Comprehensive 20-Week Full-Stack Developer Curriculum: Project Replacement Plans

Executive Summary

This document provides detailed week-by-week implementation plans for two replacement portfolio projects in a 20-week full-stack developer curriculum. Both projects address real-world business problems with validated market demand, demonstrate technical sophistication beyond typical bootcamp CRUD applications, and are scoped appropriately for the given timeframes.

Project #2 (Weeks 10-12): Multi-Channel Marketing Attribution Dashboard - Addresses a $4.8B market, (amraandelma) integrates with Google Ads and Facebook Ads APIs, implements attribution modeling algorithms, and demonstrates ETL pipeline architecture.

Project #3 (Weeks 17-18): Smart Team Scheduling & Meeting Optimization Platform - Solves a $37B annual problem, (Tbrc) implements constraint satisfaction algorithms, uses real-time WebSocket communication, and integrates with Google Calendar API.

REPLACEMENT PROJECT #2: MULTI-CHANNEL MARKETING ATTRIBUTION DASHBOARD

Timeline: Weeks 10-12 (150-200 Hours)
Tech Stack: React + TypeScript, Node.js + Express + TypeScript, PostgreSQL, OAuth2, Recharts
APIs: Google Ads API, Facebook Ads API

PROJECT OVERVIEW

Business Context

Marketing attribution is a **$4.8 billion market** projected to reach $14B by 2032. (amraandelma +3) Small businesses spend $5,000-15,000/month on marketing across multiple channels (UpFlip +4) but waste up to 30% of budget due to poor attribution. (amraandelma) (Ruler Analytics) Companies using proper attribution see **15-40% ROI improvement** and **15-30% reduction in customer acquisition costs**. (amraandelma +2)

Why This Stands Out

- **Real business problem**: Addresses validated $4.8B market (amraandelma) vs typical CRUD apps
- **API integration complexity**: Multiple data sources vs single database
- **Mathematical modeling**: Attribution algorithms vs simple queries

- **Data pipeline architecture**: ETL processes, caching, optimization for 100K+ records
- **Hiring manager appeal**: Shows business acumen + technical depth Course Report

Scope Definition

SIMPLIFIED VERSION (2-3 weeks):

- 2 platforms only (Google Ads, Facebook Ads)
- 4 attribution models (first-touch, last-touch, linear, time-decay)
- Basic dashboard with 5-7 charts
- Mock data + OAuth setup (production keys optional)
- PostgreSQL with proper indexing
- Single-user application

WEEK 10: BACKEND FOUNDATION & API INTEGRATION

Daily Schedule

Monday: Database Schema & Express Setup

- **8:00-12:00**: PostgreSQL schema design
  - Create tables: `touchpoints`, `conversions`, `attribution_results`, `oauth_tokens`
  - Implement indexing strategy for 100K+ records
  - Write migration scripts with timestamps
  - Set up database connection pooling with `pg` package
- **12:30-4:30**: Express backend setup
  - Initialize TypeScript project with `tsconfig.json` (strict mode)
  - Set up Express server with middleware (helmet, cors, express-session)
  - Create environment variable structure (.env.example)
  - Implement error handling middleware
- **5:00-6:50**: LeetCode Medium - Array/String manipulation (2 problems)
  - Focus: Data transformation patterns relevant to ETL
  - Suggested: "Group Anagrams", "3Sum"

- **7:10-8:30**: Documentation
  - Write API documentation outline
  - Document database schema with relationships diagram
  - Create README with setup instructions

Tuesday: OAuth2 & Google Ads API

- **8:00-12:00**: OAuth2 implementation
  - Set up Google Cloud Console project + OAuth credentials Google Cloud
  - Implement OAuth2 flow with @google-cloud/local-auth Google Cloud
  - Build token storage with encryption (crypto module)
  - Create token refresh middleware Permify
- **12:30-4:30**: Google Ads API integration
  - Install google-ads-api package npm
  - Implement data extraction for campaigns
  - Parse API responses and normalize data structure
  - Handle pagination and rate limiting
- **5:00-6:50**: LeetCode Medium - Hash tables (2 problems)
- **7:10-8:30**: Document OAuth flow and API integration

Wednesday: Facebook API & ETL Pipeline

- **8:00-12:00**: Facebook Ads API integration
  - Install facebook-nodejs-business-sdk
  - Set up Meta Developer account
  - Implement Insights API data extraction Damiengonot
  - Normalize Facebook data structure
- **12:30-4:30**: Data transformation layer
  - Build ETL pipeline structure
  - Create data normalization functions
  - Implement bulk insert operations
  - Add data validation and error handling

- **5:00-6:50**: LeetCode Medium - Two pointers/Sliding window
- **7:10-8:30**: Document data transformation logic

