Route Selection System - Complete Development Specification

Project Overview

Build a web-based Route Selection System for managing bi-annual driver route assignments based on seniority. The system consists of an admin portal for operations management and a mobile-responsive portal for drivers to submit route preferences.

Technology Stack

Frontend

- **Framework**: React 18+ with TypeScript
- **Styling**: Tailwind CSS
- **State Management**: React Context API + useState/useReducer
- **Forms**: React Hook Form
- **Data Tables**: TanStack Table (React Table v8)
- **File Upload**: React Dropzone
- **Date/Time**: date-fns
- **Icons**: Lucide React
- **Notifications**: React Hot Toast

Backend

- **Runtime**: Node.js 18+
- **Framework**: Express.js with TypeScript
- **Database**: SQLite (development) / PostgreSQL (production)
- **ORM**: Prisma
- **Authentication**: JWT tokens with bcrypt
- **File Processing**: xlsx (SheetJS) for Excel, papaparse for CSV
- **Validation**: Zod
- **API Documentation**: OpenAPI/Swagger

Development Tools

- **Build Tool**: Vite
- **Linting**: ESLint + Prettier
- **Testing**: Vitest + React Testing Library
- **Package Manager**: npm

Project Structure

```
route-selection-system/
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   │   ├── admin/
│   │   │   │   ├── Dashboard.tsx
│   │   │   │   ├── RouteManagement.tsx
│   │   │   │   ├── EmployeeManagement.tsx
│   │   │   │   ├── SelectionPeriodConfig.tsx
│   │   │   │   ├── AssignmentProcessor.tsx
│   │   │   │   └── Reports.tsx
│   │   │   ├── driver/
│   │   │   │   ├── DriverDashboard.tsx
│   │   │   │   ├── RouteBrowser.tsx
│   │   │   │   ├── RouteDetails.tsx
│   │   │   │   ├── SelectionSubmission.tsx
│   │   │   │   └── ConfirmationScreen.tsx
│   │   │   ├── shared/
│   │   │   │   ├── Layout.tsx
│   │   │   │   ├── Navbar.tsx
│   │   │   │   ├── Modal.tsx
│   │   │   │   ├── Table.tsx
│   │   │   │   ├── FileUpload.tsx
│   │   │   │   └── Loading.tsx
│   │   │   └── auth/
│   │   │       ├── Login.tsx
│   │   │       └── ProtectedRoute.tsx
│   │   ├── contexts/
│   │   │   ├── AuthContext.tsx
│   │   │   └── AppContext.tsx
│   │   ├── hooks/
│   │   │   ├── useAuth.ts
│   │   │   ├── useRoutes.ts
│   │   │   ├── useEmployees.ts
│   │   │   └── useSelections.ts
│   │   ├── services/
│   │   │   └── api.ts
│   │   ├── types/
│   │   │   └── index.ts
│   │   ├── utils/
│   │   │   ├── validation.ts
│   │   │   ├── formatting.ts
│   │   │   └── sorting.ts
│   │   ├── App.tsx
│   │   └── main.tsx
│   ├── public/
```

```
│   ├── package.json
│   ├── tsconfig.json
│   ├── vite.config.ts
│   └── tailwind.config.js
│
├── backend/
│   ├── src/
│   │   ├── controllers/
│   │   │   ├── authController.ts
│   │   │   ├── routeController.ts
│   │   │   ├── employeeController.ts
│   │   │   ├── selectionController.ts
│   │   │   └── assignmentController.ts
│   │   ├── middleware/
│   │   │   ├── auth.ts
│   │   │   ├── errorHandler.ts
│   │   │   └── validation.ts
│   │   ├── routes/
│   │   │   ├── authRoutes.ts
│   │   │   ├── routeRoutes.ts
│   │   │   ├── employeeRoutes.ts
│   │   │   ├── selectionRoutes.ts
│   │   │   └── assignmentRoutes.ts
│   │   ├── services/
│   │   │   ├── assignmentEngine.ts
│   │   │   ├── fileProcessor.ts
│   │   │   ├── emailService.ts
│   │   │   └── reportGenerator.ts
│   │   ├── utils/
│   │   │   ├── database.ts
│   │   │   ├── validators.ts
│   │   │   └── helpers.ts
│   │   ├── types/
│   │   │   └── index.ts
│   │   └── server.ts
│   ├── prisma/
│   │   ├── schema.prisma
│   │   └── seed.ts
│   ├── package.json
│   └── tsconfig.json
│
├── README.md
└── .gitignore
```

