Service and controller

package com.brokerportal.service;

import com.brokerportal.dto.QuoteDTO;

import com.brokerportal.entity.Client;

import com.brokerportal.entity.Location;

import com.brokerportal.entity.Quote;

import com.brokerportal.repository.ClientRepository;

import com.brokerportal.repository.LocationRepository;

import com.brokerportal.repository.QuoteRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class QuoteService {

@Autowired

private QuoteRepository quoteRepository;

@Autowired

private ClientRepository clientRepository;

@Autowired

private LocationRepository locationRepository;

public List<Quote> getAllQuotes() {

return quoteRepository.findAll();

}

public Optional<Quote> getQuoteById(Long quoteId) {

return quoteRepository.findById(quoteId);

}

public Quote createQuote(QuoteDTO quoteDTO) {

Client client = clientRepository.findById(quoteDTO.getClientId()).orElseThrow(() -> new RuntimeException("Client not found"));

Location location = locationRepository.findById(quoteDTO.getLocationId()).orElseThrow(() -> new RuntimeException("Location not found"));

Quote quote = new Quote();

quote.setClient(client);

quote.setLocation(location);

quote.setHomeType(quoteDTO.getHomeType());

quote.setFoundationType(quoteDTO.getFoundationType());

quote.setHasBasement(quoteDTO.isHasBasement());

quote.setNoOfStories(quoteDTO.getNoOfStories());

quote.setAgeOfBuilding(quoteDTO.getAgeOfBuilding());

quote.setFirstFloorHeight(quoteDTO.getFirstFloorHeight());

quote.setConstructionType(quoteDTO.getConstructionType());

return quoteRepository.save(quote);

}

public Quote updateQuote(Long quoteId, QuoteDTO quoteDTO) {

Quote quote = quoteRepository.findById(quoteId).orElseThrow(() -> new RuntimeException("Quote not found"));

quote.setHomeType(quoteDTO.getHomeType());

quote.setFoundationType(quoteDTO.getFoundationType());

quote.setHasBasement(quoteDTO.isHasBasement());

quote.setNoOfStories(quoteDTO.getNoOfStories());

quote.setAgeOfBuilding(quoteDTO.getAgeOfBuilding());

quote.setFirstFloorHeight(quoteDTO.getFirstFloorHeight());

quote.setConstructionType(quoteDTO.getConstructionType());

return quoteRepository.save(quote);

}

public void deleteQuote(Long quoteId) {

quoteRepository.deleteById(quoteId);

}

}

package com.brokerportal.controller;

import com.brokerportal.dto.QuoteDTO;

import com.brokerportal.entity.Quote;

import com.brokerportal.service.QuoteService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/quotes")

@CrossOrigin("\*")

public class QuoteController {

@Autowired

private QuoteService quoteService;

@GetMapping

public List<Quote> getAllQuotes() {

return quoteService.getAllQuotes();

}

@GetMapping("/{id}")

public ResponseEntity<Quote> getQuoteById(@PathVariable Long id) {

Optional<Quote> quote = quoteService.getQuoteById(id);

return quote.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public ResponseEntity<Quote> createQuote(@RequestBody QuoteDTO quoteDTO) {

Quote createdQuote = quoteService.createQuote(quoteDTO);

return ResponseEntity.ok(createdQuote);

}

@PutMapping("/{id}")

public ResponseEntity<Quote> updateQuote(@PathVariable Long id, @RequestBody QuoteDTO quoteDTO) {

Quote updatedQuote = quoteService.updateQuote(id, quoteDTO);

return ResponseEntity.ok(updatedQuote);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteQuote(@PathVariable Long id) {

quoteService.deleteQuote(id);

return ResponseEntity.noContent().build();

}

}

@Service

public class BrokerService {

@Autowired

private BrokerRepository brokerRepository;

public List<Broker> getAllBrokers() {

return brokerRepository.findAll();

}

public Optional<Broker> getBrokerById(Long brokerId) {

return brokerRepository.findById(brokerId);

}

public Broker saveBroker(Broker broker) {

return brokerRepository.save(broker);

}

}

@Service

public class ClientService {

@Autowired

private ClientRepository clientRepository;

public List<Client> getAllClients() {

return clientRepository.findAll();

