Developer: Tiffany Morgan
Project Name: Launch Commander
Course: INEW 2334
Date: January 28, 2026
Project Type: Full-stack MEAN Application

Launch Commander Technical Documentation

Table of Contents

1. Technical Options

Technology Stack Selection

MEAN Stack Architecture

The Launch Commander project utilizes the MEAN stack (MongoDB, Express.js, Angular, Node.js) for full-stack JavaScript development, chosen for the following technical advantages:

Frontend: Angular 13
Rationale:
- Component-Based Architecture: Modular design allows reusable components for launch cards, prediction forms, and leaderboard displays
- TypeScript Support: Strong typing reduces runtime errors and improves code maintainability
- Angular Material: Pre-built UI components ensure consistent, professional design without custom CSS overhead
- Two-Way Data Binding: Simplifies form management with `[(ngModel)]` directive
- RxJS Integration: Reactive programming patterns handle asynchronous HTTP requests elegantly
- Built-in Router: Future-proofs application for multi-page expansion

Alternatives Considered:
- React: More lightweight but requires additional libraries (Redux, Material-UI) for comparable functionality
- Vue.js: Simpler learning curve but smaller ecosystem and fewer enterprise-ready UI libraries
- Vanilla JavaScript: Would require significantly more development time for basic features

Decision: Angular chosen for comprehensive framework features and professional UI component library

Backend: Express.js + Node.js
Rationale:
- JavaScript Continuity: Single language across full stack reduces context switching
- Non-Blocking I/O: Node.js event loop handles concurrent API requests efficiently
- NPM Ecosystem: Access to thousands of packages (CORS, body-parser, MongoDB drivers)
- Lightweight Framework: Express provides routing without unnecessary overhead
- JSON-Native: Seamless JSON handling for API responses and database operations

Alternatives Considered:
- Django (Python): More opinionated framework, steeper learning curve, different language from frontend

- Spring Boot (Java): Enterprise-grade but heavyweight for this project scope
- Flask (Python): Microframework suitable but adds language complexity

Decision: Express.js for minimal configuration and JavaScript consistency

Database: MongoDB
Rationale:
- Flexible Schema: Launch data from external API varies by provider; NoSQL accommodates inconsistent structures
- JSON Documents: Native compatibility with JavaScript objects eliminates ORM complexity
- Horizontal Scaling: Can easily expand to distributed architecture if user base grows
- Fast Writes: Prediction submissions require quick insert operations during launch events
- Aggregation Pipeline: Complex leaderboard calculations benefit from MongoDB's aggregation framework

Alternatives Considered:
- PostgreSQL: Relational integrity valuable but rigid schema problematic for variable API data
- MySQL: Strong community support but lacks JSON flexibility
- Firebase: Real-time features attractive but vendor lock-in and limited query capabilities

Decision: MongoDB for schema flexibility and native JavaScript integration

---

Development Environment

Required Software:
- Node.js v16.x or higher
- MongoDB Community Edition v6.0+
- Angular CLI v13.x
- Visual Studio Code (recommended IDE)
- Git for version control
- Postman or similar for API testing

Development Workflow:
1. Backend runs on `localhost:3000`
2. Frontend development server on `localhost:4200`
3. MongoDB local instance on `localhost:27017`
4. Hot-reload enabled for both frontend and backend during development

2. Analytics and Tracking

User Engagement Metrics

Implemented Tracking (Current):
- Server-Side Logging:
  - Prediction submission counts (console logs)
  - Launch data fetch timestamps
  - API endpoint access patterns
  - Error frequency and types

Recommended Analytics Implementation (Future Phase):

Google Analytics 4 Integration
Purpose: Track user behavior and engagement patterns

Key Metrics to Monitor:
1. User Acquisition:
   - New vs. returning users
   - Traffic sources (direct, referral, social)
   - Geographic distribution

