Engineering Challenges & Technical Decisions Required

Purpose: Identify all technical problems that require engineering judgment and expertise

---

1. API Architecture & Backend Integration

Critical Questions

Q1: How do we structure API endpoints for customer-facing apps?

- Current rental software may have admin-focused APIs
- Need customer-safe versions (filtered data, permissions)
- **Decision needed:** RESTful, GraphQL, or hybrid?
- **Problem:** Exposing too much data = security risk

Q2: How do we handle authentication across platforms?

- Web and mobile need single sign-on (SSO)
- **Options:** JWT tokens, OAuth 2.0, session-based?
- **Problem:** Token refresh, secure storage, session management
- **Mobile challenge:** Where to securely store tokens on device?
- **Web challenge:** CSRF protection, XSS vulnerabilities

Q3: How do we version APIs without breaking clients?

- Mobile apps can't force updates immediately
- Some users will have old app versions
- **Problem:** Need backward compatibility strategy
- **Decision:** API versioning approach (URL, header, or content negotiation)

Q4: What's the rate limiting strategy?

- Prevent abuse and DDoS
- Different limits for authenticated vs anonymous users
- **Problem:** Where to implement? (API gateway, app layer, database)
- **Decision:** How to handle exceeded limits gracefully

Q5: How do we handle partial failures?

- User books a rental but payment processing fails
- Email notification fails but booking succeeds

- **Problem:** Data consistency across distributed operations
- **Decision:** Retry logic, transaction rollback, idempotency

Engineering Challenges

Challenge 1: Real-time Data Synchronization

- Delivery tracking needs real-time updates
- **Options:** WebSockets, Server-Sent Events (SSE), polling?
- **Mobile:** How to handle background updates?
- **Problem:** Battery drain, data usage, connection reliability
- **Technical debt:** Polling is easiest but least efficient

Challenge 2: Offline Functionality

- Users need to see rental details without internet
- **Problem:** What data to cache? How to sync when online?
- **Decision:** Which features work offline vs require connection?
- **Conflict resolution:** User modifies data offline, server has newer version

Challenge 3: API Response Optimization

- Mobile has limited bandwidth and battery
- **Problem:** Sending too much data = slow, expensive
- **Solutions:** Pagination, field filtering, response compression
- **Decision:** How to balance flexibility vs performance

Challenge 4: Legacy Integration

- Your existing rental software may not be API-first
- **Problems:**
  - How to expose internal logic safely?
  - May need to wrap old database queries
  - May need to transform legacy data formats
- **Risk:** Breaking existing functionality while adding APIs

---

2. Payment Integration & Financial Logic

Critical Questions

Q1: Which payment processor and how to integrate?

- **Options:** Stripe, PayPal, Square, Braintree
- **Considerations:** International support? Fees? Compliance?
- **Mobile challenge:** Native payment methods (Apple Pay, Google Pay)
- **Web challenge:** 3D Secure, Strong Customer Authentication (SCA)

Q2: How do we handle payment failures?

- Card declined, insufficient funds, fraud detection
- **Problem:** Do we cancel reservation? Hold it? For how long?
- **Decision:** Retry logic, fallback payment methods
- **User experience:** How to communicate failure gracefully?

Q3: How do we handle refunds and cancellations?

- Partial refunds, full refunds, cancellation fees
- **Problem:** Complex business logic with timing windows
- **Technical challenge:** Idempotency (don't refund twice)
- **Decision:** Automated vs manual approval flow

Q4: How do we ensure PCI compliance?

- Can't store credit card details directly
- **Solution:** Tokenization through payment processor
- **Problem:** Must never log or expose card details
- **Audit requirement:** Compliance documentation

Q5: How do we handle payment disputes and chargebacks?

