Tutorial: Building the CoolWallet Savings App - Full Source Code Guide

This tutorial provides a complete guide to building the CoolWallet savings app, including full source code for all key files.

Table of Contents

Project Overview

CoolWallet is a personal savings wallet app built with Ionic Angular frontend and Laravel backend, featuring offline-first synchronization.

Prerequisites

- Node.js, npm, Angular CLI, Ionic CLI
- PHP, Composer, MySQL

Frontend Setup

Step 1: Create Project

```
ionic start savings-wallet-app blank --type=angular
cd savings-wallet-app
```

Step 2: Install Dependencies

```
npm install @capacitor/core @capacitor/android @capacitor/app @capacitor/camera @capacitor/filesystem
@capacitor/haptics @capacitor/keyboard @capacitor/local-notifications @capacitor/preferences
@capacitor/splash-screen @capacitor/status-bar @capacitor-community/text-to-speech
npm install @ionic/angular ionicons rxjs tslib zone.js
npm install angular4-paystack @paystack/inline-js jspdf xlsx
```

Step 3: Capacitor Setup

```
npx cap init savings-wallet-app com.example.savingswallet
npx cap add android
```

Backend Setup (Laravel)

Step 1: Create Project

```
composer create-project laravel/laravel laravel-backend
cd laravel-backend
```

Step 2: Configure Database

Update `.env` with database credentials.

Step 3: Install Sanctum

```
composer require laravel/sanctum
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
php artisan migrate
```

Running the App

Frontend

```
ionic serve
```

Backend

```
php artisan serve
```

Full Source Code

Frontend Files

src/main.ts

```
import { enableProdMode, importProvidersFrom } from '@angular/core';
import { bootstrapApplication } from '@angular/platform-browser';
import { RouteReuseStrategy, provideRouter } from '@angular/router';
import { IonicModule, IonicRouteStrategy } from '@ionic/angular';
import { provideHttpClient } from '@angular/common/http';

import { routes } from './app/app.routes';
import { AppComponent } from './app/app.component';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

bootstrapApplication(AppComponent, {
  providers: [
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
    importProvidersFrom(IonicModule.forRoot({})),
    provideRouter(routes),
    provideHttpClient(),
  ],
});
```

src/app/app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SplashScreen } from '@capacitor/splash-screen';
import { CommonModule } from '@angular/common';
import { LocalNotifications } from '@capacitor/local-notifications';
import { SettingsService } from './services/settings.service';
import { IonicModule } from '@ionic/angular';
import { RouterModule } from '@angular/router';

import { HttpClient, HttpClientModule } from '@angular/common/http';
import { ApiService } from './services/api.service';
import { NetworkService } from './services/network.service';
import { SyncService } from './services/sync.service';
import { AuthService } from './services/auth.service';

@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html',
  styleUrls: ['app.component.scss'],
  standalone: true,
  imports: [IonicModule, CommonModule, RouterModule,HttpClientModule],
  providers:[ ApiService,
  NetworkService,
  SyncService,
  AuthService,HttpClient],
})
export class AppComponent implements OnInit {
```

```
  public appPages = [
    { title: 'Home', url: '/home', icon: 'home' },
    { title: 'Savings Goals', url: '/savings-goal', icon: 'flag' },
    { title: 'Withdraw', url: '/withdraw', icon: 'wallet' },
    { title: 'Settings', url: '/settings', icon: 'cog' },
  ];
  profilePicture: string | undefined;

  constructor(private settingsService: SettingsService) {
    this.initializeApp();
    this.loadTheme();
    this.loadProfilePicture();
    window.addEventListener('settingsChanged', () => {
      this.loadProfilePicture();
      this.loadTheme();
    });
  }

  async initializeApp() {
    await SplashScreen.show({
      showDuration: 2000,
      autoHide: true,
    });
  }

  async ngOnInit() {
    await LocalNotifications.requestPermissions();
    this.loadProfilePicture();
  }

  async loadTheme() {
    const settings = await this.settingsService.getSettings();
    document.body.classList.remove('dark', 'maroon');
    if (settings.theme !== 'light') {
      document.body.classList.add(settings.theme);
    }
  }

  async loadProfilePicture() {
    const settings = await this.settingsService.getSettings();
    this.profilePicture = settings.profilePicture;
    if(this.profilePicture=='')
    {
        this.profilePicture = 'assets/img/u1.png';
    }
    else
    {

