Rental Platform MVP Strategy Document

Date: October 9, 2025

Version: 1.0

Status: Final Strategy

Executive Summary

This document outlines the strategy for launching customer-facing mobile and web platforms that integrate with our existing rental software infrastructure. The approach prioritizes speed to market, feature parity across platforms, and optimal resource allocation.

Key Decision: Build mobile app and website in parallel using two developers to launch both platforms simultaneously within 4 months.

Current Situation

What We Have

- Deployed rental software with existing infrastructure
- APIs that can be shared with customer-facing applications
- Investment in current platform (money and human labor)
- One developer on the team

What We Need

- Customer mobile app for rental management
- Customer-facing website for bookings and management
- Both platforms with feature parity and seamless integration

Core Constraint

- Single developer cannot realistically build and maintain both platforms in a reasonable timeframe
- Sequential development would take 8+ months
- Need faster time to market

Final Recommended Strategy

Architecture Overview

```
┌─────────────────────────────────────┐
│   Existing Rental Software (Backend)   │
│   - Business Logic                 │
│   - Database                      │
│   - APIs (REST/GraphQL)              │
└──────────────┬──────────────────────┘
               │
      ┌────────┴────────┐
      ↓                 ↓
┌─────────────┐  ┌─────────────┐
│ Mobile App  │  │   Website   │
│ (Flutter)   │  │(React/Next) │
│ iOS+Android │  │             │
└─────────────┘  └─────────────┘
```

Key Principle: API-first architecture where both platforms consume the same backend APIs, ensuring feature parity and data consistency.

Team Structure

Recommended Hiring Approach

Developer #1 (Current Developer)

- **Role:** Backend/API Lead & Mobile Development
- **Responsibilities:**
  - Own and maintain API layer
  - Improve/expose APIs as needed
  - Build Flutter mobile app
  - Support both web and mobile teams
  - Review API integration on both platforms

Developer #2 (New Hire - Web Developer)

- **Role:** Web Frontend Developer
- **Responsibilities:**
  - Build React/Next.js website
  - Implement web-specific features (SEO, responsive design)

- Handle payment integrations on web
- Coordinate with API lead on requirements

Hiring Options for Developer #2:

Option	Cost	Timeline	Pros	Cons
Full-time Contract	$5k-15k/month	3-4 months	Faster delivery, full focus	Higher cost
Part-time Freelance	$2.5k-7.5k/month	4-5 months	Lower cost	Slower delivery
Dev Shop Contract	$8k-20k/month	3 months	Team support, expertise	Highest cost

Recommendation: Full-time 3-4 month contract for fastest results

Where to Find:

- Upwork or Toptal (vetted freelancers)
- Local development agencies (contract basis)
- Tech communities (part-time developers seeking side work)
- LinkedIn recruitment

Technology Stack

Mobile Application

Framework: Flutter

- **Why:** Single codebase for iOS and Android (40-50% time savings)
- **Language:** Dart
- **Target Platforms:** iOS 12+, Android 8+
- **Key Libraries:**
  - Provider/Riverpod (state management)
  - Dio (API calls)
  - Firebase (push notifications)
  - Stripe Flutter SDK (payments)

Advantages:

- One developer can manage both iOS and Android
- Fast development and hot reload
- Native performance
- Large community and package ecosystem

Website

Framework: React with Next.js

- **Why:** Production-ready, excellent SEO, fast performance
- **Language:** JavaScript/TypeScript
- **Hosting:** Vercel, Netlify, or AWS
- **Key Libraries:**
  - React Query (API state management)
  - Tailwind CSS (styling)
  - Stripe.js (payments)
  - NextAuth (authentication)

Advantages:

- Industry standard (easy to find help)
- Excellent SEO for customer acquisition
- Server-side rendering for performance
- Large talent pool

Why NOT Flutter Web:

- Performance issues for complex apps
- Poor SEO capabilities
- Large bundle sizes
- Not production-ready for customer-facing platforms

Backend/APIs

Current Setup: Use existing rental software

- **Approach:** API-first development
- **Standards:** RESTful or GraphQL (depending on current setup)
- **Documentation:** OpenAPI/Swagger specification
- **Authentication:** JWT or OAuth 2.0

