STAR Community App - Budget-Aligned Backend Architecture

1. High-Level Architecture Overview

1.1 System Architecture Pattern

- **Serverless Architecture**: Firebase-based serverless functions
- **Event-Driven Architecture**: Firestore real-time listeners
- **JAMstack Pattern**: Client-side rendering with API calls
- **NoSQL-First Design**: Optimized for Firestore's document model

1.2 Technology Stack (Budget-Aligned)

- **Backend Framework**: Firebase Cloud Functions (Node.js)
- **Database**: Firebase Firestore (NoSQL)
- **Authentication**: Firebase Authentication
- **File Storage**: Firebase Storage
- **Real-time**: Firestore real-time listeners
- **Caching**: Client-side caching + Firebase local persistence
- **Search**: Algolia free tier or client-side search
- **Monitoring**: Firebase Analytics + Google Analytics 4

2. Firebase-Based Service Architecture

2.1 Cloud Functions Structure

javascript

```javascript
// Authentication Functions
exports.createUserProfile = functions.auth.user().onCreate(async (user) => {
  // Create user document in Firestore
  return admin.firestore().collection('users').doc(user.uid).set({
    email: user.email,
    phone: user.phoneNumber,
    createdAt: admin.firestore.FieldValue.serverTimestamp(),
    userType: 'resident', // default
    status: 'active'
  });
});

// Booking Management Functions
exports.createBooking = functions.https.onCall(async (data, context) => {
  // Validate authentication
  if (!context.auth) {
    throw new functions.https.HttpsError('unauthenticated', 'User must be authenticated');
  }

  // Booking creation logic
  const booking = {
    residentId: context.auth.uid,
    serviceProviderId: data.serviceProviderId,
    serviceId: data.serviceId,
    scheduledDate: data.scheduledDate,
    scheduledTime: data.scheduledTime,
    tokenAmount: data.tokenAmount,
    status: 'pending',
    createdAt: admin.firestore.FieldValue.serverTimestamp()
  };

  return admin.firestore().collection('bookings').add(booking);
});

// Token Transaction Functions
exports.processTokenPurchase = functions.https.onCall(async (data, context) => {
  // PayFast webhook processing
  // Update user token balance
  // Record transaction
});

// Notification Functions
exports.sendBookingNotification = functions.firestore
  .document('bookings/{bookingId}')
  .onUpdate(async (change, context) => {
    const beforeData = change.before.data();
    const afterData = change.after.data();
```

```
    if (beforeData.status !== afterData.status) {
      // Send push notification via FCM
      return sendPushNotification(afterData);
    }
  });
```

