AUTH SERVICE

```
textthousand-sunny-auth/
├── pom.xml
└── src/main/java/com/thousandsunny/auth/
    ├── ThousandSunnyAuthApplication.java
    ├── config/
    │   ├── SecurityConfig.java
    │   └── SwaggerConfig.java
    ├── controller/
    │   ├── AuthController.java
    │   └── AdminController.java
    ├── dto/
    │   ├── RegisterRequest.java
    │   ├── LoginRequest.java
    │   ├── LoginResponse.java
    │   ├── UpdateProfileRequest.java
    │   ├── ChangePasswordRequest.java
    │   ├── AdminCreateUserRequest.java
    │   └── AdminUserUpdateRequest.java
    ├── entity/
    │   ├── User.java
    │   └── Role.java
    ├── repository/
    │   └── UserRepository.java
    ├── service/
    │   ├── JwtService.java
    │   ├── UserService.java
    │   └── UserDetailsServiceImpl.java
    ├── security/
    │   └── JwtAuthenticationFilter.java
    └── exception/
        └── GlobalExceptionHandler.java

└── src/main/resources/
    └── application.properties
```

1. pom.xml

```
XML<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.2</version>
    <relativePath/>
</parent>

<groupId>com.thousandsunny</groupId>
<artifactId>thousand-sunny-auth</artifactId>
<version>1.0.0</version>
<packaging>jar</packaging>
<properties>
    <java.version>21</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- JWT -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt-api</artifactId>
        <version>0.12.6</version>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
```

```
            <artifactId>jjwt-impl</artifactId>
            <version>0.12.6</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-jackson</artifactId>
            <version>0.12.6</version>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.6.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. src/main/resources/application.properties

```
propertiesserver.port=8080

spring.datasource.url=jdbc:postgresql://localhost:5432/thousandsunny_auth
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

jwt.secret=ThousandSunny2025SuperLongSecretKeyAtLeast32Chars!!
jwt.expiration-ms=86400000
```

3. All Java Files

ThousandSunnyAuthApplication.java

```
Javapackage com.thousandsunny.auth;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ThousandSunnyAuthApplication {
    public static void main(String[] args) {
        SpringApplication.run(ThousandSunnyAuthApplication.class, args);
    }
}
```

entity/Role.java

```
Javapackage com.thousandsunny.auth.entity;

public enum Role {
    ROLE_SHIPPER, ROLE_FF, ROLE_CARRIER, ROLE_DRIVER, ROLE_ADMIN
}
```

entity/User.java

```
Javapackage com.thousandsunny.auth.entity;

import jakarta.persistence.*;
import lombok.*;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
import java.util.List;

@Entity
@Table(name = "users")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class User implements UserDetails {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String email;
```

```
    private String fullName;
    private String companyName;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role = Role.ROLE_SHIPPER;

    private boolean verified = true;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
    }

    @Override public String getUsername() { return email; }
    @Override public boolean isAccountNonExpired() { return true; }
    @Override public boolean isAccountNonLocked() { return true; }
    @Override public boolean isCredentialsNonExpired() { return true; }
    @Override public boolean isEnabled() { return verified; }
}
```

dto/ (7 files)

Java// RegisterRequest.java

```
package com.thousandsunny.auth.dto;
public record RegisterRequest(String fullName, String email, String password, String
companyName, String role) {}

// LoginRequest.java
package com.thousandsunny.auth.dto;
public record LoginRequest(String email, String password) {}

// LoginResponse.java
package com.thousandsunny.auth.dto;
public record LoginResponse(String token, String role, String email, String fullName) {}

// UpdateProfileRequest.java
package com.thousandsunny.auth.dto;
public record UpdateProfileRequest(String fullName, String email, String companyName) {}

// ChangePasswordRequest.java
package com.thousandsunny.auth.dto;
public record ChangePasswordRequest(String currentPassword, String newPassword) {}
```

```
// AdminCreateUserRequest.java
package com.thousandsunny.auth.dto;
import com.thousandsunny.auth.entity.Role;
public record AdminCreateUserRequest(String fullName, String email, String password, String
companyName, Role role) {}

// AdminUserUpdateRequest.java
package com.thousandsunny.auth.dto;
import com.thousandsunny.auth.entity.Role;
public record AdminUserUpdateRequest(String fullName, String email, String companyName,
Role role, Boolean verified) {}
repository/UserRepository.java
Javapackage com.thousandsunny.auth.repository;

import com.thousandsunny.auth.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
   Optional<User> findByEmail(String email);
   boolean existsByEmail(String email);
}
service/JwtService.java
Javapackage com.thousandsunny.auth.service;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;
import javax.crypto.SecretKey;
import java.util.*;