    }
  }
}
```

src/app/app.component.html

```
<ion-app>
  <ion-split-pane contentId="main-content">
    <ion-menu contentId="main-content" type="overlay">
      <ion-header>
        <ion-toolbar>
          <ion-title>CoolWallet</ion-title>
        </ion-toolbar>
      </ion-header>
      <ion-content>
        <ion-list>
          <ion-menu-toggle auto-hide="false" *ngFor="let p of appPages">
            <ion-item [routerDirection]="'root'" [routerLink]="[p.url]">
              <ion-icon slot="start" [name]="p.icon"></ion-icon>
              <ion-label>
                {{ p.title }}
              </ion-label>
            </ion-item>
          </ion-menu-toggle>
        </ion-list>
      </ion-content>
    </ion-menu>
    <ion-router-outlet id="main-content"></ion-router-outlet>
  </ion-split-pane>
</ion-app>
```

src/app/app.routes.ts

```
import { Routes } from '@angular/router';
import { AuthGuard } from './guards/auth.guard';

export const routes: Routes = [
  {
    path: '',
    redirectTo: '/login',
    pathMatch: 'full'
  },
  {
    path: 'login',
    loadComponent: () => import('./pages/login/login.page').then(m => m.LoginPage)
  },
  {
    path: 'register',
    loadComponent: () => import('./pages/register/register.page').then(m => m.RegisterPage)
  },
  {
    path: 'home',
    loadComponent: () => import('./home/home.page').then(m => m.HomePage),
    canActivate: [AuthGuard]
  },
  {
    path: 'savings-goal',
    loadComponent: () => import('./pages/savings-goal/savings-goal.page').then(m => m.SavingsGoalPage),
    canActivate: [AuthGuard]
  },
  {
    path: 'withdraw',
    loadComponent: () => import('./pages/withdraw/withdraw.page').then(m => m.WithdrawPage),
    canActivate: [AuthGuard]
  },
  {
    path: 'settings',
    loadComponent: () => import('./pages/settings/settings.page').then(m => m.SettingsPage),
    canActivate: [AuthGuard]
  },
  {
    path: 'landing',
    loadComponent: () => import('./pages/landing/landing.page').then(m => m.LandingPage),
    canActivate: [AuthGuard]
  },
  {
    path: '**',
    redirectTo: '/login'
  }
];
```

src/app/guards/auth.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  async canActivate(): Promise<boolean> {
    const isAuthenticated = await this.authService.isAuthenticated();
    if (!isAuthenticated) {
      this.router.navigate(['/login']);
      return false;
    }
    return true;
  }
}
```

src/app/services/api.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Preferences } from '@capacitor/preferences';

export interface ApiResponse<T = any> {
  success: boolean;
  message?: string;
  data?: T;
  errors?: any;
}

@Injectable({
  providedIn: 'root'
})
export class ApiService {
  private baseUrl = 'https://ecg.codepps.online/api'; // Change this to your Laravel backend URL
  private readonly TOKEN_KEY = 'auth_token';

  constructor(private http: HttpClient) {}

  private async getHeaders(): Promise<HttpHeaders> {
    const token = await this.getToken();
    let headers = new HttpHeaders({
      'Content-Type': 'application/json',
      'Accept': 'application/json'
    });

    if (token) {
      headers = headers.set('Authorization', `Bearer ${token}`);
    }

    return headers;
  }

  async getToken(): Promise<string | null> {
    const result = await Preferences.get({ key: this.TOKEN_KEY });
    return result.value;
  }

  async setToken(token: string): Promise<void> {
    await Preferences.set({ key: this.TOKEN_KEY, value: token });
  }

  async removeToken(): Promise<void> {
    await Preferences.remove({ key: this.TOKEN_KEY });
  }

  private handleError(error: any): Observable<never> {
    console.error('API Error:', error);
    return throwError(() => error);
  }