Thursday: Attribution Algorithms

- **8:00-12:00**: Basic attribution implementation
  - Implement first-touch attribution `Google Support`
  - Implement last-touch attribution `Google Support`
  - Implement linear attribution `Google Support`
  - Create TypeScript interfaces
- **12:30-4:30**: Advanced attribution
  - Implement time-decay attribution `Google Support`
  - Build attribution calculation engine
  - Create SQL queries for aggregation
  - Optimize with EXPLAIN ANALYZE
- **5:00-6:50**: LeetCode Medium - Math/Statistics
- **7:10-8:30**: Document attribution formulas

Friday: REST API & Testing

- **8:00-12:00**: REST API endpoints
  - `/api/auth/google` - OAuth initiation
  - `/api/auth/google/callback` - OAuth callback
  - `/api/campaigns` - Get campaign data
  - `/api/attribution/:model` - Get results
- **12:30-4:30**: Testing & optimization
  - Write unit tests for algorithms
  - Test API endpoints
  - Implement Redis caching
  - Add performance monitoring
- **5:00-6:50**: LeetCode Medium - DFS
- **7:10-8:30**: Complete API documentation

Week 10 Milestones

- [ ] PostgreSQL database with 4 core tables + indexes
- [ ] OAuth2 flow working
- [ ] Google Ads API extracting data
- [ ] Facebook Ads API extracting data
- [ ] 4 attribution models implemented
- [ ] REST API with 5+ endpoints
- [ ] Redis caching operational
- [ ] 80%+ test coverage for attribution logic

Move-On Criteria

MUST HAVE:

- OAuth authentication working
- Can fetch data from at least Google Ads API
- Attribution calculations correct (unit tested)
- Database handles 10K+ inserts

BLOCKER ISSUES:

- OAuth failing → Check redirect URIs
- API rate limits → Implement exponential backoff
- Attribution math incorrect → Review formulas, add tests

WEEK 11: FRONTEND DASHBOARD & VISUALIZATION

Daily Schedule

Monday: React Setup & Authentication

- **8:00-12:00**: React project setup
  - Create React app with TypeScript (Vite)
  - Set up folder structure
  - Install dependencies: `recharts`, `axios`, `react-router-dom`, `date-fns`
  - Configure TypeScript types

- **12:30-4:30**: Authentication UI
  - Build Login component
  - Implement OAuth redirect handling
  - Create authenticated route wrapper
  - Add token storage
- **5:00-6:50**: LeetCode Medium - Dynamic programming
- **7:10-8:30**: Document component architecture

Tuesday: Dashboard Layout & API Service

- **8:00-12:00**: Dashboard layout
  - Create main Dashboard component
  - Build sidebar navigation
  - Implement date range picker
  - Add loading states
- **12:30-4:30**: API service layer
  - Create axios instance with interceptors
  - Build API service functions
  - Implement automatic token refresh
  - Add request/response logging
- **5:00-6:50**: LeetCode Medium - Binary search
- **7:10-8:30**: Document API service layer

Wednesday: Data Visualizations

- **8:00-12:00**: Basic visualizations
  - Install Recharts StackShare
  - Build multi-line chart for channel performance
  - Create stacked bar chart for attribution
  - Add pie chart for channel contribution
- **12:30-4:30**: Advanced visualizations
  - Build funnel chart

    - Create ROI comparison table
    - Implement tooltip customization
    - Add chart export functionality
- **5:00-6:50**: LeetCode Medium - Greedy algorithms
- **7:10-8:30**: Document chart components

Thursday: Attribution Model Selector & Details

- **8:00-12:00**: Attribution model selector
    - Build dropdown for model selection
    - Create side-by-side comparison view
    - Implement model explanation tooltips
    - Add educational modal
- **12:30-4:30**: Campaign performance details
    - Build campaign list table with pagination
    - Create individual campaign view
    - Add filtering and search
- **5:00-6:50**: LeetCode Medium - Graph algorithms
- **7:10-8:30**: Document component hierarchy

Friday: Polish & Testing

- **8:00-12:00**: Responsive design
    - Make dashboard mobile-responsive
    - Add dark mode toggle (optional)
    - Implement skeleton loading screens
    - Fix accessibility issues
- **12:30-4:30**: Frontend testing
    - Write component tests
    - Test user interactions
    - Add integration tests
    - Performance optimization

- **5:00-6:50**: LeetCode Medium - Backtracking
- **7:10-8:30**: Complete component documentation

Week 11 Milestones

- [ ] React app connected to backend
- [ ] OAuth authentication in UI
- [ ] Dashboard with 5-7 charts
- [ ] Attribution model selector working
- [ ] Date range filtering functional
- [ ] Responsive design
- [ ] 60%+ component test coverage

