```dart
import 'package:flutter/material.dart';
import 'package:flutter_stripe/flutter_stripe.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'package:provider/provider.dart';

// Configuration
const String apiUrl = 'https://your-backend-api.com';
const String stripePublishableKey = 'pk_test_your_publishable_key';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Initialize Stripe
  Stripe.publishableKey = stripePublishableKey;
  await Stripe.instance.applySettings();

  runApp(
    ChangeNotifierProvider(
      create: (context) => BiddingProvider(),
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Auction App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: HomePage(),
    );
  }
}

// Models
class Auction {
  final String id;
  final String title;
  final String description;
  final double startingPrice;
  final DateTime endTime;
  final String imageUrl;
```

```
  Auction({
    required this.id,
    required this.title,
    required this.description,
    required this.startingPrice,
    required this.endTime,
    required this.imageUrl,
  });

  factory Auction.fromJson(Map<String, dynamic> json) {
    return Auction(
      id: json['id'],
      title: json['title'],
      description: json['description'],
      startingPrice: json['startingPrice'].toDouble(),
      endTime: DateTime.parse(json['endTime']),
      imageUrl: json['imageUrl'],
    );
  }
}

class Bid {
  final String id;
  final String auctionId;
  final String userId;
  final double amount;
  final DateTime createdAt;
  String status; // "pending", "won", "lost", "paid"

  Bid({
    required this.id,
    required this.auctionId,
    required this.userId,
    required this.amount,
    required this.createdAt,
    required this.status,
  });

  factory Bid.fromJson(Map<String, dynamic> json) {
    return Bid(
      id: json['id'],
      auctionId: json['auctionId'],
      userId: json['userId'],
      amount: json['amount'].toDouble(),
      createdAt: DateTime.parse(json['createdAt']),
      status: json['status'],
    );
  }
```

```
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'auctionId': auctionId,
      'userId': userId,
      'amount': amount,
      'createdAt': createdAt.toIso8601String(),
      'status': status,
    };
  }
}

// State Management
class BiddingProvider with ChangeNotifier {
  List<Auction> _auctions = [];
  List<Bid> _myBids = [];
  String? _currentUserId = 'user123'; // In a real app, get from auth

  List<Auction> get auctions => _auctions;
  List<Bid> get myBids => _myBids;
  String? get currentUserId => _currentUserId;

  Future<void> fetchAuctions() async {
    final response = await http.get(Uri.parse('$apiUrl/auctions'));

    if (response.statusCode == 200) {
      final List<dynamic> auctionsJson = json.decode(response.body);
      _auctions = auctionsJson.map((json) => Auction.fromJson(json)).toList();
      notifyListeners();
    } else {
      throw Exception('Failed to load auctions');
    }
  }

  Future<void> fetchMyBids() async {
    if (_currentUserId == null) return;

    final response = await http.get(
      Uri.parse('$apiUrl/bids?userId=$_currentUserId'),
    );

    if (response.statusCode == 200) {
      final List<dynamic> bidsJson = json.decode(response.body);
      _myBids = bidsJson.map((json) => Bid.fromJson(json)).toList();
      notifyListeners();
    } else {
      throw Exception('Failed to load bids');
```

```
  }
}

Future<Bid> placeBid(String auctionId, double amount) async {
  if (_currentUserId == null) {
    throw Exception('User not logged in');
  }

  final response = await http.post(
    Uri.parse('$apiUrl/bids'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      'auctionId': auctionId,
      'userId': _currentUserId,
      'amount': amount,
    }),
  );

  if (response.statusCode == 201) {
    final bidJson = json.decode(response.body);
    final newBid = Bid.fromJson(bidJson);
    _myBids.add(newBid);
    notifyListeners();
    return newBid;
  } else {
    throw Exception('Failed to place bid');
  }
}

Future<Map<String, dynamic>> createPaymentIntent(String bidId) async {
  final response = await http.post(
    Uri.parse('$apiUrl/create-payment-intent'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({'bidId': bidId}),
  );

  if (response.statusCode == 200) {
    return json.decode(response.body);
  } else {
    throw Exception('Failed to create payment intent');
  }
}

Future<void> updateBidStatus(String bidId, String status) async {
  final response = await http.patch(
    Uri.parse('$apiUrl/bids/$bidId'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({'status': status}),
```

```
    );

    if (response.statusCode == 200) {
      final index = _myBids.indexWhere((bid) => bid.id == bidId);
      if (index != -1) {
        _myBids[index].status = status;
        notifyListeners();
      }
    } else {
      throw Exception('Failed to update bid status');
    }
  }
}

// UI Components
class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  int _selectedIndex = 0;

  static final List<Widget> _pages = [
    AuctionsPage(),
    MyBidsPage(),
    ProfilePage(),
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _selectedIndex,
        onTap: (index) {
          setState(() {
            _selectedIndex = index;
          });
        },
        items: [
          BottomNavigationBarItem(
            icon: Icon(Icons.gavel),
            label: 'Auctions',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.list),
            label: 'My Bids',
```

```
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.person),
            label: 'Profile',
          ),
        ],
      ),
    );
  }
}

class AuctionsPage extends StatefulWidget {
  @override
  _AuctionsPageState createState() => _AuctionsPageState();
}

class _AuctionsPageState extends State<AuctionsPage> {
