```java
package com.customer.persistence.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import javax.persistence.*;
import javax.validation.constraints.*;
import java.time.LocalDate;
import java.util.List;
import java.util.Objects;

@Entity
@Table(name = "customers")
public class Customer {

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

    @NotEmpty(message = "Name is required")
    @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
    @Column(nullable = false, length = 100)
    private String name;

    @NotEmpty(message = "Contact person is required")
    @Size(min = 2, max = 100, message = "Contact person must be between 2 and 100 characters")
    @Column(name = "contact_person", nullable = false, length = 100)
    private String contactPerson;

    @Size(max = 255, message = "Address must not exceed 255 characters")
    @Column(length = 255)
    private String address;

    @Email(message = "Email should be valid")
    @Size(max = 100, message = "Email must not exceed 100 characters")
    @Column(length = 100)
    private String email;

    @Pattern(regexp = "^[+]?[0-9\\s\\-()]+$", message = "Phone number format is invalid")
    @Size(max = 20, message = "Phone number must not exceed 20 characters")
    @NotBlank(message = "Phone number is required")  // Changed from @NotEmpty to @NotBlank
    @Column(length = 20, nullable = false)  // Added nullable = false
    private String phone;

    @Enumerated(EnumType.STRING)
    @Column(name = "customer_type", nullable = false)
    @NotNull(message = "Customer type is required")
    private CustomerType customerType;

    @Size(max = 100, message = "Industry must not exceed 100 characters")
    @Column(length = 100)
    private String industry;

    @PastOrPresent(message = "Last contact date cannot be in the future")
    @Column(name = "last_contact_date")
    private LocalDate lastContactDate;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    @NotNull(message = "Status is required")
    private Status status;

    @ElementCollection
    @CollectionTable(name = "customer_contact_methods", joinColumns = @JoinColumn(name = "cust
omer_id"))
```

```
    @Column(name = "contact_method")
    @Size(max = 10, message = "Maximum 10 contact methods allowed")
    private List<@NotEmpty(message = "Contact method cannot be empty") String> wantsToBeContac
tedBy;

    public enum Status {
        LEAD, COLD, WARM, CUSTOMER, CLOSED
    }

    public enum CustomerType {
        LIMITED_LIABILITY_COMPANY, SOLE_PROPRIETORSHIP,
        CORPORATION, NON_PROFIT_ORGANIZATION
    }

    // Constructors
    public Customer() {}

    public Customer(long id, String name, String contactPerson, String address, String email,
String phone,
                    CustomerType customerType, String industry, LocalDate lastContactDate, Sta
tus status,
                    List<String> wantsToBeContactedBy) {
        this.id = id;
        this.name = name;
        this.contactPerson = contactPerson;
        this.address = address;
        this.email = email;
        this.phone = phone;
        this.customerType = customerType;
        this.industry = industry;
        this.lastContactDate = lastContactDate;
        this.status = status;
        this.wantsToBeContactedBy = wantsToBeContactedBy;
    }

    // Getters and Setters
    @JsonProperty
    public long getId() { return id; }

    @JsonProperty
    public void setId(long id) { this.id = id; }

    @JsonProperty
    public String getName() { return name; }

    @JsonProperty
    public void setName(String name) { this.name = name; }

    @JsonProperty
    public String getContactPerson() { return contactPerson; }

    @JsonProperty
    public void setContactPerson(String contactPerson) { this.contactPerson = contactPerson; }

    @JsonProperty
    public String getAddress() { return address; }

    @JsonProperty
    public void setAddress(String address) { this.address = address; }

    @JsonProperty
    public String getEmail() { return email; }
```

```
    @JsonProperty
    public void setEmail(String email) { this.email = email; }

    @JsonProperty
    public String getPhone() { return phone; }

    @JsonProperty
    public void setPhone(String phone) { this.phone = phone; }

    @JsonProperty
    public CustomerType getCustomerType() { return customerType; }

    @JsonProperty
    public void setCustomerType(CustomerType customerType) { this.customerType = customerType;
 }

    @JsonProperty
    public String getIndustry() { return industry; }

    @JsonProperty
    public void setIndustry(String industry) { this.industry = industry; }

    @JsonProperty
    public LocalDate getLastContactDate() { return lastContactDate; }

    @JsonProperty
    public void setLastContactDate(LocalDate lastContactDate) { this.lastContactDate = lastCon
tactDate; }

    @JsonProperty
    public Status getStatus() { return status; }

    @JsonProperty
    public void setStatus(Status status) { this.status = status; }

    @JsonProperty
    public List<String> getWantsToBeContactedBy() { return wantsToBeContactedBy; }

    @JsonProperty
    public void setWantsToBeContactedBy(List<String> wantsToBeContactedBy) { this.wantsToBeCon
tactedBy = wantsToBeContactedBy; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Customer customer = (Customer) o;
        return id == customer.id &&
                Objects.equals(name, customer.name) &&
                Objects.equals(contactPerson, customer.contactPerson);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, contactPerson);