  // Generic HTTP methods
  async get<T>(endpoint: string): Promise<ApiResponse<T>> {
    const headers = await this.getHeaders();
    return this.http.get<ApiResponse<T>>(`${this.baseUrl}${endpoint}`, { headers })
      .pipe(catchError(this.handleError))
      .toPromise() as Promise<ApiResponse<T>>;
  }

  async post<T>(endpoint: string, data: any): Promise<ApiResponse<T>> {
    const headers = await this.getHeaders();
    return this.http.post<ApiResponse<T>>(`${this.baseUrl}${endpoint}`, data, { headers })
      .pipe(catchError(this.handleError))
      .toPromise() as Promise<ApiResponse<T>>;
```

```
}

async put<T>(endpoint: string, data: any): Promise<ApiResponse<T>> {
  const headers = await this.getHeaders();
  return this.http.put<ApiResponse<T>>(`${this.baseUrl}${endpoint}`, data, { headers })
    .pipe(catchError(this.handleError))
    .toPromise() as Promise<ApiResponse<T>>;
}

async delete<T>(endpoint: string): Promise<ApiResponse<T>> {
  const headers = await this.getHeaders();
  return this.http.delete<ApiResponse<T>>(`${this.baseUrl}${endpoint}`, { headers })
    .pipe(catchError(this.handleError))
    .toPromise() as Promise<ApiResponse<T>>;
}

// Authentication endpoints
async register(userData: any): Promise<ApiResponse> {
  return this.post('/register', userData);
}

async login(credentials: any): Promise<ApiResponse> {
  return this.post('/loginUser', credentials);
}

async logout(): Promise<ApiResponse> {
  return this.post('/user/logout', {});
}

// User endpoints
async getUserProfile(): Promise<ApiResponse> {
  return this.get('/user/profile');
}

async updateUserProfile(data: any): Promise<ApiResponse> {
  return this.put('/user/profile', data);
}

async getDashboard(): Promise<ApiResponse> {
  return this.get('/user/dashboard');
}

async getHistory(): Promise<ApiResponse> {
  return this.get('/user/history');
}

async updateNetIncome(netIncome: number): Promise<ApiResponse> {
  return this.put('/user/net-income', { net_income: netIncome });
}

// Savings Goals endpoints
async getSavingsGoals(): Promise<ApiResponse> {
  return this.get('/savings-goals');
}

async createSavingsGoal(goal: any): Promise<ApiResponse> {
  return this.post('/savings-goals', goal);
}

async updateSavingsGoal(id: number, goal: any): Promise<ApiResponse> {
  return this.put(`/savings-goals/${id}`, goal);
}

async deleteSavingsGoal(id: number): Promise<ApiResponse> {
  return this.delete(`/savings-goals/${id}`);
}

async setPrimaryGoal(id: number): Promise<ApiResponse> {
  return this.put(`/savings-goals/${id}/set-primary`, {});
```

```
  }

  async getPrimaryGoal(): Promise<ApiResponse> {
    return this.get('/savings-goals/primary');
  }

  // Savings Entries endpoints
  async getSavingsEntries(): Promise<ApiResponse> {
    return this.get('/savings-entries');
  }

  async createSavingsEntry(entry: any): Promise<ApiResponse> {
    return this.post('/savings-entries', entry);
  }

  async updateSavingsEntry(id: number, entry: any): Promise<ApiResponse> {
    return this.put(`/savings-entries/${id}`, entry);
  }

  async deleteSavingsEntry(id: number): Promise<ApiResponse> {
    return this.delete(`/savings-entries/${id}`);
  }

  async getTotalSavings(): Promise<ApiResponse> {
    return this.get('/savings-entries/total-savings');
  }