2. Engagement:
   - Session duration
   - Pages per session
   - Bounce rate by entry point

3. Prediction Behavior:
   - Predictions per user
   - Time spent on prediction forms
   - Drop-off rates during form completion

4. Launch Interest:
   - Most-viewed launches
   - Prediction rates by launch provider (SpaceX vs. others)
   - Time-to-prediction (how soon before launch users predict)

Implementation Steps:
1. Create Google Analytics property
2. Install `@angular/fire` or `ngx-google-analytics` package
3. Add tracking code to `index.html`
4. Implement event tracking for:
   - Prediction submissions (`gtag('event', 'prediction_submit')`)
   - Launch card interactions
   - Leaderboard views
   - Error occurrences

Custom Backend Analytics

Database-Driven Insights:

Create analytics collection in MongoDB:
```javascript
{
  date: ISODate("2026-01-29"),
  totalPredictions: 47,
  uniqueUsers: 23,
  averagePredictionsPerUser: 2.04,
  topLaunches: [
    { launchId: "xyz", predictions: 15 },
    { launchId: "abc", predictions: 12 }
  ],
  averageAccuracy: 68.5,
  mostPredictedParameter: "predictedSuccess"
}
```

Daily Aggregation Script:
- Runs nightly via cron job
- Calculates engagement metrics from predictions collection
- Stores summary for historical analysis
- Enables trend visualization on future admin dashboard

Error Tracking
Service Recommendation: Sentry.io

Benefits:
- Automatic error capture in frontend and backend
- Stack trace preservation for debugging
- Performance monitoring (API response times)
- Alert notifications for critical failures

Integration:
```javascript
// Frontend
import * as Sentry from "@sentry/angular";
Sentry.init({ dsn: "..." });

// Backend
const Sentry = require('@sentry/node');
Sentry.init({ dsn: "..." });
```

3. Search Functionality

Current Implementation

Implicit Filtering:
- Launch cards display in chronological order (earliest first)
- Leaderboard automatically sorts by total score (highest first)
- User prediction history retrieves by username

Limitations:
- No keyword search across launch names
- Cannot filter by launch provider (SpaceX, NASA, etc.)
- No date range filtering for past launches

Recommended Search Enhancements

Launch Search Features

1. Text Search:
```javascript
// Backend endpoint
app.get('/api/launches/search', async (req, res) => {
  const { query } = req.query;
  const launches = await db.collection('launches')
    .find({
      $text: { $search: query }
    })
    .toArray();
  res.json(launches);
});
```

MongoDB Text Index:
```javascript
```

```
db.collection('launches').createIndex({
  name: "text",
  provider: "text",
  mission: "text"
});
```

Frontend Implementation:
- Search bar in launch column header
- Real-time filtering as user types (debounced to avoid excessive API calls)
- Clear button to reset search

2. Filter by Provider:
```html
<mat-select [(ngModel)]="selectedProvider" (selectionChange)="filterLaunches()">
  <mat-option value="all">All Providers</mat-option>
  <mat-option value="SpaceX">SpaceX</mat-option>
  <mat-option value="NASA">NASA</mat-option>
  <mat-option value="ULA">United Launch Alliance</mat-option>
  <mat-option value="RocketLab">Rocket Lab</mat-option>
</mat-select>
```

3. Date Range Filter:
- "Next 7 Days" quick filter
- "Next Month" quick filter
- Custom date range picker (Material DatePicker)

Backend Query:
```javascript
app.get('/api/launches/filter', async (req, res) => {
  const { startDate, endDate, provider } = req.query;
  const filter = {
    date: {
      $gte: new Date(startDate),
      $lte: new Date(endDate)
    }
  };
  if (provider && provider !== 'all') {
    filter['lsp.name'] = provider;
  }
  const launches = await db.collection('launches')
    .find(filter)
    .sort({ date: 1 })
    .toArray();
  res.json(launches);
});
```