    }
}
```

```java
package com.customer.persistence.repo;

import com.customer.persistence.model.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    List<Customer> findByName(String name);
    List<Customer> findByStatus(Customer.Status status);
    List<Customer> findByCustomerType(Customer.CustomerType customerType);
}
```

```java
package com.customer.dto;

import com.customer.persistence.model.Customer;
import com.fasterxml.jackson.annotation.JsonProperty;

import javax.validation.constraints.*;
import java.time.LocalDate;
import java.util.List;

public class CustomerRequest {

    @JsonProperty
    private Long id;

    @NotEmpty(message = "Name is required")
    @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
    @JsonProperty
    private String name;

    @NotEmpty(message = "Contact person is required")
    @Size(min = 2, max = 100, message = "Contact person must be between 2 and 100 characters")
    @JsonProperty
    private String contactPerson;

    @Size(max = 255, message = "Address must not exceed 255 characters")
    @JsonProperty
    private String address;

    @Email(message = "Email should be valid")
    @Size(max = 100, message = "Email must not exceed 100 characters")
    @JsonProperty
    private String email;

    @NotBlank(message = "Phone number is required")  // Changed from @Size to @NotBlank
    @Pattern(regexp = "^[+]?[0-9\\s\\-()]+$", message = "Phone number format is invalid")
    @Size(max = 20, message = "Phone number must not exceed 20 characters")
    @JsonProperty
    private String phone;

    @NotNull(message = "Customer type is required")
    @JsonProperty
    private Customer.CustomerType customerType;

    @Size(max = 100, message = "Industry must not exceed 100 characters")
    @JsonProperty
    private String industry;

    @PastOrPresent(message = "Last contact date cannot be in the future")
    @JsonProperty
    private LocalDate lastContactDate;

    @NotNull(message = "Status is required")
    @JsonProperty
    private Customer.Status status;

    @Size(max = 10, message = "Maximum 10 contact methods allowed")
    @JsonProperty
    private List<String> wantsToBeContactedBy;

    // Constructors
    public CustomerRequest() {}

    // Getters and Setters
```

```
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getContactPerson() { return contactPerson; }
    public void setContactPerson(String contactPerson) { this.contactPerson = contactPerson; }

    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }

    public Customer.CustomerType getCustomerType() { return customerType; }
    public void setCustomerType(Customer.CustomerType customerType) { this.customerType = cust
omerType; }

    public String getIndustry() { return industry; }
    public void setIndustry(String industry) { this.industry = industry; }

    public LocalDate getLastContactDate() { return lastContactDate; }
    public void setLastContactDate(LocalDate lastContactDate) { this.lastContactDate = lastCon
tactDate; }

    public Customer.Status getStatus() { return status; }
    public void setStatus(Customer.Status status) { this.status = status; }

    public List<String> getWantsToBeContactedBy() { return wantsToBeContactedBy; }
    public void setWantsToBeContactedBy(List<String> wantsToBeContactedBy) { this.wantsToBeCon
tactedBy = wantsToBeContactedBy; }
}
```

```java
package com.customer.dto;

import com.customer.persistence.model.Customer;
import com.fasterxml.jackson.annotation.JsonProperty;

import java.time.LocalDate;
import java.util.List;

public class CustomerResponse {

    @JsonProperty
    private long id;

    @JsonProperty
    private String name;

    @JsonProperty
    private String contactPerson;

    @JsonProperty
    private String address;

    @JsonProperty
    private String email;

    @JsonProperty
    private String phone;

    @JsonProperty
    private Customer.CustomerType customerType;

    @JsonProperty
    private String industry;

    @JsonProperty
    private LocalDate lastContactDate;

    @JsonProperty
    private Customer.Status status;

    @JsonProperty
    private List<String> wantsToBeContactedBy;

    // Constructors
    public CustomerResponse() {}

    public CustomerResponse(Customer customer) {
        this.id = customer.getId();
        this.name = customer.getName();
        this.contactPerson = customer.getContactPerson();
        this.address = customer.getAddress();
        this.email = customer.getEmail();
        this.phone = customer.getPhone();
        this.customerType = customer.getCustomerType();
        this.industry = customer.getIndustry();
        this.lastContactDate = customer.getLastContactDate();
        this.status = customer.getStatus();
        this.wantsToBeContactedBy = customer.getWantsToBeContactedBy();
    }

    // Getters and Setters
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }
```

```
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getContactPerson() { return contactPerson; }
    public void setContactPerson(String contactPerson) { this.contactPerson = contactPerson; }

    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }

    public Customer.CustomerType getCustomerType() { return customerType; }
    public void setCustomerType(Customer.CustomerType customerType) { this.customerType = cust
omerType; }

    public String getIndustry() { return industry; }
    public void setIndustry(String industry) { this.industry = industry; }

    public LocalDate getLastContactDate() { return lastContactDate; }