- Customers dispute charges weeks/months later
- **Problem:** Need audit trail of all transactions
- **Decision:** How much transaction data to store? For how long?
- **Technical requirement:** Webhook handling for dispute notifications

Engineering Challenges

Challenge 1: Transaction State Management

- Complex state machine: pending → processing → succeeded/failed
- **Problem:** Multiple systems involved (your app, payment processor, bank)
- **Risk:** Race conditions, partial failures
- **Solution needed:** Distributed transaction handling

Challenge 2: Webhook Reliability

- Payment processor sends notifications to your server
- **Problems:**
  - What if your server is down?
  - What if webhook arrives out of order?
  - What if same webhook arrives multiple times?
- **Solution needed:** Idempotency, retry logic, queue management

Challenge 3: Multi-Currency Support (if international)

- Display prices in user's currency
- **Problem:** Exchange rates fluctuate, calculation rounding
- **Decision:** Dynamic conversion or fixed rates?
- **Tax implications:** Different VAT/tax rules per country

Challenge 4: Payment Security

- Prevent fraud, stolen cards, testing attacks
- **Decisions:**
  - Velocity checks (max transactions per hour)
  - Device fingerprinting
  - CVV requirements
  - Address verification
- **Problem:** Balance security vs user friction

---

3. Authentication & User Security

Critical Questions

Q1: How do we securely store passwords?

- **Answer:** Never store plain text
- **Decision:** Which hashing algorithm? (bcrypt, Argon2, PBKDF2)
- **Problem:** Migration from existing system if it uses weak hashing

Q2: How do we handle password resets?

- Email-based tokens with expiration
- **Problem:** Token generation, secure storage, replay attacks

- **Decision:** Token lifetime, one-time use enforcement
- **Edge case:** User requests multiple resets

Q3: How do we implement two-factor authentication (2FA)?

- **Options:** SMS, authenticator apps (TOTP), email
- **Problem:** SMS interception, app backup codes
- **Decision:** Required for all users or optional?
- **Implementation:** Time-based one-time passwords (TOTP) library

Q4: How do we handle session management?

- Keep users logged in across devices
- **Problem:** Session hijacking, token theft
- **Decisions:**
  - Session timeout duration
  - Refresh token strategy
  - Force logout on password change
- **Mobile specific:** Biometric authentication (Face ID, fingerprint)

Q5: How do we secure API endpoints?

- Prevent unauthorized access
- **Problem:** Every endpoint needs proper authorization checks
- **Decision:** Role-based access control (RBAC) implementation
- **Risk:** Forget one check = security vulnerability

Engineering Challenges

Challenge 1: Cross-Platform Session Persistence

- User logs in on web, session should work on mobile (or not?)
- **Decision:** Shared sessions or separate?
- **Problem:** Revoking access when user logs out on one device
- **Implementation:** Token blacklisting, short-lived tokens

Challenge 2: Secure Communication

- All traffic must use HTTPS/TLS
- **Problem:** Certificate management, renewal
- **Mobile:** Certificate pinning to prevent man-in-the-middle attacks

- **Decision:** How strict? Pinning can cause issues with CDNs

Challenge 3: Sensitive Data Storage

- Driver's license images, payment info (tokens)
- **Problems:**
  - Where to store? S3, database, separate service?
  - How to encrypt at rest?
  - Who can access? Audit logging
- **Compliance:** GDPR, CCPA requirements for data deletion

Challenge 4: Account Takeover Prevention

- Detect suspicious login patterns
- **Decisions:**
  - Email notification on new device?
  - Block unusual locations?
  - CAPTCHA on login failures?
- **Problem:** False positives (legitimate users blocked)

---

4. Mobile-Specific Challenges

Critical Questions

Q1: How do we handle different screen sizes and orientations?

- Phones, tablets, foldables
- **Problem:** UI layout must be responsive
- **Decision:** Portrait only or support landscape?
- **Flutter/React Native:** Different complexity than native

Q2: How do we manage app state?

- User navigates between screens, state must persist
- **Options:** Redux, MobX, Provider, Riverpod, Bloc
- **Problem:** Over-complicating vs under-structuring
- **Decision:** State management architecture affects entire codebase

Q3: How do we handle push notifications?