WEEK 12: INTEGRATION, SCALABILITY & DEPLOYMENT

Daily Schedule

Monday: Database & Performance Optimization

- **8:00-12:00**: Database optimization
  - Run EXPLAIN ANALYZE on queries
  - Add covering indexes
  - Implement query result caching
  - Set up connection pooling
- **12:30-4:30**: Backend performance
  - Implement rate limiting
  - Add compression middleware
  - Optimize ETL pipeline
  - Implement background jobs
- **5:00-6:50**: LeetCode Medium - Heap/Priority queue
- **7:10-8:30**: Document performance optimizations

Tuesday: Scalability & Security

- **8:00-12:00**: Scalability testing
  - Generate 100K+ mock touchpoints

    - Test query performance under load
    - Implement pagination
    - Add partitioning strategy
- **12:30-4:30**: Security hardening
    - Implement HTTPS
    - Add helmet.js security headers
    - Sanitize user inputs
    - Implement CSRF protection
- **5:00-6:50**: LeetCode Medium - Bit manipulation
- **7:10-8:30**: Document security measures

Wednesday: Docker & Deployment Prep

- **8:00-12:00**: Docker containerization
    - Create Dockerfile for backend
    - Create Dockerfile for frontend
    - Write docker-compose.yml
    - Test full stack in Docker
- **12:30-4:30**: Deployment preparation
    - Set up AWS/Heroku
    - Configure environment variables
    - Set up PostgreSQL database
    - Configure Redis instance
- **5:00-6:50**: LeetCode Medium - Trie/String algorithms
- **7:10-8:30**: Document Docker setup

Thursday: Deployment & Monitoring

- **8:00-12:00**: Deployment
    - Deploy backend to AWS/Heroku
    - Deploy frontend to Vercel/Netlify

    - Configure domain name (optional)
    - Set up SSL certificates
- **12:30-4:30**: Monitoring & logging
    - Implement application logging
    - Set up error tracking (Sentry)
    - Add performance monitoring
    - Create health check endpoints
- **5:00-6:50**: LeetCode Medium - Union Find
- **7:10-8:30**: Document deployment architecture

Friday: Final Testing & Demo Prep

- **8:00-12:00**: End-to-end testing
    - Test complete user flows
    - Verify attribution calculations
    - Test error scenarios
    - Performance benchmark
- **12:30-4:30**: Demo preparation
    - Create demo account with sample data
    - Prepare demo script
    - Create demo video (2-3 minutes)
    - Prepare presentation slides
- **5:00-6:50**: LeetCode Medium - Review
- **7:10-8:30**: Final documentation

Week 12 Milestones

- [ ] Application handles 100K+ touchpoints
- [ ] Sub-200ms API response times
- [ ] Deployed to production
- [ ] HTTPS enabled
- [ ] Monitoring operational
- [ ] Dockerized application

- [ ] Demo video recorded
- [ ] Complete documentation

TECHNICAL COMPLEXITY BREAKDOWN

Database Architecture (100K+ Records)

Schema Design:

```sql
CREATE TABLE touchpoints (
  id BIGSERIAL PRIMARY KEY,
  customer_id UUID NOT NULL,
  channel VARCHAR(50) NOT NULL,
  touchpoint_date TIMESTAMP NOT NULL,
  cost_micros BIGINT,
  clicks INTEGER,
  impressions INTEGER,
  metadata JSONB
);

-- Critical indexes
CREATE INDEX idx_touchpoints_customer_date ON touchpoints(customer_id, touchpoint_date DESC);
CREATE INDEX idx_touchpoints_channel_date ON touchpoints(channel, touchpoint_date DESC);
```

Performance Strategy:

- Covering indexes reduce disk I/O
- Partial indexes for recent data
- Batch inserts (1000 rows at a time)
- Redis caching (5-10 min TTL)
- Expected query time: <50ms

Attribution Algorithms

1. First-Touch: 100% credit to first interaction Properexpression **2. Last-Touch:** 100% credit to final interaction Properexpression **3. Linear:** Equal credit across touchpoints Marketing Evolution **4. Time-Decay:** Score = 2^(-(days_before_conversion) / 7) Analyticodigital

Performance Optimization

- **Caching**: Redis for API responses (5 min TTL)
- **ETL Pipeline**: Hourly scheduled jobs
- **Bulk Operations**: Batch 1000 records
- **Frontend**: Code splitting, memoization, debouncing

WHY IT STANDS OUT

Differentiation from Bootcamp Projects

Typical: CRUD todo app, single database, simple queries ITeach Recruiters

This Project:

- **Business Impact**: $4.8B market problem amraandelma
- **Technical Depth**: Multi-API integration, OAuth 2.0
- **Algorithmic Thinking**: Attribution modeling
- **Scale**: 100K+ record optimization
- **Real APIs**: Google Ads, Facebook Ads

Demo Script

"Small businesses waste 30% of marketing budgets because they don't know which channels drive sales. amraandelma Ruler Analytics I built a multi-channel attribution dashboard that integrates with Google Ads and Facebook Ads, calculates four attribution models including time-decay weighting, and visualizes ROI by channel. The backend handles 100K+ touchpoints with sub-200ms queries using PostgreSQL indexing and Redis caching."

