

The Shrey Method Fitness Platform

Implementation Guide & UI Specifications

Prepared for: Shreyas Annapureddy
Date: July 27, 2025
Version: 1.0

Table of Contents

- 1. [Introduction](#)
 - 2. [Project Overview](#)
 - 3. [Phase 1: Authentication & Dashboard Inbox](#)
 - 4. [Phase 2: Basic Client & Coach Dashboards](#)
 - 5. [Phase 3: Payment Processing](#)
 - 6. [Phase 4: Calendly Integration](#)
 - 7. [Cost Estimates](#)
 - 8. [Future Expansion Considerations](#)
 - 9. [Technical Resources](#)
-

Introduction

This document provides a comprehensive implementation guide for The Shrey Method Fitness Platform. It outlines a phased approach to development, with detailed technical specifications, UI mockups, and step-by-step implementation instructions for each phase.

The platform will transform your current static website into a full-featured coaching platform with client management, payment processing, scheduling, and content delivery capabilities.

Project Overview

Core Functionality

The Shrey Method Fitness Platform will enable:

- 1. **Client Management:** Onboarding, tracking, and managing fitness clients
- 2. **Content Delivery:** Providing personalized workout and nutrition plans
- 3. **Payment Processing:** Handling one-time and recurring payments
- 4. **Scheduling:** Booking and managing coaching sessions
- 5. **Communication:** Messaging between coach and clients

Technology Stack

- **Frontend:** HTML, CSS, JavaScript (existing website)
- **Backend:** AWS Amplify, AWS Lambda, API Gateway

- **Authentication:** AWS Cognito
- **Database:** DynamoDB
- **File Storage:** Amazon S3
- **Payment Processing:** Stripe
- **Scheduling:** Calendly (paid plan)

Implementation Approach

The platform will be built in four phases, each adding specific functionality while maintaining a working system throughout the development process.

Phase 1: Authentication & Dashboard Inbox

Overview

Phase 1 establishes the foundation of the platform by implementing user authentication and a basic messaging system. This creates immediate value by allowing clients to register and communicate with you through a centralized inbox.

UI Mockups

Client Login Page

THE SHREY METHOD FITNESS

Log In to Your Dashboard

Email: []

Password: []

[Forgot Password?] [LOG IN] button

New client? [Sign Up]

Coach Dashboard - Inbox View

THE SHREY METHOD FITNESS

[Shreyas ▼]

```
| [Dashboard]
| [Clients]          +-----+
| [★ Messages] (12 new) | Search: [  ] |
| [Programs]          +-----+
| [Settings]
|
| MESSAGES              Sort by: [Date ▼] [Filter]
|
| ● John Doe - New Client Inquiry      2 hrs ago
| "I'm interested in your 8-week program..."
|
| ● Sarah Smith - Question about diet   5 hrs ago
| "Should I be taking protein supplements..."
|
| ○ Michael Johnson - Payment Issue     Yesterday
| "I think I was charged twice for my..."
|
| ○ Lisa Wang - Program Feedback        07/24/25
| "I've completed week 3 and wanted to..."
|
| [Load More Messages]
|
+-----+
```

Step-by-Step Implementation

1. Set Up AWS Amplify Project

```
# Install Amplify CLI
npm install -g @aws-amplify/cli

# Configure Amplify
amplify configure

# Initialize Amplify in your project directory
cd /path/to/ShreyasFitnessWeb
amplify init
```

Configuration Details:

- Project name: ShreyMethodFitness
- Environment: dev
- Default editor: Visual Studio Code
- Type of app: JavaScript
- JavaScript framework: None
- Source directory: /
- Distribution directory: /
- Build command: npm run build
- Start command: npm start

2. Add Authentication

```
# Add authentication
amplify add auth

# Push changes to AWS
amplify push
```

Configuration Details:

- Authentication service: Cognito
- Default authentication flow: Email & Password
- Configure advanced settings: Yes
- User attributes: Name, Email, Phone Number
- Admin queries API: Yes
- Triggers for Lambda functions: Post Confirmation

3. Create User Roles

1. Navigate to AWS Cognito Console
2. Select your User Pool
3. Go to "Groups" and create two groups:
 - "clients"
 - "coaches"
4. Add your admin account to the "coaches" group

4. Set Up Database for Messages

```
# Add API and database
amplify add api
```

Configuration Details:

- Service: GraphQL
- API name: shreymethodapi
- Authorization type: Amazon Cognito User Pool
- Configure additional auth types: No
- Schema template: Single object with fields

Create the following schema in `amplify/backend/api/shreymethodapi/schema.graphql`:

```
type Message @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [create, read, update, delete]
},
```

```

    { allow: owner, operations: [create, read] }
  ]) {
    id: ID!
    senderName: String!
    senderEmail: String!
    subject: String!
    content: String!
    read: Boolean!
    archived: Boolean!
    createdAt: AWSDatetime!
  }

type User @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [read, update] },
  { allow: owner, operations: [read, update] }
]) {
  id: ID!
  name: String!
  email: String!
  phone: String
  userGroup: String!
  createdAt: AWSDatetime!
}

```

```

# Push changes to AWS
amplify push

```

5. Create Authentication UI Components

1. Create login page (`client-login.html`):

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <!-- Add AWS Amplify libraries -->
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
</head>
<body>
  <!-- Navigation bar (reuse existing) -->

  <div class="auth-container">
    <div class="auth-form">
      <h2>Log In to Your Dashboard</h2>

```

```

        <div id="error-message" class="error-message"></div>

        <form id="login-form">
            <div class="form-group">
                <label for="email">Email</label>
                <input type="email" id="email" required>
            </div>

            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" required>
            </div>

            <div class="form-actions">
                <a href="forgot-password.html" class="forgot-password">Forgot
Password?</a>
                <button type="submit" class="btn-primary">Log In</button>
            </div>
        </form>

        <p class="auth-redirect">New client? <a href="signup.html">Sign Up</a>
</p>
    </div>
</div>

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

```

2. Create signup page (`signup.html`) - similar structure to login page

3. Create authentication JavaScript (`js/auth.js`):

```

// Configure Amplify
const awsConfig = {
    Auth: {
        region: 'us-west-2', // Your AWS region
        userPoolId: 'us-west-2_xxxxxxxx', // Your Cognito User Pool ID
        userPoolWebClientId: 'xxxxxxxxxxxxxxxxxxxxxxxx', // Your App Client ID
    }
};

Amplify.configure(awsConfig);

// Login form handler
document.getElementById('login-form').addEventListener('submit', async
function(event) {
    event.preventDefault();

    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;
    const errorMessage = document.getElementById('error-message');

```

```

    try {
      const user = await Amplify.Auth.signIn(email, password);

      // Check user group and redirect accordingly
      const session = await Amplify.Auth.currentSession();
      const idToken = session.getIdToken().payload;

      if (idToken['cognito:groups'] &&
idToken['cognito:groups'].includes('coaches')) {
        window.location.href = 'coach-dashboard.html';
      } else {
        window.location.href = 'client-dashboard.html';
      }
    } catch (error) {
      errorMessage.textContent = error.message;
    }
  });
});

```

6. Create Coach Dashboard with Inbox

1. Create coach dashboard page (`coach-dashboard.html`):

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Coach Dashboard - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation -->
    <div class="dashboard-sidebar">
      <div class="logo">
        <h2>The Shrey Method</h2>
      </div>
      <nav class="dashboard-nav">
        <ul>
          <li><a href="coach-dashboard.html">Dashboard</a></li>
          <li><a href="coach-clients.html">Clients</a></li>
          <li class="active"><a href="coach-messages.html">Messages
<span class="badge" id="unread-count">0</span></a></li>
          <li><a href="coach-programs.html">Programs</a></li>
          <li><a href="coach-settings.html">Settings</a></li>
        </ul>
      </nav>
    </div>
  </div>

```

```

        <div class="sidebar-footer">
            <button id="logout-btn" class="btn-secondary">Log Out</button>
        </div>
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
        <header class="dashboard-header">
            <h1>Messages</h1>
            <div class="user-menu">
                <span id="user-name">Shreyas</span>
                
            </div>
        </header>

        <div class="content-container">
            <div class="toolbar">
                <div class="search-box">
                    <input type="text" id="search-messages"
placeholder="Search messages...">
                </div>
                <div class="filters">
                    <select id="sort-by">
                        <option value="date-desc">Newest First</option>
                        <option value="date-asc">Oldest First</option>
                        <option value="unread">Unread First</option>
                    </select>
                    <button id="filter-btn" class="btn-
secondary">Filter</button>
                </div>
            </div>

            <div class="messages-list" id="messages-container">
                <!-- Messages will be loaded here dynamically -->
                <div class="loading">Loading messages...</div>
            </div>

            <div class="pagination">
                <button id="load-more" class="btn-secondary">Load More
Messages</button>
            </div>
        </div>
    </div>

    <script src="js/coach-messages.js"></script>
</body>
</html>