---

4. Status Filter:
- Show only "Go for Launch" missions
- Hide completed launches
- Show all launches (default)

Leaderboard Search
- Search by username
- Filter by minimum score
- Sort by different metrics (accuracy, total predictions, score)

4. Database Options

MongoDB Schema Design

Collection 1: `launches`
```javascript
{
  _id: ObjectId("..."),                    // MongoDB auto-generated ID
  apiId: "abc-123-xyz",                    // Launch Library 2 API identifier
  name: "Falcon 9 Block 5 | Starlink Group 6-103",
  status: "Go for Launch",
  date: ISODate("2026-02-15T14:30:00Z"),
  provider: {
    name: "SpaceX",
    type: "Commercial"
  },
  vehicle: "Falcon 9 Block 5",
  site: "Kennedy Space Center LC-39A",
  mission: "Starlink satellite deployment to LEO",
  image: "https://...",

  // Actual results (populated after launch)
  actualSuccess: true,
  actualMECOTime: 8.7,                    // minutes to orbit
  actualOrbitAltitude: 525,               // kilometers
  actualPayloadSuccess: true,

  lastUpdated: ISODate("2026-01-29T12:00:00Z")
}
```

Indexes:
- `{ apiId: 1 }` - Unique, for API sync
- `{ date: 1 }` - Sort chronologically
- `{ status: 1 }` - Filter by launch status
- `{ "provider.name": 1 }` - Filter by company

Collection 2: `predictions`
```javascript
{
  _id: ObjectId("..."),
  username: "TiffanyMH",
  launchId: "67890xyz",                   // References launches._id

  // User predictions
  predictedSuccess: true,
  predictedMECOTime: 9.0,
  predictedOrbitAltitude: 500,
  predictedPayloadSuccess: true,

  // Scoring results (calculated after launch)
  pointsEarned: 35,
  breakdown: {
    successPoints: 10,
    mecoPoints: 5,
    mecoBonus: 5,
    altitudePoints: 5,
    altitudeBonus: 10,
    payloadPoints: 0
  },

  timestamp: ISODate("2026-02-10T09:15:00Z")
}
```

Indexes:
- `{ username: 1, launchId: 1 }` - Unique compound index (prevent duplicate predictions)
- `{ launchId: 1 }` - Retrieve predictions for specific launch
- `{ username: 1, timestamp: -1 }` - User history sorted by date

Collection 3: `leaderboard`
```javascript
{
  _id: ObjectId("..."),
  username: "TiffanyMH",
  totalScore: 178,
  totalPredictions: 7,
  correctPredictions: 5,
  accuracyRate: 71.4,                    // percentage
  averagePoints: 25.4,
  rank: 3,
  lastUpdated: ISODate("2026-01-29T12:00:00Z")
}
```

Indexes:
- `{ username: 1 }` - Unique, one entry per user
- `{ totalScore: -1 }` - Sort leaderboard (descending)
- `{ accuracyRate: -1 }` - Alternative sort option

Database Maintenance

Backup Strategy:
- Daily Backups: Automated MongoDB dump to external storage
- Retention: Keep 30 days of daily backups, 12 months of monthly snapshots
- Tools: `mongodump` command or MongoDB Atlas automated backups

Data Cleanup:
- Archive launches older than 1 year to separate collection
- Delete unscored predictions if launch data never received
- Prune leaderboard entries for users with zero activity in 6+ months

Performance Monitoring:
- Use `explain()` to analyze query performance
- Monitor index usage with `db.collection.stats()`
- Set up alerts for slow queries (>100ms)

5. Performance Requirements

Load Time Targets

Initial Page Load:
- Target: < 3 seconds on 4G connection
- Measured: Time from navigation to interactive content

API Response Times:
- GET /api/launches: < 200ms (typical 10-50 launches)
- POST /api/predictions: < 150ms (single document insert)
- GET /api/leaderboard: < 250ms (aggregation query with sorting)

Database Query Performance:
- Simple finds: < 50ms
- Aggregations: < 150ms
- Indexed lookups: < 10ms