  // Withdrawal Entries endpoints
  async getWithdrawalEntries(): Promise<ApiResponse> {
    return this.get('/withdrawal-entries');
  }

  async createWithdrawalEntry(entry: any): Promise<ApiResponse> {
    return this.post('/withdrawal-entries', entry);
  }

  async updateWithdrawalEntry(id: number, entry: any): Promise<ApiResponse> {
    return this.put(`/withdrawal-entries/${id}`, entry);
  }

  async deleteWithdrawalEntry(id: number): Promise<ApiResponse> {
    return this.delete(`/withdrawal-entries/${id}`);
  }
}
```

src/app/services/auth.service.ts

```
import { Injectable } from '@angular/core';
import { ApiService } from './api.service';
import { Router } from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private currentUser: any = null;

  constructor(private apiService: ApiService, private router: Router) {}

  async login(credentials: any): Promise<boolean> {
    try {
      const response = await this.apiService.login(credentials);
      if (response.success && response.data.token) {
        await this.apiService.setToken(response.data.token);
        this.currentUser = response.data.user;
        return true;
      }
      return false;
    } catch (error) {
      console.error('Login error:', error);
      return false;
    }
  }

  async register(userData: any): Promise<boolean> {
    try {
      const response = await this.apiService.register(userData);
      if (response.success && response.data.token) {
        await this.apiService.setToken(response.data.token);
        this.currentUser = response.data.user;
        return true;
      }
      return false;
    } catch (error) {
      console.error('Registration error:', error);
      return false;
    }
  }

  async logout(): Promise<void> {
    try {
      await this.apiService.logout();
    } catch (error) {
      console.error('Logout error:', error);
    } finally {
      await this.apiService.removeToken();
      this.currentUser = null;
      this.router.navigate(['/login']);
    }
  }

  async isAuthenticated(): Promise<boolean> {
    const token = await this.apiService.getToken();
    return !!token;
  }

