Software Tech Stack and Implementation

Part- I

1. Creating a PWA for Frontend and Hosting it on AWS

Progressive Web App (PWA) Creation:

- **PWA Basics:** A PWA is a web app that delivers a native app-like experience using modern web technologies. Key features include offline capabilities, push notifications, and the ability to be installed on mobile devices.
- **Key Technologies:** Use standard web technologies such as **HTML, CSS, JavaScript**, along with **Service Workers** for caching and offline support, and **Web App Manifests** for installation features.
- **Frameworks:** You can build the PWA using frameworks like **React.js**, **Vue.js**, or **Angular**. These frameworks support dynamic UI, and service workers can be added using libraries like **Workbox**.

Hosting on AWS:

- **AWS S3 + CloudFront:** Once the PWA is built, host the static files (HTML, CSS, JS) on **Amazon S3**. Use **Amazon CloudFront** as a CDN to speed up content delivery.
    - Set up an S3 bucket for your web files and configure it for static website hosting.
    - Use CloudFront for global content distribution, SSL, and caching.
- **Lambda@Edge (Optional):** You can use **Lambda@Edge** to customize responses in real-time, such as geo-blocking or A/B testing.

Deployment Pipeline:

- Use **AWS CodePipeline** or **AWS Amplify** for continuous integration and continuous deployment (CI/CD). Every time you push code to your repository, AWS can automatically deploy your updates to S3.

2. Connecting Frontend and Smart Yoga Mat via Backend

Communication Framework:

- **RESTful APIs:** Develop a backend using AWS services (like **API Gateway** and **Lambda**) that provides RESTful APIs. These APIs will serve as endpoints for the frontend to communicate with the backend for data processing.

- **WebSockets/IoT Core:** For real-time communication (e.g., sensor data from the mat), use **AWS IoT Core** or **WebSocket APIs** for bidirectional data flow. This will allow the smart mat to send data directly to the backend for processing in real-time.

Flow of Communication:

1. **Data from the Mat (Sensors):** The smart yoga mat will have pressure sensors and capacitive sensors that collect 2D pressure and capacitive maps.
2. **Backend Processing:** The sensor data is sent to the backend via IoT Core or REST APIs.
3. **Frontend Display:** The frontend (PWA) will query the backend at intervals for processed data and updates, such as posture tracking and recommendations.
4. **Data Storage:** Use **DynamoDB** or **RDS** for storing user sessions, sensor data, and activity history.

3. AI for Activity Tracking and Recommendations

AI-Powered Tracking:

- **Pose Estimation:** Use AI algorithms (such as **OpenPose** or **PoseNet**) to detect and track the user's body posture during yoga sessions.
- **Activity Recommendations:** AI models trained on various yoga datasets can offer recommendations based on posture detection, current fitness level, and previous sessions. For example, the AI can suggest which asanas to practice more often for flexibility improvement.

AI Framework:

- **TensorFlow.js:** Run AI models directly in the browser for real-time tracking and recommendations. Models can be pre-trained using TensorFlow/Keras in the backend, and then deployed to the frontend.
- **Backend AI Models:** Use **Amazon SageMaker** for training and deploying machine learning models. SageMaker can be integrated with Lambda for AI-powered recommendations during the yoga session.

4. ML for Creating Postures Using 2D Maps

ML Integration for Posture Analysis:

- **Data Collection (Pressure & Capacitive Maps):** The 2D pressure map (from pressure sensors) and capacitive map (from capacitive sensors) will serve as input data.

- **Model Training:** Train a **Convolutional Neural Network (CNN)** model using historical yoga posture data (2D maps) and the user's height to predict the correct posture. Use datasets from real yoga sessions for training the model.
- **Auto-Completion of Posture:** If some sensors provide incomplete data, a **Generative Model** (like a **Variational Autoencoder** or **Generative Adversarial Network (GAN)**) can fill in the missing sensor data by generating a full 2D map based on learned patterns.
- **Model Output:** The CNN outputs a 2D posture map for the correct pose. This can be used to provide visual feedback and recommendations on improving posture.

Workflow:

1. **Input:** 2D sensor data (pressure + capacitive) and height.
2. **Processing:** CNN processes the combined map, possibly auto-completing data.
3. **Output:** Real-time visualization and feedback on posture improvements.

5. Health Metrics from Smartwatch via Bluetooth

Bluetooth Connectivity:

- **Bluetooth Low Energy (BLE):** ESP32 can connect with smartwatches using **BLE**. It will act as a central device that collects health data such as heart rate, calorie expenditure, and workout intensity from the watch.
- **BLE GATT Profiles:** Use **GATT (Generic Attribute Profile)** to retrieve data from the smartwatch. ESP32 will send this data to the backend for further processing or visualization on the PWA frontend.

Data Flow:

- The smartwatch measures heart rate and calorie burn data.
- **ESP32** (embedded in the yoga mat) fetches this data via BLE and forwards it to the backend.
- The backend processes the health metrics and integrates them with other activity data (e.g., yoga posture tracking) to offer comprehensive session reports.

Summary of the Technology Stack:

1. **Frontend (PWA):**

- Built using React.js or Vue.js, with Service Workers and Web App Manifest for PWA features.
- Hosted on AWS S3 with CloudFront for fast, secure distribution.

2. **Backend:**
   - APIs via AWS Lambda, API Gateway, and AWS IoT Core for communication with the yoga mat.
   - AI and ML models running on SageMaker for activity tracking, recommendations, and posture correction.
3. **Smart Mat and Sensors:**
   - Sensor data processed through ESP32, which sends the 2D pressure and capacitive maps to the backend for analysis.
4. **AI/ML:**
   - CNN and Generative models to create perfect yoga postures based on sensor data, height, and user-specific datasets.
5. **Bluetooth Integration:**
   - ESP32 connects to smartwatches via BLE, collecting health data and transmitting it to the backend for analysis and integration into user performance reports.