RESOURCES

Official Documentation

- **Google Ads API:** https://developers.google.com/google-ads/api/docs/start
- **Facebook Marketing API:** https://developers.facebook.com/docs/marketing-api
- **PostgreSQL Performance:** https://www.postgresql.org/docs/current/performance-tips.html

NPM Packages

```json
{
  "dependencies": {
    "google-ads-api": "^21.0.1",
    "facebook-nodejs-business-sdk": "^22.0.2",
    "pg": "^8.11.0",
    "ioredis": "^5.3.2",
    "express": "^4.18.2",
    "recharts": "^2.10.0",
    "axios": "^1.6.0"
  }
}
```

MENTOR REVIEW TALKING POINTS

Week 10 Review

- Code review OAuth implementation
- Review attribution unit tests
- Discuss ETL pipeline architecture
- Database schema review

Questions:

- How would you handle API rate limits?
- How do you ensure attribution calculations are correct?
- What's your strategy for handling auth failures?

Week 11 Review

- Component architecture review
- Chart implementations review
- State management approach
- Accessibility audit

Questions:

- How do you handle loading states?
- What's your error boundary strategy?
- How would you optimize re-renders?

Week 12 Review

- Deployment architecture walkthrough
- Security measures implemented
- Performance benchmarks
- Monitoring strategy

Questions:

- How do you handle database migrations?
- What's your rollback strategy?
- How would you scale to 1M+ touchpoints?

AI TOOL USAGE GUIDELINES

Appropriate Usage ✅

- Generate boilerplate code
- Debug error messages
- Learn new concepts
- Code review suggestions
- Documentation generation

Inappropriate Usage ❌

- Copy-pasting entire features without understanding
- Using AI to solve LeetCode problems during practice

Best Practices

1. Always understand AI-generated code
2. Modify and adapt suggestions

3. Test thoroughly
4. Document AI tool usage
5. Use AI as a learning tool, not a shortcut

SCOPE LIMITATIONS

INCLUDED ✅

- Google Ads + Facebook Ads (2 platforms)
- 4 attribution models
- OAuth 2.0 authentication
- PostgreSQL with indexing (100K records)
- React dashboard with 5-7 charts
- Redis caching
- Production deployment
- Documentation

EXCLUDED ❌

- Additional platforms (Instagram, LinkedIn, TikTok)
- Machine learning attribution
- Real-time streaming
- Multi-user/multi-tenant
- Budget optimization
- Advanced analytics
- Mobile apps

Simplified Assumptions

- Single user (no role-based access)
- Batch processing (hourly ETL, not real-time)
- Mock data option if API credentials difficult
- Use Recharts (not custom D3.js) Hashnode

- Last 90 days only (not historical years)

REPLACEMENT PROJECT #3: SMART TEAM SCHEDULING & MEETING OPTIMIZATION PLATFORM

Timeline: Weeks 17-18 (200-250 Hours)
Tech Stack: React + TypeScript, Node.js + Express + TypeScript, PostgreSQL, Socket.io, Google Calendar API
Focus: Real-time WebSockets, Constraint Satisfaction, Calendar Integration

PROJECT OVERVIEW

Business Context

Meeting scheduling is a **$3-5 billion market** growing to $11B by 2032. (Market Research Future) **$37 billion lost annually** in the US due to meeting inefficiencies. (Impact) Employees spend **31 hours/month in unproductive meetings**, with **2-4 hours/week wasted on scheduling coordination**. (Booqed) (Ruler Analytics)

Why This Stands Out

- **Algorithmic sophistication**: Constraint satisfaction problem (NP-hard) (Cuni +2)
- **Real-time features**: WebSocket implementation
- **Multi-objective optimization**: Minimize inconvenience + maximize attendance + fairness
- **Complex domain**: Time zones, calendar APIs, availability conflicts
- **Hiring appeal**: "I solved an NP-hard scheduling problem"

Scope Definition

SIMPLIFIED VERSION (2 weeks):

- Google Calendar API only
- 3-10 participants maximum
- Weighted scoring algorithm
- Basic fairness tracking
- Real-time availability updates
- Single team/organization

WEEK 17: CORE SCHEDULING ENGINE & CALENDAR INTEGRATION

Daily Schedule

Monday: Google Calendar API Setup

- **8:00-12:00**: Google Calendar API setup
  - Create Google Cloud Project (Google)
  - Enable Calendar API
  - Set up OAuth 2.0 credentials (Google)
  - Install [googleapis] package
  - Implement OAuth flow (Google) (Permify)
- **12:30-4:30**: Calendar data retrieval
  - Implement FreeBusy API query (DZone)
  - Parse busy/free time blocks
  - Handle API rate limits
  - Test with multiple accounts
- **5:00-6:50**: LeetCode Medium - Interval problems
  - Required: "Merge Intervals", "Meeting Rooms II"
- **7:10-8:30**: Document Google Calendar integration