  getCurrentUser() {
    return this.currentUser;
  }
}
```

src/app/home/home.page.ts

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
```

```
import { IonicModule, AlertController, ModalController, ToastController } from '@ionic/angular';
import { Router } from '@angular/router';
import { AuthService } from '../services/auth.service';
import { ApiService } from '../services/api.service';
import { DataService } from '../services/data.service';
import { SettingsService } from '../services/settings.service';
import { SyncService } from '../services/sync.service';
import { NetworkService } from '../services/network.service';
import { SavingsTipService } from '../services/savings-tip.service';
import { SyncStatusComponent } from '../components/sync-status/sync-status.component';
import { PaystackService } from '../services/paystack.service';
import { jsPDF } from 'jspdf';
import * as XLSX from 'xlsx';

@Component({
  selector: 'app-home',
  templateUrl: './home.page.html',
  styleUrls: ['./home.page.scss'],
  standalone: true,
  imports: [
    CommonModule,
    FormsModule,
    IonicModule,
    SyncStatusComponent
  ],
})
export class HomePage implements OnInit, OnDestroy {
  currentUser: any = null;
  totalSavings: number = 0;
  netIncome: number = 0;
  amountSaved: number = 0;
  history: any[] = [];
  primaryGoal: any = null;
  profilePicture: string | undefined;
  showBot: boolean = false;
  botMessage: string = '';
  isManualSyncing: boolean = false;
  syncStatus: any = null;

  private syncStatusSubscription: any;
  private networkSubscription: any;

  constructor(
    private authService: AuthService,
    private apiService: ApiService,
    private dataService: DataService,
    private settingsService: SettingsService,
    private syncService: SyncService,
    private networkService: NetworkService,
    private savingsTipService: SavingsTipService,
    private alertController: AlertController,
    private modalController: ModalController,
    private toastController: ToastController,
    private router: Router,
    private paystackService: PaystackService
  ) {}

  async ngOnInit() {
    this.currentUser = this.authService.getCurrentUser();
    await this.loadData();
    this.setupSubscriptions();
  }

  ngOnDestroy() {
    if (this.syncStatusSubscription) {
      this.syncStatusSubscription.unsubscribe();
    }
    if (this.networkSubscription) {
      this.networkSubscription.unsubscribe();
```

```
  }
}

private setupSubscriptions() {
  this.syncStatusSubscription = this.syncService.syncStatus$.subscribe(status => {
    this.syncStatus = status;
  });
  this.networkSubscription = this.networkService.isOnline$.subscribe(isOnline => {
    console.log('Network status changed:', isOnline);
  });
}

async loadData() {
  try {
    await this.loadUserData();
    await this.loadSavingsData();
    await this.loadHistory();
    await this.loadProfilePicture();
  } catch (error) {
    console.error('Error loading data:', error);
  }
}

async loadUserData() {
  try {
    const response = await this.apiService.getUserProfile();
    if (response.success) {
      this.currentUser = response.data;
      this.netIncome = response.data.net_income || 0;
    }
  } catch (error) {
    console.error('Error loading user data:', error);
  }
}

async loadSavingsData() {
  try {
    const totalResponse = await this.apiService.getTotalSavings();
    if (totalResponse.success) {
      this.totalSavings = totalResponse.data.total || 0;
    }

    const goalResponse = await this.apiService.getPrimaryGoal();
    if (goalResponse.success) {
      this.primaryGoal = goalResponse.data;
    }
  } catch (error) {
    console.error('Error loading savings data:', error);
  }
}

async loadHistory() {
  try {
    const response = await this.apiService.getHistory();
    if (response.success) {
      this.history = response.data;
    }
  } catch (error) {
    console.error('Error loading history:', error);
  }
}

async loadProfilePicture() {
  const settings = await this.settingsService.getSettings();
  this.profilePicture = settings.profilePicture;
}

getUserDisplayName(): string {
  return this.currentUser?.name || 'User';
```

```
}

getUserEmail(): string {
  return this.currentUser?.email || '';
}

async addSavings() {
  if (this.amountSaved <= 0) {
    this.showToast('Please enter a valid amount to save');
    return;
  }

  try {
    const entry = {
      amount_saved: this.amountSaved,
      date: new Date().toISOString(),
      id: Date.now()
    };

    await this.dataService.addHistoryEntry({
      type: 'deposit',
      netIncome: this.netIncome,
      amountSaved: this.amountSaved,
      date: entry.date,
      id: entry.id
    });

    this.totalSavings += this.amountSaved;
    this.amountSaved = 0;
    this.showBotMessage();
    this.showToast('Savings added successfully!');
  } catch (error) {
    console.error('Error adding savings:', error);
    this.showToast('Error adding savings');
  }
}

async openWithdraw() {
  this.router.navigate(['/withdraw']);
}

async editEntry(entry: any) {
  // Implementation for editing entry
}

async deleteEntry(id: number) {
  const alert = await this.alertController.create({
    header: 'Confirm Delete',
    message: 'Are you sure you want to delete this entry?',
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel'
      },
      {
        text: 'Delete',
        handler: async () => {
          try {
            await this.dataService.deleteHistoryEntry(id);
            this.loadHistory();
            this.showToast('Entry deleted successfully');
          } catch (error) {
            console.error('Error deleting entry:', error);
            this.showToast('Error deleting entry');
          }
        }
      }
    ]
  });
```

```
    await alert.present();
  }

  async paymentInit() {
    if (this.amountSaved <= 0) {
      this.showToast('Please enter a valid amount to save');
      return;
    }

    try {
      await this.paystackService.initializePayment(this.amountSaved, this.currentUser.email);
    } catch (error) {
      console.error('Payment initialization error:', error);
      this.showToast('Payment initialization failed');
    }
  }

  async exportToPdf() {
    const doc = new jsPDF();
    doc.text('Savings History', 20, 20);
    let y = 40;
    this.history.forEach(entry => {
      doc.text(`${entry.date}: ${entry.type === 'deposit' ? '+' : '-'} ${entry.amountSaved ||
entry.amountWithdrawn}`, 20, y);
      y += 10;
    });
    doc.save('savings-history.pdf');
  }

  async exportToExcel() {
    const ws = XLSX.utils.json_to_sheet(this.history);
    const wb = XLSX.utils.book_new();
    XLSX.utils.book_append_sheet(wb, ws, 'History');
    XLSX.writeFile(wb, 'savings-history.xlsx');
  }

  async manualSync() {
    this.isManualSyncing = true;
    try {
      await this.syncService.forcSync();
      this.showToast('Sync completed successfully');
    } catch (error) {
      console.error('Sync error:', error);
      this.showToast('Sync failed');
    } finally {
      this.isManualSyncing = false;
    }
  }

  isSyncAvailable(): boolean {
    return this.networkService.isOnline;
  }

  getPendingChangesCount(): number {
    return this.syncStatus?.pendingItems || 0;
  }

  getSyncStatusText(): string {
    if (this.syncStatus?.isLoading || this.isManualSyncing) {
      return 'Syncing...';
    }
    if (this.getPendingChangesCount() > 0) {
      return `${this.getPendingChangesCount()} changes pending`;
    }
    return 'All synced';
  }

  showSyncDetails() {
    // Implementation for showing sync details
```

```
  }

  async openSettings() {
    this.router.navigate(['/settings']);
  }

  async logout() {
    await this.authService.logout();
  }

  private showBotMessage() {
    this.botMessage = this.savingsTipService.getTip(this.netIncome, this.amountSaved);
    this.showBot = true;
  }

  hideBot() {
    this.showBot = false;
  }

  private async showToast(message: string) {
    const toast = await this.toastController.create({
      message,
      duration: 2000,
      position: 'bottom'
    });
    await toast.present();
  }
}
```