    public void setLastContactDate(LocalDate lastContactDate) { this.lastContactDate = lastCon
tactDate; }

    public Customer.Status getStatus() { return status; }
    public void setStatus(Customer.Status status) { this.status = status; }

    public List<String> getWantsToBeContactedBy() { return wantsToBeContactedBy; }
    public void setWantsToBeContactedBy(List<String> wantsToBeContactedBy) { this.wantsToBeCon
tactedBy = wantsToBeContactedBy; }
}
```

```java
package com.customer.dto.error;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

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

@JsonInclude(JsonInclude.Include.NON_NULL)
public class ProblemDetail {

    @JsonProperty
    private String type;

    @JsonProperty
    private String title;

    @JsonProperty
    private int status;

    @JsonProperty
    private String detail;

    @JsonProperty
    private String instance;

    @JsonProperty
    @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'")
    private LocalDateTime timestamp;

    @JsonProperty
    private List<ValidationError> validationErrors;

    @JsonProperty
    private Map<String, Object> extensions;

    public ProblemDetail() {
        this.timestamp = LocalDateTime.now();
    }

    public ProblemDetail(String type, String title, int status, String detail, String instance
) {
        this();
        this.type = type;
        this.title = title;
        this.status = status;
        this.detail = detail;
        this.instance = instance;
    }

    // Getters and Setters
    public String getType() { return type; }
    public void setType(String type) { this.type = type; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public int getStatus() { return status; }
    public void setStatus(int status) { this.status = status; }

    public String getDetail() { return detail; }
```

```
    public void setDetail(String detail) { this.detail = detail; }

    public String getInstance() { return instance; }
    public void setInstance(String instance) { this.instance = instance; }

    public LocalDateTime getTimestamp() { return timestamp; }
    public void setTimestamp(LocalDateTime timestamp) { this.timestamp = timestamp; }

    public List<ValidationError> getValidationErrors() { return validationErrors; }
    public void setValidationErrors(List<ValidationError> validationErrors) { this.validationE
rrors = validationErrors; }

    public Map<String, Object> getExtensions() { return extensions; }
    public void setExtensions(Map<String, Object> extensions) { this.extensions = extensions;
}
}
```

```java
package com.customer.dto.error;

import com.fasterxml.jackson.annotation.JsonProperty;

public class ValidationError {

    @JsonProperty
    private String field;

    @JsonProperty
    private Object rejectedValue;

    @JsonProperty
    private String message;

    public ValidationError() {}

    public ValidationError(String field, Object rejectedValue, String message) {
        this.field = field;
        this.rejectedValue = rejectedValue;
        this.message = message;
    }

    // Getters and Setters
    public String getField() { return field; }
    public void setField(String field) { this.field = field; }

    public Object getRejectedValue() { return rejectedValue; }
    public void setRejectedValue(Object rejectedValue) { this.rejectedValue = rejectedValue; }

    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }
}
```

```java
package com.customer.controller.advice;

import com.customer.dto.error.ProblemDetail;
import com.customer.dto.error.ValidationError;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.validation.BindException;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.method.annotation.MethodArgumentTypeMismatchException;

import javax.persistence.EntityNotFoundException;
import javax.validation.ConstraintViolation;
import javax.validation.ConstraintViolationException;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@ControllerAdvice
public class GlobalExceptionHandler {

    private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class)
;

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ProblemDetail> handleValidationException(MethodArgumentNotValidExcep
tion ex, WebRequest request) {
        logger.warn("Validation error occurred: {}", ex.getMessage());

        List<ValidationError> validationErrors = ex.getBindingResult().getFieldErrors().stream
()
                .map(error -> new ValidationError(
                        error.getField(),
                        error.getRejectedValue(),
                        error.getDefaultMessage()))
                .collect(Collectors.toList());

        // Add global errors
        ex.getBindingResult().getGlobalErrors().forEach(error ->
                validationErrors.add(new ValidationError(
                        error.getObjectName(),
                        null,
                        error.getDefaultMessage())));

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/validation-error",
                "Validation Failed",
                HttpStatus.BAD_REQUEST.value(),
                "Input validation failed for one or more fields",
                request.getDescription(false).replace("uri=", "")
        );
        problemDetail.setValidationErrors(validationErrors);

        return ResponseEntity.badRequest().body(problemDetail);
    }

    @ExceptionHandler(ConstraintViolationException.class)
```

```
    public ResponseEntity<ProblemDetail> handleConstraintViolationException(ConstraintViolatio