```

2. Create message handling JavaScript (`js/coach-messages.js`):


```

// Configure Amplify (same as auth.js)
const awsConfig = {
  Auth: {
    region: 'us-west-2',
    userPoolId: 'us-west-2_XXXXXXX',
    userPoolWebClientId: 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX',
  },
  API: {
    endpoints: [{
      name: "shreymethodapi",
      endpoint: "https://XXXXXXXXXX.execute-api.us-west-2.amazonaws.com/dev"
    }]
  }
};

Amplify.configure(awsConfig);

// Check authentication
async function checkAuth() {
  try {
    await Amplify.Auth.currentAuthenticatedUser();
  } catch (error) {
    window.location.href = 'client-login.html';
  }
}

// Fetch messages
async function fetchMessages(nextToken = null) {
  const messagesContainer = document.getElementById('messages-container');
  const loadMoreBtn = document.getElementById('load-more');

  if (!nextToken) {
    messagesContainer.innerHTML = '<div class="loading">Loading messages...
</div>';
  }

  try {
    const query = `query ListMessages($limit: Int, $nextToken: String) {
      listMessages(limit: $limit, nextToken: $nextToken) {
        items {
          id
          senderName
          senderEmail
          subject
          content
          read
          createdAt
        }
        nextToken
      }
    }`;

    const variables = {

```

```
        limit: 10,
        nextToken: nextToken
    };

    const response = await Amplify.API.graphql({
        query: query,
        variables: variables
    });

    const messages = response.data.listMessages.items;
    const newNextToken = response.data.listMessages.nextToken;

    if (!nextToken) {
        messagesContainer.innerHTML = '';
    }

    // Update unread count
    const unreadCount = messages.filter(msg => !msg.read).length;
    document.getElementById('unread-count').textContent = unreadCount > 0 ?
unreadCount : '';

    // Render messages
    messages.forEach(message => {
        const messageDate = new Date(message.createdAt);
        const formattedDate = formatDate(messageDate);

        const messageEl = document.createElement('div');
        messageEl.className = `message-item ${message.read ? '' : 'unread'}`;
        messageEl.dataset.id = message.id;

        messageEl.innerHTML = `
            <div class="message-header">
                <div class="message-sender">
                    <span class="status-indicator"></span>
                    <strong>${message.senderName}</strong> -
${message.subject}
                </div>
                <div class="message-date">${formattedDate}</div>
            </div>
            <div class="message-preview">${message.content.substring(0,
100)}${message.content.length > 100 ? '...' : ''}</div>
        `;

        messageEl.addEventListener('click', () => viewMessage(message.id));
        messagesContainer.appendChild(messageEl);
    });

    // Handle "Load More" button
    if (newNextToken) {
        loadMoreBtn.style.display = 'block';
        loadMoreBtn.onclick = () => fetchMessages(newNextToken);
    } else {
        loadMoreBtn.style.display = 'none';
    }
}
```

```

    } catch (error) {
      messagesContainer.innerHTML = `

## 7. Modify Contact Form to Store Messages



1. Update the existing contact form in connect.html to send messages to the database:



11 / 64


```

```
<!-- Existing contact form HTML -->
<form id="contact-form" class="contact-form">
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" id="name" name="name" required>
  </div>

  <div class="form-group">
    <label for="email">Email</label>
    <input type="email" id="email" name="email" required>
  </div>

  <div class="form-group">
    <label for="subject">Subject</label>
    <input type="text" id="subject" name="subject" required>
  </div>

  <div class="form-group">
    <label for="message">Message</label>
    <textarea id="message" name="message" rows="5" required></textarea>
  </div>

  <button type="submit" class="btn-primary">Send Message</button>
</form>
```

2. Create JavaScript to handle form submission (`js/contact-form.js`):

```
// Configure Amplify
const awsConfig = {
  Auth: {
    region: 'us-west-2',
    userPoolId: 'us-west-2_xxxxxxxx',
    userPoolWebClientId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  },
  API: {
    endpoints: [{
      name: 'shreymethodapi',
      endpoint: 'https://xxxxxxxx.execute-api.us-west-2.amazonaws.com/dev'
    }]
  }
};

Amplify.configure(awsConfig);

// Handle form submission
document.getElementById('contact-form').addEventListener('submit', async
function(event) {
  event.preventDefault();

  const submitButton = this.querySelector('button[type="submit"]');
  const originalText = submitButton.textContent;
```

```

submitButton.textContent = 'Sending...';
submitButton.disabled = true;

const name = document.getElementById('name').value;
const email = document.getElementById('email').value;
const subject = document.getElementById('subject').value;
const content = document.getElementById('message').value;

try {
  // Create message in database
  const mutation = `mutation CreateMessage($input: CreateMessageInput!) {
    createMessage(input: $input) {
      id
    }
  }`;

  const variables = {
    input: {
      senderName: name,
      senderEmail: email,
      subject: subject,
      content: content,
      read: false,
      archived: false,
      createdAt: new Date().toISOString()
    }
  };

  await Amplify.API.graphql({
    query: mutation,
    variables: variables
  });

  // Show success message
  this.reset();
  alert('Your message has been sent successfully!');

} catch (error) {
  console.error('Error sending message:', error);
  alert('There was an error sending your message. Please try again later.');
```

```

} finally {
  submitButton.textContent = originalText;
  submitButton.disabled = false;
}
});

```

3. Add the script to `connect.html`:

```

<script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
<script src="js/contact-form.js"></script>

```

8. Testing Phase 1

1. Authentication Testing:

- Test user registration
- Test login for both client and coach accounts
- Test password reset functionality
- Verify proper redirection based on user role

2. Message System Testing:

- Submit test messages through the contact form
- Verify messages appear in coach dashboard
- Test message sorting and filtering
- Verify unread message count updates correctly

3. Security Testing:

- Verify clients cannot access coach dashboard
- Verify unauthenticated users cannot access protected pages
- Test API permissions to ensure proper access control

9. Deployment

```
# Deploy all Amplify resources
amplify publish
```

Update your AWS Amplify hosting settings to handle redirects for single-page application behavior.

Phase 2: Basic Client & Coach Dashboards

Overview

Phase 2 expands the platform with basic dashboard functionality for both clients and coaches. This includes client management for coaches and program access for clients.

UI Mockups

Client Dashboard - Main View

+-----+
| THE SHREY METHOD FITNESS [John ▼] |
+-----+
|
| [Overview] |
| [My Program] |

[Nutrition]

[Messages]

[Payments]

[Schedule]

QUICK ACCESS

[View Today's Workout]

[Log Nutrition]

[Message Coach]

Welcome Back, John!

Current Program:

8-WEEK STRENGTH FOUNDATION

Next Check-in: July 30, 2025

PROGRAM PROGRESS

Week 3 of 8

RECENT FILES

• Nutrition Plan July.pdf

• Week 3 Workouts.pdf

• Progress Chart.pdf

Coach Dashboard - Client View

THE SHREY METHOD FITNESS

[Shreyas ▼]

[Dashboard]

[★ Clients]

[Messages] (3 new)

[Programs]

[Settings]

[+ Add New Client]

CLIENT LIST

[Search:]

Sort by: [Name ▼] [Filter]

• John Doe

Program: Strength Foundation

Started: 06/15/25

Status: Active Week 3/8

Next check-in: 3 days

• Sarah Smith

Program: Custom Fat Loss

Started: 07/01/25

Status: Active Week 2/12

Next check-in: Tomorrow

○ Michael Johnson

Program: Athletic Performance

Started: 05/10/25

Status: Active Week 8/8

Next check-in: Today

[Load More Clients]

1. Extend Database Schema

Update the GraphQL schema to include program and client information:

```
# Edit the schema
nano amplify/backend/api/shreymethodapi/schema.graphql
```

Add the following types:

```
type Program @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [create, read, update, delete]
},
  { allow: owner, operations: [read] }
]) {
  id: ID!
  name: String!
  description: String!
  duration: Int!
  type: String!
  createdAt: AWSDateTime!
}

type ClientProgram @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [create, read, update, delete]
},
  { allow: owner, operations: [read] }
]) {
  id: ID!
  clientId: ID!
  programId: ID!
  startDate: AWSDateTime!
  currentWeek: Int!
  status: String!
  nextCheckIn: AWSDateTime
}

type ProgramContent @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [create, read, update, delete]
},
  { allow: owner, operations: [read] }
]) {
  id: ID!
  programId: ID!
  clientId: ID
  title: String!
  description: String
  fileUrl: String!
}
```



```
fileType: String!  
week: Int  
createdAt: AWSDatetime!  
}
```

```
# Push changes to AWS  
amplify push
```

2. Set Up S3 Storage for Program Files

```
# Add storage  
amplify add storage
```

Configuration Details:

- Service: Content (Images, audio, video, etc.)
- Resource name: shreymethodstorage
- Bucket name: shreymethodfitness-storage
- Authentication: Auth users only
- Access: Auth users only
- Lambda triggers: No

```
# Push changes to AWS  
amplify push
```

3. Create Client Dashboard

1. Create client dashboard page (`client-dashboard.html`):

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Client Dashboard - The Shrey Method Fitness</title>  
  <link rel="stylesheet" href="css/styles.css">  
  <link rel="stylesheet" href="css/dashboard.css">  
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-amplify.min.js"></script>  
</head>  
<body>  
  <div class="dashboard-container">  
    <!-- Sidebar navigation -->
```

```

<div class="dashboard-sidebar">
  <div class="logo">
    <h2>The Shrey Method</h2>
  </div>
  <nav class="dashboard-nav">
    <ul>
      <li class="active"><a href="client-
dashboard.html">Overview</a></li>
      <li><a href="client-program.html">My Program</a></li>
      <li><a href="client-nutrition.html">Nutrition</a></li>
      <li><a href="client-messages.html">Messages</a></li>
      <li><a href="client-payments.html">Payments</a></li>
      <li><a href="client-schedule.html">Schedule</a></li>
    </ul>
  </nav>
  <div class="sidebar-footer">
    <button id="logout-btn" class="btn-secondary">Log Out</button>
  </div>
</div>