src/app/home/home.page.html

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-menu-button></ion-menu-button>
    </ion-buttons>
    <ion-title>Welcome, {{ getUserDisplayName() }}</ion-title>
    <ion-buttons slot="end">
      <!-- Sync Status Button -->
      <ion-button fill="clear" (click)="showSyncDetails()" [disabled]="!isSyncAvailable()">
        <ion-icon
          [name]="syncStatus?.isLoading || isManualSyncing ? 'sync' : (getPendingChangesCount() > 0 ?
'cloud-upload-outline' : 'cloud-done-outline')"
          [class.spinning]="syncStatus?.isLoading || isManualSyncing">
        </ion-icon>
        <ion-badge color="warning" *ngIf="getPendingChangesCount() > 0">{{ getPendingChangesCount() }}
</ion-badge>
      </ion-button>
      <ion-button fill="clear" (click)="openSettings()">
        <ion-icon name="settings-outline"></ion-icon>
      </ion-button>
      <ion-button fill="clear" (click)="logout()">
        <ion-icon name="log-out-outline"></ion-icon>
      </ion-button>
      <ion-avatar style="width: 42px; height: 42px; margin-left: 8px;">
        <img [src]="profilePicture || 'assets/img/u1.png'" />
      </ion-avatar>
    </ion-buttons>
  </ion-toolbar>
</ion-header>

<ion-content>
  <!-- Sync Status Component -->
  <app-sync-status></app-sync-status>