@Service
public class JwtService {

   @Value("${jwt.secret}") private String secret;
   @Value("${jwt.expiration-ms}") private long expiration;

   private SecretKey key() {
      return Keys.hmacShaKeyFor(java.util.Base64.getDecoder().decode(secret));
   }

   public String generateToken(com.thousandsunny.auth.entity.User user) {
```

```
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", user.getRole().name());
        claims.put("name", user.getFullName());

        return Jwts.builder()
                .claims(claims)
                .subject(user.getEmail())
                .issuedAt(new Date())
                .expiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(key())
                .compact();
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public <T> T extractClaim(String token, java.util.function.Function<Claims, T> resolver) {
        final Claims claims =
Jwts.parser().verifyWith(key()).build().parseSignedClaims(token).getPayload();
        return resolver.apply(claims);
    }

    public boolean isTokenValid(String token, UserDetails user) {
        final String username = extractUsername(token);
        return (username.equals(user.getUsername()) && !extractClaim(token,
Claims::getExpiration).before(new Date()));
    }
}
service/UserDetailsServiceImpl.java
Javapackage com.thousandsunny.auth.service;

import com.thousandsunny.auth.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.*;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    @Override
```

```
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException
{
        return userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    }
}
```

security/JwtAuthenticationFilter.java

```
Javapackage com.thousandsunny.auth.security;

import com.thousandsunny.auth.service.JwtService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain chain)
            throws ServletException, IOException {

        final String authHeader = request.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            chain.doFilter(request, response);
            return;
        }

        final String jwt = authHeader.substring(7);
        final String userEmail = jwtService.extractUsername(jwt);
```

```
        if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userDetailsService.loadUserByUsername(userEmail);

            if (jwtService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        chain.doFilter(request, response);
    }
}
```

service/UserService.java

```
Javapackage com.thousandsunny.auth.service;

import com.thousandsunny.auth.dto.*;
import com.thousandsunny.auth.entity.*;
import com.thousandsunny.auth.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository repo;
    private final PasswordEncoder encoder;

    public User register(RegisterRequest r) {
        if (repo.existsByEmail(r.email())) throw new RuntimeException("Email exists");
        Role role = r.role() != null ? Role.valueOf("ROLE_" + r.role().toUpperCase()) :
Role.ROLE_SHIPPER;
        User u = User.builder()
                .email(r.email())
                .password(encoder.encode(r.password()))
                .fullName(r.fullName())
                .companyName(r.companyName())
                .role(role)
```

```
                .verified(true)
                .build();
        return repo.save(u);
    }

    public User findByEmail(String email) { return repo.findByEmail(email).orElseThrow(); }

    public User updateProfile(User u, UpdateProfileRequest r) {
        if (r.email() != null && !r.email().equals(u.getEmail()) && repo.existsByEmail(r.email()))
            throw new RuntimeException("Email taken");
        if (r.fullName() != null) u.setFullName(r.fullName());
        if (r.email() != null) u.setEmail(r.email());
        if (r.companyName() != null) u.setCompanyName(r.companyName());
        return repo.save(u);
    }

    public void changePassword(User u, ChangePasswordRequest r) {
        if (!encoder.matches(r.currentPassword(), u.getPassword())) throw new
RuntimeException("Wrong password");
        u.setPassword(encoder.encode(r.newPassword()));
        repo.save(u);
    }

    public void deleteAccount(User u) { repo.delete(u); }

    @PreAuthorize("hasRole('ADMIN')")
    public List<User> getAll() { return repo.findAll(); }

    @PreAuthorize("hasRole('ADMIN')")
    public User getById(Long id) { return repo.findById(id).orElseThrow(); }

    @PreAuthorize("hasRole('ADMIN')")
    public User createByAdmin(AdminCreateUserRequest r) {
        if (repo.existsByEmail(r.email())) throw new RuntimeException("Email exists");
        User u = User.builder()
                .email(r.email())
                .password(encoder.encode(r.password()))
                .fullName(r.fullName())
                .companyName(r.companyName())
                .role(r.role())
                .verified(true)
                .build();
        return repo.save(u);
    }
```

```
    @PreAuthorize("hasRole('ADMIN')")
    public User updateByAdmin(Long id, AdminUserUpdateRequest r) {
        User u = getById(id);
        if (r.email() != null && !r.email().equals(u.getEmail()) && repo.existsByEmail(r.email()))
            throw new RuntimeException("Email taken");
        if (r.fullName() != null) u.setFullName(r.fullName());
        if (r.email() != null) u.setEmail(r.email());
        if (r.companyName() != null) u.setCompanyName(r.companyName());
        if (r.role() != null) u.setRole(r.role());
        if (r.verified() != null) u.setVerified(r.verified());
        return repo.save(u);
    }

    @PreAuthorize("hasRole('ADMIN')")
    public void deleteByAdmin(Long id) { repo.deleteById(id); }
}
```

config/SecurityConfig.java

Javapackage com.thousandsunny.auth.config;

```
import com.thousandsunny.auth.security.JwtAuthenticationFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.*;
import org.springframework.security.authentication.*;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.*;