<!-- Main content area -->
<div class="dashboard-content">
  <header class="dashboard-header">
    <h1>Welcome Back, <span id="client-name">Client</span>!</h1>
    <div class="user-menu">
      <span id="user-name"></span>
      
    </div>
  </header>

  <div class="content-container">
    <div class="dashboard-grid">
      <!-- Program Overview Card -->
      <div class="dashboard-card">
        <h2>Current Program</h2>
        <div id="program-info">
          <div class="loading">Loading program information...
</div>
        </div>
      </div>

      <!-- Progress Card -->
      <div class="dashboard-card">
        <h2>Program Progress</h2>
        <div id="progress-container">
          <div class="loading">Loading progress...</div>
        </div>
      </div>

      <!-- Quick Access Card -->
      <div class="dashboard-card">
        <h2>Quick Access</h2>
        <div class="quick-actions">

```

```

        <button id="view-workout-btn" class="btn-primary">View
Today's Workout</button>
        <button id="log-nutrition-btn" class="btn-primary">Log
Nutrition</button>
        <button id="message-coach-btn" class="btn-
primary">Message Coach</button>
    </div>
</div>

<!-- Recent Files Card -->
<div class="dashboard-card">
    <h2>Recent Files</h2>
    <div id="recent-files">
        <div class="loading">Loading recent files...</div>
    </div>
</div>
</div>
</div>
</div>

<script src="js/client-dashboard.js"></script>
</body>
</html>

```

2. Create client dashboard JavaScript (js/client-dashboard.js):

```

// Configure Amplify
const awsConfig = {
  Auth: {
    region: 'us-west-2',
    userPoolId: 'us-west-2_XXXXXXX',
    userPoolWebClientId: 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX',
  },
  API: {
    endpoints: [{
      name: "shreymethodapi",
      endpoint: "https://XXXXXXXXXX.execute-api.us-west-2.amazonaws.com/dev"
    }]
  },
  Storage: {
    AWSS3: {
      bucket: 'shreymethodfitness-storage',
      region: 'us-west-2'
    }
  }
};

Amplify.configure(awsConfig);

// Check authentication
async function checkAuth() {

```

```

    try {
      const user = await Amplify.Auth.currentAuthenticatedUser();
      return user;
    } catch (error) {
      window.location.href = 'client-login.html';
    }
  }

  // Fetch client information
  async function fetchClientInfo() {
    try {
      const user = await checkAuth();
      const clientName = document.getElementById('client-name');
      const userName = document.getElementById('user-name');

      // Set user name from Cognito attributes
      clientName.textContent = user.attributes.name || user.username;
      userName.textContent = user.attributes.name || user.username;

      // Fetch client program
      fetchClientProgram(user.username);
      fetchRecentFiles(user.username);

    } catch (error) {
      console.error('Error fetching client info:', error);
    }
  }

  // Fetch client program
  async function fetchClientProgram(clientId) {
    const programInfo = document.getElementById('program-info');
    const progressContainer = document.getElementById('progress-container');

    try {
      // Query for client program
      const query = `query GetClientProgram($clientId: ID!) {
        listClientPrograms(filter: {clientId: {eq: $clientId}}) {
          items {
            id
            programId
            startDate
            currentWeek
            status
            nextCheckIn
            program {
              name
              description
              duration
            }
          }
        }
      }`;

      const variables = {

```

```

        clientId: clientId
    };

    const response = await Amplify.API.graphql({
        query: query,
        variables: variables
    });

    const clientProgram = response.data.listClientPrograms.items[0];

    if (clientProgram) {
        // Display program info
        programInfo.innerHTML = `
            <h3>${clientProgram.program.name}</h3>
            <p>${clientProgram.program.description}</p>
            <p class="next-checkin">Next Check-in: ${new
Date(clientProgram.nextCheckIn).toLocaleDateString()}</p>
        `;

        // Display progress
        const progress = Math.round((clientProgram.currentWeek /
clientProgram.program.duration) * 100);
        progressContainer.innerHTML = `
            <div class="progress-bar">
                <div class="progress" style="width: ${progress}%></div>
            </div>
            <p>Week ${clientProgram.currentWeek} of
${clientProgram.program.duration}</p>
        `;
    } else {
        programInfo.innerHTML = '<p>No active program found. Contact your
coach to get started.</p>';
        progressContainer.innerHTML = '<p>No program progress to display.
</p>';
    }

    } catch (error) {
        programInfo.innerHTML = `<div class="error">Error loading program:
${error.message}</div>`;
        progressContainer.innerHTML = `<div class="error">Error loading progress.
</div>`;
    }
}

// Fetch recent files
async function fetchRecentFiles(clientId) {
    const recentFiles = document.getElementById('recent-files');

    try {
        // Query for program content
        const query = `query GetProgramContent($clientId: ID!) {
            listProgramContents(filter: {clientId: {eq: $clientId}}, limit: 5,
sort: {field: "createdAt", direction: "desc"}) {
                items {

```

```

        id
        title
        fileUrl
        fileType
        createdAt
      }
    }
  }
};

const variables = {
  clientId: clientId
};

const response = await Amplify.API.graphql({
  query: query,
  variables: variables
});

const files = response.data.listProgramContents.items;

if (files.length > 0) {
  const filesList = document.createElement('ul');
  filesList.className = 'files-list';

  files.forEach(file => {
    const fileItem = document.createElement('li');
    fileItem.innerHTML = `
      <a href="#" class="file-link" data-url="${file.fileUrl}">
        <i class="file-icon ${getFileIconClass(file.fileType)}">

          </i>

          ${file.title}
        </a>
      `;
    filesList.appendChild(fileItem);
  });

  recentFiles.innerHTML = '';
  recentFiles.appendChild(filesList);

  // Add event listeners to file links
  document.querySelectorAll('.file-link').forEach(link => {
    link.addEventListener('click', async (e) => {
      e.preventDefault();
      const url = e.currentTarget.dataset.url;
      const fileUrl = await Amplify.Storage.get(url);
      window.open(fileUrl, '_blank');
    });
  });

} else {
  recentFiles.innerHTML = '<p>No files available yet.</p>';
}

} catch (error) {

```

```
        recentFiles.innerHTML = `

23 / 64


```

4. Create Coach Client Management

1. Create coach clients page (`coach-clients.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Client Management - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation (same as coach-dashboard.html) -->
    <div class="dashboard-sidebar">
      <!-- ... sidebar content ... -->
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
      <header class="dashboard-header">
        <h1>Client Management</h1>
        <div class="header-actions">
          <button id="add-client-btn" class="btn-primary">+ Add New
Client</button>
        </div>
      </header>

      <div class="content-container">
        <div class="toolbar">
          <div class="search-box">
            <input type="text" id="search-clients" placeholder="Search
clients...">
          </div>
          <div class="filters">
            <select id="sort-by">
              <option value="name">Name</option>
              <option value="program">Program</option>
              <option value="status">Status</option>
              <option value="check-in">Next Check-in</option>
            </select>
            <button id="filter-btn" class="btn-
secondary">Filter</button>
          </div>
        </div>

        <div class="clients-stats">
          <div class="stat-card">
```



```

        <h3>Active Clients</h3>
        <p id="active-clients-count">0</p>
    </div>
    <div class="stat-card">
        <h3>Check-ins Today</h3>
        <p id="checkins-today-count">0</p>
    </div>
    <div class="stat-card">
        <h3>New This Month</h3>
        <p id="new-clients-count">0</p>
    </div>
</div>

<div class="clients-list" id="clients-container">
    <!-- Clients will be loaded here dynamically -->
    <div class="loading">Loading clients...</div>
</div>

<div class="pagination">
    <button id="load-more" class="btn-secondary">Load More
Clients</button>
    </div>
</div>
</div>
</div>
</div>

<!-- Add Client Modal -->
<div id="add-client-modal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2>Add New Client</h2>
        <form id="add-client-form">
            <!-- Form fields for adding a new client -->
            <!-- ... -->
        </form>
    </div>
</div>

<script src="js/coach-clients.js"></script>
</body>
</html>

```

2. Create client management JavaScript (`js/coach-clients.js`):

```

// Configure Amplify (same as other files)
const awsConfig = { /* ... */ };
Amplify.configure(awsConfig);

// Check authentication
async function checkAuth() {
    try {
        const user = await Amplify.Auth.currentAuthenticatedUser();
    }
}

```

```

    // Check if user is in coaches group
    const session = await Amplify.Auth.currentSession();
    const idToken = session.getIdToken().payload;

    if (!idToken['cognito:groups'] ||
!idToken['cognito:groups'].includes('coaches')) {
        window.location.href = 'client-login.html';
    }

    return user;
} catch (error) {
    window.location.href = 'client-login.html';
}
}

// Fetch clients
async function fetchClients(nextToken = null) {
    const clientsContainer = document.getElementById('clients-container');
    const loadMoreBtn = document.getElementById('load-more');

    if (!nextToken) {
        clientsContainer.innerHTML = '<div class="loading">Loading clients...
</div>';
    }

    try {
        // Query for all users in the clients group
        // This would typically be done through a Lambda function with admin
privileges
        // For this example, we'll assume a simplified approach

        // Fetch client programs to get program information
        const query = `query ListClientPrograms($limit: Int, $nextToken: String) {
            listClientPrograms(limit: $limit, nextToken: $nextToken) {
                items {
                    id
                    clientId
                    programId
                    startDate
                    currentWeek
                    status
                    nextCheckIn
                    program {
                        name
                        duration
                    }
                    client {
                        name
                        email
                    }
                }
            }
            nextToken
        }`;
    }
}