Database Schema (Prisma)

prisma

```
// prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql" // or "sqlite" for development
  url      = env("DATABASE_URL")
}

model User {
  id        String   @id @default(uuid())
  email     String   @unique
  password  String
  role      Role     @default(DRIVER)
  employeeId String? @unique
  employee  Employee? @relation(fields: [employeeId], references: [id])
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

enum Role {
  ADMIN
  DRIVER
}

model Employee {
  id                  String    @id @default(uuid())
  employeeNumber      String    @unique
  firstName           String
  lastName            String
  email               String    @unique
  phone               String?
  hireDate            DateTime
  cdlNumber           String
  hasDoublesEndorsement Boolean  @default(false)
  hasChainExperience  Boolean   @default(false)
  currentRouteId      String?
  currentRoute        Route?    @relation("CurrentRoute", fields: [currentRouteId], references: [id])
  isActive            Boolean   @default(true)
  isEligible          Boolean   @default(true)
  ineligibilityReason String?
  ineligibilityDate   DateTime?
```

```
  user                User?
  selections          Selection[]
  assignments         Assignment[]
  disqualifications   Disqualification[]
  createdAt           DateTime  @default(now())
  updatedAt           DateTime  @updatedAt
}

model Route {
  id                  String   @id @default(uuid())
  runNumber           Int      @unique
  runType             RunType
  origin              String
  destination         String
  days                String
  startTime           String
  endTime             String
  dispatchGoal        String?
  arrivalGoal         String?
  distance            Int
  rateType            RateType
  workTime            Float
  requiresDoublesEndorsement Boolean @default(false)
  requiresChainExperience   Boolean @default(false)
  isActive            Boolean @default(true)
  currentEmployees    Employee[] @relation("CurrentRoute")
  selections          Selection[]
  assignments         Assignment[]
  createdAt           DateTime @default(now())
  updatedAt           DateTime @updatedAt
}

enum RunType {
  DOUBLES
  SINGLES
}

enum RateType {
  MILES
  FLAT_RATE
}

model SelectionPeriod {
  id            String      @id @default(uuid())
  name          String
  openingDate   DateTime
  closingDate   DateTime
```

```
  notificationDate  DateTime
  status            PeriodStatus @default(DRAFT)
  isActive          Boolean     @default(false)
  selections        Selection[]
  assignments       Assignment[]
  createdAt         DateTime    @default(now())
  updatedAt         DateTime    @updatedAt
}

enum PeriodStatus {
  DRAFT
  OPEN
  CLOSED
  PROCESSED
}

model Selection {
  id                String          @id @default(uuid())
  employeeId        String
  employee          Employee        @relation(fields: [employeeId], references: [id])
  selectionPeriodId String
  selectionPeriod   SelectionPeriod @relation(fields: [selectionPeriodId], references: [id])
  firstChoiceId     String?
  firstChoice       Route?          @relation(fields: [firstChoiceId], references: [id])
  secondChoiceId    String?
  thirdChoiceId     String?
  submittedAt       DateTime?
  confirmationNumber String?        @unique
  createdAt         DateTime        @default(now())
  updatedAt         DateTime        @updatedAt

  @@unique([employeeId, selectionPeriodId])
}

model Assignment {
  id                String          @id @default(uuid())
  employeeId        String
  employee          Employee        @relation(fields: [employeeId], references: [id])
  routeId           String?
  route             Route?          @relation(fields: [routeId], references: [id])
  selectionPeriodId String
  selectionPeriod   SelectionPeriod @relation(fields: [selectionPeriodId], references: [id])
  choiceReceived    Int?            // 1, 2, 3, or null for float pool
  assignedAt        DateTime        @default(now())
  effectiveDate     DateTime
  createdAt         DateTime        @default(now())
```

```
  updatedAt        DateTime       @updatedAt

  @@unique([employeeId, selectionPeriodId])
}

model Disqualification {
  id          String   @id @default(uuid())
  employeeId  String
  employee    Employee @relation(fields: [employeeId], references: [id])
  reason      String
  description String?
  date        DateTime @default(now())
  resolvedAt  DateTime?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

model AuditLog {
  id        String   @id @default(uuid())
  userId    String
  action    String
  entity    String
  entityId  String?
  changes   String?  // JSON string
  ipAddress String?
  createdAt DateTime @default(now())
}
```

API Endpoints

Authentication

```
POST   /api/auth/login
POST   /api/auth/logout
GET    /api/auth/me
POST   /api/auth/refresh
```

Routes

```
GET    /api/routes                # List all routes
GET    /api/routes/:id             # Get route details
POST   /api/routes                # Create route (admin)
PUT    /api/routes/:id            # Update route (admin)
DELETE /api/routes/:id             # Delete route (admin)
POST   /api/routes/import          # Import routes from Excel/CSV (admin)
GET    /api/routes/export          # Export routes to Excel
GET    /api/routes/available       # Get available routes for current period
```

Employees

```
GET    /api/employees             # List all employees (admin)
GET    /api/employees/:id          # Get employee details
POST   /api/employees              # Create employee (admin)
PUT    /api/employees/:id          # Update employee (admin)
DELETE /api/employees/:id          # Delete employee (admin)
POST   /api/employees/import       # Import employees from Excel/CSV (admin)
GET    /api/employees/export       # Export employees to Excel (admin)
PUT    /api/employees/:id/eligibility # Update eligibility status (admin)
GET    /api/employees/:id/seniority   # Get seniority ranking
```

Selection Periods

```
GET    /api/periods               # List all periods (admin)
GET    /api/periods/:id            # Get period details
POST   /api/periods                # Create period (admin)
PUT    /api/periods/:id            # Update period (admin)
DELETE /api/periods/:id            # Delete period (admin)
GET    /api/periods/active         # Get current active period
PUT    /api/periods/:id/status     # Update period status (admin)
```

