```
// main.dart
import 'package:flutter/material.dart';
import 'dart:async';

void main() {
  runApp(BiddingApp());
}

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

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

class _HomePageState extends State<HomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Bidding System'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => CreateAuctionPage()),
                );
              },
              child: Text('Create Auction'),
            ),
            SizedBox(height: 20),
```

```
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => BrowseAuctionsPage()),
                );
              },
              child: Text('Browse Auctions'),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => PaymentHistoryPage()),
                );
              },
              child: Text('Payment History'),
            ),
          ],
        ),
      ),
    );
  }
}

// Models
class Bid {
  final String id;
  final String bidderId;
  final String bidderName;
  final double amount;
  final DateTime timestamp;

  Bid({
    required this.id,
    required this.bidderId,
    required this.bidderName,
    required this.amount,
    required this.timestamp,
  });
}

class Auction {
  final String id;
  final String itemName;
  final String description;
  final String imageUrl;
```

```
  final double startingPrice;
  final DateTime endTime;
  final List<Bid> bids;
  Bid? winningBid;

  Auction({
    required this.id,
    required this.itemName,
    required this.description,
    required this.imageUrl,
    required this.startingPrice,
    required this.endTime,
    required this.bids,
    this.winningBid,
  });
}

class Payment {
  final String id;
  final String auctionId;
  final String itemName;
  final String buyerId;
  final String buyerName;
  final double amount;
  final DateTime timestamp;
  final PaymentStatus status;

  Payment({
    required this.id,
    required this.auctionId,
    required this.itemName,
    required this.buyerId,
    required this.buyerName,
    required this.amount,
    required this.timestamp,
    required this.status,
  });
}

enum PaymentStatus { pending, completed, failed, refunded }

// Service layer
class AuctionService {
  // Simulating a database with in-memory lists
  static List<Auction> _auctions = [];
  static List<Payment> _payments = [];

  // Get all auctions
```

```
  List<Auction> getAllAuctions() {
    return _auctions;
  }

  // Get active auctions
  List<Auction> getActiveAuctions() {
    final now = DateTime.now();
    return _auctions.where((auction) => auction.endTime.isAfter(now)).toList();
  }

  // Create a new auction
  Auction createAuction(String itemName, String description, String imageUrl, double
startingPrice, int durationInMinutes) {
    final id = DateTime.now().millisecondsSinceEpoch.toString();
    final endTime = DateTime.now().add(Duration(minutes: durationInMinutes));

    final newAuction = Auction(
      id: id,
      itemName: itemName,
      description: description,
      imageUrl: imageUrl,
      startingPrice: startingPrice,
      endTime: endTime,
      bids: [],
    );

    _auctions.add(newAuction);
    return newAuction;
  }

  // Place a bid
  bool placeBid(String auctionId, String bidderId, String bidderName, double amount) {
    final auctionIndex = _auctions.indexWhere((auction) => auction.id == auctionId);

    if (auctionIndex == -1) return false;

    final auction = _auctions[auctionIndex];

    // Check if auction is still active
    if (DateTime.now().isAfter(auction.endTime)) return false;

    // Check if bid is high enough
    double minimumBid = auction.startingPrice;
    if (auction.bids.isNotEmpty) {
      minimumBid = auction.bids.map((bid) => bid.amount).reduce((a, b) => a > b ? a : b) + 1;
    }

    if (amount < minimumBid) return false;
```

```
  // Create and add the bid
  final newBid = Bid(
    id: DateTime.now().millisecondsSinceEpoch.toString(),
    bidderId: bidderId,
    bidderName: bidderName,
    amount: amount,
    timestamp: DateTime.now(),
  );

  _auctions[auctionIndex].bids.add(newBid);
  return true;
}

// End auction and determine winner
Bid? endAuction(String auctionId) {
  final auctionIndex = _auctions.indexWhere((auction) => auction.id == auctionId);
  if (auctionIndex == -1) return null;

  final auction = _auctions[auctionIndex];
  if (auction.bids.isEmpty) return null;

  // Find highest bid
  auction.bids.sort((a, b) => b.amount.compareTo(a.amount));
  final winningBid = auction.bids.first;

  _auctions[auctionIndex].winningBid = winningBid;

  // Create payment record
  final payment = Payment(
    id: DateTime.now().millisecondsSinceEpoch.toString(),
    auctionId: auctionId,
    itemName: auction.itemName,
    buyerId: winningBid.bidderId,
    buyerName: winningBid.bidderName,
    amount: winningBid.amount,
    timestamp: DateTime.now(),
    status: PaymentStatus.pending,
  );

  _payments.add(payment);
  return winningBid;
}

// Process payment
bool processPayment(String paymentId, bool isSuccessful) {
  final paymentIndex = _payments.indexWhere((payment) => payment.id == paymentId);