2.2 Firestore Database Structure

javascript

```
// Users Collection
users: {
  [userId]: {
    email: string,
    phone: string,
    firstName: string,
    lastName: string,
    userType: 'resident' | 'service_provider' | 'agent' | 'admin',
    status: 'active' | 'suspended' | 'pending_verification',
    createdAt: timestamp,
    updatedAt: timestamp,
    profileImageUrl: string,
    location: geopoint,
    address: string,
    areaId: string,
    preferences: object,
    // For Service Providers
    businessName?: string,
    businessDescription?: string,
    verificationStatus?: 'pending' | 'verified' | 'rejected',
    operatingHours?: object,
    serviceAreas?: array,
    ratingAverage?: number,
    ratingCount?: number,
    portfolioImages?: array
  }
}

// Services Collection
services: {
  [serviceId]: {
    serviceProviderId: string,
    categoryId: string,
    name: string,
    description: string,
    basePrice: number,
    priceUnit: string,
    durationMinutes: number,
    isActive: boolean,
    requirements: string,
    inclusions: array,
    exclusions: array,
    createdAt: timestamp,
    updatedAt: timestamp
  }
}

// Bookings Collection
```

```
// Bookings Collection
bookings: {
  [bookingId]: {
    residentId: string,
    serviceProviderId: string,
    serviceId: string,
    status: 'pending' | 'confirmed' | 'in_progress' | 'completed' | 'cancelled',
    scheduledDate: timestamp,
    scheduledTimeStart: string,
    scheduledTimeEnd: string,
    actualStartTime: timestamp,
    actualEndTime: timestamp,
    serviceLocation: string,
    specialRequirements: string,
    tokenAmount: number,
    serviceIdCode: string, // SI code
    createdAt: timestamp,
    updatedAt: timestamp,
    cancellationReason?: string,
    cancelledBy?: string,
    cancelledAt?: timestamp
  }
}

// Token Wallets Collection
tokenWallets: {
  [userId]: {
    balance: number,
    totalEarned: number,
    totalSpent: number,
    totalPurchased: number,
    escrowBalance: number,
    createdAt: timestamp,
    updatedAt: timestamp,
    lastTransactionAt: timestamp
  }
}

// Token Transactions Collection
tokenTransactions: {
  [transactionId]: {
    userId: string,
    transactionType: 'purchase' | 'redemption' | 'payment' | 'refund' | 'bonus',
    amount: number,
    balanceBefore: number,
    balanceAfter: number,
    referenceId: string,
    referenceType: string,
    description: string,
    externalTransactionId: string,
```

```
    paymentMethod: string,
    status: 'pending' | 'completed' | 'failed' | 'cancelled',
    createdAt: timestamp,
    processedAt: timestamp,
    metadata: object
  }
}

// STAR Projects Collection
starProjects: {
  [projectId]: {
    creatorId: string,
    title: string,
    description: string,
    category: string,
    targetAmount: number,
    currentAmount: number,
    minContributors: number,
    currentContributors: number,
    deadline: timestamp,
    status: 'draft' | 'active' | 'funded' | 'in_progress' | 'completed' | 'cancelled',
    geographicScope: string,
    selectedSpId: string,
    createdAt: timestamp,
    updatedAt: timestamp,
    completedAt: timestamp,
    impactReport: string,
    images: array,
    votingSettings: object,
    participants: {
      [userId]: {
        contributionAmount: number,
        joinedAt: timestamp,
        isAdmin: boolean,
        votingWeight: number
      }
    }
  }
}

// STAR Causes Collection
starCauses: {
  [causeId]: {
    applicantId: string,
    championId: string,
    title: string,
    description: string,
    beneficiaryInfo: object,
    targetAmount: number,
```

```
    currentAmount: number,
    deadline: timestamp,
    status: 'pending' | 'approved' | 'active' | 'completed' | 'rejected',
    urgencyLevel: 'low' | 'medium' | 'high' | 'critical',
    category: string,
    supportingDocuments: array,
    createdAt: timestamp,
    updatedAt: timestamp,
    approvedAt: timestamp,
    completedAt: timestamp,
    impactReport: string
  }
}

// Service Categories Collection
serviceCategories: {
  [categoryId]: {
    name: string,
    description: string,
    parentId: string,
    iconUrl: string,
    isActive: boolean,
    sortOrder: number
  }
}

// Reviews Collection
reviews: {
  [reviewId]: {
    bookingId: string,
    reviewerId: string,
    revieweeId: string,
    rating: number, // 1-5
    comment: string,
    createdAt: timestamp,
    isPublic: boolean
  }
}

// Areas Collection
areas: {
  [areaId]: {
    name: string,
    type: 'city' | 'suburb' | 'township' | 'village',
    parentId: string,
    boundaries: geopoint, // simplified for budget
    isActive: boolean
  }
}
```

```
// Chat Messages Collection
chatMessages: {
  [chatId]: {
    messages: {
      [messageId]: {
        senderId: string,
        recipientId: string,
        message: string,
        timestamp: timestamp,
        read: boolean,
        messageType: 'text' | 'image' | 'system'
      }
    }
  }
}
```