Selections

```
GET    /api/selections             # List all selections for period (admin)
GET    /api/selections/my          # Get my selections (driver)
POST   /api/selections             # Submit/update selections (driver)
DELETE /api/selections/:id         # Delete selection (before deadline)
GET    /api/selections/stats       # Get selection statistics (admin)
```

Assignments

```
POST   /api/assignments/process       # Run assignment algorithm (admin)
GET    /api/assignments               # Get all assignments for period (admin)
GET    /api/assignments/my            # Get my assignment (driver)
GET    /api/assignments/preview       # Preview assignments before finalizing (admin)
PUT    /api/assignments/finalize      # Finalize and notify assignments (admin)
GET    /api/assignments/export        # Export assignments to Excel (admin)
```

Reports

```
GET    /api/reports/selection-summary
GET    /api/reports/assignment-distribution
GET    /api/reports/eligibility
GET    /api/reports/seniority-list
```

Disqualifications

```
GET    /api/disqualifications         # List disqualifications (admin)
POST   /api/disqualifications         # Create disqualification (admin)
PUT    /api/disqualifications/:id/resolve # Resolve disqualification (admin)
```

Core Algorithm: Assignment Engine

typescript

```typescript
// backend/src/services/assignmentEngine.ts

interface Employee {
  id: string;
  firstName: string;
  lastName: string;
  hireDate: Date;
  hasDoublesEndorsement: boolean;
  hasChainExperience: boolean;
  isEligible: boolean;
}

interface Route {
  id: string;
  runNumber: number;
  runType: 'DOUBLES' | 'SINGLES';
  requiresDoublesEndorsement: boolean;
  requiresChainExperience: boolean;
}

interface Selection {
  employeeId: string;
  firstChoiceId: string | null;
  secondChoiceId: string | null;
  thirdChoiceId: string | null;
}

interface Assignment {
  employeeId: string;
  routeId: string | null; // null = float pool
  choiceReceived: number | null; // 1, 2, 3, or null
}

export class AssignmentEngine {

  async processAssignments(
    employees: Employee[],
    routes: Route[],
    selections: Selection[]
  ): Promise<Assignment[]> {

    // Step 1: Filter and sort employees by seniority
    const eligibleEmployees = employees
      .filter(emp => emp.isEligible)
```

```
  .sort((a, b) => {
    // Sort by hire date (earliest = most senior)
    const dateCompare = a.hireDate.getTime() - b.hireDate.getTime();
    if (dateCompare !== 0) return dateCompare;
    // Tie-breaker: last name alphabetically
    return a.lastName.localeCompare(b.lastName);
  });

// Step 2: Create a pool of available routes
const availableRoutes = new Map(routes.map(r => [r.id, r]));

// Step 3: Process each employee by seniority
const assignments: Assignment[] = [];

for (const employee of eligibleEmployees) {
  const selection = selections.find(s => s.employeeId === employee.id);

  if (!selection) {
    // No selection submitted - assign to float pool
    assignments.push({
      employeeId: employee.id,
      routeId: null,
      choiceReceived: null
    });
    continue;
  }

  // Try to assign in preference order
  const preferences = [
    { routeId: selection.firstChoiceId, choice: 1 },
    { routeId: selection.secondChoiceId, choice: 2 },
    { routeId: selection.thirdChoiceId, choice: 3 }
  ].filter(p => p.routeId !== null);

  let assigned = false;

  for (const pref of preferences) {
    const route = availableRoutes.get(pref.routeId!);

    if (!route) {
      // Route already assigned to someone else
      continue;
    }

    // Check qualifications
    if (route.requiresDoublesEndorsement && !employee.hasDoublesEndorsement) {
      continue;
```

```
      }

      if (route.requiresChainExperience && !employee.hasChainExperience) {
        continue;
      }

      // Award the route
      assignments.push({
        employeeId: employee.id,
        routeId: route.id,
        choiceReceived: pref.choice
      });

      availableRoutes.delete(route.id);
      assigned = true;
      break;
    }

    if (!assigned) {
      // No preferences available or qualified - float pool
      assignments.push({
        employeeId: employee.id,
        routeId: null,
        choiceReceived: null
      });
    }
  }

  return assignments;
}

async validateAssignments(assignments: Assignment[]): Promise<{
  valid: boolean;
  errors: string[];
}> {
  const errors: string[] = [];
  const assignedRoutes = new Set<string>();

  for (const assignment of assignments) {
    if (assignment.routeId) {
      if (assignedRoutes.has(assignment.routeId)) {
        errors.push(`Route ${assignment.routeId} assigned to multiple employees`);
      }
      assignedRoutes.add(assignment.routeId);
    }
  }
```

```
    return {
      valid: errors.length === 0,
      errors
    };
  }
}
```