Optimization Strategies

Frontend:
1. Code Splitting: Lazy-load Angular modules not needed on initial page
2. Asset Optimization:
   - Minify JavaScript/CSS in production build
   - Compress images (WebP format)
   - Enable gzip compression on server
3. Caching:
   - Service Worker for offline functionality
   - Browser caching headers for static assets (1 week TTL)
4. Bundle Size: Keep main bundle < 500KB gzipped

Backend:
1. Connection Pooling: MongoDB driver maintains connection pool (default 10 connections)
2. Response Compression: Use `compression` middleware for large JSON responses
3. Query Optimization:
   - Use projection to return only needed fields
   - Limit results with `.limit()` for large datasets
4. Caching Strategy:
   - Redis cache for frequently accessed leaderboard (5-minute TTL)
   - In-memory cache for launch data (refresh every 10 minutes)

Database:
1. Indexing: All search and sort fields indexed (see Database Options section)
2. Aggregation Optimization: Use `$match` early in pipeline to reduce documents
3. Schema Design: Embedded documents for related data (avoid joins)

Scalability Considerations

Horizontal Scaling:
- Load Balancer: Nginx distributes traffic across multiple Node.js instances
- Database Sharding: Partition predictions by date range if collection exceeds 10M documents

- CDN Distribution: Serve static frontend assets from CloudFlare or Fastly

Vertical Scaling:
- Increase server RAM for larger MongoDB working set
- Multi-core CPU utilization with Node.js cluster module

Concurrency Handling:
- Expected Load: 100-500 concurrent users during major launch events
- Stress Testing: Use Artillery.io to simulate 1000 concurrent predictions
- Rate Limiting: Throttle prediction submissions to 10 per minute per user

Monitoring and Alerts

Tools:
- New Relic or Datadog: Application performance monitoring (APM)
- MongoDB Atlas Monitoring: Database metrics and slow query logs
- Pingdom or UptimeRobot: Uptime monitoring with alerts

Alert Thresholds:
- API response time > 1 second
- Database CPU usage > 80%
- Error rate > 5% of requests
- Server disk space < 20% remaining

---

6. Hosting Information
Development Environment

Local Setup:
- Backend: `http://localhost:3000`
  - Started via `node server.js` from backend directory
  - Requires MongoDB running on port 27017
- Frontend: `http://localhost:4200`
  - Started via `npm start` from frontend directory
  - Angular CLI dev server with hot-reload
- Database: MongoDB Community Edition (local install)
  - Data stored in default directory: `C:\data\db` (Windows) or `/data/db` (Mac/Linux)

Production Hosting Options

Option 1: Cloud Platform as a Service (Recommended)

Frontend Hosting: Vercel
- Cost: Free tier (hobby projects)
- Features:
  - Automatic deployments from Git repository
  - Global CDN (Edge network)
  - Free SSL certificates
  - Serverless functions for API routes (if needed)
- Deployment:
  1. Connect GitHub repository to Vercel
  2. Configure build command: `cd frontend && npm run build`
  3. Set output directory: `frontend/dist/frontend`
  4. Deploy automatically on Git push

Alternative: Netlify (similar features and pricing)

Backend Hosting: Railway or Render
- Cost:
  - Railway: $5/month (hobby tier)
  - Render: Free tier available (sleeps after inactivity)
- Features:
  - Node.js runtime environment
  - Automatic scaling
  - Environment variable management
  - Git integration for CI/CD
  - Health check monitoring
- Deployment:
  1. Connect GitHub repository
  2. Configure start command: `cd backend && node server.js`
  3. Set environment variables (MongoDB URI)
  4. Deploy on Git push

Alternative: Heroku ($7/month), DigitalOcean App Platform ($12/month)

