ChartCraft AI - Backend Developer Specification

Current Tech Stack

Layer	Technology
Framework	FastAPI (Python 3.11+)
Database	PostgreSQL 15+
Cache	Redis 7+
ORM	SQLAlchemy 2.0 + Alembic
Authentication	JWT + bcrypt
File Storage	AWS S3 / MinIO (local dev)
Task Queue	Celery + Redis
API Documentation	FastAPI auto-generated OpenAPI
Web Server	Uvicorn + Gunicorn (production)
Data Processing	Pandas, NumPy, openpyxl
Chart Generation	Matplotlib, Plotly
AI Integration	OpenAI API (GPT-4o-mini)

Environment & Deployment

All configuration must be stored in environment variables:

```
# Database
DATABASE_URL=postgresql://user:password@localhost:5432/chartcraft
REDIS_URL=redis://localhost:6379/0

# AWS/Storage
AWS_ACCESS_KEY_ID=your_access_key
AWS_SECRET_ACCESS_KEY=your_secret_key
AWS_S3_BUCKET=chartcraft-files
AWS_REGION=us-east-1
```

```
# Authentication
JWT_SECRET_KEY=your-secret-key-here
JWT_ALGORITHM=HS256
JWT_EXPIRATION_HOURS=24

# AI Services
OPENAI_API_KEY=your_openai_key

# App Settings
DEBUG=true
ALLOWED_ORIGINS=http://localhost:3000,https://yourdomain.com
MAX_FILE_SIZE_MB=10
```

- **Local development:** `uvicorn main:app --reload --port 3001`
- **Production:** Docker + Gunicorn on AWS/GCP/Azure
- **Database migrations:** Alembic for schema versioning

Part 2 – Phase 1 Development Roadmap (6 Months)

Month 1: Foundation & Core Infrastructure

Week 1-2: Project Setup & Database

Deliverables:

- FastAPI project structure with proper module organization
- PostgreSQL database setup with SQLAlchemy models
- Redis cache configuration
- Docker development environment
- Basic authentication system (JWT)

Key Tasks:

- [ ] Set up FastAPI project structure
- [ ] Create database models and relationships
- [ ] Implement Alembic migrations
- [ ] Configure Redis for caching and sessions
- [ ] Set up Docker Compose for local development
- [ ] Implement JWT authentication endpoints

Components:

```
app/
├── models/          # SQLAlchemy models
├── schemas/         # Pydantic request/response models
```

```
├── api/            # API route handlers
├── core/           # Configuration and utilities
├── services/       # Business logic
└── tests/          # Test suites
```

Week 3-4: File Upload & Storage System

Deliverables:

- S3 integration for file storage
- File upload validation and processing
- Data source management endpoints
- Basic error handling and logging

Key Tasks:

- [ ] Implement S3 upload/download functionality
- [ ] Create file validation system (CSV, Excel)
- [ ] Build data source CRUD operations
- [ ] Set up structured logging
- [ ] Implement file cleanup and lifecycle management

Month 2: Data Processing Engine

Week 1-2: Data Analysis & Processing

Deliverables:

- CSV/Excel file processing pipeline
- Column type detection and validation
- Data cleaning and preprocessing utilities
- Basic statistical analysis

Key Tasks:

- [ ] Build robust file parsing (pandas + openpyxl)
- [ ] Implement column type inference
- [ ] Create data validation rules
- [ ] Add statistical analysis functions
- [ ] Handle missing data and outliers

Core Functions:

```
class DataProcessor:
    async def process_file(self, file_path: str) -> ProcessedDataResult
    async def analyze_columns(self, df: pd.DataFrame) -> ColumnAnalysis
    async def validate_data(self, df: pd.DataFrame) -> ValidationResult
```

```
async def clean_data(self, df: pd.DataFrame) -> pd.DataFrame
```

Week 3-4: Chart Configuration Engine

Deliverables:

- Chart type recommendation system
- Chart.js configuration generator
- Data aggregation and transformation
- Chart optimization algorithms

Key Tasks:

- [ ] Build chart type suggestion logic
- [ ] Create Chart.js config templates
- [ ] Implement data aggregation functions
- [ ] Add chart optimization rules
- [ ] Create chart preview generation

Month 3: AI Integration & Intelligence

Week 1-2: OpenAI Integration

Deliverables:

- OpenAI API client with error handling
- Prompt engineering for chart suggestions
- AI-powered data insights generation
- Response parsing and validation

Key Tasks:

- [ ] Set up OpenAI API client with retries
- [ ] Design prompts for chart recommendations
- [ ] Create insight generation pipeline
- [ ] Implement response caching
- [ ] Add fallback mechanisms for AI failures

AI Service Architecture:

```
class AIService:
    async def suggest_chart_types(self, data_summary: dict) -> List[ChartSuggestion]
    async def generate_insights(self, df: pd.DataFrame, chart_config: dict) -> AIInsights
    async def create_headlines(self, insights: dict) -> List[str]
    async def optimize_chart(self, chart_config: dict) -> OptimizationSuggestions
```

Week 3-4: Chart Intelligence & Optimization

Deliverables:

- Smart chart type selection
- Automatic color scheme suggestions
- Data-driven layout optimization
- Performance optimization for large datasets

Key Tasks:

- [ ] Implement intelligent chart type matching
- [ ] Create color palette generation
- [ ] Add automatic axis optimization
- [ ] Implement data sampling for large files
- [ ] Create chart performance benchmarks

Month 4: Export & Integration Services

Week 1-2: Export Engine

Deliverables:

- PNG/SVG image generation
- PDF report creation
- HTML embed code generation
- Export job queue system

Key Tasks:

- [ ] Build image export using Matplotlib/Plotly
- [ ] Create PDF generation with charts
- [ ] Implement HTML embed templates
- [ ] Set up Celery for background exports
- [ ] Add export job status tracking

Export Service:

```
class ExportService:
    async def export_to_image(self, chart_config: dict, format: str) -> bytes
    async def generate_pdf_report(self, charts: List[dict]) -> bytes
    async def create_embed_code(self, chart_config: dict) -> str
    async def batch_export(self, export_requests: List[dict]) -> BatchExportResult
```

Week 3-4: Newsletter Platform Integration

Deliverables:

- Mailchimp API integration
- ConvertKit API integration
- Generic webhook system
- Integration testing framework

Key Tasks:

- [ ] Implement Mailchimp template insertion
- [ ] Create ConvertKit chart embedding
- [ ] Build generic webhook handler
- [ ] Add OAuth flow for platform connections
- [ ] Create integration health monitoring

Month 5: Performance & Scalability

Week 1-2: Caching & Optimization

Deliverables:

- Redis caching strategy
- Database query optimization
- API response caching
- Background job optimization

Key Tasks:

- [ ] Implement multi-level caching
- [ ] Optimize database queries with indexes
- [ ] Add API response caching
- [ ] Optimize file processing pipeline
- [ ] Implement connection pooling

Week 3-4: Testing & Documentation

Deliverables:

- Comprehensive test suite
- API documentation
- Performance benchmarks
- Load testing results

Key Tasks:

- [ ] Write unit tests (>80% coverage)
- [ ] Create integration tests
- [ ] Add performance tests
- [ ] Generate API documentation
- [ ] Conduct load testing

Month 6: Production Readiness

Week 1-2: Security & Monitoring

Deliverables:

- Security audit and hardening
- Monitoring and alerting setup
- Error tracking and logging
- Rate limiting and DDoS protection

Key Tasks:

- [ ] Implement rate limiting
- [ ] Add security headers and CORS
- [ ] Set up monitoring (Prometheus/Grafana)
- [ ] Configure error tracking (Sentry)
- [ ] Add health check endpoints

Week 3-4: Deployment & Launch

Deliverables:

- Production deployment pipeline
- Database backup strategy
- Scaling configuration
- Launch preparation

Key Tasks:

- [ ] Set up CI/CD pipeline
- [ ] Configure auto-scaling
- [ ] Implement database backups
- [ ] Create rollback procedures
- [ ] Prepare launch monitoring

API Endpoint Specification

Authentication Endpoints

```
POST /api/auth/register
Body: {
  "email": "user@example.com",
  "password": "securepassword",
  "first_name": "John",
```

```
    "last_name": "Doe"
}
Response: {
    "user": UserResponse,
    "access_token": "jwt_token",
    "token_type": "bearer"
}

POST /api/auth/login
Body: {
    "email": "user@example.com",
    "password": "securepassword"
}
Response: {
    "access_token": "jwt_token",
    "token_type": "bearer",
    "expires_in": 86400
}

GET /api/auth/me
Headers: Authorization: Bearer jwt_token
Response: UserResponse
```

Data Management Endpoints

```
POST /api/data/upload
Form: file: UploadFile
Response: {
    "data_source_id": "uuid",
    "file_name": "data.csv",
    "row_count": 1000,
    "column_info": [
        {
            "name": "revenue",
            "type": "numeric",
            "sample_values": [100, 200, 150]
        }
    ],
    "suggested_charts": ["bar", "line"]
}

GET /api/data/sources/{source_id}/preview
Response: {
    "data": [...],
    "total_rows": 1000,
    "columns": [...]
}
```

```
POST /api/data/analyze
Body: {
  "data_source_id": "uuid",
  "analysis_type": "basic"
}
Response: {
  "statistics": {...},
  "patterns": [...],
  "recommendations": [...]
}
```

Chart Generation Endpoints

```
POST /api/charts/
Body: {
  "data_source_id": "uuid",
  "chart_type": "bar",
  "title": "Monthly Revenue",
  "config": {
    "x_column": "month",
    "y_column": "revenue",
    "color_scheme": "blue"
  },
  "generate_insights": true
}
Response: {
  "chart_id": "uuid",
  "chart_config": {...},
  "ai_insights": {
    "headline": "Revenue grew 25% in Q4",
    "insights": [...],
    "recommendations": [...]
  }
}

GET /api/charts/suggest-type
Query: data_source_id=uuid
Response: {
  "suggestions": [
    {
      "chart_type": "bar",
      "confidence": 0.9,
      "reason": "Best for comparing categories"
    }
  ]
}
```

```
PUT /api/charts/{chart_id}
Body: ChartUpdateRequest
Response: ChartResponse
```

Export Endpoints

```
POST /api/export/image
Body: {
    "chart_id": "uuid",
    "format": "png",
    "width": 800,
    "height": 600,
    "dpi": 300
}
Response: {
    "export_id": "uuid",
    "download_url": "https://...",
    "expires_at": "2024-01-01T00:00:00Z"
}

POST /api/export/embed
Body: {
    "chart_id": "uuid",
    "theme": "light",
    "responsive": true
}
Response: {
    "html_code": "<div>...</div>",
    "css_code": "...",
    "js_code": "..."
}
```