- iOS (APNs) and Android (FCM) have different systems

- **Problems:**
  - User permissions (can be denied)
  - Token management and device registration
  - Notification when app is closed, background, foreground
  - Deep linking (notification opens specific screen)
- **Decision:** Which events trigger notifications? Frequency limits?

Q4: How do we handle app updates?

- Force users to update for critical bugs/security?
- **Problem:** Can't force iOS users (App Store review takes days)
- **Decision:** Soft prompts vs hard blocks
- **Technical:** Version checking mechanism

Q5: How do we handle permissions?

- Camera (document scanning), location, notifications, contacts
- **Problem:** Users can deny permissions at any time
- **Decision:** Graceful degradation when permissions denied
- **UX challenge:** When and how to request permissions

Engineering Challenges

Challenge 1: Platform-Specific Features

- iOS Face ID vs Android fingerprint
- Apple Pay vs Google Pay
- Different design guidelines (Material vs Human Interface)
- **Problem:** Code that works on one platform, fails on other
- **Solution needed:** Platform-specific code, testing on both

Challenge 2: App Size Optimization

- Large apps = users won't download
- **Problems:**
  - Image assets, fonts, libraries add up
  - Code splitting and lazy loading
- **Target:** Under 50MB for better download rates
- **Decision:** What features are worth the size increase?

Challenge 3: Background Processing

- Tracking delivery status when app is closed
- **Problems:**
  - iOS is very restrictive on background tasks
  - Android battery optimization kills background processes
  - Different rules per OS version
- **Technical complexity:** Background job scheduling, wake locks

Challenge 4: Deep Linking and Universal Links

- User clicks email link, opens in app (not browser)
- **Problems:**
  - iOS universal links, Android app links setup
  - Handling when app is not installed
  - Routing to correct screen with context
- **Decision:** Which flows need deep linking?

Challenge 5: Crash Reporting and Debugging

- Apps crash on user devices, you need to know why
- **Options:** Sentry, Firebase Crashlytics, Bugsnag
- **Problem:** Production debugging without access to device
- **Decision:** How much telemetry to collect? Privacy concerns

Challenge 6: App Store Compliance

- Apple and Google have strict guidelines
- **Common rejection reasons:**
  - Privacy policy missing/inadequate
  - Age ratings incorrect
  - Subscription handling (must use in-app purchase)
  - Metadata (screenshots, descriptions)
  - Using deprecated APIs
- **Problem:** Rejection delays launch by weeks
- **Solution needed:** Pre-submission review checklist

---

5. Web-Specific Challenges

Critical Questions

Q1: How do we optimize for SEO?

- Need customers to find you via Google
- **Problem:** Single Page Apps (React) are initially poor for SEO
- **Solution:** Server-side rendering (Next.js, Nuxt.js)
- **Decision:** Which pages need SEO? (Blog, landing, search results)

Q2: How do we handle browser compatibility?

- Chrome, Safari, Firefox, Edge + mobile browsers
- **Problem:** Different JavaScript support, CSS rendering
- **Decision:** Which browser versions to support?
- **Testing burden:** Cross-browser testing strategy

Q3: How do we optimize page load speed?

- Google penalizes slow sites in SEO
- **Problems:**
  - Large JavaScript bundles
  - Unoptimized images
  - Too many API calls
- **Solutions:** Code splitting, lazy loading, CDN, image optimization
- **Target:** Under 2 seconds for first contentful paint

Q4: How do we handle responsive design?

- Desktop, tablet, mobile browser views
- **Problem:** Different layouts, different interactions
- **Decision:** Mobile-first or desktop-first design?
- **Testing:** Must test all breakpoints

Q5: How do we manage client-side state?