Part – II

1. Creating a Progressive Web App (PWA) and Hosting it on AWS

PWA Development:

- **Service Workers:** These allow the PWA to function offline by caching assets like HTML, CSS, JS, and images. By intercepting network requests, service workers deliver cached resources when a user is offline, ensuring the app works seamlessly even with no internet connection.
- **Web App Manifest:** A JSON file that defines the app's metadata, such as the name, theme color, icons, and display mode. This enables the app to be installable on a user's device, giving it an app-like experience.
- **Push Notifications:** Notifications can be sent through the **Notification API** and **Push API**. These features allow real-time updates like reminders for yoga sessions, progress reports, or suggestions based on previous sessions.

Hosting on AWS:

- **Amazon S3:** The PWA's static assets are hosted on S3. Configure S3 buckets to support **static website hosting**. Your PWA files (HTML, CSS, JS) are stored here and accessed by users via a web browser.
- **Amazon CloudFront:** Acts as a CDN to distribute your static content globally with low latency. CloudFront provides SSL certificates for HTTPS traffic, enhancing security and performance.
- **AWS Amplify:** Amplify provides a simple way to connect the PWA to AWS backend resources like Lambda functions, APIs, and databases. It also automates CI/CD pipelines, enabling faster iterations and updates.

2. Connecting Frontend and Smart Yoga Mat through Backend

Backend Architecture:

- **AWS IoT Core:** Use **AWS IoT Core** to manage and communicate with your yoga mat's sensors. IoT Core handles the connection of ESP32 (embedded in the mat) to the AWS cloud securely. It can receive sensor data (pressure, capacitance) and forward it to processing services.
- **API Gateway and Lambda:** If you prefer HTTP-based communication, **API Gateway** will serve as the bridge between the frontend PWA and backend services. **AWS Lambda** functions handle requests and responses. Lambda functions are stateless, serverless compute services that process sensor data from the mat.
- **WebSocket API:** For real-time feedback (e.g., correcting a yoga posture while the user is still in the pose), **AWS WebSocket APIs** offer bidirectional communication between the yoga mat and PWA, ensuring low-latency updates.

3. AI for Activity Tracking and Recommendations

Pose Estimation and Correction:

- **AI Libraries:** The AI uses frameworks like **PoseNet** or **TensorFlow.js** to detect yoga poses from the user's body movements. This can be run on the client side (browser) for real-time feedback or on the server side (backend) for heavier computations.
- **Model Training:** AI models trained on various yoga poses can compare the detected pose with an ideal pose dataset. Recommendations for improvements (e.g., adjusting posture or alignment) are then generated and fed back to the user.

- **Real-time Feedback:** Use WebSocket for real-time feedback, ensuring the user receives suggestions while performing the yoga pose. For more complex pose analysis, the data is processed on the backend and then sent back to the PWA.

AI Framework:

- **TensorFlow/Keras for Model Training:** Models can be trained using yoga datasets to recognize poses. After training, models can be deployed using **Amazon SageMaker** or directly on the PWA using **TensorFlow.js** for real-time predictions.
- **Data Analytics for Recommendations:** AI-powered recommendations can be based on past user performance, current posture detection, and goals (e.g., flexibility, balance). The AI analyzes trends over multiple sessions to provide personalized yoga routines.

4. ML for Posture Creation Using Sensor Data

Pressure & Capacitive Sensor Data Integration:

- **2D Map Creation:** The mat uses a grid of pressure and capacitive sensors to create two separate 2D maps representing where the user's body is pressing on the mat and where capacitive changes are detected.
- **Model Training:** A **Convolutional Neural Network (CNN)** is trained on a dataset of yoga postures, using pressure and capacitive maps as input. The CNN learns the relationship between sensor patterns and specific yoga poses.
- **Generative Model for Data Filling:** When sensor data is incomplete or noisy, a **Generative Adversarial Network (GAN)** or **Variational Autoencoder (VAE)** can auto-complete the map by filling in missing sections based on known patterns.

Workflow:

1. **Sensor Data Collection:** ESP32 gathers 2D pressure and capacitive data.
2. **ML Processing:** The combined data is fed into a CNN to determine which yoga pose is being performed.
3. **Pose Completion:** In case of missing sensor data, generative models can fill in the gaps to create a more accurate posture representation.
4. **Feedback:** The model outputs recommendations or pose corrections back to the PWA in real-time.

5. Bluetooth Communication for Health Metrics

Bluetooth Integration:

- **ESP32 with BLE:** ESP32 will act as a central device that connects to BLE devices (like a smartwatch) to collect health data (heart rate, calorie burn, etc.).
- **BLE Protocols:** Use **GATT (Generic Attribute Profile)** to access the smartwatch's health metrics. For instance, the ESP32 can read the heart rate characteristic from the watch and transmit this data to the backend.

Data Flow:

1. **Smartwatch Data Collection:** The smartwatch collects heart rate, calories, and other metrics.
2. **ESP32 BLE Communication:** The ESP32 receives this data via BLE and forwards it to the backend.
3. **Backend Processing:** The backend integrates this data with yoga session metrics (e.g., pose detection) to generate a comprehensive activity report.

Specific Technologies and Tools Overview:

- **PWA Development:** React.js/Vue.js, Service Workers, Web App Manifest.
- **AWS Hosting:** S3, CloudFront, Amplify, API Gateway, Lambda, IoT Core.
- **AI/ML:** TensorFlow.js, PoseNet, CNNs, GANs, SageMaker.
- **Bluetooth:** ESP32 with BLE, GATT Profiles.
- **Data Handling:** DynamoDB or RDS for session, sensor data, and user analytics.