```

```

const variables = {
  limit: 10,
  nextToken: nextToken
};

const response = await Amplify.API.graphql({
  query: query,
  variables: variables
});

const clientPrograms = response.data.listClientPrograms.items;
const newNextToken = response.data.listClientPrograms.nextToken;

if (!nextToken) {
  clientsContainer.innerHTML = '';
}

// Update stats
updateClientStats(clientPrograms);

// Render clients
clientPrograms.forEach(clientProgram => {
  const checkInDate = new Date(clientProgram.nextCheckIn);
  const today = new Date();
  const diffDays = Math.ceil((checkInDate - today) / (1000 * 60 * 60 *
24));

  let checkInText;
  if (diffDays === 0) {
    checkInText = 'Today';
  } else if (diffDays === 1) {
    checkInText = 'Tomorrow';
  } else if (diffDays > 1) {
    checkInText = `${diffDays} days`;
  } else {
    checkInText = 'Overdue';
  }

  const clientEl = document.createElement('div');
  clientEl.className = 'client-item';
  clientEl.dataset.id = clientProgram.clientId;

  clientEl.innerHTML = `
    <div class="client-header">
      <div class="client-name">
        <span class="status-indicator
${clientProgram.status.toLowerCase()}></span>
        <strong>${clientProgram.client.name}</strong>
      </div>
      <div class="client-actions">
        <button class="btn-icon view-client" title="View Client">
          <i class="fas fa-eye"></i>
        </button>

```

```

        <button class="btn-icon edit-client" title="Edit Client">
            <i class="fas fa-edit"></i>
        </button>
    </div>
</div>
<div class="client-details">
    <div class="client-program">
        <span class="label">Program:</span>
        ${clientProgram.program.name}
    </div>
    <div class="client-progress">
        <span class="label">Status:</span> ${clientProgram.status}
        Week ${clientProgram.currentWeek}/${clientProgram.program.duration}
    </div>
    <div class="client-checkin">
        <span class="label">Next check-in:</span> ${checkInText}
    </div>
</div>
`;

// Add event listeners
clientEl.querySelector('.view-client').addEventListener('click', () =>
viewClient(clientProgram.clientId));
clientEl.querySelector('.edit-client').addEventListener('click', () =>
editClient(clientProgram.clientId));

clientsContainer.appendChild(clientEl);
});

// Handle "Load More" button
if (newNextToken) {
    loadMoreBtn.style.display = 'block';
    loadMoreBtn.onclick = () => fetchClients(newNextToken);
} else {
    loadMoreBtn.style.display = 'none';
}

} catch (error) {
    clientsContainer.innerHTML = `<div class="error">Error loading clients:
    ${error.message}</div>`;
}
}

// Update client statistics
function updateClientStats(clientPrograms) {
    const activeClientsCount = document.getElementById('active-clients-count');
    const checkinsToday = document.getElementById('checkins-today-count');
    const newClientsCount = document.getElementById('new-clients-count');

    // Count active clients
    const activeClients = clientPrograms.filter(cp => cp.status ===
'Active').length;
    activeClientsCount.textContent = activeClients;

```

```
// Count check-ins today
const today = new Date().toDateString();
const todayCheckins = clientPrograms.filter(cp => {
  const checkInDate = new Date(cp.nextCheckIn).toDateString();
  return checkInDate === today;
}).length;
checkinsToday.textContent = todayCheckins;

// Count new clients this month
const thisMonth = new Date().getMonth();
const thisYear = new Date().getFullYear();
const newClients = clientPrograms.filter(cp => {
  const startDate = new Date(cp.startDate);
  return startDate.getMonth() === thisMonth && startDate.getFullYear() ===
thisYear;
}).length;
newClientsCount.textContent = newClients;
}

// View client details
function viewClient(clientId) {
  window.location.href = `coach-client-details.html?id=${clientId}`;
}

// Edit client
function editClient(clientId) {
  window.location.href = `coach-client-edit.html?id=${clientId}`;
}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
  checkAuth();
  fetchClients();

  // Set up add client modal
  const modal = document.getElementById('add-client-modal');
  const addClientBtn = document.getElementById('add-client-btn');
  const closeBtn = document.querySelector('.close');

  addClientBtn.addEventListener('click', () => {
    modal.style.display = 'block';
  });

  closeBtn.addEventListener('click', () => {
    modal.style.display = 'none';
  });

  window.addEventListener('click', (event) => {
    if (event.target === modal) {
      modal.style.display = 'none';
    }
  });
});

// Set up add client form
```

```
document.getElementById('add-client-form').addEventListener('submit', async
(event) => {
    event.preventDefault();
    // Implementation for adding a new client
});
});
```

5. Create Program Content Management

1. Create program content upload functionality in the coach dashboard
2. Create program content viewing functionality in the client dashboard

6. Testing Phase 2

1. Client Dashboard Testing:

- Test program information display
- Test file access and download
- Verify proper permissions for client users

2. Coach Dashboard Testing:

- Test client management functionality
- Test adding new clients
- Test program assignment
- Test file upload and management

3. Security Testing:

- Verify clients can only access their own content
- Verify coaches can access all client information
- Test API permissions for content access

7. Deployment

```
# Deploy all Amplify resources
amplify publish
```

Phase 3: Payment Processing

Overview

Phase 3 adds payment processing capabilities to the platform, allowing clients to make one-time payments and manage subscriptions. This phase integrates Stripe for secure payment processing.

UI Mockups

Client Dashboard - Payments View

+-----+ THE SHREY METHOD FITNESS [John ▼] +-----+	
[Overview]	
[My Program]	PAYMENT INFORMATION
[Nutrition]	
[Messages]	Current Plan:
[★ Payments]	MONTHLY COACHING (\$199/month)
[Schedule]	Next billing: August 5, 2025
	PAYMENT METHOD
	Visa ending in 4242
	[Update Payment Method]
PAYMENT HISTORY	PAYMENT OPTIONS
July 5, 2025	[Cancel Subscription]
\$199 - Monthly Coaching	
	[Upgrade to 3-month plan]
June 5, 2025	Save 10% (\$179/month)
\$199 - Monthly Coaching	
	[Add 1:1 Session]
[View All Transactions]	\$99 single session
+-----+	

Coach Dashboard - Finances View

+-----+ THE SHREY METHOD FITNESS [Shreyas ▼] +-----+	
[Dashboard]	
[Clients]	FINANCE OVERVIEW
[Messages]	
[Programs]	Monthly Revenue: \$4,975
[★ Finances]	Clients: 24 active
[Settings]	Avg. Client Value: \$207
	UPCOMING PAYMENTS
RECENT TRANSACTIONS	
	Next 7 days: \$1,390
07/26/25	Next 30 days: \$4,975
John Doe - \$199	
	PAYMENT ALERTS
07/25/25	
Sarah Smith - \$249	⚠ Michael Johnson

	Payment failed (2 days ago)
07/25/25	
Michael Johnson - \$99	⚠ Lisa Wang
(FAILED)	Subscription ending
[Export Financial Data]	
+-----+	

Step-by-Step Implementation

1. Set Up Stripe Integration

1. Create a Stripe account at <https://stripe.com>
2. Get API keys from the Stripe Dashboard
3. Install Stripe libraries:

```
npm install stripe @stripe/stripe-js
```

2. Create Lambda Functions for Payment Processing

```
# Add function for payment processing
amplify add function
```

Configuration Details:

- Function name: processPayment
- Runtime: NodeJS
- Template: Hello World
- Advanced settings: Yes
- Access other resources: Yes
- Categories: API, Storage
- Environment variables: STRIPE_SECRET_KEY

Create the Lambda function code:

```
// amplify/backend/function/processPayment/src/index.js
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event) => {
  try {
    const { operation, payload } = JSON.parse(event.body);
```



```
        switch (operation) {
            case 'createPaymentIntent':
                return await createPaymentIntent(payload);
            case 'createSubscription':
                return await createSubscription(payload);
            case 'cancelSubscription':
                return await cancelSubscription(payload);
            case 'updatePaymentMethod':
                return await updatePaymentMethod(payload);
            default:
                return {
                    statusCode: 400,
                    body: JSON.stringify({ error: 'Invalid operation' })
                };
        }
    } catch (error) {
        return {
            statusCode: 500,
            body: JSON.stringify({ error: error.message })
        };
    }
};

async function createPaymentIntent(payload) {
    const { amount, currency, customerId, description } = payload;

    const paymentIntent = await stripe.paymentIntents.create({
        amount,
        currency,
        customer: customerId,
        description
    });

    return {
        statusCode: 200,
        body: JSON.stringify({
            clientSecret: paymentIntent.client_secret
        })
    };
}

async function createSubscription(payload) {
    const { customerId, priceId } = payload;

    const subscription = await stripe.subscriptions.create({
        customer: customerId,
        items: [{ price: priceId }],
        expand: ['latest_invoice.payment_intent']
    });

    return {
        statusCode: 200,
        body: JSON.stringify({
            subscriptionId: subscription.id,