Core Features (Feature Parity)

Phase 1 - MVP Features (Both Platforms)

Booking & Rentals

- Browse available vehicles
- Book rentals with date/time selection
- View rental agreements
- Request vehicle changes/upgrades

Payments

- Process payments (credit card, mobile wallets)
- View payment history
- Payment notifications
- Invoice downloads

Rental Management

- Active rental dashboard
- Delivery tracking (real-time status)
- Rental modifications
- Extension requests

Account Management

- User registration/login
- Profile management
- Document uploads (driver's license, etc.)
- Support/help center

Platform-Specific Optimizations

Mobile App Advantages:

- Push notifications (rental reminders, delivery updates, payment due)
- One-tap access (home screen icon)
- Location services integration
- Mobile wallet integration (Apple Pay, Google Pay)
- Offline mode (cached rental details)
- Camera integration (document scanning)

Website Advantages:

- Detailed search and filtering (larger screen)
- Better for initial research and comparison

- No download required (lower friction)
- Works on any device with browser
- Better for lengthy forms and documentation
- Superior for desktop users

Development Timeline

Pre-Development (Weeks 1-2)

Goal: Align both developers on technical foundation

- Define complete API requirements and contracts
- Create API documentation (OpenAPI/Swagger)
- Set up shared development environment
- Design system and UI/UX guidelines
- Authentication/authorization specifications
- Payment integration specifications
- Set up project management tools

Deliverable: Signed-off API specification document

Parallel Development (Months 1-4)

Mobile Development (Developer #1):

- **Month 1:** Core app structure, authentication, API integration
- **Month 2:** Booking flow, payment integration, rental dashboard
- **Month 3:** Advanced features (tracking, notifications, profile)
- **Month 4:** Testing, bug fixes, app store submission

Web Development (Developer #2):

- **Month 1:** Project setup, authentication, responsive design
- **Month 2:** Booking flow, payment integration, SEO optimization
- **Month 3:** Rental management, user dashboard, testing
- **Month 4:** Performance optimization, cross-browser testing, deployment

API Development (Developer #1 - Ongoing):

- Continuous API improvements as needed
- Support both mobile and web teams

- Performance optimization
- Security hardening

Coordination Rituals

Weekly Sync (1 hour):

- Progress updates from both developers
- Blockers and dependencies
- API changes or requirements
- Alignment on features

Daily Async Updates:

- Slack/Discord check-ins
- Shared documentation updates
- Code repository activity

Bi-weekly Demo:

- Show working features
- Gather stakeholder feedback
- Adjust priorities if needed

Launch Preparation (Month 4-5)

Both Platforms:

- User acceptance testing
- Load testing and performance optimization
- Security audit
- Analytics setup
- Customer support documentation
- Marketing materials preparation

Mobile Specific:

- App Store submission (2-3 weeks review time)
- Google Play submission (few days review time)
- Beta testing with TestFlight/Play Console

Web Specific:

- Domain and hosting setup

- SSL certificates
- SEO optimization and meta tags
- Google Analytics/search console setup

Soft Launch: Limited user group (beta testers)
Full Launch: Public availability both platforms

Budget Considerations

Development Costs (4-Month Timeline)

Item	Estimated Cost	Notes
Web Developer Contract (3-4 months)	$15,000 - $60,000	Depends on location and experience
Current Developer (existing)	$0 additional	Already on payroll
API improvements/infrastructure	$2,000 - $5,000	Cloud costs, tools, services
Design resources (if needed)	$3,000 - $10,000	UI/UX design, assets
App Store fees	$100/year (Apple) + $25 one-time (Google)	Required for mobile
Testing devices	$1,000 - $3,000	Various phones/tablets if not available
Third-party services	$500 - $2,000	Payment processing setup, analytics, etc.

