Expense Insurance', duration: '120', amount: '₹1000,000' },
    { policyNo: 'POL005', policyName: 'LifePlus Retirement Plan', duration: '480', amount: '₹100,000' },
    { policyNo: 'POL005', policyName: 'LifePlus Child Education Plan', duration: '180', amount: '₹200,000' },
  ]);

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setNewPolicy({ ...newPolicy, [name]: value });
  };

  const handleAddPolicy = () => {
    const newPolicyNo = `POL${(policies.length + 1).toString().padStart(3, '0')}`;
    setPolicies([...policies, { policyNo: newPolicyNo, ...newPolicy }]);
    toast.success(`Policy ${newPolicyNo} added successfully`);
    setOpen(false);
```

```
    setNewPolicy({ policyName: '', duration: '', amount: '' });
  };

  const handleDeletePolicy = (policy) => {
    if (window.confirm(`Are you sure you want to delete the policy
${policy.policyName}?`)) {
      setPolicies(policies.filter(p => p.policyNo !== policy.policyNo));
      toast.error(`Policy ${policy.policyName} successfully deleted`);
    }
  };

  const openEditSheet = (policy) => {
    setEditMode(true);
    setSelectedPolicy(policy);
    setNewPolicy({
      policyName: policy.policyName,
      duration: policy.duration,
      amount: policy.amount,
    });
    setOpen(true);
  };

  const handleEditPolicy = () => {
    setPolicies(policies.map(policy =>
      policy.policyNo === selectedPolicy.policyNo ? { ...selectedPolicy, ...newPolicy } :
policy
    ));
    toast.success(`Policy ${selectedPolicy.policyName} successfully updated`);
    setOpen(false);
    setEditMode(false);
```

```
  setNewPolicy({ policyName: '', duration: '', amount: '' });
};

return (
  <div className="p-6 space-y-6">
    <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
      <DashboardCard
        title="Total Users"
        value="120"
        icon={<BarChart className="h-8 w-8 text-primary" />}
      />
      <DashboardCard
        title="Total Policies"
        value="5"
        icon={<User2 className="h-8 w-8 text-primary" />}
      />
      <DashboardCard
        title="Pending Claims"
        value="24"
        icon={<FileText className="h-8 w-8 text-primary" />}
      />
    </div>
    <Card>
      <CardHeader className='w-full flex flex-row justify-between items-center'>
        <CardTitle>User Policies</CardTitle>
        <Button onClick={() => {
          setEditMode(false);
          setOpen(true);
        }}>
          <Plus className='h-5 w-5 mr-2' /> Add Policy
```

```
          </Button>
        </CardHeader>
        <CardContent>
          <Table>
            <TableHeader>
              <TableRow>
                <TableHead className="w-[100px]">Policy No</TableHead>
                <TableHead>Policy Name</TableHead>
                <TableHead>Duration (Months)</TableHead>
                <TableHead className="text-right">Amount</TableHead>
                <TableHead className="text-right">Actions</TableHead>
              </TableRow>
            </TableHeader>
            <TableBody>
              {policies.map((policy) => (
                <TableRow key={policy.policyNo}>
                  <TableCell className="font-medium">{policy.policyNo}</TableCell>
                  <TableCell>{policy.policyName}</TableCell>
                  <TableCell>{policy.duration}</TableCell>
                  <TableCell className="text-right">{policy.amount}</TableCell>
                  <TableCell className="text-right">
                    {/* <Button variant="ghost" size="icon" className="mr-2" onClick={() =>
openEditSheet(policy)}>
                      <Edit className="h-4 w-4 text-blue-500" />
                    </Button> */}
                    <Button variant="ghost" size="icon" className="text-red-600" onClick={()
=> handleDeletePolicy(policy)}>
                      <Trash2 className="h-4 w-4" />
                    </Button>
                  </TableCell>
```

```
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </CardContent>
  </Card>