```

```

        clientSecret: subscription.latest_invoice.payment_intent.client_secret
      })
    };
  }

  async function cancelSubscription(payload) {
    const { subscriptionId } = payload;

    const canceledSubscription = await stripe.subscriptions.del(subscriptionId);

    return {
      statusCode: 200,
      body: JSON.stringify({
        subscriptionId: canceledSubscription.id,
        status: canceledSubscription.status
      })
    };
  }

  async function updatePaymentMethod(payload) {
    const { customerId, paymentMethodId } = payload;

    await stripe.paymentMethods.attach(paymentMethodId, {
      customer: customerId
    });

    await stripe.customers.update(customerId, {
      invoice_settings: {
        default_payment_method: paymentMethodId
      }
    });

    return {
      statusCode: 200,
      body: JSON.stringify({
        success: true
      })
    };
  }
}

```

3. Create API Endpoint for Payment Processing

```

# Add API endpoint
amplify add api

```

Configuration Details:

- Service: REST
- API name: paymentapi
- Path: /payments

- Lambda function: processPayment
- Additional paths: No

```
# Push changes to AWS
amplify push
```

4. Extend Database Schema for Payments

Update the GraphQL schema to include payment information:

```
type Payment @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [read] },
  { allow: owner, operations: [read] }
]) {
  id: ID!
  clientId: ID!
  amount: Float!
  currency: String!
  status: String!
  type: String!
  description: String
  stripePaymentId: String
  createdAt: AWSDatetime!
}

type Subscription @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [read, update] },
  { allow: owner, operations: [read] }
]) {
  id: ID!
  clientId: ID!
  planId: ID!
  status: String!
  stripeSubscriptionId: String
  currentPeriodEnd: AWSDatetime
  createdAt: AWSDatetime!
}

type Plan @model
@auth(rules: [
  { allow: groups, groups: ["coaches"], operations: [create, read, update, delete] },
  { allow: owner, operations: [read] }
]) {
  id: ID!
  name: String!
  description: String
  price: Float!
}
```

```

currency: String!
interval: String!
stripePriceId: String
active: Boolean!
}

```

```

# Push changes to AWS
amplify push

```

5. Create Client Payment UI

1. Create client payments page (`client-payments.html`):

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Payments - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
  <script src="https://js.stripe.com/v3/"></script>
</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation (same as client-dashboard.html) -->
    <div class="dashboard-sidebar">
      <!-- ... sidebar content ... -->
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
      <header class="dashboard-header">
        <h1>Payments</h1>
      </header>

      <div class="content-container">
        <div class="dashboard-grid">
          <!-- Current Plan Card -->
          <div class="dashboard-card">
            <h2>Payment Information</h2>
            <div id="current-plan">
              <div class="loading">Loading plan information...</div>
            </div>
          </div>

          <!-- Payment Method Card -->

```

```

        <div class="dashboard-card">
            <h2>Payment Method</h2>
            <div id="payment-method">
                <div class="loading">Loading payment method...</div>
            </div>
        </div>

        <!-- Payment History Card -->
        <div class="dashboard-card">
            <h2>Payment History</h2>
            <div id="payment-history">
                <div class="loading">Loading payment history...</div>
            </div>
            <button id="view-all-transactions" class="btn-
secondary">View All Transactions</button>
        </div>

        <!-- Payment Options Card -->
        <div class="dashboard-card">
            <h2>Payment Options</h2>
            <div id="payment-options">
                <div class="loading">Loading payment options...</div>
            </div>
        </div>
    </div>
</div>
</div>
</div>

<!-- Payment Method Modal -->
<div id="payment-method-modal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2>Update Payment Method</h2>
        <form id="payment-form">
            <div id="card-element">
                <!-- Stripe Card Element will be inserted here -->
            </div>
            <div id="card-errors" class="error-message"></div>
            <button type="submit" class="btn-primary">Update Payment
Method</button>
        </form>
    </div>
</div>

<script src="js/client-payments.js"></script>
</body>
</html>

```

2. Create client payments JavaScript (`js/client-payments.js`):

```

// Configure Amplify
const awsConfig = {
  Auth: {
    region: 'us-west-2',
    userPoolId: 'us-west-2_xxxxxxxx',
    userPoolWebClientId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  },
  API: {
    endpoints: [{
      name: "shreymethodapi",
      endpoint: "https://xxxxxxxx.execute-api.us-west-2.amazonaws.com/dev"
    }, {
      name: "paymentapi",
      endpoint: "https://xxxxxxxx.execute-api.us-west-2.amazonaws.com/dev"
    }]
  }
};

Amplify.configure(awsConfig);

// Initialize Stripe
const stripe = Stripe('pk_test_xxxxxxxxxxxxxxxxxxxxxxxx'); // Your Stripe
publishable key

// Check authentication
async function checkAuth() {
  try {
    const user = await Amplify.Auth.currentAuthenticatedUser();
    return user;
  } catch (error) {
    window.location.href = 'client-login.html';
  }
}

// Fetch client subscription
async function fetchSubscription() {
  const currentPlan = document.getElementById('current-plan');

  try {
    const user = await checkAuth();

    // Query for client subscription
    const query = `query GetSubscription($clientId: ID!) {
      listSubscriptions(filter: {clientId: {eq: $clientId}, status: {eq:
"active"}}) {
        items {
          id
          planId
          status
          stripeSubscriptionId
          currentPeriodEnd
          plan {
            name

```

```

            price
            currency
            interval
        }
    }
}
};

const variables = {
  clientId: user.username
};

const response = await Amplify.API.graphql({
  query: query,
  variables: variables
});

const subscription = response.data.listSubscriptions.items[0];

if (subscription) {
  const nextBillingDate = new Date(subscription.currentPeriodEnd);
  const formattedPrice = formatCurrency(subscription.plan.price,
subscription.plan.currency);

  currentPlan.innerHTML = `
    <h3>${subscription.plan.name}</h3>
    <p class="plan-
price">${formattedPrice}/${subscription.plan.interval}</p>
    <p class="next-billing">Next billing:
${nextBillingDate.toLocaleDateString()}</p>
  `;

  // Fetch payment method
  fetchPaymentMethod(user.username);

  // Fetch payment history
  fetchPaymentHistory(user.username);

  // Load payment options
  loadPaymentOptions(subscription);
} else {
  currentPlan.innerHTML = '<p>No active subscription found.</p>';

  // Load available plans
  loadAvailablePlans();
}

} catch (error) {
  currentPlan.innerHTML = `<div class="error">Error loading subscription:
${error.message}</div>`;
}
}

```

```
// Format currency helper
function formatCurrency(amount, currency) {
  return new Intl.NumberFormat('en-US', {
    style: 'currency',
    currency: currency
  }).format(amount);
}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
  fetchSubscription();

  // Set up payment method modal
  const modal = document.getElementById('payment-method-modal');
  const updatePaymentBtn = document.getElementById('update-payment-btn');
  const closeBtn = document.querySelector('.close');

  if (updatePaymentBtn) {
    updatePaymentBtn.addEventListener('click', () => {
      modal.style.display = 'block';
      setupStripeElements();
    });
  }

  closeBtn.addEventListener('click', () => {
    modal.style.display = 'none';
  });

  window.addEventListener('click', (event) => {
    if (event.target === modal) {
      modal.style.display = 'none';
    }
  });
});
```

6. Create Coach Finance Dashboard

1. Create coach finances page (`coach-finances.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Finances - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-amplify.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@3.7.1/dist/chart.min.js">
</script>
```



```

</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation (same as coach-dashboard.html) -->
    <div class="dashboard-sidebar">
      <!-- ... sidebar content ... -->
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
      <header class="dashboard-header">
        <h1>Finances</h1>
        <div class="header-actions">
          <button id="export-data-btn" class="btn-secondary">Export
Financial Data</button>
        </div>
      </header>

      <div class="content-container">
        <div class="dashboard-grid">
          <!-- Finance Overview Card -->
          <div class="dashboard-card">
            <h2>Finance Overview</h2>
            <div class="finance-stats">
              <div class="stat-item">
                <h3>Monthly Revenue</h3>
                <p id="monthly-revenue">$0</p>
              </div>
              <div class="stat-item">
                <h3>Active Clients</h3>
                <p id="active-clients">0</p>
              </div>
              <div class="stat-item">
                <h3>Avg. Client Value</h3>
                <p id="avg-client-value">$0</p>
              </div>
            </div>
            <div class="chart-container">
              <canvas id="revenue-chart"></canvas>
            </div>
          </div>

          <!-- Recent Transactions Card -->
          <div class="dashboard-card">
            <h2>Recent Transactions</h2>
            <div id="recent-transactions">
              <div class="loading">Loading transactions...</div>
            </div>
          </div>

          <!-- Upcoming Payments Card -->
          <div class="dashboard-card">
            <h2>Upcoming Payments</h2>
            <div id="upcoming-payments">

```

```

        <div class="loading">Loading upcoming payments...
    </div>

    </div>
</div>

    <!-- Payment Alerts Card -->
    <div class="dashboard-card">
        <h2>Payment Alerts</h2>
        <div id="payment-alerts">
            <div class="loading">Loading payment alerts...</div>
        </div>
    </div>
</div>
</div>
</div>
</div>

    <script src="js/coach-finances.js"></script>
</body>
</html>

