```
// models/fare.js
const mongoose = require('mongoose');

const fareSchema = new mongoose.Schema({
  basePrice: {
    type: Number,
    required: true,
    default: 2.0
  },
  perKilometerRate: {
    type: Number,
    required: true,
    default: 1.5
  },
  perMinuteRate: {
    type: Number,
    required: true,
    default: 0.25
  },
  surgeMultiplier: {
    type: Number,
    required: true,
    default: 1.0,
    min: 1.0,
    max: 5.0
  },
  currency: {
    type: String,
    default: 'USD'
  },
  location: {
    type: {
      city: String,
      state: String,
      country: String
    },
    required: true
  },
  serviceType: {
    type: String,
    enum: ['economy', 'premium', 'luxury'],
    default: 'economy'
  },
  isActive: {
    type: Boolean,
    default: true
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
});

const Fare = mongoose.model('Fare', fareSchema);

module.exports = Fare;

// services/fareCalculator.js
const geolib = require('geolib');

class FareCalculator {
  static async calculateFare(pickupCoords, dropoffCoords, duration, fareDetails,
trafficMultiplier = 1.0) {
    try {
      // Calculate distance in kilometers
      const distance = geolib.getDistance(
        { latitude: pickupCoords.latitude, longitude: pickupCoords.longitude },
        { latitude: dropoffCoords.latitude, longitude: dropoffCoords.longitude }
      ) / 1000; // Convert meters to kilometers

      // Apply fare formula
      const baseFare = fareDetails.basePrice;
      const distanceCost = distance * fareDetails.perKilometerRate;
      const timeCost = duration * fareDetails.perMinuteRate;

      // Apply surge pricing and traffic conditions
      const dynamicMultiplier = fareDetails.surgeMultiplier * trafficMultiplier;

      // Calculate total fare
      const totalFare = (baseFare + distanceCost + timeCost) * dynamicMultiplier;

      return {
        baseFare,
        distanceCost,
        timeCost,
        distance,
        duration,
        surgeMultiplier: fareDetails.surgeMultiplier,
        trafficMultiplier,
        totalFare: Math.round(totalFare * 100) / 100, // Round to 2 decimal places
        currency: fareDetails.currency
      };
    } catch (error) {
      console.error('Error calculating fare:', error);
      throw new Error('Failed to calculate fare');
    }
  }

  static async getSurgeMultiplier(location, time) {
    // Implementation of surge pricing algorithm based on:
    // 1. Current demand in the area
    // 2. Available drivers
    // 3. Time of day (peak hours)
    // 4. Special events

    // This is a simplified example
    const hour = time.getHours();

    // Peak hours: 7-9 AM and 5-7 PM
    const isPeakHour = (hour >= 7 && hour <= 9) || (hour >= 17 && hour <= 19);

    // Weekend
    const isWeekend = time.getDay() === 0 || time.getDay() === 6;

    // Example calculation (should be replaced with actual demand/supply algorithm)
    let surgeMultiplier = 1.0;

    if (isPeakHour) surgeMultiplier += 0.5;
    if (isWeekend && hour >= 20) surgeMultiplier += 0.3;

    return Math.min(surgeMultiplier, 3.0); // Cap at 3x
  }

  static async getTrafficMultiplier(pickupCoords, dropoffCoords, time) {
    // In a real implementation, this would call a traffic API service
    // For this example, we'll use a simplified model

    const hour = time.getHours();

    // Rush hours
    if ((hour >= 7 && hour <= 9) || (hour >= 16 && hour <= 19)) {
      return 1.2; // 20% increase during rush hour
    }

    return 1.0;
  }
}

module.exports = FareCalculator;

// routes/fares.js
const express = require('express');
const router = express.Router();
const Fare = require('../models/fare');
const FareCalculator = require('../services/fareCalculator');
const { authenticateToken } = require('../middleware/auth');

// Get fare rates for a specific location
router.get('/rates/:location', async (req, res) => {
  try {
    const location = req.params.location;
    const serviceType = req.query.service || 'economy';

    const fareDetails = await Fare.findOne({
      'location.city': location,
      'serviceType': serviceType,
      'isActive': true
    });

    if (!fareDetails) {
      return res.status(404).json({
        status: 'error',
        message: 'Fare details not found for this location and service type'
      });
    }

    res.json({
      status: 'success',
      data: fareDetails
    });
  } catch (error) {
    res.status(500).json({
      status: 'error',
      message: error.message
    });
  }
});

// Calculate fare estimate
router.post('/estimate', authenticateToken, async (req, res) => {
  try {
    const {
      pickupCoordinates,
      dropoffCoordinates,
      estimatedDuration,
      serviceType,
      city
    } = req.body;

    // Validate request body
    if (!pickupCoordinates || !dropoffCoordinates || !estimatedDuration || !city) {
      return res.status(400).json({
```

```
    status: 'error',
    message: 'Missing required parameters'
  });
}

// Get fare details for the city
const fareDetails = await Fare.findOne({
  'location.city': city,
  'serviceType': serviceType || 'economy',
  'isActive': true
});

if (!fareDetails) {
  return res.status(404).json({
    status: 'error',
    message: 'Fare details not found for this location'
  });
}

// Get surge multiplier based on current demand
const currentTime = new Date();
const surgeMultiplier = await FareCalculator.getSurgeMultiplier(
  { city, coordinates: pickupCoordinates },
  currentTime
);

// Update the fare details with the current surge multiplier
fareDetails.surgeMultiplier = surgeMultiplier;

// Get traffic multiplier
const trafficMultiplier = await FareCalculator.getTrafficMultiplier(
  pickupCoordinates,
  dropoffCoordinates,
  currentTime
);

// Calculate fare estimate
const fareEstimate = await FareCalculator.calculateFare(
  pickupCoordinates,
  dropoffCoordinates,
  estimatedDuration,
  fareDetails,
  trafficMultiplier
);

res.json({
  status: 'success',
  data: fareEstimate
```

```
    });
  } catch (error) {
    res.status(500).json({
      status: 'error',
      message: error.message
    });
  }
});

// Create or update fare settings (admin only)
router.post('/settings', authenticateToken, async (req, res) => {
  try {
    // Check if user is admin
    if (req.user.role !== 'admin') {
      return res.status(403).json({
        status: 'error',
        message: 'Unauthorized access'
      });
    }

    const {
      basePrice,
      perKilometerRate,
      perMinuteRate,
      location,
      serviceType,
      currency
    } = req.body;

    // Find existing fare settings or create new one
    let fareSettings = await Fare.findOne({
      'location.city': location.city,
      'serviceType': serviceType
    });

    if (fareSettings) {
      // Update existing settings
      fareSettings.basePrice = basePrice || fareSettings.basePrice;
      fareSettings.perKilometerRate = perKilometerRate || fareSettings.perKilometerRate;
      fareSettings.perMinuteRate = perMinuteRate || fareSettings.perMinuteRate;
      fareSettings.currency = currency || fareSettings.currency;
      fareSettings.location = location || fareSettings.location;
      fareSettings.updatedAt = Date.now();

      await fareSettings.save();
    } else {
      // Create new settings
      fareSettings = new Fare({
        basePrice,
        perKilometerRate,
        perMinuteRate,
        location,
        serviceType,
        currency
      });

      await fareSettings.save();
    }

    res.json({
      status: 'success',
      message: 'Fare settings updated successfully',
      data: fareSettings
    });
  } catch (error) {
    res.status(500).json({
      status: 'error',
      message: error.message
    });
  }
});

module.exports = router;
```