- User data, form state, API cache
- **Options:** React Query, SWR, Redux, Zustand
- **Problem:** Over-fetching, stale data, cache invalidation
- **Decision:** State management strategy affects performance

Engineering Challenges

Challenge 1: Progressive Web App (PWA) Features

- Should web app work offline like mobile?
- **Decision:** Service workers for caching?
- **Problem:** Cache invalidation, storage limits
- **Trade-off:** Complexity vs functionality

Challenge 2: Analytics and Tracking

- Need to understand user behavior
- **Options:** Google Analytics, Mixpanel, Heap, PostHog
- **Problems:**
  - Privacy regulations (GDPR cookie consent)
  - Ad blockers blocking analytics
  - Performance impact
- **Decision:** What to track? How to respect privacy?

Challenge 3: Form Handling and Validation

- Complex booking forms with many fields
- **Problems:**
  - Client-side vs server-side validation
  - User experience for errors
  - Accessibility (screen readers)
- **Decision:** Validation library? Custom solution?

Challenge 4: Session Management

- Keep users logged in across browser tabs
- **Problems:**
  - Cookie vs localStorage for tokens
  - XSS attacks stealing tokens
  - CSRF protection
- **Technical:** Secure cookie configuration

Challenge 5: Performance Monitoring

- Detect slow pages, errors in production
- **Options:** Sentry, LogRocket, DataDog

- **Problem:** Production issues invisible without monitoring
- **Decision:** What level of detail? Cost vs value

---

6. Data Management & Scalability

Critical Questions

Q1: How do we structure the database for new customer data?

- User accounts, bookings, payment history
- **Decision:** New tables in existing DB or separate database?
- **Problem:** Data relationships, foreign keys to existing rental data
- **Normalization:** How much to duplicate vs reference?

Q2: How do we handle data migrations?

- Schema changes as features evolve
- **Problem:** Zero-downtime deployments
- **Decision:** Migration strategy, rollback plans
- **Risk:** Data loss, inconsistent state

Q3: How do we cache data for performance?

- Reduce database load, faster responses
- **Options:** Redis, Memcached, in-memory caching
- **Problems:**
  - Cache invalidation (hardest problem in CS)
  - Stale data shown to users
  - Memory constraints
- **Decision:** What to cache? Cache duration?

Q4: How do we handle concurrent operations?

- Multiple users booking same vehicle at same time
- **Problem:** Race conditions, double-booking
- **Solution:** Database locks, optimistic locking, transactions
- **Decision:** Performance vs consistency trade-off

Q5: How do we scale as user base grows?

- More users = more database load, API calls

- **Options:** Horizontal scaling, load balancing, database sharding
- **Problem:** Most solutions add significant complexity
- **Decision:** Over-engineer early or optimize when needed?

Engineering Challenges

Challenge 1: Data Consistency Across Systems

- Customer books on mobile, views on web
- Changes in rental software must sync to apps
- **Problem:** Eventually consistent vs strongly consistent
- **Decision:** Real-time sync or acceptable delay?

Challenge 2: File Storage and Management

- User uploads (driver's license, profile photos)
- **Options:** AWS S3, Google Cloud Storage, local storage
- **Problems:**
  - Cost, bandwidth, geographic distribution
  - Image processing (resize, compress)
  - Security (signed URLs, access control)
- **Decision:** Storage strategy, CDN usage

Challenge 3: Search Functionality

- Users search for available vehicles by criteria
- **Problems:**
  - Full-text search in database is slow
  - Complex filters (date range, location, vehicle type)
- **Options:** Elasticsearch, Algolia, database full-text search
- **Decision:** Complexity vs performance vs cost

Challenge 4: Audit Logging

- Track all important actions for compliance and debugging
- **Problems:**
  - Log everything = storage costs, performance impact
  - Log too little = can't debug issues
- **Decision:** What to log? How long to retain?
- **Implementation:** Structured logging, log aggregation

Challenge 5: Data Backup and Recovery

- Need backups in case of data loss
- **Problems:**
  - How often? Where to store?
  - Testing recovery (often forgotten)
  - Point-in-time recovery for data corruption
- **Decision:** Backup strategy, disaster recovery plan

---

7. Third-Party Integrations

Critical Questions

Q1: How do we integrate with payment gateways?