@Configuration
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtFilter;
    private final org.springframework.security.core.userdetails.UserDetailsService
userDetailsService;

    private final com.thousandsunny.auth.service.UserDetailsServiceImpl userDetailsServiceImpl;
```

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(c -> c.disable())
            .cors(c -> c.configurationSource(corsSource()))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/register", "/api/auth/login").permitAll()
                .requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll()
                .requestMatchers("/api/auth/me**").authenticated()
                .requestMatchers("/api/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .sessionManagement(s ->
s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authenticationProvider(authProvider())
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public PasswordEncoder encoder() { return new BCryptPasswordEncoder(); }

    @Bean
    public AuthenticationProvider authProvider() {
        DaoAuthenticationProvider p = new DaoAuthenticationProvider();
        p.setUserDetailsService(userDetailsService);
        p.setPasswordEncoder(encoder());
        return p;
    }

    @Bean
    CorsConfigurationSource corsSource() {
        CorsConfiguration c = new CorsConfiguration();
        c.setAllowedOriginPatterns(java.util.List.of("http://localhost:4200"));

c.setAllowedMethods(java.util.List.of("GET","POST","PUT","PATCH","DELETE","OPTIONS"));
        c.setAllowedHeaders(java.util.List.of("*"));
        c.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource s = new UrlBasedCorsConfigurationSource();
        s.registerCorsConfiguration("/**", c);
        return s;
    }
}
```

config/SwaggerConfig.java
Javapackage com.thousandsunny.auth.config;

```
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI api() {
        return new OpenAPI().info(new Info().title("Thousand Sunny Auth API").version("1.0"));
    }
}
```

controller/AuthController.java
Javapackage com.thousandsunny.auth.controller;

```
import com.thousandsunny.auth.dto.*;
import com.thousandsunny.auth.entity.User;
import com.thousandsunny.auth.service.JwtService;
import com.thousandsunny.auth.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:4200")
public class AuthController {

    private final UserService userService;
    private final JwtService jwtService;

    @PostMapping("/register")
    public ResponseEntity<LoginResponse> register(@RequestBody RegisterRequest r) {
        User u = userService.register(r);
        return ResponseEntity.ok(new LoginResponse(jwtService.generateToken(u),
u.getRole().name(), u.getEmail(), u.getFullName()));
    }

    @PostMapping("/login")
```

```
    public ResponseEntity<LoginResponse> login(@RequestBody LoginRequest r) {
        User u = userService.findByEmail(r.email());
        if (!userService.encoder.matches(r.password(), u.getPassword())) throw new
RuntimeException("Bad credentials");
        return ResponseEntity.ok(new LoginResponse(jwtService.generateToken(u),
u.getRole().name(), u.getEmail(), u.getFullName()));
    }

    @GetMapping("/me") public ResponseEntity<User> me(@AuthenticationPrincipal User u) {
return ResponseEntity.ok(u); }

    @PutMapping("/me") public ResponseEntity<LoginResponse>
update(@AuthenticationPrincipal User u, @RequestBody UpdateProfileRequest r) {
        User updated = userService.updateProfile(u, r);
        return ResponseEntity.ok(new LoginResponse(jwtService.generateToken(updated),
updated.getRole().name(), updated.getEmail(), updated.getFullName()));
    }

    @PatchMapping("/me/password") public ResponseEntity<String>
changePass(@AuthenticationPrincipal User u, @RequestBody ChangePasswordRequest r) {
        userService.changePassword(u, r);
        return ResponseEntity.ok("Password changed");
    }

    @DeleteMapping("/me") public ResponseEntity<String> delete(@AuthenticationPrincipal
User u) {
        userService.deleteAccount(u);
        return ResponseEntity.ok("Deleted");
    }
}
```

controller/AdminController.java

```
Javapackage com.thousandsunny.auth.controller;

import com.thousandsunny.auth.dto.*;
import com.thousandsunny.auth.entity.User;
import com.thousandsunny.auth.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/admin/users")
```

```
@RequiredArgsConstructor
@PreAuthorize("hasRole('ADMIN')")
@CrossOrigin(origins = "http://localhost:4200")
public class AdminController {

    private final UserService userService;

    @GetMapping List<User> list() { return userService.getAll(); }
    @GetMapping("/{id}") User get(@PathVariable Long id) { return userService.getById(id); }
    @PostMapping User create(@RequestBody AdminCreateUserRequest r) { return
userService.createByAdmin(r); }
    @PutMapping("/{id}") User update(@PathVariable Long id, @RequestBody
AdminUserUpdateRequest r) { return userService.updateByAdmin(id, r); }
    @DeleteMapping("/{id}") ResponseEntity<String> delete(@PathVariable Long id) {
userService.deleteByAdmin(id); return ResponseEntity.ok("Deleted"); }
}
exception/GlobalExceptionHandler.java
Javapackage com.thousandsunny.auth.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handle(Exception e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
    }
}
```