  if (paymentIndex == -1) return false;
```

```
    _payments[paymentIndex] = _payments[paymentIndex].copyWith(
      status: isSuccessful ? PaymentStatus.completed : PaymentStatus.failed,
    );

    return true;
  }

  // Get payment history
  List<Payment> getPaymentHistory() {
    return _payments;
  }
}

// UI Pages
class CreateAuctionPage extends StatefulWidget {
  @override
  _CreateAuctionPageState createState() => _CreateAuctionPageState();
}

class _CreateAuctionPageState extends State<CreateAuctionPage> {
  final _formKey = GlobalKey<FormState>();
  final _auctionService = AuctionService();

  String _itemName = '';
  String _description = '';
  String _imageUrl = '';
  double _startingPrice = 0.0;
  int _duration = 60; // minutes

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Create Auction'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                TextFormField(
                  decoration: InputDecoration(labelText: 'Item Name'),
                  validator: (value) {
                    if (value == null || value.isEmpty) {
```

```
      return 'Please enter item name';
    }
    return null;
  },
  onSaved: (value) {
    _itemName = value!;
  },
),
SizedBox(height: 12),
TextFormField(
  decoration: InputDecoration(labelText: 'Description'),
  maxLines: 3,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter description';
    }
    return null;
  },
  onSaved: (value) {
    _description = value!;
  },
),
SizedBox(height: 12),
TextFormField(
  decoration: InputDecoration(labelText: 'Image URL'),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter image URL';
    }
    return null;
  },
  onSaved: (value) {
    _imageUrl = value!;
  },
),
SizedBox(height: 12),
TextFormField(
  decoration: InputDecoration(labelText: 'Starting Price'),
  keyboardType: TextInputType.number,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter starting price';
    }
    if (double.tryParse(value) == null) {
      return 'Please enter a valid number';
    }
    return null;
  },
```

```
        onSaved: (value) {
          _startingPrice = double.parse(value!);
        },
      ),
      SizedBox(height: 12),
      TextFormField(
        decoration: InputDecoration(labelText: 'Duration (minutes)'),
        keyboardType: TextInputType.number,
        initialValue: '60',
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter duration';
          }
          if (int.tryParse(value) == null) {
            return 'Please enter a valid number';
          }
          return null;
        },
        onSaved: (value) {
          _duration = int.parse(value!);
        },
      ),
      SizedBox(height: 24),
      ElevatedButton(
        onPressed: () {
          if (_formKey.currentState!.validate()) {
            _formKey.currentState!.save();

            _auctionService.createAuction(
              _itemName,
              _description,
              _imageUrl,
              _startingPrice,
              _duration,
            );

            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('Auction created successfully')),
            );

            Navigator.pop(context);
          }
        },
        child: Text('Create Auction'),
      ),
    ],
  ),
),
```

```
        ),
      ),
    );
  }
}

class BrowseAuctionsPage extends StatefulWidget {
  @override
  _BrowseAuctionsPageState createState() => _BrowseAuctionsPageState();
}

class _BrowseAuctionsPageState extends State<BrowseAuctionsPage> {
  final _auctionService = AuctionService();
  late Timer _timer;

  @override
  void initState() {
    super.initState();
    // Refresh the UI every second to update remaining time
    _timer = Timer.periodic(Duration(seconds: 1), (timer) {
      setState(() {});
    });
  }

  @override
  void dispose() {
    _timer.cancel();
    super.dispose();
  }

  String _getRemainingTime(DateTime endTime) {
    final remaining = endTime.difference(DateTime.now());
    if (remaining.isNegative) return 'Ended';

    final hours = remaining.inHours;
    final minutes = remaining.inMinutes % 60;
    final seconds = remaining.inSeconds % 60;

    return '$hours:${minutes.toString().padLeft(2, '0')}:${seconds.toString().padLeft(2, '0')}';
  }

  @override
  Widget build(BuildContext context) {
    final activeAuctions = _auctionService.getActiveAuctions();

    return Scaffold(
      appBar: AppBar(
        title: Text('Browse Auctions'),
```

```
      ),
      body: activeAuctions.isEmpty
          ? Center(child: Text('No active auctions available'))
          : ListView.builder(
              itemCount: activeAuctions.length,
              itemBuilder: (context, index) {
                final auction = activeAuctions[index];

                return Card(
                  margin: EdgeInsets.symmetric(horizontal: 16, vertical: 8),
                  child: ListTile(
                    leading: Image.network(
                      auction.imageUrl,
                      width: 50,
                      height: 50,
                      fit: BoxFit.cover,
                      errorBuilder: (context, error, stackTrace) {