- Already covered in section 2, but worth emphasizing
- **Problem:** Each gateway has different APIs, webhooks
- **Decision:** Single gateway or multiple for redundancy?

Q2: How do we send transactional emails?

- Booking confirmations, password resets, receipts
- **Options:** SendGrid, Mailgun, AWS SES, Postmark
- **Problems:**
  - Email deliverability (spam filters)
  - Templates, personalization
  - Tracking opens/clicks
- **Decision:** DIY or use email service?

Q3: How do we send SMS notifications?

- Optional: SMS for pickup reminders
- **Options:** Twilio, AWS SNS, Plivo
- **Problems:**
  - International phone numbers
  - Cost per SMS
  - Carrier restrictions
- **Decision:** SMS vs push notifications vs email

Q4: How do we integrate mapping/location services?

- Show rental locations, delivery tracking
- **Options:** Google Maps, Mapbox, Apple Maps
- **Problems:**
  - API costs (Google Maps can be expensive)
  - Rate limits
  - Offline maps?
- **Decision:** Which mapping service? Cost vs features

Q5: How do we handle analytics and crash reporting?

- Already mentioned, but needs integration decision
- **Problem:** Multiple services = multiple SDKs = larger app size
- **Decision:** Minimal viable analytics vs comprehensive

Engineering Challenges

Challenge 1: Webhook Security

- Third parties send data to your server
- **Problems:**
  - Verify webhook is authentic (not attacker)
  - Signature validation
  - Replay attack prevention
- **Implementation:** HMAC signatures, timestamp checking

Challenge 2: API Rate Limits

- Third-party services limit requests
- **Problems:**
  - Exceeding limits = blocked or charged more
  - Need retry logic with exponential backoff
- **Decision:** Caching strategy, request batching

Challenge 3: Service Outages

- Third-party service goes down (happens regularly)
- **Problems:**
  - Your app functionality breaks
  - Need graceful degradation
- **Decision:** Fallback strategies, circuit breakers

Challenge 4: Cost Management

- Many services charge per API call
- **Problems:**
  - Costs can spiral unexpectedly
  - Need monitoring and alerting
- **Decision:** Which features are worth the cost?

---

8. Testing & Quality Assurance

Critical Questions

Q1: What's our testing strategy?

- Unit tests, integration tests, end-to-end tests
- **Problem:** Testing takes time, slows development
- **Decision:** What coverage is sufficient? 80%? 90%?
- **Trade-off:** Speed vs quality

Q2: How do we test payment flows without real money?

- Payment gateways have test modes
- **Problem:** Test mode doesn't catch all edge cases
- **Decision:** When to test with real (small) transactions?

Q3: How do we test on different devices?

- Can't buy every Android phone model
- **Options:** Device farms (BrowserStack, Sauce Labs), physical devices
- **Problem:** Cost vs coverage
- **Decision:** Which devices to prioritize?

Q4: How do we handle beta testing?

- Need real users before full launch
- **Options:** TestFlight (iOS), Google Play beta tracks (Android)
- **Problems:**
  - Recruiting beta testers
  - Gathering feedback
  - Bug reporting process

- **Decision:** Open beta vs closed beta? How many testers?

Q5: How do we test API changes without breaking production?

- Need to update APIs while apps are in use
- **Options:** Feature flags, canary deployments, blue-green deployment
- **Problem:** Coordination complexity
- **Decision:** Deployment strategy

Engineering Challenges

Challenge 1: Test Data Management

- Need realistic data for testing
- **Problems:**
  - Privacy (can't use real customer data)
  - Creating synthetic data
  - Resetting test databases
- **Solution:** Seed scripts, data anonymization

Challenge 2: Automated Testing Infrastructure

- CI/CD pipelines for automatic testing
- **Options:** GitHub Actions, GitLab CI, Jenkins, CircleCI
- **Problems:**
  - Setup complexity
  - Test flakiness (tests fail randomly)
  - Slow test suites
- **Decision:** What to automate? Where's the ROI?