  <Sheet open={open} onOpenChange={() => setOpen(false)}>
    <SheetContent>
      <SheetHeader>
        <SheetTitle>{editMode ? "Edit Policy" : "Add New Policy"}</SheetTitle>
        <SheetDescription>
          {editMode ? "Update the details below to edit the policy." : "Fill out the details
below to add a new policy."}
        </SheetDescription>
      </SheetHeader>
      <div className="grid gap-4 py-4">
        <div className="grid grid-cols-4 items-center gap-4">
          <Label htmlFor="policyName" className="text-right">
            Policy Name
          </Label>
          <Input
            id="policyName"
            name="policyName"
            value={newPolicy.policyName}
            onChange={handleInputChange}
            className="col-span-3"
          />
        </div>
        <div className="grid grid-cols-4 items-center gap-4">
```

```
            <Label htmlFor="duration" className="text-right">
              Duration (Months)
            </Label>
            <Input
              id="duration"
              name="duration"
              value={newPolicy.duration}
              onChange={handleInputChange}
              className="col-span-3"
            />
          </div>
          <div className="grid grid-cols-4 items-center gap-4">
            <Label htmlFor="amount" className="text-right">
              Amount
            </Label>
            <Input
              id="amount"
              name="amount"
              value={newPolicy.amount}
              onChange={handleInputChange}
              className="col-span-3"
            />
          </div>
        </div>
        <SheetFooter className='flex flex-col flex-1'>
          <Button className='w-1/2 bg-destructive hover:bg-destructive/80' onClick={()
=> setOpen(false)}>Cancel</Button>
          <Button type="button" className='w-1/2' onClick={editMode ?
handleEditPolicy : handleAddPolicy}>
            {editMode ? "Edit" : "Submit"}
```

```
            </Button>
          </SheetFooter>
        </SheetContent>
      </Sheet>

      <ToastContainer />
    </div>
  );
};

const DashboardCard = ({ title, value, icon }) => (
  <Card className="bg-gradient-to-r from-primary to-secondary text-white shadow-lg">
    <CardHeader className="flex flex-row justify-between items-center">
      <CardTitle className="text-xl font-bold">{title}</CardTitle>
      {icon}
    </CardHeader>
    <CardContent>
      <div className="text-3xl font-extrabold">{value}</div>
    </CardContent>
  </Card>
);

export default AdminDashboard;
```

CHAPTER 6

CONCLUSION

This chapter tells about the conclusion that anyone can drive from the project and the learning we learnt by taking over this project.

6.1 CONCLUSION

In conclusion, the proposed Life Plus Insurance Management System is designed to streamline policy management and claims processing, enhancing interactions between users and administrators. This scalable system meets the needs of both small agencies and large insurance companies, simplifying policy applications, expediting claims processing, and ensuring data security. With real-time updates and adaptability, Life Plus improves operational efficiency, reduces processing times, and enhances user satisfaction. By providing a user-friendly interface and comprehensive management tools, the system fosters trust, builds strong relationships with policyholders, and ensures a reliable and responsive insurance service for all stakeholders.

6.2 FUTURE SCOPE

Integration of AI and Machine Learning: Implementing AI algorithms to predict policy demand, optimize premium pricing, and provide personalized policy recommendations based on user data and behavior.

Enhanced Security Measures: Continual improvements in security protocols to safeguard sensitive policyholder data and prevent unauthorized access or breaches, ensuring the highest standards of data protection.

Mobile Application Development: Creating a mobile app version of the system to offer greater accessibility and convenience for users to manage their policies, submit claims, and receive notifications on-the-go.

Analytics and Reporting: Developing robust analytics tools to provide insights into policy trends, claims data, and customer feedback. This data will be valuable for insurers to refine their offerings and strategies.

Integration with Financial Management Systems: Seamlessly integrating with existing financial management and CRM platforms to streamline operations, sync policyholder information, and enhance customer service.

Adaptive Risk Assessment Models: Implementing adaptive risk assessment models that adjust based on the policyholder's lifestyle changes and behavior, providing more accurate and dynamic policy adjustments.

Collaborations with Healthcare Providers: Partnering with healthcare providers and organizations to offer integrated health and life insurance packages, providing comprehensive coverage and benefits.

Feedback Mechanisms: Incorporating feedback mechanisms for policyholders to share their experiences, suggestions, and concerns, facilitating continuous improvement and customer satisfaction.

REFERENCES : -

- https://github.com/coderomm/LIC
- https://github.com/JoaquinAuza/DetLifeInsurance
- https://github.com/Gianatmaja/Life-Insurance-Premium-Calculator
- https://docs.spring.io/spring-security/site/docs/current/reference/html5/
- https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.security