                        return Icon(Icons.image, size: 50);
                      },
                    ),
                    title: Text(auction.itemName),
                    subtitle: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Text('Current bid: \$${auction.bids.isEmpty ?
auction.startingPrice.toStringAsFixed(2) : auction.bids.map((e) => e.amount).reduce((a, b)
=> a > b ? a : b).toStringAsFixed(2)}'),
                        Text('Time left: ${_getRemainingTime(auction.endTime)}'),
                      ],
                    ),
                    onTap: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => AuctionDetailPage(auction: auction),
                        ),
                      ).then((_) {
                        setState(() {});
                      });
                    },
                  ),
                );
              },
            ),
    );
  }
}
```

```
class AuctionDetailPage extends StatefulWidget {
  final Auction auction;

  AuctionDetailPage({required this.auction});

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

class _AuctionDetailPageState extends State<AuctionDetailPage> {
  final _formKey = GlobalKey<FormState>();
  final _auctionService = AuctionService();
  double _bidAmount = 0.0;
  late Timer _timer;

  @override
  void initState() {
    super.initState();
    _timer = Timer.periodic(Duration(seconds: 1), (timer) {
      setState(() {});
    });
  }

  @override
  void dispose() {
    _timer.cancel();
    super.dispose();
  }

  String _getRemainingTime(DateTime endTime) {
    final remaining = endTime.difference(DateTime.now());
    if (remaining.isNegative) return 'Auction ended';

    final hours = remaining.inHours;
    final minutes = remaining.inMinutes % 60;
    final seconds = remaining.inSeconds % 60;

    return '$hours:${minutes.toString().padLeft(2, '0')}:${seconds.toString().padLeft(2, '0')}';
  }

  double _getMinimumBid() {
    if (widget.auction.bids.isEmpty) {
      return widget.auction.startingPrice;
    }

    return widget.auction.bids.map((e) => e.amount).reduce((a, b) => a > b ? a : b) + 1.0;
  }
```

```
  @override
  Widget build(BuildContext context) {
    final isAuctionActive = DateTime.now().isBefore(widget.auction.endTime);
    final minimumBid = _getMinimumBid();

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.auction.itemName),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Center(
                child: Image.network(
                  widget.auction.imageUrl,
                  height: 200,
                  fit: BoxFit.contain,
                  errorBuilder: (context, error, stackTrace) {
                    return Icon(Icons.image, size: 200);
                  },
                ),
              ),
              SizedBox(height: 16),
              Text(
                widget.auction.itemName,
                style: Theme.of(context).textTheme.headline5,
              ),
              SizedBox(height: 8),
              Text(widget.auction.description),
              SizedBox(height: 16),
              Text(
                'Time remaining: ${_getRemainingTime(widget.auction.endTime)}',
                style: Theme.of(context).textTheme.subtitle1,
              ),
              SizedBox(height: 16),
              Text(
                'Current highest bid: \$${widget.auction.bids.isEmpty ? "No bids yet" :
widget.auction.bids.map((e) => e.amount).reduce((a, b) => a > b ? a :
b).toStringAsFixed(2)}',
                style: Theme.of(context).textTheme.subtitle1,
              ),
              SizedBox(height: 8),
              Text('Minimum bid required: \$${minimumBid.toStringAsFixed(2)}'),
              SizedBox(height: 24),
```

```
if (isAuctionActive) ...[
  Form(
    key: _formKey,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        TextFormField(
          decoration: InputDecoration(
            labelText: 'Your Bid Amount',
            prefixText: '\$',
          ),
          keyboardType: TextInputType.number,
          validator: (value) {
            if (value == null || value.isEmpty) {
              return 'Please enter bid amount';
            }

            final amount = double.tryParse(value);
            if (amount == null) {
              return 'Please enter a valid number';
            }

            if (amount < minimumBid) {
              return 'Bid must be at least \$${minimumBid.toStringAsFixed(2)}';
            }

            return null;
          },
          onSaved: (value) {
            _bidAmount = double.parse(value!);
          },
        ),
        SizedBox(height: 16),
        ElevatedButton(
          onPressed: () {
            if (_formKey.currentState!.validate()) {
              _formKey.currentState!.save();

              // In a real app, we would get this from user authentication
              final userId = 'user123';
              final userName = 'John Doe';

              final success = _auctionService.placeBid(
                widget.auction.id,
                userId,
                userName,
                _bidAmount,
              );
```