  bool _isLoading = true;

  @override
  void initState() {
    super.initState();
    _loadAuctions();
  }

  Future<void> _loadAuctions() async {
    final provider = Provider.of<BiddingProvider>(context, listen: false);
    setState(() => _isLoading = true);
    try {
      await provider.fetchAuctions();
    } finally {
      setState(() => _isLoading = false);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Active Auctions'),
        actions: [
          IconButton(
            icon: Icon(Icons.refresh),
            onPressed: _loadAuctions,
          ),
        ],
      ),
      body: _isLoading
```

```
        ? Center(child: CircularProgressIndicator())
        : Consumer<BiddingProvider>(
            builder: (context, provider, child) {
              final auctions = provider.auctions;

              if (auctions.isEmpty) {
                return Center(
                  child: Text('No active auctions found'),
                );
              }

              return ListView.builder(
                itemCount: auctions.length,
                itemBuilder: (context, index) {
                  final auction = auctions[index];
                  return AuctionCard(auction: auction);
                },
              );
            },
          ),
    );
  }
}

class AuctionCard extends StatelessWidget {
  final Auction auction;

  const AuctionCard({Key? key, required this.auction}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Image.network(
            auction.imageUrl,
            height: 180,
            width: double.infinity,
            fit: BoxFit.cover,
          ),
          Padding(
            padding: EdgeInsets.all(16),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
```

```
  auction.title,
  style: TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
  ),
),
SizedBox(height: 8),
Text(auction.description),
SizedBox(height: 16),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text('Starting Price'),
        Text(
          '\$${auction.startingPrice.toStringAsFixed(2)}',
          style: TextStyle(
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
    Column(
      crossAxisAlignment: CrossAxisAlignment.end,
      children: [
        Text('Ends'),
        Text(
          _formatEndTime(auction.endTime),
          style: TextStyle(
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ],
),
SizedBox(height: 16),
SizedBox(
  width: double.infinity,
  child: ElevatedButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => AuctionDetailPage(auction: auction),
        ),
```

```
                    );
                  },
                  child: Text('View Details'),
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  );
 }

 String _formatEndTime(DateTime endTime) {
  final now = DateTime.now();
  final difference = endTime.difference(now);

  if (difference.isNegative) {
   return 'Ended';
  }

  if (difference.inDays > 0) {
   return '${difference.inDays}d ${difference.inHours % 24}h';
  }

  if (difference.inHours > 0) {
   return '${difference.inHours}h ${difference.inMinutes % 60}m';
  }

  return '${difference.inMinutes}m ${difference.inSeconds % 60}s';
 }
}

class AuctionDetailPage extends StatefulWidget {
 final Auction auction;

 const AuctionDetailPage({Key? key, required this.auction}) : super(key: key);

 @override
 _AuctionDetailPageState createState() => _AuctionDetailPageState();
}

class _AuctionDetailPageState extends State<AuctionDetailPage> {
 final _bidController = TextEditingController();
 bool _placingBid = false;

 @override
 Widget build(BuildContext context) {
```

```
return Scaffold(
  appBar: AppBar(
    title: Text('Auction Details'),
  ),
  body: SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Image.network(
          widget.auction.imageUrl,
          height: 240,
          width: double.infinity,
          fit: BoxFit.cover,
        ),
        Padding(
          padding: EdgeInsets.all(16),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                widget.auction.title,
                style: TextStyle(
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                ),
              ),
              SizedBox(height: 16),
              Text(
                widget.auction.description,
                style: TextStyle(fontSize: 16),
              ),
              SizedBox(height: 24),
              Card(
                child: Padding(
                  padding: EdgeInsets.all(16),
                  child: Column(
                    children: [
                      Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          Text('Starting Price:'),
                          Text(
                            '\$${widget.auction.startingPrice.toStringAsFixed(2)}',
                            style: TextStyle(fontWeight: FontWeight.bold),
                          ),
                        ],
                      ),
                      SizedBox(height: 8),
```

```
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Text('Auction Ends:'),
                    Text(
                      '${widget.auction.endTime.toString().substring(0, 16)}',
                      style: TextStyle(fontWeight: FontWeight.bold),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
        SizedBox(height: 32),
        Text(
          'Place Your Bid',
          style: TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.bold,
          ),
        ),
        SizedBox(height: 16),
        TextField(
          controller: _bidController,
          keyboardType: TextInputType.numberWithOptions(decimal: true),
          decoration: InputDecoration(
            labelText: 'Bid Amount',
            prefixText: '\$',
            border: OutlineInputBorder(),
          ),
        ),
        SizedBox(height: 16),