Tuesday: Scheduling Algorithm Design

- **8:00-12:00**: Algorithm design
  - Research constraint satisfaction problems (Cuni) (ScienceDirect)
  - Design weighted scoring system
  - Identify variables and constraints
  - Define scoring components
- **12:30-4:30**: Weighted scoring implementation
  - Implement [TimeSlotScorer] class
  - Build scoring functions (availability, preference, timezone, fairness)
  - Create composite score calculation
- **5:00-6:50**: LeetCode Medium - Greedy algorithms

- **7:10-8:30**: Document scoring algorithm

Wednesday: Time Zone Handling

- **8:00-12:00**: Time zone implementation
  - Install Luxon `npm`
  - Implement timezone conversion utilities
  - Build `findBusinessHourOverlaps()` function
  - Handle DST transitions
- **12:30-4:30**: Optimal time slot finder
  - Implement `findOptimalTimeSlots()` function
  - Generate candidate time slots
  - Score each slot
  - Return top 5 ranked options
- **5:00-6:50**: LeetCode Medium - Binary search on answer
- **7:10-8:30**: Document timezone handling

Thursday: Database Schema & Fairness

- **8:00-12:00**: PostgreSQL schema
  - Create tables: users, meetings, meeting_participants, fairness_scores
  - Add indexes for performance
  - Implement migration scripts
- **12:30-4:30**: Fairness tracking
  - Implement `FairnessTracker` class
  - Track compromise scores
  - Calculate fairness bonus
  - Store/retrieve from PostgreSQL
- **5:00-6:50**: LeetCode Medium - Heap/Priority queue
- **7:10-8:30**: Document fairness algorithm

Friday: REST API & Testing

- **8:00-12:00**: Express API endpoints
  - `POST /api/schedule/find-slots` - Find optimal times
  - `POST /api/schedule/create-meeting` - Book meeting
  - `GET /api/availability/:userId` - Get availability
  - `GET /api/fairness/:teamId` - Get fairness scores
- **12:30-4:30**: Integration testing
  - Test end-to-end flow
  - Test multiple participants
  - Verify Google Calendar creation
  - Test fairness updates
- **5:00-6:50**: LeetCode Medium - Dynamic programming
- **7:10-8:30**: Complete API documentation

Week 17 Milestones

- [ ] Google Calendar OAuth working
- [ ] Can query availability for multiple users
- [ ] Weighted scoring algorithm implemented
- [ ] Time zone handling functional (3+ zones)
- [ ] Optimal time slot finder working
- [ ] Fairness tracking operational
- [ ] REST API functional
- [ ] Can create meetings in Google Calendar

WEEK 18: REAL-TIME FEATURES, FRONTEND & DEPLOYMENT

Daily Schedule

Monday: WebSocket Setup

- **8:00-12:00**: Socket.io setup
  - Install Socket.io v4.6+
  - Create Socket.io server
  - Implement room-based architecture

    - Set up authentication middleware

- **12:30-4:30**: Real-time availability
    - Implement `availability:update` event
    - Broadcast changes to team
    - Handle concurrent requests (locking)
    - Test real-time updates
- **5:00-6:50**: LeetCode Medium - Graph algorithms
- **7:10-8:30**: Document WebSocket architecture

Tuesday: React Frontend

- **8:00-12:00**: React setup
    - Create React + TypeScript app
    - Install dependencies
    - Implement OAuth authentication UI
    - Create calendar visualization
- **12:30-4:30**: Meeting scheduling UI
    - Build participant selector
    - Create duration selector
    - Add date range picker
    - Implement "Find Times" button
- **5:00-6:50**: LeetCode Medium - Sliding window
- **7:10-8:30**: Document component hierarchy

Wednesday: Real-Time UI

- **8:00-12:00**: Optimal time slots display
    - Build time slot cards
    - Display ranked options with details
    - Implement "Select Time" button
    - Add visual indicators
- **12:30-4:30**: Real-time updates UI

    - Connect Socket.io client
    - Subscribe to team room
    - Update UI on availability changes
    - Show live indicator
- **5:00-6:50**: LeetCode Medium - Two pointers
- **7:10-8:30**: Document WebSocket integration

Thursday: Meeting Creation & Polish

- **8:00-12:00**: Meeting creation flow
    - Implement meeting creation UI
    - Show confirmation with time zones
    - Display success message
    - Update fairness scores
- **12:30-4:30**: Testing & polish
    - Test full flow end-to-end
    - Add loading states and error handling
    - Implement responsive design
    - Fix accessibility issues
- **5:00-6:50**: LeetCode Medium - Monotonic stack
- **7:10-8:30**: Document user flows