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

                      setState(() {});
                    } else {
                      ScaffoldMessenger.of(context).showSnackBar(
                        SnackBar(content: Text('Failed to place bid')),
                      );
                    }
                  }
                },
                child: Text('Place Bid'),
              ),
            ],
          ),
        ),
      ] else ...[
        Center(
          child: Text(
            'This auction has ended',
            style: Theme.of(context).textTheme.subtitle1?.copyWith(
              color: Colors.red,
            ),
          ),
        ),

        if (widget.auction.winningBid != null) ...[
          SizedBox(height: 16),
          Center(
            child: Text(
              'Winning bid: \$${widget.auction.winningBid!.amount.toStringAsFixed(2)} by
${widget.auction.winningBid!.bidderName}',
              style: Theme.of(context).textTheme.subtitle1,
            ),
          ),
        ],
      ],

      SizedBox(height: 32),
      Text(
        'Bid History',
        style: Theme.of(context).textTheme.headline6,
      ),
      SizedBox(height: 8),
```

```
            if (widget.auction.bids.isEmpty) ...[
              Center(child: Text('No bids placed yet')),
            ] else ...[
              ListView.builder(
                shrinkWrap: true,
                physics: NeverScrollableScrollPhysics(),
                itemCount: widget.auction.bids.length,
                itemBuilder: (context, index) {
                  final sortedBids = widget.auction.bids.toList()
                    ..sort((a, b) => b.timestamp.compareTo(a.timestamp));
                  final bid = sortedBids[index];

                  return ListTile(
                    title: Text('\$${bid.amount.toStringAsFixed(2)} by ${bid.bidderName}'),
                    subtitle: Text(
                      'at ${bid.timestamp.toString().substring(0, 19)}',
                    ),
                  );
                },
              ),
            ],
          ],
        ),
      ),
    ),
  );
 }
}

class PaymentHistoryPage extends StatelessWidget {
  final _auctionService = AuctionService();

  @override
  Widget build(BuildContext context) {
    final payments = _auctionService.getPaymentHistory();

    return Scaffold(
      appBar: AppBar(
        title: Text('Payment History'),
      ),
      body: payments.isEmpty
          ? Center(child: Text('No payment history available'))
          : ListView.builder(
              itemCount: payments.length,
              itemBuilder: (context, index) {
                final payment = payments[index];

                return ListTile(
```

```
            title: Text(payment.itemName),
            subtitle: Text('Amount: \$${payment.amount.toStringAsFixed(2)}'),
            trailing: _getPaymentStatusChip(payment.status),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => PaymentDetailPage(payment: payment),
                ),
              );
            },
          );
        },
      ),
  );
}