File Processing: Excel/CSV Import

typescript

```typescript
// backend/src/services/fileProcessor.ts

import * as XLSX from 'xlsx';
import Papa from 'papaparse';

interface RouteImportRow {
  'Run Number': number;
  'Run Type': string;
  'Orig': string;
  'Dest': string;
  'Days': string;
  'Start': string;
  'End': string;
  'Distance': number;
  'Rate Type': string;
  'Work Time': number;
}

interface EmployeeImportRow {
  'Employee ID': string;
  'First Name': string;
  'Last Name': string;
  'Hire Date': string;
  'CDL Number': string;
  'Email': string;
  'Phone': string;
  'Has Doubles Endorsement'?: string;
  'Has Chain Experience'?: string;
}

export class FileProcessor {

  async processRouteFile(buffer: Buffer, filename: string): Promise<any[]> {
    const ext = filename.split('.').pop()?.toLowerCase();

    if (ext === 'csv') {
      return this.processRouteCSV(buffer);
    } else if (ext === 'xlsx' || ext === 'xls') {
      return this.processRouteExcel(buffer);
    }

    throw new Error('Unsupported file format. Please use CSV or Excel.');
  }
```

```
private async processRouteExcel(buffer: Buffer): Promise<any[]> {
  const workbook = XLSX.read(buffer, { type: 'buffer' });
  const sheetName = workbook.SheetNames[0];
  const sheet = workbook.Sheets[sheetName];
  const data = XLSX.utils.sheet_to_json<RouteImportRow>(sheet);

  return data.map(row => this.mapRouteRow(row));
}

private async processRouteCSV(buffer: Buffer): Promise<any[]> {
  const text = buffer.toString('utf-8');

  return new Promise((resolve, reject) => {
    Papa.parse<RouteImportRow>(text, {
      header: true,
      dynamicTyping: true,
      skipEmptyLines: true,
      complete: (results) => {
        const mapped = results.data.map(row => this.mapRouteRow(row));
        resolve(mapped);
      },
      error: (error) => {
        reject(error);
      }
    });
  });
}

private mapRouteRow(row: RouteImportRow): any {
  return {
    runNumber: row['Run Number'],
    runType: row['Run Type'].toUpperCase() as 'DOUBLES' | 'SINGLES',
    origin: row['Orig'],
    destination: row['Dest'],
    days: row['Days'],
    startTime: row['Start'],
    endTime: row['End'],
    distance: row['Distance'],
    rateType: row['Rate Type'].toUpperCase().replace(' ', '_') as 'MILES' | 'FLAT_RATE',
    workTime: row['Work Time'],
    requiresDoublesEndorsement: row['Run Type'].toUpperCase() === 'DOUBLES',
    requiresChainExperience: false,
    isActive: true
  };
}

async processEmployeeFile(buffer: Buffer, filename: string): Promise<any[]> {
```

```
  const ext = filename.split('.').pop()?.toLowerCase();

  if (ext === 'csv') {
    return this.processEmployeeCSV(buffer);
  } else if (ext === 'xlsx' || ext === 'xls') {
    return this.processEmployeeExcel(buffer);
  }

  throw new Error('Unsupported file format. Please use CSV or Excel.');
}

private async processEmployeeExcel(buffer: Buffer): Promise<any[]> {
  const workbook = XLSX.read(buffer, { type: 'buffer' });
  const sheetName = workbook.SheetNames[0];
  const sheet = workbook.Sheets[sheetName];
  const data = XLSX.utils.sheet_to_json<EmployeeImportRow>(sheet);

  return data.map(row => this.mapEmployeeRow(row));
}

private async processEmployeeCSV(buffer: Buffer): Promise<any[]> {
  const text = buffer.toString('utf-8');

  return new Promise((resolve, reject) => {
    Papa.parse<EmployeeImportRow>(text, {
      header: true,
      skipEmptyLines: true,
      complete: (results) => {
        const mapped = results.data.map(row => this.mapEmployeeRow(row));
        resolve(mapped);
      },
      error: (error) => {
        reject(error);
      }
    });
  });
}

private mapEmployeeRow(row: EmployeeImportRow): any {
  const parseBoolean = (val?: string): boolean => {
    if (!val) return false;
    return val.toLowerCase() === 'true' || val === '1' || val.toLowerCase() === 'yes';
  };

  return {
    employeeNumber: row['Employee ID'].toString(),
```

```
    firstName: row['First Name'],
    lastName: row['Last Name'],
    email: row['Email'],
    phone: row['Phone'] || null,
    hireDate: new Date(row['Hire Date']),
    cdlNumber: row['CDL Number'],
    hasDoublesEndorsement: parseBoolean(row['Has Doubles Endorsement']),
    hasChainExperience: parseBoolean(row['Has Chain Experience']),
    isActive: true,
    isEligible: true
  };
}

validateRouteData(routes: any[]): { valid: boolean; errors: string[] } {
  const errors: string[] = [];
  const runNumbers = new Set<number>();

  routes.forEach((route, index) => {
    if (!route.runNumber) {
      errors.push(`Row ${index + 1}: Missing Run Number`);
    } else if (runNumbers.has(route.runNumber)) {
      errors.push(`Row ${index + 1}: Duplicate Run Number ${route.runNumber}`);
    } else {
      runNumbers.add(route.runNumber);
    }

    if (!route.origin || route.origin.length !== 3) {
      errors.push(`Row ${index + 1}: Invalid Origin code`);
    }

    if (!route.destination || route.destination.length !== 3) {
      errors.push(`Row ${index + 1}: Invalid Destination code`);
    }

    if (!route.distance || route.distance <= 0) {
      errors.push(`Row ${index + 1}: Invalid Distance`);
    }
  });

  return { valid: errors.length === 0, errors };
}

validateEmployeeData(employees: any[]): { valid: boolean; errors: string[] } {
  const errors: string[] = [];
  const employeeNumbers = new Set<string>();
  const emails = new Set<string>();
```

```
    employees.forEach((emp, index) => {
      if (!emp.employeeNumber) {
        errors.push(`Row ${index + 1}: Missing Employee ID`);
      } else if (employeeNumbers.has(emp.employeeNumber)) {
        errors.push(`Row ${index + 1}: Duplicate Employee ID ${emp.employeeNumber}`);
      } else {
        employeeNumbers.add(emp.employeeNumber);
      }

      if (!emp.email) {
        errors.push(`Row ${index + 1}: Missing Email`);
      } else if (emails.has(emp.email)) {
        errors.push(`Row ${index + 1}: Duplicate Email ${emp.email}`);
      } else {
        emails.add(emp.email);
      }

      if (!emp.hireDate || isNaN(emp.hireDate.getTime())) {
        errors.push(`Row ${index + 1}: Invalid Hire Date`);
      }
    });

    return { valid: errors.length === 0, errors };
  }
}
```