}

public Optional<Client> getClientById(Long clientId) {

return clientRepository.findById(clientId);

}

public Client saveClient(Client client) {

return clientRepository.save(client);

}

}

@Service

public class LocationService {

@Autowired

private LocationRepository locationRepository;

public Location getLocationByZipCode(String zipCode) {

return locationRepository.findByZipCode(zipCode);

}

public Location fetchAndStoreLocationDetails(String zipCode) {

// Fetch hardcoded or API-based data (this is an example)

Location location = new Location();

location.setZipCode(zipCode);

location.setDistanceToRiver(2.5);

location.setDistanceToCoast(10.0);

location.setElevation(15.0);

return locationRepository.save(location);

}

}

@Service

public class QuoteService {

@Autowired

private QuoteRepository quoteRepository;

@Autowired

private ClientRepository clientRepository;

@Autowired

private LocationRepository locationRepository;

public List<Quote> getAllQuotes() {

return quoteRepository.findAll();

}

public Optional<Quote> getQuoteById(Long quoteId) {

return quoteRepository.findById(quoteId);

}

public List<Quote> getQuotesByClientId(Long clientId) {

return quoteRepository.findByClientClientId(clientId);

}

public Quote createQuote(QuoteDTO quoteDTO) {

Client client = clientRepository.findById(quoteDTO.getClientId()).orElseThrow(() -> new RuntimeException("Client not found"));

Location location = locationRepository.findById(quoteDTO.getLocationId()).orElseThrow(() -> new RuntimeException("Location not found"));

Quote quote = new Quote();

quote.setClient(client);

quote.setLocation(location);

quote.setHomeType(quoteDTO.getHomeType());

quote.setFoundationType(quoteDTO.getFoundationType());

quote.setHasBasement(quoteDTO.isHasBasement());

quote.setNoOfStories(quoteDTO.getNoOfStories());

quote.setAgeOfBuilding(quoteDTO.getAgeOfBuilding());

quote.setFirstFloorHeight(quoteDTO.getFirstFloorHeight());

quote.setConstructionType(quoteDTO.getConstructionType());

return quoteRepository.save(quote);

}

}

@Service

public class PremiumService {

@Autowired

private PremiumRepository premiumRepository;

@Autowired

private QuoteRepository quoteRepository;

public List<Premium> getAllPremiums() {

return premiumRepository.findAll();

}

public double calculatePremium(Quote quote) {

// Step 1: Base Rates

double buildingRate = 1.50;

double contentsRate = 1.20;

// Step 2: Apply Geographic Rating Factors

buildingRate \*= (1 + (quote.getDistanceToCoast() / 1000) \* 0.10);

buildingRate \*= (1 + (quote.getDistanceToRiver() / 1000) \* 0.05);

buildingRate \*= (1 + (quote.getElevationRelativeToRiver() / 10) \* 0.15);

contentsRate \*= (1 + (quote.getDistanceToCoast() / 1000) \* 0.10);

contentsRate \*= (1 + (quote.getDistanceToRiver() / 1000) \* 0.05);

contentsRate \*= (1 + (quote.getElevationRelativeToRiver() / 10) \* 0.15);

// Step 3: Apply Property Characteristics Rating Factors

double foundationFactor = quote.getFoundationType().equalsIgnoreCase("Crawlspace") ? 1.20 : 1.00;

double firstFloorHeightFactor = (quote.getFirstFloorHeight() >= 5.0) ? 1.05 : 1.10;

buildingRate \*= firstFloorHeightFactor \* foundationFactor;

contentsRate \*= firstFloorHeightFactor \* foundationFactor;

// Step 5: Calculate Initial Premium

double buildingPremium = buildingRate \* (quote.getBuildingCoverage() / 1000);

double contentsPremium = contentsRate \* (quote.getContentsCoverage() / 1000);

double initialPremium = buildingPremium + contentsPremium;

double finalPremium = initialPremium;

return finalPremium;

}

public Premium savePremium(Long quoteId) {

Quote quote = quoteRepository.findById(quoteId).orElseThrow(() -> new RuntimeException("Quote not found"));

double calculatedPremium = calculatePremium(quote);

Premium premium = new Premium();

premium.setQuote(quote);

premium.setCalculatedPremium(calculatedPremium);

return premiumRepository.save(premium);

}

}