  <!-- Enhanced Sync Status Card -->
  <ion-card *ngIf="syncStatus?.isLoading || isManualSyncing || getPendingChangesCount() > 0">
    <ion-card-content>
      <div class="sync-status-card">
```

```
      <ion-icon
        [name]="syncStatus?.isLoading || isManualSyncing ? 'sync' : 'cloud-upload-outline'"
        [class.spinning]="syncStatus?.isLoading || isManualSyncing"
        color="primary">
      </ion-icon>
      <div class="sync-info">
        <h3>{{ getSyncStatusText() }}</h3>
        <p *ngIf="getPendingChangesCount() > 0">{{ getPendingChangesCount() }} changes pending sync</p>
        <p *ngIf="syncStatus?.error" class="error-text">{{ syncStatus.error }}</p>
      </div>
      <ion-button
        *ngIf="isSyncAvailable() && getPendingChangesCount() > 0"
        fill="clear"
        size="small"
        (click)="manualSync()">
        Sync Now
      </ion-button>
    </div>
    <ion-progress-bar
      *ngIf="syncStatus?.isLoading || isManualSyncing"
      [value]="syncStatus?.progress ? syncStatus.progress / 100 : 0">
    </ion-progress-bar>
  </ion-card-content>
</ion-card>

<!-- User Welcome Card -->
<div class="welcome-card">
  <ion-card>
    <ion-card-content>
      <div class="user-info">
        <ion-avatar class="user-avatar">
          <img [src]="profilePicture || 'assets/img/u1.png'" />
        </ion-avatar>
        <div class="user-details">
          <h2>{{ getUserDisplayName() }}</h2>
          <p>{{ getUserEmail() }}</p>
          <ion-chip color="primary" *ngIf="currentUser?.net_income">
            <ion-icon name="cash-outline"></ion-icon>
            <ion-label>{{ netIncome | currency:'GHS' }} monthly</ion-label>
          </ion-chip>
        </div>
      </div>
    </ion-card-content>
  </ion-card>
</div>

<div class="card">
  <ion-card-header>
    <ion-card-title>Total Savings</ion-card-title>
  </ion-card-header>
  <ion-card-content>
    <h2>{{ totalSavings | currency:'GHS' }}</h2>
  </ion-card-content>
</div>

<div class="card" *ngIf="primaryGoal">
  <ion-card-header>
    <ion-card-title>{{ primaryGoal.name }}</ion-card-title>
  </ion-card-header>
  <ion-card-content>
    <p>Target: {{ primaryGoal.targetAmount | currency:'GHS' }}</p>
    <p>Saved: {{ primaryGoal.currentAmount | currency:'GHS' }}</p>
    <ion-progress-bar [value]="primaryGoal.currentAmount / primaryGoal.targetAmount"></ion-progress-bar>
  </ion-card-content>
</div>

<div class="card">
  <ion-item>
    <ion-label color="medium" position="floating">Net Income</ion-label>
```

```
      <ion-input type="number" disabled [(ngModel)]="netIncome"></ion-input>
    </ion-item>

    <ion-item>
      <ion-label color="medium" position="floating">Amount Saved</ion-label>
      <ion-input type="number" [(ngModel)]="amountSaved"></ion-input>
    </ion-item>



    <button (click)="paymentInit()">Deposit <ion-text *ngIf="amountSaved! > 0" style="margin-left:8px;">
- <span style="margin-left:8px;">GHS {{ amountSaved}}</span></ion-text>
    </button>

    <!-- <ion-button expand="full" (click)="addSavings()">Save</ion-button> -->
    <ion-button expand="full" color="secondary" (click)="openWithdraw()">Withdraw</ion-button>
  </div>