Frontend Components

Admin Dashboard Component

typescript

```typescript
// frontend/src/components/admin/Dashboard.tsx

import React, { useEffect, useState } from 'react';
import { Users, Route, Calendar, TrendingUp } from 'lucide-react';

interface DashboardStats {
  totalEmployees: number;
  eligibleEmployees: number;
  totalRoutes: number;
  selectionsSubmitted: number;
  selectionPercentage: number;
}

export const Dashboard: React.FC = () => {
  const [stats, setStats] = useState<DashboardStats | null>(null);
  const [activePeriod, setActivePeriod] = useState<any>(null);

  useEffect(() => {
    fetchDashboardData();
  }, []);

  const fetchDashboardData = async () => {
    // API calls to get stats and active period
    const response = await fetch('/api/dashboard/stats');
    const data = await response.json();
    setStats(data.stats);
    setActivePeriod(data.activePeriod);
  };

  if (!stats) return <div>Loading...</div>;

  return (
    <div className="p-6">
      <h1 className="text-3xl font-bold mb-6">Dashboard</h1>

      {/* Stats Cards */}
      <div className="grid grid-cols-1 md:grid-cols-4 gap-4 mb-8">
        <StatCard
          icon={<Users className="w-8 h-8 text-blue-600" />}
          title="Total Employees"
          value={stats.totalEmployees}
          subtitle={`${stats.eligibleEmployees} eligible`}
        />
        <StatCard
```

```
          icon={<Route className="w-8 h-8 text-green-600" />}
          title="Available Routes"
          value={stats.totalRoutes}
        />
        <StatCard
          icon={<Calendar className="w-8 h-8 text-purple-600" />}
          title="Selections Submitted"
          value={stats.selectionsSubmitted}
        />
        <StatCard
          icon={<TrendingUp className="w-8 h-8 text-orange-600" />}
          title="Participation"
          value={`${stats.selectionPercentage}%`}
        />
      </div>

      {/* Active Period Info */}
      {activePeriod && (
        <div className="bg-white rounded-lg shadow p-6 mb-6">
          <h2 className="text-xl font-semibold mb-4">Current Selection Period</h2>
          <div className="grid grid-cols-3 gap-4">
            <div>
              <p className="text-sm text-gray-600">Opening Date</p>
              <p className="font-semibold">{new Date(activePeriod.openingDate).toLocaleDateString()}</p>
            </div>
            <div>
              <p className="text-sm text-gray-600">Closing Date</p>
              <p className="font-semibold">{new Date(activePeriod.closingDate).toLocaleDateString()}</p>
            </div>
            <div>
              <p className="text-sm text-gray-600">Status</p>
              <span className={`px-2 py-1 rounded text-sm ${
                activePeriod.status === 'OPEN' ? 'bg-green-100 text-green-800' : 'bg-gray-100'
              }`}>
                {activePeriod.status}
              </span>
            </div>
          </div>
        </div>
      )}

      {/* Recent Activity */}
      <div className="bg-white rounded-lg shadow p-6">
        <h2 className="text-xl font-semibold mb-4">Recent Activity</h2>
        {/* Activity feed would go here */}
      </div>
    </div>
```

```
  );
};

const StatCard: React.FC<{
  icon: React.ReactNode;
  title: string;
  value: string | number;
  subtitle?: string;
}> = ({ icon, title, value, subtitle }) => {
  return (
    <div className="bg-white rounded-lg shadow p-6">
      <div className="flex items-center justify-between mb-2">
        {icon}
        <span className="text-3xl font-bold">{value}</span>
      </div>
      <h3 className="text-gray-600 text-sm">{title}</h3>
      {subtitle && <p className="text-xs text-gray-500 mt-1">{subtitle}</p>}
    </div>
  );
};
```