3. API Design with Firebase

3.1 Cloud Functions HTTP Endpoints

javascript

```javascript
// Authentication (handled by Firebase Auth SDK)
// No custom endpoints needed - use Firebase Auth directly

// User Management
exports.updateUserProfile = functions.https.onCall(async (data, context) => {
  // Update user profile
});

exports.uploadDocument = functions.https.onCall(async (data, context) => {
  // Handle document uploads for verification
});

// Service Provider Management
exports.registerServiceProvider = functions.https.onCall(async (data, context) => {
  // Convert user to service provider
});

exports.updateServiceProviderProfile = functions.https.onCall(async (data, context) => {
  // Update SP profile and services
});

// Booking Management
exports.createBooking = functions.https.onCall(async (data, context) => {
  // Create new booking
});

exports.updateBookingStatus = functions.https.onCall(async (data, context) => {
  // Update booking status
});

exports.cancelBooking = functions.https.onCall(async (data, context) => {
  // Cancel booking and handle refunds
});

// Token Economy
exports.purchaseTokens = functions.https.onCall(async (data, context) => {
  // Initiate PayFast payment
});

exports.processPayment = functions.https.onRequest(async (req, res) => {
  // PayFast webhook handler
});

// Community Features
exports.createProject = functions.https.onCall(async (data, context) => {
  // Create STAR Project
});
```

```
exports.joinProject = functions.https.onCall(async (data, context) => {
  // Join project and contribute tokens
});

exports.applyCause = functions.https.onCall(async (data, context) => {
  // Apply for STAR Cause
});
```

3.2 Real-time Data Flow

javascript

```
// Client-side real-time listeners
// Booking updates
firebase.firestore()
  .collection('bookings')
  .where('residentId', '==', currentUserId)
  .onSnapshot((snapshot) => {
    // Real-time booking updates
  });

// Chat messages
firebase.firestore()
  .collection('chatMessages')
  .doc(chatId)
  .collection('messages')
  .orderBy('timestamp', 'desc')
  .limit(50)
  .onSnapshot((snapshot) => {
    // Real-time chat updates
  });

// Project updates
firebase.firestore()
  .collection('starProjects')
  .onSnapshot((snapshot) => {
    // Real-time project updates
  });

// Wallet balance
firebase.firestore()
  .collection('tokenWallets')
  .doc(userId)
  .onSnapshot((doc) => {
    // Real-time wallet updates
  });
```

4. Security Implementation

4.1 Firestore Security Rules

javascript

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users can read/write their own profile
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
      allow read: if request.auth != null; // Others can read basic profile
    }

    // Service providers can manage their services
    match /services/{serviceId} {
      allow read: if request.auth != null;
      allow write: if request.auth != null &&
        resource.data.serviceProviderId == request.auth.uid;
    }

    // Booking access control
    match /bookings/{bookingId} {
      allow read, write: if request.auth != null &&
        (resource.data.residentId == request.auth.uid ||
         resource.data.serviceProviderId == request.auth.uid);
    }

    // Token wallets - users can only access their own
    match /tokenWallets/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }

    // Token transactions - users can only read their own
    match /tokenTransactions/{transactionId} {
      allow read: if request.auth != null && resource.data.userId == request.auth.uid;
      allow write: if false; // Only Cloud Functions can write
    }

    // STAR Projects - public read, creator write
    match /starProjects/{projectId} {
      allow read: if request.auth != null;
      allow write: if request.auth != null &&
        (resource == null || resource.data.creatorId == request.auth.uid);
    }

    // STAR Causes - public read, restricted write
    match /starCauses/{causeId} {
      allow read: if request.auth != null;
      allow create: if request.auth != null;
      allow update: if request.auth != null &&
        (resource.data.applicantId == request.auth.uid ||
```

```
      hasRole('agent') || hasRole('admin'));
    }

    // Reviews - booking participants only
    match /reviews/{reviewId} {
      allow read: if request.auth != null;
      allow write: if request.auth != null &&
        resource.data.reviewerId == request.auth.uid;
    }

    // Chat messages - participants only
    match /chatMessages/{chatId} {
      allow read, write: if request.auth != null &&
        (request.auth.uid in resource.data.participants);
    }