```

2. Create coach finances JavaScript (`js/coach-finances.js`):

```

// Configure Amplify (same as other files)
const awsConfig = { /* ... */ };
Amplify.configure(awsConfig);

// Check authentication (same as coach-clients.js)
async function checkAuth() { /* ... */ }

// Fetch financial data
async function fetchFinancialData() {
    try {
        await checkAuth();

        // Fetch payments
        const paymentsQuery = `query ListPayments {
            listPayments(limit: 100, sort: {field: "createdAt", direction:
"desc"}) {
                items {
                    id
                    clientId
                    amount
                    currency
                    status
                    type
                    description
                    createdAt
                    client {
                        name
                    }
                }
            }
        }`;
    }
}

```

```
    }
  }`;

  const paymentsResponse = await Amplify.API.graphql({
    query: paymentsQuery
  });

  const payments = paymentsResponse.data.listPayments.items;

  // Fetch subscriptions
  const subscriptionsQuery = `query ListSubscriptions {
    listSubscriptions(filter: {status: {eq: "active"}}) {
      items {
        id
        clientId
        planId
        status
        currentPeriodEnd
        client {
          name
        }
        plan {
          name
          price
          currency
        }
      }
    }
  }`;

  const subscriptionsResponse = await Amplify.API.graphql({
    query: subscriptionsQuery
  });

  const subscriptions = subscriptionsResponse.data.listSubscriptions.items;

  // Update UI with financial data
  updateFinanceOverview(payments, subscriptions);
  updateRecentTransactions(payments);
  updateUpcomingPayments(subscriptions);
  updatePaymentAlerts(payments, subscriptions);

} catch (error) {
  console.error('Error fetching financial data:', error);
}

// Update finance overview
function updateFinanceOverview(payments, subscriptions) {
  const monthlyRevenueEl = document.getElementById('monthly-revenue');
  const activeClientsEl = document.getElementById('active-clients');
  const avgClientValueEl = document.getElementById('avg-client-value');

  // Calculate monthly revenue
```

```
const currentMonth = new Date().getMonth();
const currentYear = new Date().getFullYear();

const monthlyPayments = payments.filter(payment => {
  const paymentDate = new Date(payment.createdAt);
  return paymentDate.getMonth() === currentMonth &&
  paymentDate.getFullYear() === currentYear;
});

const monthlyRevenue = monthlyPayments.reduce((total, payment) => {
  return total + payment.amount;
}, 0);

// Count active clients
const activeClients = new Set(subscriptions.map(sub => sub.clientId)).size;

// Calculate average client value
const avgClientValue = activeClients > 0 ? monthlyRevenue / activeClients : 0;

// Update UI
monthlyRevenueEl.textContent = formatCurrency(monthlyRevenue, 'USD');
activeClientsEl.textContent = activeClients;
avgClientValueEl.textContent = formatCurrency(avgClientValue, 'USD');

// Create revenue chart
createRevenueChart(payments);
}

// Create revenue chart
function createRevenueChart(payments) {
  const ctx = document.getElementById('revenue-chart').getContext('2d');

  // Group payments by month
  const monthlyData = {};

  payments.forEach(payment => {
    const date = new Date(payment.createdAt);
    const monthYear = `${date.getMonth() + 1}/${date.getFullYear()}`;

    if (!monthlyData[monthYear]) {
      monthlyData[monthYear] = 0;
    }

    monthlyData[monthYear] += payment.amount;
  });

  // Sort months chronologically
  const sortedMonths = Object.keys(monthlyData).sort((a, b) => {
    const [aMonth, aYear] = a.split('/').map(Number);
    const [bMonth, bYear] = b.split('/').map(Number);

    if (aYear !== bYear) {
      return aYear - bYear;
    }
  })
```

```
        return aMonth - bMonth;
    });

    // Create chart
    new Chart(ctx, {
        type: 'line',
        data: {
            labels: sortedMonths,
            datasets: [{
                label: 'Monthly Revenue',
                data: sortedMonths.map(month => monthlyData[month]),
                backgroundColor: 'rgba(76, 175, 80, 0.2)',
                borderColor: 'rgba(76, 175, 80, 1)',
                borderWidth: 2,
                tension: 0.3
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                y: {
                    beginAtZero: true,
                    ticks: {
                        callback: function(value) {
                            return '$' + value;
                        }
                    }
                }
            }
        }
    });
}

// Format currency helper
function formatCurrency(amount, currency) {
    return new Intl.NumberFormat('en-US', {
        style: 'currency',
        currency: currency
    }).format(amount);
}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
    fetchFinancialData();

    // Set up export data button
    document.getElementById('export-data-btn').addEventListener('click', () => {
        // Implementation for exporting financial data
    });
});
```

7. Testing Phase 3

1. Payment Processing Testing:

- Test Stripe integration
- Test payment method updates
- Test subscription creation and cancellation
- Verify payment history display

2. Financial Dashboard Testing:

- Test revenue calculations
- Test transaction history display
- Test upcoming payments display
- Test payment alerts functionality

3. Security Testing:

- Verify secure handling of payment information
- Test API permissions for payment operations
- Verify proper error handling for payment failures

8. Deployment

```
# Deploy all Amplify resources
amplify publish
```

Phase 4: Calendly Integration

Overview

Phase 4 integrates Calendly for scheduling capabilities, allowing clients to book sessions with you directly through the platform. This phase leverages Calendly's existing functionality rather than building a custom scheduling system.

UI Mockups

Client Dashboard - Schedule View

THE SHREY METHOD FITNESS

[John ▼]

[Overview]

[My Program]SCHEDULE A SESSION

[Nutrition]

[Messages][Embedded Calendly Widget]

[Payments]	
[★ Schedule]	<ul style="list-style-type: none">• Select session type• Choose available date• Pick available time slot
UPCOMING SESSIONS	
Check-in Call	
July 30, 2025	
3:00 PM - 3:30 PM	
[Reschedule] [Cancel]	
PAST SESSIONS	
Progress Review	
July 15, 2025	
[View Session Notes]	
+-----+	

Coach Dashboard - Calendar View

+-----+	
THE SHREY METHOD FITNESS	[Shreyas ▼]
+-----+	
[Dashboard]	CALENDAR
[Clients]	
[Messages]	[July 2025] [Week ▼]
[Programs]	
[Finances]	+-----+
[★ Schedule]	DAILY SCHEDULE
[Settings]	+-----+
TODAY'S SCHEDULE	9:00 AM: Sarah Smith
	Check-in Call
9:00 AM	11:30 AM: John Doe
Sarah Smith	Program Review
Check-in Call	2:00 PM: New Client
	Consultation
11:30 AM	4:30 PM: Michael Johnson
John Doe	Final Assessment
Program Review	
2:00 PM	+-----+
New Client	
Consultation	
[Manage Availability] [Block Time Off]	
+-----+	

Step-by-Step Implementation

1. Set Up Calendly Account

1. Sign up for a Calendly Pro or Teams account at <https://calendly.com>
2. Configure your availability and event types:
 - 15-minute consultation (free)
 - 30-minute check-in call
 - 60-minute assessment
 - 90-minute in-person coaching

2. Create Client Scheduling Page

1. Create client schedule page (`client-schedule.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Schedule - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
  <script src="https://assets.calendly.com/assets/external/widget.js" async>
</script>
</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation (same as client-dashboard.html) -->
    <div class="dashboard-sidebar">
      <!-- ... sidebar content ... -->
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
      <header class="dashboard-header">
        <h1>Schedule</h1>
      </header>

      <div class="content-container">
        <div class="dashboard-grid">
          <!-- Calendly Widget Card -->
          <div class="dashboard-card full-width">
            <h2>Schedule a Session</h2>
            <div class="calendly-container">
              <!-- Calendly inline widget -->
              <div id="calendly-widget" style="min-
width:320px;height:630px;"></div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
```



```

        </div>

        <!-- Upcoming Sessions Card -->
        <div class="dashboard-card">
            <h2>Upcoming Sessions</h2>
            <div id="upcoming-sessions">
                <div class="loading">Loading upcoming sessions...
</div>

            </div>
        </div>

        <!-- Past Sessions Card -->
        <div class="dashboard-card">
            <h2>Past Sessions</h2>
            <div id="past-sessions">
                <div class="loading">Loading past sessions...</div>
            </div>
        </div>
    </div>
</div>

    <script src="js/client-schedule.js"></script>
</body>
</html>

```

2. Create client scheduling JavaScript (`js/client-schedule.js`):

```

// Configure Amplify
const awsConfig = {
  Auth: {
    region: 'us-west-2',
    userPoolId: 'us-west-2_xxxxxxxx',
    userPoolWebClientId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  },
  API: {
    endpoints: [{
      name: "shreymethodapi",
      endpoint: "https://xxxxxxxx.execute-api.us-west-2.amazonaws.com/dev"
    }]
  }
};

Amplify.configure(awsConfig);

// Check authentication
async function checkAuth() {
  try {
    const user = await Amplify.Auth.currentAuthenticatedUser();
    return user;
  } catch (error) {

```

```
        window.location.href = 'client-login.html';
    }
}

// Initialize Calendly
async function initializeCalendly() {
    try {
        const user = await checkAuth();

        // Get user information to prefill Calendly
        const name = user.attributes.name || '';
        const email = user.attributes.email || '';

        // Initialize Calendly inline widget
        Calendly.initInlineWidget({
            url: 'https://calendly.com/shreymethodfitness',
            parentElement: document.getElementById('calendly-widget'),
            prefill: {
                name: name,
                email: email
            },
            utm: {
                utmSource: 'website'
            }
        });