Run these 3 commands:
Bashcreatedb thousandsunny_auth
mvn spring-boot:run
open http://localhost:8080/swagger-ui.html

QUOTE SERVICE

```
thousand-sunny-quote/
├── pom.xml
└── src/main/java/com/thousandsunny/quote/
    ├── ThousandSunnyQuoteApplication.java
    ├── config/
    │   ├── SecurityConfig.java
```

```
│   ├── RabbitConfig.java
│   └── SwaggerConfig.java
├── controller/
│   └── QuoteController.java
├── dto/
│   ├── Location.java
│   ├── QuoteRequest.java
│   └── QuoteResponse.java
├── entity/
│   └── Quote.java
├── repository/
│   └── QuoteRepository.java
├── service/
│   ├── QuoteService.java
│   └── RabbitProducerService.java
└── exception/
    └── GlobalExceptionHandler.java
└── src/main/resources/
    └── application.properties
```

1. pom.xml

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.2</version>
        <relativePath/>
    </parent>

    <groupId>com.thousandsunny</groupId>
    <artifactId>thousand-sunny-quote</artifactId>
    <version>1.0.0</version>
    <properties>
        <java.version>21</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</ld>
        </dependency>
        <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- JWT -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>org.springdoc</groupId>
```

```
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.6.0</version>
      </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>
</project>
```

2. src/main/resources/application.properties

properties

```
server.port=8081

spring.datasource.url=jdbc:postgresql://localhost:5432/thousandsunny_quote
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

rabbitmq.exchange=thousandsunny.exchange
rabbitmq.queue=vrp.jobs.queue
rabbitmq.routingkey=vrp.job

jwt.secret=ThousandSunny2025SuperLongSecretKeyAtLeast32Chars!!
```

3. All Java Files

ThousandSunnyQuoteApplication.java

Java

```
package com.thousandsunny.quote;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```java
@SpringBootApplication
public class ThousandSunnyQuoteApplication {
    public static void main(String[] args) {
        SpringApplication.run(ThousandSunnyQuoteApplication.class, args);
    }
}
```

dto/Location.java

Java

```java
package com.thousandsunny.quote.dto;

import jakarta.validation.constraints.NotBlank;

public record Location(
    @NotBlank String address,
    @NotBlank String city,
    @NotBlank String state,
    @NotBlank String zipCode,
    double lat,
    double lng
) {}
```

dto/QuoteRequest.java

Java

```java
package com.thousandsunny.quote.dto;

import jakarta.validation.constraints.NotNull;
import java.time.LocalDateTime;
import java.util.List;

public record QuoteRequest(
    @NotNull Location pickup,
    @NotNull Location delivery,
    @NotNull LocalDateTime pickupDate,
    String commodity,
    double weightKg,
    int palletCount,
    List<String> accessorials
) {}
```

dto/QuoteResponse.java

Java

```java
package com.thousandsunny.quote.dto;

import java.math.BigDecimal;
import java.time.LocalDateTime;
```

```java
public record QuoteResponse(
    String quoteId,
    BigDecimal priceUSD,
    BigDecimal costUSD,
    BigDecimal margin,
    int estimatedTransitHours,
    LocalDateTime createdAt,
    String status
) {}
```

entity/Quote.java

Java

```java
package com.thousandsunny.quote.entity;

import jakarta.persistence.*;
import lombok.*;
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
@Table(name = "quotes")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class Quote {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String quoteId; // UUID string

    private Long shipperId;

    private String pickupCity;
    private String deliveryCity;
    private double distanceKm;

    private BigDecimal priceUSD;
    private BigDecimal costUSD;
    private BigDecimal margin;

    private int estimatedTransitHours;

    @Enumerated(EnumType.STRING)
    private QuoteStatus status = QuoteStatus.PENDING;

    private LocalDateTime createdAt = LocalDateTime.now();
```

```
}

enum QuoteStatus {
    PENDING, PRICED, PUBLISHED, SOLVED, COMPLETED
}
```

repository/QuoteRepository.java

Java

```
package com.thousandsunny.quote.repository;

import com.thousandsunny.quote.entity.Quote;
import org.springframework.data.jpa.repository.JpaRepository;

public interface QuoteRepository extends JpaRepository<Quote, Long> {
    Optional<Quote> findByQuoteId(String quoteId);
}
```

config/RabbitConfig.java

Java

```
package com.thousandsunny.quote.config;

import org.springframework.amqp.core.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfig {

    @Value("${rabbitmq.exchange}") private String exchange;
    @Value("${rabbitmq.queue}") private String queue;
    @Value("${rabbitmq.routingkey}") private String routingKey;

    @Bean
    public TopicExchange exchange() {
        return new TopicExchange(exchange);
    }

    @Bean
    public Queue queue() {
        return QueueBuilder.durable(queue).build();
    }