    // Helper functions
    function hasRole(role) {
      return request.auth != null &&
        get(/databases/$(database)/documents/users/$(request.auth.uid)).data.userType == role;
    }
  }
}
```

4.2 Firebase Storage Security Rules

javascript

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    // User profile images
    match /profiles/{userId}/{allPaths=**} {
      allow read: if request.auth != null;
      allow write: if request.auth != null && request.auth.uid == userId;
    }

    // Service provider portfolios
    match /portfolios/{userId}/{allPaths=**} {
      allow read: if request.auth != null;
      allow write: if request.auth != null && request.auth.uid == userId;
    }

    // Project images
    match /projects/{projectId}/{allPaths=**} {
      allow read: if request.auth != null;
      allow write: if request.auth != null; // Validated in Firestore rules
    }

    // Verification documents
    match /documents/{userId}/{allPaths=**} {
      allow read: if request.auth != null &&
        (request.auth.uid == userId || hasAdminRole());
      allow write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

5. Performance Optimization for Free Tier

5.1 Query Optimization

javascript

```javascript
// Efficient Firestore queries to minimize reads
// Use compound indexes for complex queries
// Pagination to limit document reads
const getServiceProviders = async (location, category, limit = 10) => {
  return firebase.firestore()
    .collection('users')
    .where('userType', '==', 'service_provider')
    .where('status', '==', 'active')
    .where('serviceAreas', 'array-contains', location)
    .limit(limit)
    .get();
};

// Use subcollections to optimize data structure
const getUserBookings = async (userId, limit = 20) => {
  return firebase.firestore()
    .collection('bookings')
    .where('residentId', '==', userId)
    .orderBy('createdAt', 'desc')
    .limit(limit)
    .get();
};
```

5.2 Client-Side Optimization

javascript

```javascript
// Firestore offline persistence
firebase.firestore().enablePersistence()
  .catch((err) => {
    if (err.code == 'failed-precondition') {
      // Multiple tabs open
    } else if (err.code == 'unimplemented') {
      // Browser doesn't support
    }
  });

// Efficient real-time listeners with proper cleanup
useEffect(() => {
  const unsubscribe = firebase.firestore()
    .collection('bookings')
    .where('residentId', '==', currentUserId)
    .where('status', 'in', ['pending', 'confirmed', 'in_progress'])
    .onSnapshot((snapshot) => {
      // Handle updates
    });

  return () => unsubscribe(); // Cleanup
}, [currentUserId]);
```

6. Integration with External Services

6.1 PayFast Integration

javascript

```javascript
// PayFast payment initiation
exports.initiatePayment = functions.https.onCall(async (data, context) => {
  const paymentData = {
    merchant_id: functions.config().payfast.merchant_id,
    merchant_key: functions.config().payfast.merchant_key,
    amount: data.amount,
    item_name: `${data.tokens} STAR Tokens`,
    return_url: `${functions.config().app.url}/payment-success`,
    cancel_url: `${functions.config().app.url}/payment-cancel`,
    notify_url: `${functions.config().app.url}/api/payfast-webhook`,
    custom_str1: context.auth.uid, // User ID
    custom_str2: data.tokens // Token amount
  };

  // Generate signature and return payment URL
  return {
    paymentUrl: generatePayFastUrl(paymentData),
    paymentId: paymentData.m_payment_id
  };
});

// PayFast webhook handler
exports.payfastWebhook = functions.https.onRequest(async (req, res) => {
  // Verify payment signature
  // Update user token balance
  // Record transaction
  // Send confirmation notification
});
```

6.2 SMS Integration (Twilio)

javascript