Challenge 3: Load Testing

- How does system perform under high traffic?
- **Problems:**
  - Simulating realistic user behavior
  - Cost of load testing infrastructure
  - Interpreting results
- **Tools:** JMeter, k6, Locust
- **Decision:** When to load test? What's acceptable performance?

Challenge 4: Security Testing

- Vulnerability scanning, penetration testing
- **Problems:**
  - Specialized expertise required
  - Can be expensive
  - Ongoing (not one-time)
- **Decision:** DIY vs hire security firm?

---

9. DevOps & Infrastructure

Critical Questions

Q1: Where do we host the applications?

- **Options:** AWS, Google Cloud, Azure, DigitalOcean, Heroku
- **Considerations:** Cost, complexity, scaling, support
- **Decision:** Managed services (expensive but easy) vs self-managed (cheap but complex)

Q2: How do we deploy updates without downtime?

- Zero-downtime deployment strategy
- **Options:** Rolling updates, blue-green deployment, canary releases
- **Problem:** Database migrations during deployment
- **Decision:** Deployment automation level

Q3: How do we monitor production systems?

- Know when something breaks before users complain
- **Monitoring needs:**
  - Server health (CPU, memory, disk)
  - API response times
  - Error rates
  - Database performance
- **Options:** DataDog, New Relic, AWS CloudWatch, Prometheus + Grafana
- **Decision:** What to monitor? Alert thresholds?

Q4: How do we handle secrets and configuration?

- API keys, database passwords, certificates
- **Problem:** Can't commit to Git (security risk)
- **Options:** Environment variables, secrets management (Vault, AWS Secrets Manager)

- **Decision:** Secrets management strategy

Q5: How do we ensure high availability?

- System uptime target: 99.9%? 99.99%?
- **Problems:**
  - Load balancing
  - Database replication
  - Disaster recovery
- **Trade-off:** Cost vs uptime guarantee

Engineering Challenges

Challenge 1: Continuous Integration/Continuous Deployment (CI/CD)

- Automate build, test, deploy pipeline
- **Problems:**
  - Pipeline configuration complexity
  - Test flakiness blocking deployments
  - Rollback mechanisms
- **Decision:** Deployment frequency, automation level

Challenge 2: Database Scaling

- As data grows, queries slow down
- **Options:**
  - Vertical scaling (bigger server)
  - Horizontal scaling (read replicas)
  - Database sharding
- **Problem:** Sharding is complex, hard to reverse
- **Decision:** When to scale? How?

Challenge 3: CDN Configuration

- Serve images, CSS, JS from edge locations
- **Options:** CloudFlare, AWS CloudFront, Fastly
- **Problems:**
  - Cache invalidation
  - SSL certificate management
  - Cost optimization

- **Decision:** What to CDN? What to serve directly?

Challenge 4: Logging and Debugging

- Production issues need investigation
- **Problems:**
  - Log volume (expensive to store)
  - Finding relevant logs among millions
  - Sensitive data in logs (PII, passwords)
- **Options:** ELK stack, Splunk, CloudWatch Logs, Papertrail
- **Decision:** Log retention, searching strategy

Challenge 5: Disaster Recovery

- What if entire region goes down?
- **Needs:**
  - Multi-region deployment
  - Database backups in different locations
  - Failover automation
- **Problem:** Significant cost and complexity
- **Decision:** How much redundancy is worth it?

---

10. Compliance & Legal

Critical Questions

Q1: How do we handle GDPR/privacy compliance?

- European users have data rights
- **Requirements:**
  - Data export (user downloads their data)
  - Data deletion (right to be forgotten)
  - Consent management
  - Privacy policy
- **Problem:** Implementation complexity, ongoing maintenance
- **Decision:** Which markets require which compliance?

Q2: How do we handle PCI DSS for payments?

- Payment card industry security standards

- **Good news:** Using Stripe/payment processor handles most of this
- **Still need:** Secure network, access control, regular testing
- **Problem:** Annual compliance audit
- **Decision:** Self-assessment or hire auditor?