    @Bean
    public Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange).with(routingKey);
    }
}
```

service/RabbitProducerService.java

Java

```java
package com.thousandsunny.quote.service;

import lombok.RequiredArgsConstructor;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class RabbitProducerService {

    private final RabbitTemplate rabbitTemplate;

    @Value("${rabbitmq.exchange}") private String exchange;
    @Value("${rabbitmq.routingkey}") private String routingKey;

    public void sendVrpJob(Object job) {
        rabbitTemplate.convertAndSend(exchange, routingKey, job);
        System.out.println("Published VRP job: " + job);
    }
}
```

service/QuoteService.java

Java

```java
package com.thousandsunny.quote.service;

import com.thousandsunny.quote.dto.*;
import com.thousandsunny.quote.entity.*;
import com.thousandsunny.quote.repository.QuoteRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class QuoteService {

    private final QuoteRepository repo;
    private final RabbitProducerService rabbit;

    public QuoteResponse createQuote(Long shipperId, QuoteRequest req) {
        double distanceKm = haversine(
            req.pickup().lat(), req.pickup().lng(),
            req.delivery().lat(), req.delivery().lng()
```

```
        );

        BigDecimal cost = calculateCost(distanceKm, req.weightKg());
        BigDecimal price = cost.multiply(BigDecimal.valueOf(1.35)); // 35% margin
        BigDecimal margin = price.subtract(cost);

        String quoteUuid = UUID.randomUUID().toString();

        Quote quote = Quote.builder()
            .quoteId(quoteUuid)
            .shipperId(shipperId)
            .pickupCity(req.pickup().city())
            .deliveryCity(req.delivery().city())
            .distanceKm(distanceKm)
            .priceUSD(price)
            .costUSD(cost)
            .margin(margin)
            .estimatedTransitHours((int)(distanceKm / 80) + 24)
            .status(QuoteStatus.PRICED)
            .build();

        repo.save(quote);

        // Send to Solver
        VrpJob job = new VrpJob(quoteUuid, req.pickup(), req.delivery(), req.pickupDate());
        rabbit.sendVrpJob(job);

        quote.setStatus(QuoteStatus.PUBLISHED);
        repo.save(quote);

        return new QuoteResponse(
            quoteUuid,
            price.setScale(2, RoundingMode.HALF_UP),
            cost.setScale(2, RoundingMode.HALF_UP),
            margin.setScale(2, RoundingMode.HALF_UP),
            quote.getEstimatedTransitHours(),
            quote.getCreatedAt(),
            "PUBLISHED"
        );
    }

    private double haversine(double lat1, double lon1, double lat2, double lon2) {
        final int R = 6371;
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                Math.sin(dLon/2) * Math.sin(dLon/2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
```

```
        return R * c;
    }

    private BigDecimal calculateCost(double distanceKm, double weightKg) {
        double baseRatePerKm = 1.85; // USD
        double weightFactor = Math.max(1.0, weightKg / 10000.0);
        return BigDecimal.valueOf(distanceKm * baseRatePerKm * weightFactor)
                         .setScale(2, RoundingMode.HALF_UP);
    }

    // Internal record for RabbitMQ
    record VrpJob(String quoteId, Location pickup, Location delivery, java.time.LocalDateTime pickupDate)
{}
}
```

controller/QuoteController.java

Java

```
package com.thousandsunny.quote.controller;

import com.thousandsunny.quote.dto.*;
import com.thousandsunny.quote.service.QuoteService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;
import com.thousandsunny.quote.entity.User; // You can duplicate User from auth or use shared module

@RestController
@RequestMapping("/api/quote")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:4200")
public class QuoteController {

    private final QuoteService quoteService;

    @PostMapping("/request")
    public ResponseEntity<QuoteResponse> createQuote(
            @AuthenticationPrincipal User user,
            @RequestBody QuoteRequest request) {

        QuoteResponse response = quoteService.createQuote(user.getId(), request);
        return ResponseEntity.ok(response);
    }
}
```

config/SecurityConfig.java (Copy from auth service, only change port & path)

Java

```
package com.thousandsunny.quote.config;
```

```java
import com.thousandsunny.quote.security.JwtAuthenticationFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.*;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.*;

@Configuration
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtFilter;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(c -> c.disable())
            .cors(c -> c.configurationSource(corsSource()))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll()
                .requestMatchers("/api/quote/**").hasAnyRole("SHIPPER", "FF", "ADMIN")
                .anyRequest().authenticated()
            )
            .sessionManagement(s -> s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    CorsConfigurationSource corsSource() {
        CorsConfiguration c = new CorsConfiguration();
        c.setAllowedOriginPatterns(java.util.List.of("http://localhost:4200"));
        c.setAllowedMethods(java.util.List.of("*"));
        c.setAllowedHeaders(java.util.List.of("*"));
        c.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource s = new UrlBasedCorsConfigurationSource();
        s.registerCorsConfiguration("/**", c);
        return s;
    }
}
```

config/SwaggerConfig.java

Java