        // Set up Calendly event listener
        window.addEventListener('message', function(e) {
            if (e.data.event && e.data.event.indexOf('calendly') === 0) {
                // Handle Calendly events
                if (e.data.event === 'calendly.event_scheduled') {
                    // Refresh upcoming sessions after scheduling
                    fetchUpcomingSessions();
                }
            }
        });

    } catch (error) {
        console.error('Error initializing Calendly:', error);
    }
}

// Fetch upcoming sessions
async function fetchUpcomingSessions() {
    const upcomingSessions = document.getElementById('upcoming-sessions');

    try {
        // In a real implementation, you would fetch this data from your database
        // or from Calendly's API using a server-side function

        // For this example, we'll use mock data
        const mockSessions = [
            {
                id: '1',
            }
        ]
    }
}
```

```
        title: 'Check-in Call',
        date: new Date('2025-07-30T15:00:00'),
        duration: 30,
        status: 'confirmed'
    }
];

if (mockSessions.length > 0) {
    const sessionsList = document.createElement('div');
    sessionsList.className = 'sessions-list';

    mockSessions.forEach(session => {
        const sessionItem = document.createElement('div');
        sessionItem.className = 'session-item';

        const formattedDate = session.date.toLocaleDateString();
        const startTime = session.date.toLocaleTimeString([], { hour: '2-
digit', minute: '2-digit' });

        const endTime = new Date(session.date.getTime() + session.duration
* 60000)
            .toLocaleTimeString([], { hour: '2-digit', minute: '2-digit'
});

        sessionItem.innerHTML = `
            <h3>${session.title}</h3>
            <p class="session-date">${formattedDate}</p>
            <p class="session-time">${startTime} - ${endTime}</p>
            <div class="session-actions">
                <button class="btn-secondary reschedule-btn" data-
id="${session.id}">Reschedule</button>
                <button class="btn-secondary cancel-btn" data-
id="${session.id}">Cancel</button>
            </div>
        `;

        sessionsList.appendChild(sessionItem);
    });

    upcomingSessions.innerHTML = '';
    upcomingSessions.appendChild(sessionsList);

    // Add event listeners to buttons
    document.querySelectorAll('.reschedule-btn').forEach(btn => {
        btn.addEventListener('click', () =>
rescheduleSession(btn.dataset.id));
    });

    document.querySelectorAll('.cancel-btn').forEach(btn => {
        btn.addEventListener('click', () =>
cancelSession(btn.dataset.id));
    });
} else {
```

```
    upcomingSessions.innerHTML = '<p>No upcoming sessions scheduled.</p>';
  }

  } catch (error) {
    upcomingSessions.innerHTML = `<div class="error">Error loading sessions:
    ${error.message}</div>`;
  }

  // Also fetch past sessions
  fetchPastSessions();
}

// Fetch past sessions
async function fetchPastSessions() {
  const pastSessions = document.getElementById('past-sessions');

  try {
    // In a real implementation, you would fetch this data from your database
    // or from Calendly's API using a server-side function

    // For this example, we'll use mock data
    const mockSessions = [
      {
        id: '2',
        title: 'Progress Review',
        date: new Date('2025-07-15T14:00:00'),
        duration: 45,
        status: 'completed',
        hasNotes: true
      }
    ];

    if (mockSessions.length > 0) {
      const sessionsList = document.createElement('div');
      sessionsList.className = 'sessions-list';

      mockSessions.forEach(session => {
        const sessionItem = document.createElement('div');
        sessionItem.className = 'session-item past';

        const formattedDate = session.date.toLocaleDateString();

        sessionItem.innerHTML = `
          <h3>${session.title}</h3>
          <p class="session-date">${formattedDate}</p>
          ${session.hasNotes ? `<button class="btn-secondary view-notes-
          btn" data-id="${session.id}">View Session Notes</button>` : ''}
        `;

        sessionsList.appendChild(sessionItem);
      });

      pastSessions.innerHTML = '';
      pastSessions.appendChild(sessionsList);
    }
  }
}
```

```

        // Add event listeners to buttons
        document.querySelectorAll('.view-notes-btn').forEach(btn => {
            btn.addEventListener('click', () =>
viewSessionNotes(btn.dataset.id));
        });

    } else {
        pastSessions.innerHTML = '<p>No past sessions found.</p>';
    }

    } catch (error) {
        pastSessions.innerHTML = `<div class="error">Error loading sessions:
${error.message}</div>`;
    }
}

// Reschedule session
function rescheduleSession(sessionId) {
    // In a real implementation, this would open Calendly's reschedule flow
    // For this example, we'll just scroll to the Calendly widget
    document.getElementById('calendly-widget').scrollIntoView({ behavior: 'smooth'
});
}

// Cancel session
function cancelSession(sessionId) {
    // In a real implementation, this would cancel the session via Calendly's API
    if (confirm('Are you sure you want to cancel this session?')) {
        alert('Session cancelled. In a real implementation, this would cancel via
Calendly\'s API.');
```

```

        fetchUpcomingSessions(); // Refresh the list
    }
}

// View session notes
function viewSessionNotes(sessionId) {
    // In a real implementation, this would fetch and display session notes
    alert('In a real implementation, this would show session notes from your
database.');
```

```

}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
    initializeCalendly();
    fetchUpcomingSessions();
});
```

3. Create Coach Calendar View

1. Create coach schedule page ([coach-schedule.html](#)):

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Schedule - The Shrey Method Fitness</title>
  <link rel="stylesheet" href="css/styles.css">
  <link rel="stylesheet" href="css/dashboard.css">
  <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
</head>
<body>
  <div class="dashboard-container">
    <!-- Sidebar navigation (same as coach-dashboard.html) -->
    <div class="dashboard-sidebar">
      <!-- ... sidebar content ... -->
    </div>

    <!-- Main content area -->
    <div class="dashboard-content">
      <header class="dashboard-header">
        <h1>Schedule</h1>
        <div class="header-actions">
          <button id="manage-availability-btn" class="btn-
secondary">Manage Availability</button>
          <button id="block-time-btn" class="btn-secondary">Block Time
Off</button>
        </div>
      </header>

      <div class="content-container">
        <div class="calendar-controls">
          <div class="date-selector">
            <button id="prev-month" class="btn-icon"><i class="fas fa-
chevron-left"></i></button>
            <h2 id="current-month">July 2025</h2>
            <button id="next-month" class="btn-icon"><i class="fas fa-
chevron-right"></i></button>
          </div>
          <div class="view-selector">
            <select id="calendar-view">
              <option value="day">Day</option>
              <option value="week" selected>Week</option>
              <option value="month">Month</option>
            </select>
          </div>
        </div>

        <div class="calendar-container">
          <div id="calendar">
            <!-- Calendar will be rendered here -->
            <div class="loading">Loading calendar...</div>
          </div>
        </div>
      </div>
    </div>
  </div>

```

```

        </div>

        <div class="dashboard-grid">
            <!-- Today's Schedule Card -->
            <div class="dashboard-card">
                <h2>Today's Schedule</h2>
                <div id="today-schedule">
                    <div class="loading">Loading today's schedule...</div>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Calendly Admin Panel Iframe Modal -->
<div id="calendly-admin-modal" class="modal">
    <div class="modal-content large">
        <span class="close">&times;</span>
        <iframe id="calendly-admin-iframe" src="about:blank"
style="width:100%; height:600px; border:none;"></iframe>
    </div>
</div>

<script src="js/coach-schedule.js"></script>
</body>
</html>

```

2. Create coach schedule JavaScript ([js/coach-schedule.js](#)):

```

// Configure Amplify (same as other files)
const awsConfig = { /* ... */ };
Amplify.configure(awsConfig);

// Check authentication (same as coach-clients.js)
async function checkAuth() { /* ... */ }

// Initialize calendar
async function initializeCalendar() {
    try {
        await checkAuth();

        // In a real implementation, this would integrate with Calendly's API
        // to fetch and display your calendar

        // For this example, we'll use mock data and a simplified calendar view
        renderCalendar();
        fetchTodaySchedule();

    } catch (error) {
        console.error('Error initializing calendar:', error);
    }
}