Q3: How do we handle Terms of Service and licensing?

- Legal agreements users must accept
- **Problems:**
  - Version tracking (user accepted which version?)
  - Force re-acceptance on major changes
  - Age verification (COPPA for under 13)
- **Technical:** Acceptance tracking in database

Q4: How do we handle accessibility (ADA/WCAG)?

- Legal requirement in many jurisdictions
- **Requirements:**
  - Screen reader support
  - Keyboard navigation
  - Color contrast
  - Alt text for images
- **Problem:** Often forgotten until sued
- **Decision:** Compliance level (AA vs AAA)?

Q5: How do we handle data retention and deletion?

- Can't keep user data forever (GDPR)
- **Problems:**
  - Which data to delete when?
  - Legal holds (can't delete if under investigation)
  - Audit trails vs privacy
- **Decision:** Retention policies per data type

Engineering Challenges

Challenge 1: Data Encryption

- At rest (database) and in transit (API calls)
- **Requirements:**

    - SSL/TLS certificates
    - Database encryption
    - Encryption key management
- **Problem:** Performance impact, key rotation
- **Decision:** What needs encryption? What's sufficient?

Challenge 2: User Consent Management

- Cookie consent, marketing opt-ins, data processing
- **Problems:**
    - UI/UX for consent (annoying but necessary)
    - Tracking consent state
    - Respecting preferences
- **Technical:** Consent management platform or DIY?

Challenge 3: Audit Trails

- Track who did what when (for compliance)
- **Problems:**
    - Every action needs logging
    - Immutable logs (can't be tampered)
    - Long-term storage
- **Implementation:** Append-only log storage

Challenge 4: Localization and Internationalization

- If serving multiple countries
- **Problems:**
    - Multiple languages
    - Date/time formats
    - Currency display
    - Legal requirements per country
- **Decision:** Which markets to support?

---

11. User Experience & Design

Critical Questions

Q1: How do we handle errors gracefully?

- API fails, network down, payment declined
- **Problem:** Technical errors need user-friendly messages
- **Decision:** Error messaging strategy
- **Examples:**
  - "Payment declined" vs "Your card was declined. Please try another payment method."
  - "500 error" vs "Something went wrong. We're looking into it."

Q2: How do we handle loading states?

- API calls take time
- **Options:** Spinners, skeleton screens, progress indicators
- **Problem:** Perceived performance vs actual performance
- **Decision:** Loading UX patterns per context

Q3: How do we handle edge cases in UI?

- Empty states (no bookings yet)
- Error states (failed to load)
- Long content (user has 100 bookings)
- **Problem:** Developers forget these, ship broken UI
- **Decision:** Design for every state

Q4: How do we ensure accessibility?

- Already mentioned in compliance, but UX impact
- **Problems:**
  - Screen reader navigation
  - Color blindness considerations
  - Motor impairment (large touch targets)
- **Testing:** How to validate accessibility?

Q5: How do we handle onboarding?

- First-time user experience
- **Decision:** Tutorials, tooltips, or minimal guidance?
- **Problem:** Balance education vs friction

Engineering Challenges

Challenge 1: Animation Performance

- Smooth animations = professional feel

- **Problem:** Janky animations on low-end devices
- **Decision:** Animation complexity vs performance
- **Technical:** Hardware acceleration, frame rate optimization

Challenge 2: Form Validation UX

- When to show errors? As they type or on submit?
- **Problem:** Too early = annoying, too late = frustrating
- **Decision:** Validation timing strategy

Challenge 3: Internationalization (i18n)

- Supporting multiple languages
- **Problems:**
  - Text length varies (German longer than English)
  - Right-to-left languages (Arabic, Hebrew)
  - Date/number formatting
- **Technical:** i18n library, translation management

Challenge 4: Dark Mode

- Increasingly expected feature
- **Problem:** Every screen needs dark variant
- **Decision:** Support dark mode or not? (Adds development time)

---

12. Business Logic & Domain-Specific Challenges

Critical Questions

Q1: How do we handle booking conflicts?