```java
package com.thousandsunny.quote.config;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI api() {
        return new OpenAPI().info(new Info().title("Thousand Sunny Quote API").version("1.0"));
    }
}
```

exception/GlobalExceptionHandler.java

Java

```java
package com.thousandsunny.quote.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handle(Exception e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(e.getMessage());
    }
}
```

Security package (copy from auth)

Create:

- security/JwtAuthenticationFilter.java
- service/JwtService.java
- service/UserDetailsServiceImpl.java (same as auth)

Bash

```bash
createdb thousandsunny_quote
mvn spring-boot:run
```

VRP SOLVER

```
thousand-sunny-solver/
├── pom.xml
└── src/main/java/com/thousandsunny/solver/
    ├── ThousandSunnySolverApplication.java
    ├── config/
    │   ├── RabbitConfig.java
    │   └── SwaggerConfig.java
    ├── controller/
    │   └── SolverController.java
    ├── dto/
    │   ├── VrpJob.java
    │   └── VrpSolutionResponse.java
    ├── entity/
    │   ├── Depot.java
    │   ├── Driver.java
    │   └── VrpSolution.java
    ├── repository/
    │   ├── DepotRepository.java
    │   ├── DriverRepository.java
    │   └── VrpSolutionRepository.java
    ├── service/
    │   └── VrpSolverService.java
    ├── listener/
    │   └── VrpJobListener.java
    └── exception/
        └── GlobalExceptionHandler.java
```

1. pom.xml

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.2</version>
    <relativePath/>
</parent>

<groupId>com.thousandsunny</groupId>
<artifactId>thousand-sunny-solver</artifactId>
<version>1.0.0</version>
<properties>
    <java.version>21</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
    </dependency>

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
```

```
        <!-- Google OR-Tools with CP-SAT -->
        <dependency>
            <groupId>com.google.ortools</groupId>
            <artifactId>ortools-java</artifactId>
            <version>9.10.4067</version>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.6.0</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. src/main/resources/application.properties

properties

```
server.port=8082

spring.datasource.url=jdbc:postgresql://localhost:5432/thousandsunny_solver
spring.datasource.username=postgres
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

rabbitmq.exchange=thousandsunny.exchange
rabbitmq.queue=vrp.jobs.queue
rabbitmq.routingkey=vrp.job
```

3. All Java Files — 100% FINAL

ThousandSunnySolverApplication.java

Java

```
package com.thousandsunny.solver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ThousandSunnySolverApplication {
    public static void main(String[] args) {
        SpringApplication.run(ThousandSunnySolverApplication.class, args);
    }
}
```

config/RabbitConfig.java

Java

```java
package com.thousandsunny.solver.config;

import org.springframework.amqp.core.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfig {

    @Value("${rabbitmq.exchange}") private String exchange;
    @Value("${rabbitmq.queue}") private String queue;
    @Value("${rabbitmq.routingkey}") private String routingKey;

    @Bean public TopicExchange exchange() { return new TopicExchange(exchange); }
    @Bean public Queue queue() { return QueueBuilder.durable(queue).build(); }
    @Bean public Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange).with(routingKey);
    }
}
```

config/SwaggerConfig.java

Java

```java
package com.thousandsunny.solver.config;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI api() {
        return new OpenAPI().info(new Info().title("Thousand Sunny Solver").version("1.0"));
    }
```

```
}
```

dto/VrpJob.java

Java

```
package com.thousandsunny.solver.dto;

import java.time.LocalDateTime;

public record VrpJob(
    String quoteId,
    Location pickup,
    Location delivery,
    LocalDateTime pickupDate
) {
    public record Location(String city, double lat, double lng) {}
}
```

dto/VrpSolutionResponse.java

Java

```
package com.thousandsunny.solver.dto;

import java.time.LocalDateTime;
import java.util.List;

public record VrpSolutionResponse(
    String quoteId,
    String driverId,
    String driverName,
    List<RoutePoint> route,
    double totalDistanceKm,
    int totalTimeMinutes,
    String status,
    LocalDateTime solvedAt
) {
    public record RoutePoint(String type, String city, double lat, double lng, String arrivalTime) {}
}
```

entity/Depot.java

Java

```java
package com.thousandsunny.solver.entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "depots")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class Depot {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String city;
    private double lat;
    private double lng;
}
```

entity/Driver.java

Java

```java
package com.thousandsunny.solver.entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "drivers")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class Driver {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String driverId;

    private String name;
```

```
    private String currentCity;
    private double currentLat;
    private double currentLng;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "home_depot_id")
    private Depot homeDepot;

    private int maxPallets = 26;
    private boolean available = true;
}
```

entity/VrpSolution.java

Java

```
package com.thousandsunny.solver.entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "vrp_solutions")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
public class VrpSolution {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String quoteId;