Database Hosting: MongoDB Atlas
- Cost: Free tier (512MB storage, shared cluster)
- Features:
  - Managed MongoDB service (no maintenance)
  - Automated backups
  - Monitoring and alerts
  - Connection pooling
  - IP whitelisting for security
- Setup:
  1. Create Atlas account and cluster (5-10 minutes)
  2. Whitelist server IP address
  3. Create database user with read/write permissions
  4. Get connection string:
`mongodb+srv://username:password@cluster.mongodb.net/launchCommanderDB`
  5. Update backend environment variable

Option 2: Virtual Private Server (Advanced)

Provider: DigitalOcean Droplet
- Cost: $6/month (basic tier: 1GB RAM, 25GB SSD)
- Configuration:
  - Ubuntu 22.04 LTS
  - Nginx reverse proxy (frontend static files + backend API proxy)
  - PM2 process manager (keeps Node.js running, auto-restart)
  - Let's Encrypt SSL certificate (free)
  - Self-hosted MongoDB or connect to Atlas

Pros:
- Full control over server configuration
- Can host multiple projects on same server
- Learning opportunity for DevOps skills

Cons:
- Manual security updates required
- More complex setup
- Responsible for backups and monitoring

Domain and SSL

Domain Registration:
- Providers: Namecheap, Google Domains, Cloudflare
- Cost: $10-15/year for `.com` domain
- Suggestion: `launchcommander.app` or `launchcommander.io`

SSL Certificate:
- Automatic with Vercel/Netlify/Railway (no action required)
- Free via Let's Encrypt for VPS setup
- Enforces HTTPS for secure data transmission

Deployment Checklist

Pre-Deployment:
- [ ] Environment variables configured (MongoDB URI, API keys)
- [ ] Production build tested locally (`ng build --prod`)
- [ ] Database indexes created
- [ ] Error logging configured (Sentry or similar)
- [ ] CORS settings updated for production domain
- [ ] API rate limiting implemented

Post-Deployment:
- [ ] DNS records point to hosting provider
- [ ] SSL certificate active (HTTPS working)
- [ ] Database connection successful
- [ ] All API endpoints responding
- [ ] Frontend loads and displays data
- [ ] Predictions saving to database
- [ ] Monitoring alerts configured
- [ ] Backup schedule active

Estimated Hosting Costs

Minimal Setup (Free/Low-Cost):
- Frontend (Vercel): $0
- Backend (Render Free): $0
- Database (Atlas Free): $0
- Domain: $12/year
- Total: $1/month

Recommended Setup (Reliable):
- Frontend (Vercel): $0
- Backend (Railway): $5/month
- Database (Atlas Free): $0
- Domain: $12/year
- Monitoring (Free tiers): $0
- Total: $6/month

Professional Setup (Production-Ready):
- Frontend (Vercel Pro): $20/month
- Backend (Railway Pro): $20/month
- Database (Atlas M10): $57/month
- Domain: $12/year
- Monitoring (Sentry Team): $26/month
- Total: $124/month

Backup and Disaster Recovery

Backup Strategy:
- MongoDB Atlas: Automated daily backups (retention: 7 days on free tier)
- Git Repository: All code version-controlled on GitHub
- Database Exports: Weekly manual exports via `mongodump` stored on external drive

Recovery Plan:
1. Restore database from Atlas snapshot
2. Redeploy backend from Git repository
3. Rebuild frontend from source
4. Update DNS if hosting provider changed
5. Test all functionality before announcing service restoration

Expected Recovery Time: 1-2 hours

Maintenance Schedule

Daily:
- Automated health checks (UptimeRobot pings)
- Error log review (Sentry dashboard)

Weekly:
- Manual database backup export
- Review performance metrics (New Relic)
- Check for security updates (npm audit)

Monthly:
- Dependency updates (`npm update`)
- Database index analysis and optimization
- Review hosting costs and usage

Quarterly:
- Major framework updates (Angular minor versions)
- Performance audit and optimization
- User feedback review and feature prioritization

Document Version: 1.0
Last Updated: February 3, 2026
Next Review: February 29, 2026