        SizedBox(
          width: double.infinity,
          child: ElevatedButton(
            onPressed: _placingBid ? null : _placeBid,
            child: _placingBid
                ? CircularProgressIndicator(color: Colors.white)
                : Text('Place Bid'),
          ),
        ),
      ],
    ),
  ),
),
```

```
    );
  }

  Future<void> _placeBid() async {
    final bidAmount = double.tryParse(_bidController.text);
    if (bidAmount == null || bidAmount <= 0) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Please enter a valid bid amount')),
      );
      return;
    }

    if (bidAmount < widget.auction.startingPrice) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Bid must be higher than starting price')),
      );
      return;
    }

    setState(() => _placingBid = true);

    try {
      final provider = Provider.of<BiddingProvider>(context, listen: false);
      await provider.placeBid(widget.auction.id, bidAmount);

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Bid placed successfully')),
      );

      _bidController.clear();
      Navigator.pop(context);
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Failed to place bid: ${e.toString()}')),
      );
    } finally {
      setState(() => _placingBid = false);
    }
  }
}

class MyBidsPage extends StatefulWidget {
  @override
  _MyBidsPageState createState() => _MyBidsPageState();
}

class _MyBidsPageState extends State<MyBidsPage> {
  bool _isLoading = true;
```

```
@override
void initState() {
  super.initState();
  _loadBids();
}

Future<void> _loadBids() async {
  final provider = Provider.of<BiddingProvider>(context, listen: false);
  setState(() => _isLoading = true);
  try {
    await provider.fetchMyBids();
  } finally {
    setState(() => _isLoading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('My Bids'),
      actions: [
        IconButton(
          icon: Icon(Icons.refresh),
          onPressed: _loadBids,
        ),
      ],
    ),
    body: _isLoading
        ? Center(child: CircularProgressIndicator())
        : Consumer<BiddingProvider>(
            builder: (context, provider, child) {
              final bids = provider.myBids;

              if (bids.isEmpty) {
                return Center(
                  child: Text('You have not placed any bids yet'),
                );
              }

              return ListView.builder(
                itemCount: bids.length,
                itemBuilder: (context, index) {
                  final bid = bids[index];
                  return BidCard(bid: bid);
                },
              );
```

```
            },
          ),
    );
  }
}

class BidCard extends StatelessWidget {
  final Bid bid;

  const BidCard({Key? key, required this.bid}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      child: Padding(
        padding: EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Text(
                  'Auction: ${bid.auctionId.substring(0, 8)}...',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                  ),
                ),
                _buildStatusChip(bid.status),
              ],
            ),
            SizedBox(height: 16),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Text('Bid Amount:'),
                Text(
                  '\$${bid.amount.toStringAsFixed(2)}',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16,
                  ),
                ),
              ],
            ),
            SizedBox(height: 8),
            Row(
```

```
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text('Date:'),
              Text(bid.createdAt.toString().substring(0, 16)),
            ],
          ),
          SizedBox(height: 16),
          if (bid.status == 'won')
            SizedBox(
              width: double.infinity,
              child: ElevatedButton(
                onPressed: () => _handlePayment(context, bid),
                child: Text('Complete Payment'),
              ),
            ),
        ],
      ),
    ),
  );
}

Widget _buildStatusChip(String status) {
  Color color;
  String label;

  switch (status) {
    case 'pending':
      color = Colors.orange;
      label = 'Pending';
      break;
    case 'won':
      color = Colors.green;
      label = 'Won';
      break;
    case 'lost':
      color = Colors.red;
      label = 'Lost';
      break;
    case 'paid':
      color = Colors.blue;
      label = 'Paid';
      break;
    default:
      color = Colors.grey;
      label = status;
  }

  return Chip(
```

```
      label: Text(
        label,
        style: TextStyle(color: Colors.white),
      ),
      backgroundColor: color,
    );
  }

  Future<void> _handlePayment(BuildContext context, Bid bid) async {
    final provider = Provider.of<BiddingProvider>(context, listen: false);

    try {
      // Get payment intent from your backend
      final paymentIntentData = await provider.createPaymentIntent(bid.id);

      // Initialize payment sheet
      await Stripe.instance.initPaymentSheet(
        paymentSheetParameters: SetupPaymentSheetParameters(
          paymentIntentClientSecret: paymentIntentData['clientSecret'],
          merchantDisplayName: 'Auction App',
          style: ThemeMode.light,
        ),
      );

      // Present payment sheet
      await Stripe.instance.presentPaymentSheet();

      // If we get here, payment succeeded
      await provider.updateBidStatus(bid.id, 'paid');

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Payment successful')),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Error: ${e.toString()}')),
      );
    }
  }
}

class ProfilePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Profile'),
      ),
```

```
      body: Center(
        child: Text('Profile page - implement user settings, history, etc.'),
      ),
    );
  }
}
```