```javascript
// SMS notifications for critical events
exports.sendSMSNotification = functions.firestore
  .document('bookings/{bookingId}')
  .onUpdate(async (change, context) => {
    const booking = change.after.data();

    if (booking.status === 'confirmed') {
      const twilioClient = require('twilio')(
        functions.config().twilio.sid,
        functions.config().twilio.token
      );

      await twilioClient.messages.create({
        body: `Your STAR booking is confirmed for ${booking.scheduledDate}`,
        from: functions.config().twilio.phone,
        to: booking.residentPhone
      });
    }
  });
```

7. Monitoring and Analytics (Free Tier)

7.1 Custom Analytics Events

javascript

```javascript
// Track business metrics with Firebase Analytics
const trackTokenPurchase = (amount, method) => {
  firebase.analytics().logEvent('token_purchase', {
    value: amount,
    currency: 'ZAR',
    payment_method: method
  });
};

const trackBookingCreated = (serviceCategory, tokenAmount) => {
  firebase.analytics().logEvent('booking_created', {
    service_category: serviceCategory,
    token_amount: tokenAmount
  });
};

const trackProjectJoined = (projectId, contributionAmount) => {
  firebase.analytics().logEvent('project_joined', {
    project_id: projectId,
    contribution_amount: contributionAmount
  });
};
```

7.2 Error Tracking

javascript

```javascript
// Custom error logging to Firestore
const logError = async (error, context) => {
  await firebase.firestore().collection('errorLogs').add({
    error: error.message,
    stack: error.stack,
    context: context,
    userId: firebase.auth().currentUser?.uid,
    timestamp: firebase.firestore.FieldValue.serverTimestamp()
  });
};
```

8. Deployment Configuration

8.1 Firebase Configuration

json

```
{
  "functions": {
    "source": "functions",
    "runtime": "nodejs18"
  },
  "firestore": {
    "rules": "firestore.rules",
    "indexes": "firestore.indexes.json"
  },
  "storage": {
    "rules": "storage.rules"
  },
  "hosting": {
    "public": "build",
    "ignore": ["firebase.json", "**/.*", "**/node_modules/**"],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

8.2 Environment Configuration

javascript

```
// functions/index.js
const functions = require('firebase-functions');
const admin = require('firebase-admin');

admin.initializeApp({
  credential: admin.credential.applicationDefault(),
  storageBucket: functions.config().app.storage_bucket
});

// Set configuration
// firebase functions:config:set payfast.merchant_id="your_id"
// firebase functions:config:set payfast.merchant_key="your_key"
// firebase functions:config:set twilio.sid="your_sid"
// firebase functions:config:set twilio.token="your_token"
```

9. Cost Optimization Strategies

9.1 Firestore Usage Optimization

```javascript
// Batch operations to reduce function calls
const batch = firebase.firestore().batch();

// Update multiple documents in one batch
batch.update(userRef, { tokenBalance: newBalance });
batch.set(transactionRef, transactionData);
batch.update(bookingRef, { status: 'paid' });

await batch.commit(); // Single operation
```

9.2 Storage Optimization

```javascript
// Compress images before upload
const compressImage = async (file) => {
  const canvas = document.createElement('canvas');
  const ctx = canvas.getContext('2d');

  // Resize and compress logic
  canvas.width = Math.min(file.width, 800);
  canvas.height = Math.min(file.height, 600);

  return canvas.toBlob(blob => blob, 'image/jpeg', 0.7);
};
```

10. Migration Strategy

10.1 From Complex to Simple Architecture

1. **Remove Microservices**: Consolidate into Cloud Functions
2. **Simplify Database**: Move from PostgreSQL to Firestore
3. **Eliminate Message Queues**: Use Firestore triggers
4. **Remove Redis**: Use Firestore local persistence
5. **Simplify Authentication**: Use Firebase Auth

10.2 Future Scaling Path

When revenue reaches thresholds:

- **$100/month**: Upgrade to Firebase Blaze plan
- **$500/month**: Add advanced monitoring
- **$1000/month**: Consider hybrid architecture
- **$5000/month**: Evaluate dedicated infrastructure

This architecture maintains all the functionality of the original design while staying within the ultra-budget constraints, leveraging Firebase's generous free tiers and pay-as-you-go pricing model.