Friday: Deployment & Demo

- **8:00-10:00**: Docker containerization
    - Create Dockerfiles
    - Write docker-compose.yml
    - Test locally
- **10:00-12:00**: AWS deployment
    - Deploy backend to AWS
    - Deploy frontend to Vercel
    - Configure databases

    - Set up Redis
- **12:30-2:30**: Production configuration
    - Configure environment variables
    - Set up HTTPS/SSL
    - Test WebSocket over HTTPS
    - Configure CORS
- **2:30-4:30**: Demo prep & final testing
    - Create demo account
    - Record demo video (3-5 minutes)
    - Test all features
    - Performance check
- **5:00-6:50**: LeetCode Medium - Review
- **7:10-8:30**: Final documentation

Week 18 Milestones

- [ ] WebSocket real-time updates working
- [ ] React frontend with calendar visualization
- [ ] Complete meeting scheduling flow functional
- [ ] Google Calendar events created
- [ ] Fairness tracking updating
- [ ] Real-time updates across clients tested
- [ ] Deployed to production
- [ ] Demo video recorded

TECHNICAL COMPLEXITY BREAKDOWN

Handling 1000+ Employees & 10K+ Slots

Challenge: Finding optimal time across 1000 employees

Solution:

1. **Database Optimization:**

```sql
CREATE INDEX idx_meetings_time_range ON meetings
  USING GIST (tsrange(start_time, end_time));
```

2. **Query Optimization:**

- PostgreSQL range types for overlap detection
- Pre-compute availability patterns
- Cache calendar data (15-min TTL)
- Pagination for large results

3. **Algorithmic Optimization:**

- Check business hours only (9am-5pm)
- Early termination when finding 5 good slots
- Check most constrained participants first
- Parallel slot scoring

Performance Targets:

- Availability query: <200ms for 10 participants
- Optimal time finding: <5 seconds
- WebSocket latency: <100ms

Multi-Objective Optimization

Weighted Scoring:

```
Score(slot) = 0.4 * Availability(slot)
            + 0.2 * Preference(slot)
            + 0.2 * TimezoneScore(slot)
            + 0.1 * Fairness(slot)
```

Component Functions:

- **Availability:** % of participants free
- **Preference:** Proximity to preferred times

- **Timezone:** Avoid early/late hours locally
- **Fairness:** Bonus for participants who've compromised

Algorithm Complexity

- **Time:** O(n) where n = participants
- **Space:** O(n)
- **Optimizations:** Early termination, pruning, caching, heuristics

---

WHY IT STANDS OUT

Differentiation from Bootcamp Projects

Typical: Task trackers, simple CRUD (ITeach Recruiters)

This Project:

- **Solves NP-hard problem:** Constraint satisfaction
- **Real-time architecture:** WebSocket bi-directional communication
- **Complex domain:** Time zones, calendars, conflicts (Harvard Business School +2)
- **Multi-user coordination:** Concurrent requests, broadcasts
- **Business impact:** $37B annual problem

Demo Script

"Meeting scheduling wastes 2-4 hours weekly. (Booqed) (Ruler Analytics) I built an intelligent scheduler that solves a constraint satisfaction problem using weighted scoring across multiple objectives: availability, time zone preferences, and fairness rotation. The system uses WebSocket for real-time updates, integrates with Google Calendar API, and handles concurrent scheduling requests. The algorithm finds optimal meeting times in under 5 seconds for 10 participants across multiple time zones."

RESOURCES

Official Documentation

- **Google Calendar API:** https://developers.google.com/calendar/api/quickstart/nodejs
- **Socket.io:** https://socket.io/docs/v4/

- **Luxon:** https://moment.github.io/luxon/

NPM Packages

```json
{
  "dependencies": {
    "googleapis": "^105.0.0",
    "@google-cloud/local-auth": "^2.1.0",
    "socket.io": "^4.6.0",
    "socket.io-client": "^4.6.0",
    "express": "^4.18.2",
    "luxon": "^3.4.0",
    "pg": "^8.11.0",
    "ioredis": "^5.3.2",
    "react-big-calendar": "^1.8.5"
  }
}
```

MENTOR REVIEW TALKING POINTS

Week 17 Review

- Code review scoring algorithm
- Review time zone handling
- Discuss Google Calendar integration
- Database schema review

Questions:

- How does scoring handle edge cases?
- Explain time zone conversion strategy
- Walk through finding optimal time for 10 people
- What happens if Google Calendar API is down?

Week 18 Review

- WebSocket architecture review

- Frontend component architecture
- Deployment architecture
- Performance testing results

Questions:

- How do you prevent race conditions?
- Explain WebSocket authentication
- What happens if connection drops?
- How would you scale to 10,000 users?