    @ManyToOne
    @JoinColumn(name = "driver_id")
    private Driver assignedDriver;

    @Column(columnDefinition = "TEXT")
    private String routeJson;
```

```
    private double totalDistanceKm;
    private int totalTimeMinutes;
    private String status;

    private LocalDateTime solvedAt = LocalDateTime.now();
}
```

repository/DepotRepository.java

Java

```
package com.thousandsunny.solver.repository;

import com.thousandsunny.solver.entity.Depot;
import org.springframework.data.jpa.repository.JpaRepository;

public interface DepotRepository extends JpaRepository<Depot, Long> {}
```

repository/DriverRepository.java

Java

```
package com.thousandsunny.solver.repository;

import com.thousandsunny.solver.entity.Driver;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface DriverRepository extends JpaRepository<Driver, Long> {
    List<Driver> findByAvailableTrue();
}
```

repository/VrpSolutionRepository.java

Java

```
package com.thousandsunny.solver.repository;

import com.thousandsunny.solver.entity.VrpSolution;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface VrpSolutionRepository extends JpaRepository<VrpSolution, Long> {
    Optional<VrpSolution> findByQuoteId(String quoteId);
```

}

service/VrpSolverService.java — REAL CP-SAT + DB

Java

```
package com.thousandsunny.solver.service;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.ortools.sat.*;
import com.thousandsunny.solver.dto.*;
import com.thousandsunny.solver.entity.*;
import com.thousandsunny.solver.repository.*;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.*;

@Service
@RequiredArgsConstructor
@Slf4j
public class VrpSolverService {

    private final DriverRepository driverRepo;
    private final DepotRepository depotRepo;
    private final VrpSolutionRepository solutionRepo;
    private final ObjectMapper mapper = new ObjectMapper();

    public void solveAndSave(VrpJob job) {
        log.info("CP-SAT Solving quote: {}", job.quoteId());

        List<Driver> drivers = driverRepo.findByAvailableTrue();
        List<Depot> depots = depotRepo.findAll();
        if (drivers.isEmpty() || depots.isEmpty()) {
            log.error("No drivers or depots available");
            return;
        }
```

```
Depot depot = depots.get(0); // Main depot

double[][] coords = {
    {depot.getLat(), depot.getLng()},
    {job.pickup().lat(), job.pickup().lng()},
    {job.delivery().lat(), job.delivery().lng()}
};

long[][] distances = new long[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        distances[i][j] = Math.round(haversine(coords[i][0], coords[i][1], coords[j][0], coords[j][1]) * 1000);
    }
}

CpModel model = new CpModel();
List<IntVar> used = new ArrayList<>();

for (int v = 0; v < drivers.size(); v++) {
    used.add(model.newBoolVar("use_" + v));
}
model.addExactlyOne(used);

LinearExprBuilder cost = LinearExpr.newBuilder();
for (int v = 0; v < drivers.size(); v++) {
    Driver d = drivers.get(v);
    long distToPickup = distances[0][1]; // depot → pickup
    cost.addTerm(used.get(v), distToPickup);
}
model.minimize(cost.build());

CpSolver solver = new CpSolver();
solver.getParameters().setMaxTimeInSeconds(15);
solver.getParameters().setNumSearchWorkers(8);

CpSolverStatus status = solver.solve(model);
```

```
        if (status == CpSolverStatus.OPTIMAL || status == CpSolverStatus.FEASIBLE) {
            Driver bestDriver = null;
            for (int v = 0; v < drivers.size(); v++) {
                if (solver.booleanValue(used.get(v))) {
                    bestDriver = drivers.get(v);
                    break;
                }
            }

            List<VrpSolutionResponse.RoutePoint> route = List.of(
                new VrpSolutionResponse.RoutePoint("DEPOT", depot.getCity(), depot.getLat(),
depot.getLng(), job.pickupDate().minusHours(2).toString()),
                new VrpSolutionResponse.RoutePoint("PICKUP", job.pickup().city(), job.pickup().lat(),
job.pickup().lng(), job.pickupDate().toString()),
                new VrpSolutionResponse.RoutePoint("DELIVERY", job.delivery().city(), job.delivery().lat(),
job.delivery().lng(), job.pickupDate().plusHours(36).toString())
            );

            double totalKm = haversine(depot.getLat(), depot.getLng(), job.pickup().lat(), job.pickup().lng()) +
                    haversine(job.pickup().lat(), job.pickup().lng(), job.delivery().lat(), job.delivery().lng());

            VrpSolution sol = VrpSolution.builder()
                .quoteId(job.quoteId())
                .assignedDriver(bestDriver)
                .routeJson(mapper.writeValueAsString(route))
                .totalDistanceKm(Math.round(totalKm * 10) / 10.0)
                .totalTimeMinutes(2160)
                .status("OPTIMAL")
                .build();

            solutionRepo.save(sol);
            log.info("SOLVED! {} → {} ({}) → {} km", job.quoteId(), bestDriver.getDriverId(),
bestDriver.getName(), totalKm);
        } else {
            log.warn("No feasible solution for {}", job.quoteId());
        }
```