```

```
}

// Render calendar
function renderCalendar() {
  const calendar = document.getElementById('calendar');
  const currentMonthEl = document.getElementById('current-month');

  // Get current date
  const now = new Date();
  const currentYear = now.getFullYear();
  const currentMonth = now.getMonth();

  // Set current month display
  currentMonthEl.textContent = new Date(currentYear,
currentMonth).toLocaleDateString('en-US', { month: 'long', year: 'numeric' });

  // Get selected view
  const view = document.getElementById('calendar-view').value;

  // Render appropriate view
  switch (view) {
    case 'day':
      renderDayView(calendar, now);
      break;
    case 'week':
      renderWeekView(calendar, now);
      break;
    case 'month':
      renderMonthView(calendar, now);
      break;
  }
}

// Fetch today's schedule
async function fetchTodaySchedule() {
  const todaySchedule = document.getElementById('today-schedule');

  try {
    // In a real implementation, this would fetch from Calendly's API
    // For this example, we'll use mock data
    const mockSchedule = [
      {
        id: '1',
        clientName: 'Sarah Smith',
        title: 'Check-in Call',
        time: '9:00 AM'
      },
      {
        id: '2',
        clientName: 'John Doe',
        title: 'Program Review',
        time: '11:30 AM'
      },
    ]
  }
}
```



```
        id: '3',
        clientName: 'New Client',
        title: 'Consultation',
        time: '2:00 PM'
    }
];

if (mockSchedule.length > 0) {
    const scheduleList = document.createElement('div');
    scheduleList.className = 'schedule-list';

    mockSchedule.forEach(session => {
        const sessionItem = document.createElement('div');
        sessionItem.className = 'schedule-item';

        sessionItem.innerHTML = `
            <div class="schedule-time">${session.time}</div>
            <div class="schedule-details">
                <h3>${session.clientName}</h3>
                <p>${session.title}</p>
            </div>
        `;

        scheduleList.appendChild(sessionItem);
    });

    todaySchedule.innerHTML = '';
    todaySchedule.appendChild(scheduleList);
} else {
    todaySchedule.innerHTML = '<p>No sessions scheduled for today.</p>';
}

} catch (error) {
    todaySchedule.innerHTML = `<div class="error">Error loading schedule:
    ${error.message}</div>`;
}

}

// Open Calendly admin panel
function openCalendlyAdmin(page = 'scheduled_events') {
    const modal = document.getElementById('calendly-admin-modal');
    const iframe = document.getElementById('calendly-admin-iframe');

    // Set iframe source to Calendly admin page
    iframe.src = `https://calendly.com/app/${page}`;

    // Show modal
    modal.style.display = 'block';
}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
    initializeCalendar();
});
```

```
// Set up calendar view selector
document.getElementById('calendar-view').addEventListener('change',
renderCalendar);

// Set up month navigation
document.getElementById('prev-month').addEventListener('click', () => {
  // Implementation for navigating to previous month
});

document.getElementById('next-month').addEventListener('click', () => {
  // Implementation for navigating to next month
});

// Set up admin buttons
document.getElementById('manage-availability-btn').addEventListener('click',
() => {
  openCalendlyAdmin('availability');
});

document.getElementById('block-time-btn').addEventListener('click', () => {
  openCalendlyAdmin('event_types/timeoffs/new');
});

// Set up modal close button
document.querySelector('.close').addEventListener('click', () => {
  document.getElementById('calendly-admin-modal').style.display = 'none';
});
});
```

4. Create Webhook for Calendly Events

To keep your platform in sync with Calendly events (bookings, cancellations, etc.), set up a webhook:

```
# Add function for Calendly webhook
amplify add function
```

Configuration Details:

- Function name: calendlyWebhook
- Runtime: NodeJS
- Template: Hello World
- Advanced settings: Yes
- Access other resources: Yes
- Categories: API, Storage
- Environment variables: None

Create the Lambda function code:

```
// amplify/backend/function/calendlyWebhook/src/index.js
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event) => {
  try {
    // Verify Calendly webhook signature (in a real implementation)
    // ...

    const payload = JSON.parse(event.body);
    const eventType = payload.event;

    switch (eventType) {
      case 'invitee.created':
        return await handleBookingCreated(payload);
      case 'invitee.canceled':
        return await handleBookingCanceled(payload);
      default:
        return {
          statusCode: 200,
          body: JSON.stringify({ message: 'Event type not handled' })
        };
    }
  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify({ error: error.message })
    };
  }
};

async function handleBookingCreated(payload) {
  // Extract booking information
  const booking = {
    id: payload.payload.invitee.uuid,
    eventType: payload.payload.event_type.name,
    clientEmail: payload.payload.invitee.email,
    clientName: payload.payload.invitee.name,
    startTime: payload.payload.scheduled_event.start_time,
    endTime: payload.payload.scheduled_event.end_time,
    status: 'confirmed',
    createdAt: new Date().toISOString()
  };

  // Store booking in database
  await docClient.put({
    TableName: process.env.BOOKINGS_TABLE,
    Item: booking
  }).promise();

  return {
    statusCode: 200,
    body: JSON.stringify({ message: 'Booking created successfully' })
  };
}
```

```
};  
}  
  
async function handleBookingCanceled(payload) {  
  // Extract booking ID  
  const bookingId = payload.payload.invitee.uuid;  
  
  // Update booking status in database  
  await docClient.update({  
    TableName: process.env.BOOKINGS_TABLE,  
    Key: { id: bookingId },  
    UpdateExpression: 'set #status = :status',  
    ExpressionAttributeNames: {  
      '#status': 'status'  
    },  
    ExpressionAttributeValues: {  
      ':status': 'canceled'  
    }  
  }).promise();  
  
  return {  
    statusCode: 200,  
    body: JSON.stringify({ message: 'Booking canceled successfully' })  
  };  
}
```

5. Create API Endpoint for Calendly Webhook

```
# Add API endpoint  
amplify add api
```

Configuration Details:

- Service: REST
- API name: calendlyapi
- Path: /webhook
- Lambda function: calendlyWebhook
- Additional paths: No

```
# Push changes to AWS  
amplify push
```

6. Configure Calendly Webhook

1. Go to Calendly admin dashboard
2. Navigate to Integrations > Webhooks

3. Add a new webhook with the API Gateway URL
4. Select events to monitor (invitee.created, invitee.canceled)

7. Testing Phase 4

1. Calendly Integration Testing:

- Test booking a session through the client dashboard
- Test viewing upcoming sessions
- Test canceling a session
- Verify webhook functionality for event synchronization

2. Coach Calendar Testing:

- Test calendar view functionality
- Test managing availability
- Test blocking time off
- Verify today's schedule display

3. Security Testing:

- Verify proper authentication for Calendly admin access
- Test webhook signature verification
- Verify proper error handling for scheduling operations

8. Deployment

```
# Deploy all Amplify resources
amplify publish
```

Cost Estimates

AWS Services Monthly Costs

Service	Free Tier	Beyond Free Tier	Estimated Monthly Cost
AWS Amplify Hosting	First 12 months: 5GB storage, 15GB data transfer	\$0.023 per GB stored, \$0.15 per GB transferred	\$0-25
Cognito	First 50,000 MAUs free	\$0.0055 per MAU	\$0 (< 50k users)
DynamoDB	25GB storage	On-demand capacity: ~\$1.25 per million read requests	\$0-20
Lambda	1M free requests, 400,000 GB-seconds	\$0.20 per 1M requests	\$0-15
API Gateway	1M API calls (first 12 months)	\$3.50 per million API calls	\$0-10

Service	Free Tier	Beyond Free Tier	Estimated Monthly Cost
S3 Storage	5GB storage, 20,000 GET requests	\$0.023 per GB stored	\$1-15
SES (Email)	N/A	\$0.10 per 1,000 emails	\$1-5
Total AWS			\$1-90

Third-Party Services

Service	Plan	Monthly Cost
Calendly	Pro	\$12
Calendly	Teams	\$16 per user
Stripe	Standard	2.9% + \$0.30 per transaction
Total Third-Party		\$12-16 + transaction fees

Total Estimated Monthly Operating Costs

User Base	Monthly Cost Range
Initial (< 100 clients)	\$13-50
Growing (100-500 clients)	\$50-150
Established (500+ clients)	\$150-300+

One-Time Development Costs

If hiring developers for implementation:

Implementation Scope	Estimated Cost
Basic implementation (Phase 1 only)	\$5,000-10,000
Full implementation (All phases)	\$15,000-30,000+

Cost Optimization Tips

- Leverage AWS Free Tier:** The free tier covers most services for the first 12 months
- Use Serverless Architecture:** Pay only for what you use
- Implement Caching:** Reduce database reads and API calls
- Monitor Usage:** Set up AWS Budgets to track spending
- Phase Implementation:** Start with core features, add more as you grow

Future Expansion Considerations

Advanced Client Features

1. **Mobile App:**

- Native mobile experience for clients
- Push notifications for appointments and check-ins
- Offline access to workout plans

2. **Progress Tracking:**

- Body measurements tracking
- Progress photos with comparison tools
- Achievement badges and milestones

3. **Nutrition Tracking:**

- Meal planning and recipes
- Food diary with macro tracking
- Grocery list generation

Advanced Coach Features

1. **Analytics Dashboard:**

- Client retention metrics
- Revenue forecasting
- Program effectiveness analysis

2. **Content Management System:**

- Exercise video library
- Template workout plans
- Drag-and-drop program builder

3. **Team Management:**

- Support for multiple coaches
- Client assignment and transfers
- Team performance metrics

Technical Enhancements

1. **Real-time Features:**

- Live chat with clients
- Real-time workout tracking
- Instant notifications

2. **AI Integration:**

- Automated check-in reminders
- Personalized workout recommendations

- Nutrition analysis

3. Integration Ecosystem:

- Fitness tracker integration (Fitbit, Apple Health)
- Nutrition app integration (MyFitnessPal)
- Accounting software integration (QuickBooks, Xero)

Implementation Approach

When considering these expansions, we recommend:

1. **Prioritize based on client feedback:** Let actual user needs drive development
 2. **Implement incrementally:** Add features in small, testable batches
 3. **Measure impact:** Track usage and ROI of new features
 4. **Maintain performance:** Ensure the platform remains fast and responsive
 5. **Consider scalability:** Design with growth in mind from the beginning
-

Technical Resources

AWS Documentation

- [AWS Amplify Documentation](#)
- [Amazon Cognito Developer Guide](#)
- [DynamoDB Developer Guide](#)
- [Lambda Developer Guide](#)

Third-Party Services

- [Stripe Documentation](#)
- [Calendly API Documentation](#)

Development Resources

- [AWS Amplify JavaScript Library](#)
- [Stripe.js Documentation](#)
- [Calendly Embed Documentation](#)

Security Best Practices

- [AWS Security Best Practices](#)
- [Stripe Security](#)
- [OWASP Top 10](#)