Final Demo Questions

1. **"Explain your scheduling algorithm"**
   - Start with CSP basics
   - Describe weighted scoring
   - Show example calculation
2. **"How is this different from Calendly?"**
   - "Calendly is one-to-many. Mine optimizes groups." SaaSworthy
   - "I handle multi-party optimization with fairness rotation."
3. **"What was the hardest part?"**
   - "Time zone handling across DST transitions" Harvard Business School Tivazo
   - "Concurrent scheduling with locking"
   - "Algorithm performance optimization"
4. **"How would you scale this?"**
   - "Database indexing with GIST indexes"
   - "Redis caching for calendar data"
   - "Horizontal scaling with Redis adapter for Socket.io"
   - "Background jobs for heavy computations"

AI TOOL USAGE GUIDELINES

Appropriate Usage ✅

- Generate boilerplate Socket.io setup
- Debug time zone conversion issues
- Learn constraint satisfaction concepts
- Code review suggestions
- Generate TypeScript interfaces

Examples

- "Generate TypeScript interface for Google Calendar FreeBusy response"
- "Explain how to handle WebSocket reconnection"
- "Debug this Luxon timezone conversion"

Inappropriate Usage ❌

- Copy-pasting entire scheduling algorithm
- Using AI for LeetCode problems
- Submitting AI code without understanding

Best Practices

1. Understand AI-generated code before using
2. Adapt to your specific needs
3. Test thoroughly
4. Document AI usage with mentor
5. Use AI as learning tool

SCOPE LIMITATIONS

INCLUDED ✅

- Google Calendar API (OAuth + FreeBusy + Events)
- 3-10 participants maximum

- Weighted scoring algorithm
- Basic fairness tracking
- Real-time availability updates (WebSocket)
- Time zone handling (3+ zones)
- React frontend with calendar visualization
- Production deployment
- Documentation

EXCLUDED ❌

- Multi-calendar support (Outlook, Apple)
- Complex CSP solver (use simplified scoring)
- Machine learning preference learning
- Mobile apps
- Slack/Teams integration
- Advanced fairness algorithms
- Meeting analytics dashboard
- Budget/cost optimization

Simplified Assumptions

- Single organization (no multi-tenant)
- Google Calendar only (not Outlook/Apple)
- Basic fairness (compromise hours, not complex rotation)
- Next 7 days only (not long-term scheduling)
- Small teams (3-10 people, not 100+)

Time-Saving Strategies

1. Use official Google Calendar API examples
2. Use Socket.io boilerplate
3. Mock calendar data for frontend development
4. Use react-big-calendar for visualization

5. Deploy to Heroku/Vercel (not complex AWS setup)

IMPLEMENTATION SUCCESS FACTORS

Critical Success Patterns

For Both Projects

Week-by-Week Execution:

1. **Backend first** (APIs, databases, algorithms)
2. **Frontend second** (visualization, UI)
3. **Integration third** (deployment, testing)

Daily LeetCode Practice:

- Maintains algorithmic thinking
- Patterns relevant to project work
- Prepares for technical interviews
- 2 Medium problems daily = 10/week

Documentation Throughout:

- Write as you code (not at end)
- Include diagrams (architecture, data flow)
- Capture decisions and trade-offs
- Prepare for mentor reviews

Technical Debt Management

When Behind Schedule:

1. **Prioritize MUST HAVEs** over GOOD TO HAVEs
2. **Use mock data** if APIs are blocking
3. **Simplify features** (fewer charts, basic UI)
4. **Skip advanced optimizations** (focus on working first)
5. **Deploy early** to catch issues

When Ahead of Schedule:

1. **Add polish** (animations, dark mode)
2. **Improve testing** (higher coverage)
3. **Optimize performance** (caching, queries)
4. **Enhance documentation** (blog posts, videos)

Portfolio Presentation Strategy

Project #2 (Marketing Attribution) Presentation

Opening Hook: "Small businesses waste $1,500 monthly—30% of their marketing budget—because they can't track which channels drive sales. I built a solution."

Technical Highlights:

- "Integrated Google Ads and Facebook Ads APIs with OAuth 2.0"
- "Implemented 4 attribution models including time-decay weighting"
- "Optimized PostgreSQL to handle 100K+ touchpoints with sub-200ms queries"
- "Built ETL pipeline with Redis caching for dashboard performance"

Business Impact: "Companies using proper attribution see 15-40% ROI improvement. My tool provides enterprise-level attribution at a fraction of the cost of $890/month solutions like HubSpot."

What I Learned:

- "OAuth 2.0 token management and refresh patterns"
- "Database optimization for scale (indexing, caching)"
- "ETL pipeline architecture and error handling"
- "Making complex data accessible through visualization"

Project #3 (Scheduling Platform) Presentation

Opening Hook: "Employees waste 2-4 hours weekly coordinating meetings. That's $37 billion lost annually. I built an intelligent scheduler that reduces this to seconds."