Total Estimated Range: $21,625 - $80,125

Cost Savings vs Alternatives:

- Building sequentially: +4 months opportunity cost
- Hiring two new developers: +$100k+ in salaries
- Using agency for everything: $150k - $300k

Ongoing Costs (Post-Launch)

- Hosting: $100 - $500/month
- Payment processing: 2.9% + $0.30 per transaction (Stripe standard)
- Push notifications: $0 - $100/month (Firebase free tier usually sufficient)
- Maintenance: Existing developer + contract web dev as needed

---

Risk Mitigation

Technical Risks

Risk: API bottlenecks or missing functionality
Mitigation: Define complete API requirements in Week 1-2, allow buffer time

Risk: Integration issues between platforms
Mitigation: Weekly sync meetings, shared API documentation, early integration testing

Risk: Mobile app store rejections
Mitigation: Follow Apple and Google guidelines strictly, submit early for beta review

Risk: Performance issues under load
Mitigation: Load testing before launch, scalable infrastructure planning

Team Risks

Risk: Web developer doesn't work out
Mitigation:

- Contract with clear milestones and exit clauses
- Weekly progress reviews
- Have backup candidates identified

Risk: Current developer overwhelmed with API + mobile work
Mitigation:

- Clear priority: API stability first
- Web developer handles their own API integration questions
- Consider API-only contractor if needed

Risk: Communication breakdown between developers
Mitigation:

- Required weekly syncs
- Shared documentation (Notion, Confluence)
- Code reviews for API changes

Business Risks

Risk: Feature creep delays launch
Mitigation:

- Lock MVP features in pre-development phase
- "Phase 2" parking lot for additional features

- Ruthless prioritization

Risk: Customers don't adopt new platforms
Mitigation:

- Beta testing with real customers before full launch
- Incentivize early adoption (discounts, features)
- Gather feedback and iterate quickly

Risk: Competitors launch first
Mitigation:

- Parallel development for speed
- Focus on MVP, not perfection
- Marketing and customer outreach during development

Success Metrics

Development Phase Metrics

Velocity:

- Features completed vs planned per sprint
- API endpoint completion rate
- Bug resolution time

Quality:

- Test coverage >80%
- Critical bugs: 0 before launch
- App store pre-submission reviews passed

Timeline:

- Launch date target: End of Month 4
- Milestone completion on schedule

Post-Launch Metrics (First 3 Months)

Adoption:

- Mobile app downloads: Target 500+ (adjust to your customer base)
- Website bookings: Target 200+ (adjust to your customer base)
- Active users (monthly): Target 60% of downloads

Engagement:

- Daily active users (mobile)
- Booking completion rate
- Feature usage analytics

Business:

- Revenue through new platforms
- Customer acquisition cost
- Customer satisfaction scores (NPS)
- Support ticket volume

Technical:

- App crash rate: <1%
- Website load time: <2 seconds
- API response time: <200ms average
- Uptime: >99.5%

Decision Framework

What We're Building

✅ **YES - MVP Features:**

- Core booking and rental management
- Payment processing
- Delivery tracking
- Notifications (push on app, email on web)
- User authentication and profiles
- Essential customer support features

❌ **NO - Phase 2 Features:**

- Advanced analytics dashboards
- Loyalty programs
- Referral systems
- Social features
- Complex vehicle comparison tools

- AI-powered recommendations

Platform Approach

✅ **Feature Parity:** Every critical feature works on both platforms
✅ **Platform Optimization:** UX tailored to mobile vs desktop
✅ **Unified Backend:** Single source of truth via APIs
✅ **Optional Choice:** Customers choose their preferred platform

❌ **NOT Doing:**

- Forcing app download after web booking
- Separate databases/infrastructure
- Different feature sets (causing customer frustration)
- Building sequentially (too slow)

Next Steps (Action Items)

Immediate (Week 1)

Decision-Making:

- [ ] Approve budget allocation
- [ ] Confirm timeline expectations with stakeholders
- [ ] Decide on web developer hiring approach (full-time vs part-time contract)

Hiring:

- [ ] Write job description for web developer
- [ ] Post on Upwork/Toptal/LinkedIn
- [ ] Review current developer bandwidth and API readiness

Week 2

Preparation:

- [ ] Hire web developer
- [ ] Schedule kick-off meeting with both developers
- [ ] Set up project management tools (Jira, Linear, or Trello)
- [ ] Set up communication channels (Slack, Discord)
- [ ] Create shared documentation space (Notion, Confluence)

Weeks 3-4 (Pre-Development)

Technical Foundation:

- [ ] Current developer audits existing APIs
- [ ] Document all API endpoints and data models
- [ ] Create API specification document (OpenAPI/Swagger)
- [ ] Both developers review and sign off on API spec
- [ ] Set up version control (GitHub/GitLab)
- [ ] Define authentication/authorization approach
- [ ] Select and configure payment gateway (Stripe recommended)
- [ ] Design database schema for any new tables needed

Design:

- [ ] Create or finalize UI/UX designs for both platforms
- [ ] Define design system (colors, typography, components)
- [ ] Prepare all necessary assets (logos, icons, images)

Project Setup:

- [ ] Define sprint cycles (recommended: 2-week sprints)
- [ ] Set up testing environments
- [ ] Configure CI/CD pipelines
- [ ] Set up error tracking (Sentry, Bugsnag)
- [ ] Configure analytics (Google Analytics, Mixpanel)

Month 1-4 (Development)

- [ ] Weekly sync meetings every Monday
- [ ] Sprint planning and retrospectives every 2 weeks
- [ ] Continuous API improvements and support
- [ ] Regular progress updates to stakeholders
- [ ] Beta testing preparation starting Month 3

Month 5 (Launch)

- [ ] Soft launch with beta users
- [ ] Gather feedback and fix critical issues
- [ ] Prepare marketing materials
- [ ] Customer support training
- [ ] Full public launch

Communication Plan

Internal Stakeholders

Weekly Updates:

- Progress on development milestones
- Blockers or risks
- Budget tracking
- Timeline adjustments if needed

Monthly Demos:

- Live demonstration of working features
- Gather feedback from business stakeholders
- Adjust priorities based on business needs

Development Team

Daily:

- Async status updates in Slack/Discord
- Code commits and pull requests
- Documentation updates

Weekly:

- 1-hour sync meeting
- API changes discussion
- Blocker resolution

Bi-weekly:

- Sprint planning
- Sprint retrospective
- Demo of completed features

Key Principles to Remember

1. **API-First:** Everything flows from well-designed APIs
2. **Feature Parity:** Don't create second-class citizens on either platform
3. **Platform Optimization:** Leverage each platform's strengths
4. **Customer Choice:** Let customers use what they prefer
5. **Quality Over Speed:** But don't let perfect kill good
6. **Communication:** Over-communicate between developers
7. **Feedback Loops:** Test with real customers early and often

8. **Technical Debt:** Take some shortcuts for speed, document them for later
9. **Parallel Work:** Keep developers unblocked and working independently
10. **Ruthless Prioritization:** MVP means minimum, not maximum

---

Appendix A: Technology Alternatives Considered

Why NOT React Native for Mobile?

- Considered but rejected because Flutter has better performance
- Flutter's single codebase is cleaner than React Native's platform-specific code
- Dart is easier to learn than maintaining React Native's bridging

Why NOT Native iOS/Android?

- Too slow: 6-8 months vs 3-4 months with Flutter
- Requires two specialists or one developer learning two platforms
- Higher maintenance cost

Why NOT Flutter Web?

- Performance issues for production apps
- Poor SEO (critical for customer acquisition)
- Not industry standard for web development
- Harder to hire Flutter Web specialists

Why NOT WordPress/No-Code?

- Limited customization for rental-specific flows
- Poor integration with existing rental software APIs
- Scaling issues
- Less professional appearance

Appendix B: Glossary

API (Application Programming Interface): The connection layer that allows your mobile app and website to communicate with your rental software backend

MVP (Minimum Viable Product): The simplest version of your product that delivers core value to customers

Flutter: Google's framework for building mobile apps that work on both iOS and Android from a single codebase

React/Next.js: Popular JavaScript frameworks for building fast, SEO-friendly websites

Feature Parity: Ensuring all critical features work the same way on both mobile and web platforms

API-First Development: Designing and building the backend APIs before or alongside the frontend applications

CI/CD: Continuous Integration/Continuous Deployment - automated testing and deployment pipelines

Push Notifications: Alerts sent directly to users' mobile devices even when app is closed

---

Document Control

Version History:

Version	Date	Author	Changes
1.0	Oct 9, 2025	Strategy Team	Initial strategy document

Next Review Date: After web developer hired
Owner: [Your Name/Role]
Stakeholders: Development team, Management, Product team

END OF DOCUMENT