Driver Route Browser Component

typescript

```typescript
// frontend/src/components/driver/RouteBrowser.tsx

import React, { useState, useEffect } from 'react';
import { Search, Filter } from 'lucide-react';

interface Route {
  id: string;
  runNumber: number;
  runType: 'DOUBLES' | 'SINGLES';
  origin: string;
  destination: string;
  distance: number;
  startTime: string;
  endTime: string;
  workTime: number;
  requiresDoublesEndorsement: boolean;
}

export const RouteBrowser: React.FC = () => {
  const [routes, setRoutes] = useState<Route[]>([]);
  const [filteredRoutes, setFilteredRoutes] = useState<Route[]>([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [filterType, setFilterType] = useState<'ALL' | 'DOUBLES' | 'SINGLES'>('ALL');

  useEffect(() => {
    fetchRoutes();
  }, []);

  useEffect(() => {
    applyFilters();
  }, [searchTerm, filterType, routes]);

  const fetchRoutes = async () => {
    const response = await fetch('/api/routes/available');
    const data = await response.json();
    setRoutes(data);
  };

  const applyFilters = () => {
    let filtered = routes;

    if (filterType !== 'ALL') {
      filtered = filtered.filter(r => r.runType === filterType);
    }
```

```
  if (searchTerm) {
    filtered = filtered.filter(r =>
      r.runNumber.toString().includes(searchTerm) ||
      r.origin.toLowerCase().includes(searchTerm.toLowerCase()) ||
      r.destination.toLowerCase().includes(searchTerm.toLowerCase())
    );
  }

  setFilteredRoutes(filtered);
};

return (
  <div className="p-4">
    <h1 className="text-2xl font-bold mb-4">Available Routes</h1>

    {/* Search and Filter */}
    <div className="mb-4 space-y-3">
      <div className="relative">
        <Search className="absolute left-3 top-3 w-5 h-5 text-gray-400" />
        <input
          type="text"
          placeholder="Search by run number or location..."
          className="w-full pl-10 pr-4 py-2 border rounded-lg"
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
        />
      </div>

      <div className="flex gap-2">
        <button
          className={`px-4 py-2 rounded-lg ${
            filterType === 'ALL' ? 'bg-blue-600 text-white' : 'bg-gray-200'
          }`}
          onClick={() => setFilterType('ALL')}
        >
          All Routes
        </button>
        <button
          className={`px-4 py-2 rounded-lg ${
            filterType === 'DOUBLES' ? 'bg-yellow-600 text-white' : 'bg-gray-200'
          }`}
          onClick={() => setFilterType('DOUBLES')}
        >
          Doubles
        </button>
        <button
```

```
            className={`px-4 py-2 rounded-lg ${
              filterType === 'SINGLES' ? 'bg-blue-600 text-white' : 'bg-gray-200'
            }`}
            onClick={() => setFilterType('SINGLES')}
          >
            Singles
          </button>
        </div>
      </div>

      {/* Route Cards */}
      <div className="space-y-3">
        {filteredRoutes.map(route => (
          <RouteCard key={route.id} route={route} />
        ))}
      </div>

      {filteredRoutes.length === 0 && (
        <div className="text-center text-gray-500 py-8">
          No routes found matching your criteria
        </div>
      )}
    </div>
  );
};

const RouteCard: React.FC<{ route: Route }> = ({ route }) => {
  return (
    <div className="bg-white rounded-lg shadow p-4 hover:shadow-md transition">
      <div className="flex justify-between items-start mb-2">
        <div>
          <h3 className="font-bold text-lg">Run #{route.runNumber}</h3>
          <p className="text-sm text-gray-600">
            {route.origin} → {route.destination}
          </p>
        </div>
        <span className={`px-3 py-1 rounded text-sm font-medium ${
          route.runType === 'DOUBLES'
            ? 'bg-yellow-100 text-yellow-800'
            : 'bg-blue-100 text-blue-800'
        }`}>
          {route.runType}
        </span>
      </div>

      <div className="grid grid-cols-3 gap-2 text-sm mb-3">
```

```
      <div>
        <p className="text-gray-500">Distance</p>
        <p className="font-semibold">{route.distance} mi</p>
      </div>
      <div>
        <p className="text-gray-500">Work Time</p>
        <p className="font-semibold">{route.workTime} hrs</p>
      </div>
      <div>
        <p className="text-gray-500">Schedule</p>
        <p className="font-semibold text-xs">{route.startTime} - {route.endTime}</p>
      </div>
    </div>

    {route.requiresDoublesEndorsement && (
      <div className="text-xs text-orange-600 mb-2">
        ⚠ Requires Doubles Endorsement
      </div>
    )}

    <button
      className="w-full bg-blue-600 text-white py-2 rounded-lg hover:bg-blue-700 transition"
      onClick={() => {/* Navigate to details or add to selection */}}
    >
      View Details
    </button>
  </div>
 );
};
```

