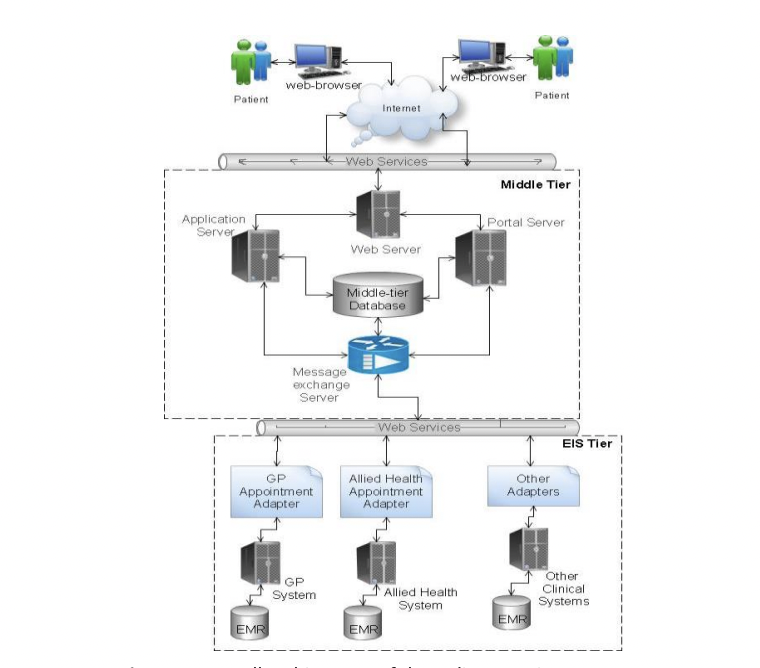
**Requirement Gathering and Analysis Phase**

**Technology Stack (Architecture & Stack)**

|  |  |
| --- | --- |
| Date | 06 July 2024 |
| Team ID | SWTID1720010842 |
| Project Name | Book a Doctor |
| Maximum Marks | 3 Marks |

**Technical Architecture:**



**Table-1 : Components & Technologies:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Component** | **Description** | **Technology** |
|  | User Interface | How user interacts with application e.g.  Web UI, Mobile App,etc. | HTML, CSS, JavaScript, React.js |
|  | Application Logic-1 | Logic for handling user registration, authentication, and authorization | Node.js, Express.js |
|  | Application Logic-2 | Logic for managing doctor availability and appointment scheduling | Node.js, Express.js |
|  | Application Logic-3 | Logic for appointment notifications and reminders | Node.js, Express.js |
|  | Database | Data storage for users, appointments, and doctor information | MongoDB (NoSQL) |
|  | Cloud Database | Managed cloud database service for scalable and secure data storage | MongoDB Atlas |
|  | File Storage | Storage solution for user-uploaded files such as profile pictures and documents | AWS S3, Local Filesystem |
|  | External API-1 | API used for providing location-based services such as finding nearby doctors | Google Maps API |
|  | External API-2 | API used for sending SMS notifications for appointment reminders | Twilio API |
|  | Machine Learning Model | Not applicable for basic functionality, could be added for predictive analytics in the future | No specific ML model required |
|  | Infrastructure (Server / Cloud) | Deployment environment for the application | Local: Node.js server,  Cloud: AWS EC2, AWS Elastic Beanstalk, Kubernetes |

**Table-2: Application Characteristics:**

| **S.No** | **Characteristics** | **Description** | **Technology** |
| --- | --- | --- | --- |
|  | Open-Source Frameworks | List the open-source frameworks used | MongoDB, Express.js, React, Node.js |
|  | Security Implementations | List all the security/access controls implemented, use of firewalls, etc. | JWT (JSON Web Tokens), bcrypt, HTTPS, OWASP, Helmet, CORS |
|  | Scalable Architecture | Justify the scalability of architecture (3-tier, Micro-services) | Microservices architecture, Docker, Kubernetes |
|  | Availability | Justify the availability of application (e.g., use of load balancers, distributed servers, etc.) | AWS EC2, AWS Elastic Load Balancing (ELB), Auto Scaling Groups |
|  | Performance | Design considerations for the performance of the application (number of requests per sec, use of Cache, use of CDN’s) | Redis (for caching), CloudFront (CDN), Nginx (reverse proxy), MongoDB Atlas (performance optimization) |