  <div class="card">
    <ion-list>
      <ion-list-header>
        <h3>History</h3>
      </ion-list-header>
      <ion-item-sliding *ngFor="let entry of history">
        <ion-item>
          <ion-label>
            <h2 [ngStyle]="{'color': entry.type === 'deposit' ? 'green' : 'red'}">
              {{ entry.type === 'deposit' ? '+' : '-' }} {{ (entry.type === 'deposit' ? entry.amountSaved
: entry.amountWithdrawn) | currency:'GHS' }}
            </h2>
            <p>{{ entry.date | date }} <span *ngIf="entry.type === 'withdrawal' && entry.goalName">from {{
entry.goalName }}</span></p>
          </ion-label>
        </ion-item>
        <ion-item-options side="end">
          <ion-item-option (click)="editEntry(entry)">Edit</ion-item-option>
          <ion-item-option color="danger" (click)="deleteEntry(entry.id)">Delete</ion-item-option>
        </ion-item-options>
      </ion-item-sliding>
    </ion-list>
    <div class="export-buttons">
      <ion-button expand="full" (click)="exportToPdf()">Export as PDF</ion-button>
      <ion-button expand="full" (click)="exportToExcel()">Export as Excel</ion-button>
      <ion-button
        expand="full"
        fill="outline"
        color="primary"
        (click)="manualSync()"
        [disabled]="!isSyncAvailable()">
        <ion-icon name="sync" slot="start" [class.spinning]="isManualSyncing"></ion-icon>
        {{ isManualSyncing ? 'Syncing...' : 'Sync Data' }}
      </ion-button>
    </div>
  </div>

  <div class="bot-popup" *ngIf="showBot">
    <div class="bot-message">
      <p>{{ botMessage }}</p>
      <ion-button (click)="hideBot()">Close</ion-button>
    </div>
  </div>
</ion-content>
```

Backend Files

laravel-backend/app/Http/Controllers/Api/AuthController.php

```
<?php
```

```php
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\SUser;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:susers',
            'password' => 'required|string|min:8',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation failed',
                'errors' => $validator->errors()
            ], 422);
        }

        $user = SUser::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'success' => true,
            'message' => 'User registered successfully',
            'data' => [
                'user' => $user,
                'token' => $token
            ]
        ]);
    }

    public function login(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'email' => 'required|string|email',
            'password' => 'required|string',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation failed',
                'errors' => $validator->errors()
            ], 422);
        }

        if (!Auth::attempt($request->only('email', 'password'))) {
            return response()->json([
                'success' => false,
                'message' => 'Invalid credentials'
            ], 401);
        }

        $user = Auth::user();
        $token = $user->createToken('auth_token')->plainTextToken;
```

```
        return response()->json([
            'success' => true,
            'message' => 'Login successful',
            'data' => [
                'user' => $user,
                'token' => $token
            ]
        ]);
    }

    public function logout(Request $request)
    {
        $request->user()->currentAccessToken()->delete();

        return response()->json([
            'success' => true,
            'message' => 'Logged out successfully'
        ]);
    }

    public function profile(Request $request)
    {
        return response()->json([
            'success' => true,
            'data' => $request->user()
        ]);
    }

    public function updateProfile(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'name' => 'string|max:255',
            'email' => 'string|email|max:255|unique:susers,email,' . $request->user()->id,
            'net_income' => 'numeric|min:0',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
                'message' => 'Validation failed',
                'errors' => $validator->errors()
            ], 422);
        }

        $user = $request->user();
        $user->update($request->only(['name', 'email', 'net_income']));

        return response()->json([
            'success' => true,
            'message' => 'Profile updated successfully',
            'data' => $user
        ]);
    }

    public function dashboard(Request $request)
    {
        $user = $request->user();
        $totalSavings = $user->savingsEntries()->sum('amount_saved');
        $totalWithdrawals = $user->withdrawalEntries()->sum('amount_withdrawn');
        $goals = $user->savingsGoals;

        return response()->json([
            'success' => true,
            'data' => [
                'total_savings' => $totalSavings,
                'total_withdrawals' => $totalWithdrawals,
                'goals' => $goals
            ]
```

```
        ]);
    }
```