Database Schema

Core Models

```
class User(SQLAlchemyBase):
    __tablename__ = "users"

    id: UUID = Column(UUID(as_uuid=True), primary_key=True)
    email: str = Column(String(255), unique=True, nullable=False)
    password_hash: str = Column(String(255), nullable=False)
    first_name: str = Column(String(100))
    last_name: str = Column(String(100))
```

```
    subscription_tier: str = Column(String(50), default="free")
    is_active: bool = Column(Boolean, default=True)
    created_at: datetime = Column(DateTime, default=datetime.utcnow)

class DataSource(SQLAlchemyBase):
    __tablename__ = "data_sources"

    id: UUID = Column(UUID(as_uuid=True), primary_key=True)
    user_id: UUID = Column(UUID(as_uuid=True), ForeignKey("users.id"))
    file_name: str = Column(String(255), nullable=False)
    file_type: str = Column(String(10), nullable=False)
    s3_key: str = Column(String(500))
    column_metadata: dict = Column(JSON)
    row_count: int = Column(Integer)
    created_at: datetime = Column(DateTime, default=datetime.utcnow)

class Chart(SQLAlchemyBase):
    __tablename__ = "charts"

    id: UUID = Column(UUID(as_uuid=True), primary_key=True)
    user_id: UUID = Column(UUID(as_uuid=True), ForeignKey("users.id"))
    data_source_id: UUID = Column(UUID(as_uuid=True), ForeignKey("data_sources.id"))
    title: str = Column(String(255), nullable=False)
    chart_type: str = Column(String(50), nullable=False)
    chart_config: dict = Column(JSON, nullable=False)
    ai_insights: dict = Column(JSON)
    is_public: bool = Column(Boolean, default=False)
    view_count: int = Column(Integer, default=0)
    created_at: datetime = Column(DateTime, default=datetime.utcnow)
```

Security & Performance Requirements

Security Measures

- JWT token authentication with refresh tokens
- Password hashing using bcrypt
- Input validation and sanitization
- Rate limiting (100 requests/minute per user)
- CORS configuration for frontend origins
- SQL injection prevention via ORM
- File upload validation and scanning

Performance Targets

- API response time: <200ms for 95th percentile

- File upload processing: <30 seconds for 10MB files
- Chart generation: <5 seconds for datasets up to 50K rows
- Database connection pooling: 10-50 connections
- Cache hit ratio: >80% for frequently accessed data

Monitoring & Logging

- Structured JSON logging
- API performance metrics
- Error tracking and alerting
- Resource usage monitoring
- User activity analytics

Testing Strategy

Unit Tests (Week 20-22)

```
# Test coverage targets
- Models and database operations: 95%
- API endpoints: 90%
- Business logic services: 95%
- Utility functions: 100%

# Key test areas
- Authentication flow
- File upload and processing
- Chart generation logic
- AI service integration
- Export functionality
```

Integration Tests (Week 23-24)

```
# End-to-end workflows
- Complete user registration to chart export
- File upload to chart generation
- AI suggestion to final export
- Error handling and recovery
```

Deployment Architecture

Development Environment

```
# docker-compose.yml
```

```
services:
  app:
    build: .
    ports: ["3001:8000"]
    environment:
      - DATABASE_URL=postgresql://postgres:password@db:5432/chartcraft
      - REDIS_URL=redis://redis:6379/0

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: chartcraft
      POSTGRES_PASSWORD: password

  redis:
    image: redis:7-alpine
```

Production Requirements

- **Container orchestration:** Docker + Kubernetes
- **Load balancer:** Nginx or AWS ALB
- **Database:** Managed PostgreSQL (RDS/Cloud SQL)
- **Cache:** Managed Redis (ElastiCache/Cloud Memorystore)
- **File storage:** AWS S3 or Google Cloud Storage
- **Monitoring:** Prometheus + Grafana
- **Logging:** ELK Stack or Google Cloud Logging

Success Metrics & KPIs

Development KPIs

- [ ] Core API completion: Month 4
- [ ] AI integration: Month 3
- [ ] Export functionality: Month 4
- [ ] Production deployment: Month 6
- [ ] Test coverage: >90%
- [ ] API documentation: 100% complete

Performance KPIs

- [ ] File processing: <30s for 10MB files
- [ ] Chart generation: <5s for 50K rows
- [ ] API uptime: 99.9%
- [ ] Response time: <200ms (95th percentile)

Business KPIs

- [ ] Support 1000 concurrent users
- [ ] Process 10K charts/day
- [ ] Handle 100GB data storage
- [ ] 99.9% data integrity

This specification provides a complete 6-month development roadmap for building ChartCraft AI's backend, with clear milestones, technical requirements, and success criteria for each phase.