Environment Variables

```bash
# .env.example

# Database
DATABASE_URL="postgresql://user:password@localhost:5432/route_selection"

# JWT
JWT_SECRET="your-secret-key-change-in-production"
JWT_EXPIRES_IN="7d"

# Server
PORT=3000
NODE_ENV="development"

# CORS
CORS_ORIGIN="http://localhost:5173"

# Email (for notifications)
SMTP_HOST="smtp.gmail.com"
SMTP_PORT=587
SMTP_USER="your-email@company.com"
SMTP_PASSWORD="your-app-password"
SMTP_FROM="noreply@company.com"

# Frontend
VITE_API_URL="http://localhost:3000"
```

Package.json Files

Backend package.json

json

```
{
  "name": "route-selection-backend",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "tsx watch src/server.ts",
    "build": "tsc",
    "start": "node dist/server.js",
    "prisma:generate": "prisma generate",
    "prisma:migrate": "prisma migrate dev",
    "prisma:seed": "tsx prisma/seed.ts"
  },
  "dependencies": {
    "@prisma/client": "^5.7.0",
    "bcrypt": "^5.1.1",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "multer": "^1.4.5-lts.1",
    "nodemailer": "^6.9.7",
    "papaparse": "^5.4.1",
    "xlsx": "^0.18.5",
    "zod": "^3.22.4"
  },
  "devDependencies": {
    "@types/bcrypt": "^5.0.2",
    "@types/cors": "^2.8.17",
    "@types/express": "^4.17.21",
    "@types/jsonwebtoken": "^9.0.5",
    "@types/multer": "^1.4.11",
    "@types/node": "^20.10.5",
    "@types/nodemailer": "^6.4.14",
    "@types/papaparse": "^5.3.14",
    "prisma": "^5.7.0",
    "tsx": "^4.7.0",
    "typescript": "^5.3.3"
  }
}
```

Frontend package.json

json

```
{
  "name": "route-selection-frontend",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext ts,tsx"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.20.1",
    "react-hook-form": "^7.49.2",
    "react-hot-toast": "^2.4.1",
    "@tanstack/react-table": "^8.10.7",
    "date-fns": "^3.0.6",
    "lucide-react": "^0.294.0",
    "papaparse": "^5.4.1",
    "xlsx": "^0.18.5",
    "zod": "^3.22.4"
  },
  "devDependencies": {
    "@types/react": "^18.2.45",
    "@types/react-dom": "^18.2.18",
    "@types/papaparse": "^5.3.14",
    "@vitejs/plugin-react": "^4.2.1",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.32",
    "tailwindcss": "^3.3.6",
    "typescript": "^5.3.3",
    "vite": "^5.0.8"
  }
}
```