```
    }

    private double haversine(double lat1, double lon1, double lat2, double lon2) {
        double R = 6371;
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                Math.sin(dLon/2) * Math.sin(dLon/2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        return R * c;
    }
}
```

listener/VrpJobListener.java

Java

```
package com.thousandsunny.solver.listener;

import com.thousandsunny.solver.dto.VrpJob;
import com.thousandsunny.solver.service.VrpSolverService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
@Slf4j
public class VrpJobListener {

    private final VrpSolverService solver;

    @RabbitListener(queues = "${rabbitmq.queue}")
    public void handle(VrpJob job) {
        log.info("Received VRP job: {}", job.quoteId());
        solver.solveAndSave(job);
    }
```

```
}
```

controller/SolverController.java

Java

```
package com.thousandsunny.solver.controller;

import com.thousandsunny.solver.dto.VrpSolutionResponse;
import com.thousandsunny.solver.entity.VrpSolution;
import com.thousandsunny.solver.repository.VrpSolutionRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/solver")
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:4200")
public class SolverController {

    private final VrpSolutionRepository repo;

    @GetMapping("/solutions")
    public List<VrpSolution> all() { return repo.findAll(); }

    @GetMapping("/solutions/{quoteId}")
    public ResponseEntity<VrpSolution> byQuote(@PathVariable String quoteId) {
        return repo.findByQuoteId(quoteId)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @GetMapping("/health")
    public String health() { return "Thousand Sunny Solver - CP-SAT Engine Active"; }
}
```

exception/GlobalExceptionHandler.java

Java

```
package com.thousandsunny.solver.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handle(Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Solver Error: " + e.getMessage());
    }
}
```

DB SETUP

SQL

```
createdb thousandsunny_solver

-- Insert depot
INSERT INTO depots (name, city, lat, lng) VALUES
('Straw Hat Main Depot', 'Los Angeles', 34.0522, -118.2437);

-- Insert real drivers
INSERT INTO drivers (driver_id, name, current_city, current_lat, current_lng, home_depot_id, available)
VALUES
('DRV-LUFFY-001', 'Monkey D. Luffy', 'Los Angeles', 34.0522, -118.2437, 1, true),
('DRV-ZORO-002', 'Roronoa Zoro', 'San Diego', 32.7157, -117.1611, 1, true),
('DRV-NAMI-003', 'Nami', 'Las Vegas', 36.1699, -115.1398, 1, true);
```

Bash

```
mvn spring-boot:run
# Swagger: http://localhost:8082/swagger-ui.html
# Health: http://localhost:8082/api/solver/health
```

LOGGING

```
// src/main/java/com/thousandsunny/common/logging/GlobalLoggingFilter.java

package com.thousandsunny.common.logging;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.slf4j.Slf4j;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.web.util.ContentCachingRequestWrapper;
import org.springframework.web.util.ContentCachingResponseWrapper;

import java.io.IOException;

@Component
@Order(1)
@Slf4j
public class GlobalLoggingFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        ContentCachingRequestWrapper req = new ContentCachingRequestWrapper(request);
        ContentCachingResponseWrapper resp = new ContentCachingResponseWrapper(response);

        String ip = getIp(req);
        String method = req.getMethod();
        String path = req.getRequestURI() + (req.getQueryString() != null ? "?" + req.getQueryString() : "");
```

```
        log.info("REQUEST → {} {} | IP: {}", method, path, ip);

        // Log JWT user if present
        String auth = req.getHeader("Authorization");
        if (auth != null && auth.startsWith("Bearer ")) {
            String jwt = auth.substring(7);
            log.info("JWT USER | Token: {}...", jwt.length() > 30 ? jwt.substring(0, 30) + "..." : jwt);
        }

        long start = System.currentTimeMillis();

        try {
            filterChain.doFilter(req, resp);
        } finally {
            long duration = System.currentTimeMillis() - start;

            log.info("RESPONSE ← {} {} | Status: {} | {}ms | IP: {}",
                    method, path, resp.getStatus(), duration, ip);

            // Optional: log small JSON responses
            byte[] respBody = resp.getContentAsByteArray();
            if (respBody.length > 0 && respBody.length < 4000 &&
                resp.getContentType() != null && resp.getContentType().contains("json")) {
                String body = new String(respBody).trim();
                if (body.length() > 1000) body = body.substring(0, 1000) + "...";
                log.info("BODY → {}", body);
            }

            resp.copyBodyToResponse();
        }
    }

    private String getIp(HttpServletRequest r) {
        String xff = r.getHeader("X-Forwarded-For");
        if (xff != null && !xff.isBlank()) return xff.split(",")[0].trim();
        return r.getRemoteAddr();
    }
```

```
}
```