Widget _getPaymentStatusChip(PaymentStatus status) {
  Color color;
  String label;

  switch (status) {
    case PaymentStatus.pending:
      color = Colors.orange;
      label = 'Pending';
      break;
    case PaymentStatus.completed:
      color = Colors.green;
      label = 'Completed';
      break;
    case PaymentStatus.failed:
      color = Colors.red;
      label = 'Failed';
      break;
    case PaymentStatus.refunded:
      color = Colors.blue;
      label = 'Refunded';
      break;
  }

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

```
}

class PaymentDetailPage extends StatefulWidget {
  final Payment payment;

  PaymentDetailPage({required this.payment});

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

class _PaymentDetailPageState extends State<PaymentDetailPage> {
  final _auctionService = AuctionService();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Payment Details'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Card(
              child: Padding(
                padding: const EdgeInsets.all(16.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Item: ${widget.payment.itemName}',
                      style: Theme.of(context).textTheme.headline6,
                    ),
                    SizedBox(height: 8),
                    Text('Payment ID: ${widget.payment.id}'),
                    Text('Buyer: ${widget.payment.buyerName}'),
                    Text('Amount: \$${widget.payment.amount.toStringAsFixed(2)}'),
                    Text('Date: ${widget.payment.timestamp.toString().substring(0, 19)}'),
                    SizedBox(height: 16),
                    Row(
                      children: [
                        Text('Status: '),
                        _getPaymentStatusChip(widget.payment.status),
                      ],
                    ),
                  ],
```

```
            ),
          ),
        ),
        SizedBox(height: 24),

        if (widget.payment.status == PaymentStatus.pending) ...[
          Text(
            'Process Payment',
            style: Theme.of(context).textTheme.headline6,
          ),
          SizedBox(height: 16),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              ElevatedButton(
                onPressed: () {
                  _processPayment(true);
                },
                style: ElevatedButton.styleFrom(
                  primary: Colors.green,
                ),
                child: Text('Complete Payment'),
              ),
              ElevatedButton(
                onPressed: () {
                  _processPayment(false);
                },
                style: ElevatedButton.styleFrom(
                  primary: Colors.red,
                ),
                child: Text('Mark as Failed'),
              ),
            ],
          ),
        ],
      ],
    ),
  ),
  );
}

void _processPayment(bool isSuccessful) {
  final success = _auctionService.processPayment(widget.payment.id, isSuccessful);

  if (success) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(
```

```
            isSuccessful ? 'Payment completed successfully' : 'Payment marked as failed'
          ),
        ),
      );

      Navigator.pop(context);
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Failed to process payment')),
      );
    }
  }

  Widget _getPaymentStatusChip(PaymentStatus status) {
    Color color;
    String label;

    switch (status) {
      case PaymentStatus.pending:
        color = Colors.orange;
        label = 'Pending';
        break;
      case PaymentStatus.completed:
        color = Colors.green;
        label = 'Completed';
        break;
      case PaymentStatus.failed:
        color = Colors.red;
        label = 'Failed';
        break;
      case PaymentStatus.refunded:
        color = Colors.blue;
        label = 'Refunded';
        break;
    }

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

// Extension for creating copies of immutable objects
extension PaymentCopyWith on Payment {
```

```
  Payment copyWith({
    String? id,
    String? auctionId,
    String? itemName,
    String? buyerId,
    String? buyerName,
    double? amount,
    DateTime? timestamp,
    PaymentStatus? status,
  }) {
    return Payment(
      id: id ?? this.id,
      auctionId: auctionId ?? this.auctionId,
      itemName: itemName ?? this.itemName,
      buyerId: buyerId ?? this.buyerId,
      buyerName: buyerName ?? this.buyerName,
      amount: amount ?? this.amount,
      timestamp: timestamp ?? this.timestamp,
      status: status ?? this.status,
    );
  }
}
```