Seed Data Script

typescript

```typescript
// prisma/seed.ts

import { PrismaClient } from '@prisma/client';
import bcrypt from 'bcrypt';

const prisma = new PrismaClient();

async function main() {
  console.log('Seeding database...');

  // Create admin user
  const hashedPassword = await bcrypt.hash('admin123', 10);

  const adminUser = await prisma.user.create({
    data: {
      email: 'admin@company.com',
      password: hashedPassword,
      role: 'ADMIN'
    }
  });

  console.log('Created admin user:', adminUser.email);

  // Create sample employees
  const employees = await Promise.all([
    prisma.employee.create({
      data: {
        employeeNumber: 'EMP001',
        firstName: 'John',
        lastName: 'Smith',
        email: 'john.smith@company.com',
        phone: '555-0101',
        hireDate: new Date('2020-01-15'),
        cdlNumber: 'CDL123456',
        hasDoublesEndorsement: true,
        hasChainExperience: true,
        isActive: true,
        isEligible: true
      }
    }),
    prisma.employee.create({
      data: {
        employeeNumber: 'EMP002',
        firstName: 'Jane',
```

```
      lastName: 'Doe',
      email: 'jane.doe@company.com',
      phone: '555-0102',
      hireDate: new Date('2019-06-01'),
      cdlNumber: 'CDL789012',
      hasDoublesEndorsement: false,
      hasChainExperience: true,
      isActive: true,
      isEligible: true
    }
  }),
  prisma.employee.create({
    data: {
      employeeNumber: 'EMP003',
      firstName: 'Mike',
      lastName: 'Johnson',
      email: 'mike.johnson@company.com',
      phone: '555-0103',
      hireDate: new Date('2021-03-20'),
      cdlNumber: 'CDL345678',
      hasDoublesEndorsement: true,
      hasChainExperience: false,
      isActive: true,
      isEligible: true
    }
  })
]);

console.log(`Created ${employees.length} employees`);

// Create sample routes
const routes = await Promise.all([
  prisma.route.create({
    data: {
      runNumber: 1,
      runType: 'DOUBLES',
      origin: 'DEN',
      destination: 'WAM',
      days: 'M,T,W,TH,F',
      startTime: '9:15 PM',
      endTime: '8:30 AM',
      distance: 562,
      rateType: 'MILES',
      workTime: 11.25,
      requiresDoublesEndorsement: true,
      requiresChainExperience: false,
      isActive: true
```

```
    }
  }),
  prisma.route.create({
    data: {
      runNumber: 10,
      runType: 'SINGLES',
      origin: 'DEN',
      destination: 'NPL',
      days: 'M,T,W,TH,F',
      startTime: '8:15 PM',
      endTime: '6:45 AM',
      distance: 524,
      rateType: 'MILES',
      workTime: 11.25,
      requiresDoublesEndorsement: false,
      requiresChainExperience: false,
      isActive: true
    }
  }),
  prisma.route.create({
    data: {
      runNumber: 15,
      runType: 'SINGLES',
      origin: 'DEN',
      destination: 'OAK',
      days: 'M,T,W,TH,F',
      startTime: '7:15 AM',
      endTime: '5:00 PM',
      distance: 510,
      rateType: 'MILES',
      workTime: 11.25,
      requiresDoublesEndorsement: false,
      requiresChainExperience: false,
      isActive: true
    }
  })
]);

console.log(`Created ${routes.length} routes`);

// Create a selection period
const period = await prisma.selectionPeriod.create({
  data: {
    name: 'February 2025 Selection',
    openingDate: new Date('2025-02-01'),
    closingDate: new Date('2025-02-15'),
```

```
      notificationDate: new Date('2025-02-20'),
      status: 'DRAFT',
      isActive: false
    }
  });

  console.log('Created selection period:', period.name);

  console.log('Seeding completed!');
}

main()
  .catch((e) => {
    console.error(e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

README.md

markdown

Route Selection System

A web-based application for managing bi-annual driver route assignments based on seniority.

Features

- **Admin Portal**: Manage routes, employees, and selection periods
- **Driver Portal**: Browse routes and submit preferences
- **Automated Assignment**: Seniority-based algorithm with qualification checking
- **File Import/Export**: Excel and CSV support
- **Mobile Responsive**: Works on all devices

Prerequisites

- Node.js 18+
- PostgreSQL (or SQLite for development)
- npm or yarn

Installation

1. **Clone the repository**
```bash
git clone
cd route-selection-system
```

2. **Install backend dependencies**
```bash
cd backend
npm install
```

3. **Install frontend dependencies**
```bash
cd ../frontend
npm install
```

4. **Set up environment variables**
```bash
# In backend directory
cp .env.example .env
# Edit .env with your configuration
```

5. **Set up database**
```bash
cd backend
npx prisma generate
npx prisma migrate dev
npx prisma db seed
```

Running the Application

Development Mode

Start Backend:
```bash
cd backend
npm run dev
```

Start Frontend:
```bash
cd frontend
npm run dev
```

Access the application at `http://localhost:5173`

Production Build

Build Backend:
```bash
cd backend
npm run build
npm start
```

Build Frontend:
```bash
cd frontend
npm run build
npm run preview
```

Default Credentials

Admin:
- Email: admin@company.com

- Password: admin123

API Documentation

API documentation is available at `http://localhost:3000/api-docs` when running in development mode.

File Import Formats

Routes (Excel/CSV)
Required columns: Run Number, Run Type, Orig, Dest, Days, Start, End, Distance, Rate Type, Work Time

Employees (Excel/CSV)
Required columns: Employee ID, First Name, Last Name, Hire Date, CDL Number, Email

Testing
```bash
# Backend tests
cd backend
npm test

# Frontend tests
cd frontend
npm test
```

Deployment

See `DEPLOYMENT.md` for detailed deployment instructions.

Support

For issues or questions, contact: support@company.com

License

Key Implementation Notes

1. **Start with Backend First**: Set up the database schema and API endpoints before building the frontend
2. **Use the Seed Script**: Run the seed script to populate initial data for testing
3. **File Upload Size Limits**: Configure appropriate limits in Express for Excel file uploads (recommend 10MB)
4. **Authentication**: Implement JWT-based auth with refresh tokens for security
5. **Error Handling**: Use try-catch blocks and proper error middleware throughout
6. **Validation**: Use Zod schemas for request validation on both frontend and backend
7. **Testing Strategy**: Write unit tests for the assignment algorithm and integration tests for API endpoints
8. **Mobile First**: Design the driver portal mobile-first, then enhance for desktop
9. **Caching**: Consider Redis for session management and caching frequently accessed data
10. **Logging**: Implement comprehensive audit logging for all administrative actions

---

Development Checklist

- [ ] Set up project structure
- [ ] Configure database and Prisma
- [ ] Implement authentication system
- [ ] Create CRUD APIs for all entities
- [ ] Build assignment algorithm
- [ ] Implement file upload/processing
- [ ] Create admin dashboard
- [ ] Build driver mobile portal
- [ ] Add form validation
- [ ] Implement notifications
- [ ] Create reports and exports
- [ ] Write tests
- [ ] Add error handling
- [ ] Optimize performance
- [ ] Security audit
- [ ] Documentation
- [ ] Deployment preparation

This specification provides everything needed to build the Route Selection System. Import this document into Claude Code and begin development with the backend API layer, followed by the admin portal, and finally the driver mobile interface.