- Vehicle already booked for those dates
- **Problem:** User starts booking, someone else completes first
- **Decision:** Optimistic UI or real-time availability check?
- **Technical:** Locking mechanism, timeout strategy

Q2: How do we calculate pricing?

- Base rate + extras + taxes + discounts
- **Problems:**
  - Complex business rules

    - Tax calculation per location
    - Promotional codes
- **Decision:** Server-side calculation (secure) vs client-side (responsive)

Q3: How do we handle delivery tracking?

- Show vehicle location, estimated arrival
- **Options:** GPS integration, manual updates, third-party logistics
- **Problem:** Real-time location = privacy concerns, battery drain
- **Decision:** Tracking granularity

Q4: How do we handle vehicle availability?

- Vehicles in maintenance, damaged, already rented
- **Problem:** Availability changes in real-time
- **Decision:** Cache strategy, stale data handling

Q5: How do we handle customer support?

- In-app chat, email, phone?
- **Options:** Intercom, Zendesk, custom solution
- **Decision:** Support channel integration

Engineering Challenges

Challenge 1: Booking State Machine

- Complex states: reserved → confirmed → vehicle assigned → in-progress → completed → disputed
- **Problem:** State transitions need validation
- **Decision:** State machine implementation

Challenge 2: Notification Strategy

- When to notify users? What triggers?
- **Problems:**
    - Too many = users disable notifications
    - Too few = users miss important updates
- **Decision:** Notification rules, frequency limits

Challenge 3: Document Management

- Driver's license, insurance, rental agreements
- **Problems:**

- Verification workflow
- Document expiration tracking
- Storage and retrieval

- **Decision:** OCR for document reading? Manual review?

Challenge 4: Damage Reporting

- Users report vehicle damage with photos
- **Problems:**
  - Photo upload and storage
  - Dispute resolution workflow
  - Insurance integration
- **Decision:** Process design, technical implementation

Summary: Why These Problems Need Engineers

What This List Shows

1. These are judgment calls, not coding tasks

- AI can't decide your business rules
- Trade-offs require context and experience
- No "right answer" - depends on priorities

2. These problems are interconnected

- Payment choice affects mobile integration
- Authentication affects API design
- Caching affects consistency
- One wrong decision cascades

3. These need ongoing maintenance

- Not "build once and done"
- APIs evolve, platforms update, regulations change
- Need someone who understands the system

4. Debugging requires expertise

- When production breaks at 2am
- When subtle bugs appear

- When performance degrades
- AI can't troubleshoot what it can't see

Key Takeaway

This is why PM + Claude Code won't work:

- PM doesn't know how to answer these questions
- Claude Code can implement solutions, but can't decide which solution
- Each question has 3-10 possible answers
- Wrong choices = technical debt, security holes, or project failure

This is why you need experienced developers:

- They've made these mistakes before
- They know the trade-offs
- They can debug the inevitable problems
- They can maintain the system long-term

What to Do With This Document

If You're Hiring Developers

Use these questions in interviews:

- "How would you handle payment webhooks?"
- "What's your approach to API versioning?"
- "How do you ensure mobile app security?"

Good answers show:

- Experience with these specific problems
- Understanding of trade-offs
- Practical solutions, not just theoretical

If You're Scoping the Project

Each decision here affects:

- Development time (complexity)
- Ongoing costs (infrastructure, third-party services)
- Maintenance burden (technical debt)

You need to decide:

- Which features are must-have vs nice-to-have
- Which trade-offs you're willing to make
- Where to spend time vs where to cut corners

If You're Evaluating Options

This list explains why:

- Option E (No-Code) can't handle many of these
- Option A (Two Developers) can address most of these
- Option I (Agency) has experience with all of these

Bottom line: Building production software is about making hundreds of technical decisions correctly. That requires experience, judgment, and expertise - not just the ability to write code.