Technical Highlights:

- "Implemented constraint satisfaction algorithm with weighted scoring"

- "Integrated Google Calendar API for availability querying and event creation"
- "Built real-time collaboration layer with WebSocket (Socket.io)"
- "Handled complex time zone scenarios across multiple regions"

Business Impact: "My algorithm finds optimal meeting times in under 5 seconds for 10 participants across time zones, considering availability, preferences, and fairness rotation."

What I Learned:

- "Constraint satisfaction problems and multi-objective optimization"
- "Real-time architecture with WebSocket and room patterns"
- "Time zone complexity (DST transitions, business hour overlaps)"
- "Google Calendar API integration and rate limit handling"

Career Positioning

How These Projects Set You Apart

Technical Differentiation:

1. **Beyond CRUD:** Demonstrates algorithmic thinking and optimization
2. **API Integration:** Shows ability to work with external services (80% of jobs require)
3. **Real-Time Systems:** WebSocket experience valuable for collaborative tools
4. **Scale Considerations:** Database optimization, caching, performance
5. **Business Acumen:** Understanding of real-world problems and ROI

Interview Talking Points:

"Tell me about a challenging project" → Discuss constraint satisfaction algorithm or attribution modeling

"How do you handle performance issues?" → Share database optimization strategies, caching, query analysis

"Have you worked with external APIs?" → Detail OAuth 2.0, rate limiting, error handling, token refresh

"What's your experience with real-time features?" → Explain WebSocket architecture, room patterns, concurrent handling

Canadian Job Market Alignment

In-Demand Skills Demonstrated:

- TypeScript (growing demand)
- React (standard for frontend)
- Node.js/Express (popular backend)
- PostgreSQL (enterprise databases)
- API integration (critical skill)
- Real-time features (collaborative tools)
- Cloud deployment (AWS/Docker)

Target Companies:

- **FinTech** (attribution analytics, data pipelines)
- **SaaS** (scheduling tools, productivity apps)
- **Marketing Tech** (attribution, analytics)
- **Enterprise Software** (collaboration tools, calendar apps)

FINAL CHECKLIST

Project #2 Completion Criteria

- [ ] OAuth working with both Google and Facebook
- [ ] Can fetch campaign data from APIs
- [ ] 4 attribution models implemented correctly
- [ ] Dashboard displays data with 5+ charts
- [ ] Database handles 100K+ records efficiently
- [ ] Deployed to production with HTTPS
- [ ] Demo video showing full flow (2-3 min)
- [ ] README with screenshots and setup guide
- [ ] Portfolio case study written

Project #3 Completion Criteria

- [ ] Google Calendar OAuth functional
- [ ] Can query availability for multiple users

- [ ] Scheduling algorithm finds optimal times
- [ ] WebSocket real-time updates working
- [ ] Can create meetings in Google Calendar
- [ ] Time zones handled correctly (3+ zones tested)
- [ ] Frontend shows ranked time slot options
- [ ] Deployed to production
- [ ] Demo video showing full flow (3-5 min)
- [ ] README with architecture diagram

Portfolio Readiness

- [ ] Both projects deployed and publicly accessible
- [ ] Demo videos recorded and uploaded
- [ ] GitHub repos clean with good commit history
- [ ] READMEs include problem, solution, tech stack, setup
- [ ] Portfolio website updated with project cards
- [ ] Technical blog posts written (optional but impressive)
- [ ] Prepared to discuss algorithms and architecture
- [ ] Can quantify business impact with statistics

CONCLUSION

These two replacement projects transform your portfolio from "bootcamp graduate" to "job-ready developer" by demonstrating:

1. **Real-world problem solving** (validated market problems with clear ROI)
2. **Technical sophistication** (algorithms, real-time systems, API integration)
3. **Scale considerations** (database optimization, caching, performance)
4. **Business acumen** (understanding marketing ROI, productivity gains)
5. **Production readiness** (deployment, monitoring, security)

Project #2 showcases data pipeline architecture, API integration, and analytical thinking through marketing attribution modeling.

Project #3 demonstrates algorithmic problem-solving, real-time systems, and complex domain mastery through intelligent scheduling optimization.

Together, these projects provide comprehensive evidence of full-stack capabilities, algorithmic thinking, and the ability to build production-ready applications that solve real business problems—exactly what Canadian

employers are seeking in 2025.

Time Investment: 350-450 hours total (Project #2: 150-200h, Project #3: 200-250h)

Career Impact: Portfolio that stands out from 90% of bootcamp graduates, demonstrates skills aligned with $70K-$90K entry-level positions in Canadian tech market.

Next Steps After Completion:

1. Apply to companies building SaaS, FinTech, MarTech, or productivity tools
2. Leverage projects in technical interviews to demonstrate problem-solving
3. Write technical blog posts explaining algorithms (boosts visibility)
4. Continue building: Add features, optimize further, learn from user feedback

Your math background combined with these algorithmically sophisticated projects positions you uniquely in the Canadian job market. Emphasize your problem-solving abilities and analytical thinking in interviews—these projects prove it conclusively.