nException ex, WebRequest request) {
        logger.warn("Constraint violation occurred: {}", ex.getMessage());

        List<ValidationError> validationErrors = ex.getConstraintViolations().stream()
                .map(violation -> new ValidationError(
                        getFieldName(violation),
                        violation.getInvalidValue(),
                        violation.getMessage()))
                .collect(Collectors.toList());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/constraint-violation",
                "Constraint Violation",
                HttpStatus.BAD_REQUEST.value(),
                "One or more constraints were violated",
                request.getDescription(false).replace("uri=", "")
        );
        problemDetail.setValidationErrors(validationErrors);

        return ResponseEntity.badRequest().body(problemDetail);
    }

    @ExceptionHandler(BindException.class)
    public ResponseEntity<ProblemDetail> handleBindException(BindException ex, WebRequest requ
est) {
        logger.warn("Binding error occurred: {}", ex.getMessage());

        List<ValidationError> validationErrors = ex.getFieldErrors().stream()
                .map(error -> new ValidationError(
                        error.getField(),
                        error.getRejectedValue(),
                        error.getDefaultMessage()))
                .collect(Collectors.toList());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/binding-error",
                "Binding Error",
                HttpStatus.BAD_REQUEST.value(),
                "Data binding failed",
                request.getDescription(false).replace("uri=", "")
        );
        problemDetail.setValidationErrors(validationErrors);

        return ResponseEntity.badRequest().body(problemDetail);
    }

    @ExceptionHandler(HttpMessageNotReadableException.class)
    public ResponseEntity<ProblemDetail> handleHttpMessageNotReadableException(HttpMessageNotR
eadableException ex, WebRequest request) {
        logger.warn("Message not readable: {}", ex.getMessage());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/malformed-request",
                "Malformed JSON Request",
                HttpStatus.BAD_REQUEST.value(),
                "Request body could not be read or parsed",
                request.getDescription(false).replace("uri=", "")
        );

        return ResponseEntity.badRequest().body(problemDetail);
    }
```

```
    @ExceptionHandler(MethodArgumentTypeMismatchException.class)
    public ResponseEntity<ProblemDetail> handleMethodArgumentTypeMismatchException(MethodArgum
entTypeMismatchException ex, WebRequest request) {
        logger.warn("Method argument type mismatch: {}", ex.getMessage());

        List<ValidationError> validationErrors = new ArrayList<>();
        validationErrors.add(new ValidationError(
                ex.getName(),
                ex.getValue(),
                String.format("Parameter '%s' should be of type %s", ex.getName(), ex.getRequi
redType().getSimpleName())
        ));

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/type-mismatch",
                "Type Mismatch",
                HttpStatus.BAD_REQUEST.value(),
                "Method argument type mismatch",
                request.getDescription(false).replace("uri=", "")
        );
        problemDetail.setValidationErrors(validationErrors);

        return ResponseEntity.badRequest().body(problemDetail);
    }

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<ProblemDetail> handleEntityNotFoundException(EntityNotFoundException
 ex, WebRequest request) {
        logger.warn("Entity not found: {}", ex.getMessage());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/entity-not-found",
                "Entity Not Found",
                HttpStatus.NOT_FOUND.value(),
                ex.getMessage(),
                request.getDescription(false).replace("uri=", "")
        );

        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(problemDetail);
    }

    @ExceptionHandler(DataIntegrityViolationException.class)
    public ResponseEntity<ProblemDetail> handleDataIntegrityViolationException(DataIntegrityVi
olationException ex, WebRequest request) {
        logger.error("Data integrity violation: {}", ex.getMessage());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/data-integrity-violation",
                "Data Integrity Violation",
                HttpStatus.CONFLICT.value(),
                "A database constraint was violated",
                request.getDescription(false).replace("uri=", "")
        );

        return ResponseEntity.status(HttpStatus.CONFLICT).body(problemDetail);
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<ProblemDetail> handleIllegalArgumentException(IllegalArgumentExcepti
on ex, WebRequest request) {
        logger.warn("Illegal argument: {}", ex.getMessage());

        ProblemDetail problemDetail = new ProblemDetail(
```

```
                "https://problems.customer-crm.com/illegal-argument",
                "Illegal Argument",
                HttpStatus.BAD_REQUEST.value(),
                ex.getMessage(),
                request.getDescription(false).replace("uri=", "")
        );

        return ResponseEntity.badRequest().body(problemDetail);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ProblemDetail> handleGenericException(Exception ex, WebRequest reque
st) {
        logger.error("Unexpected error occurred", ex);

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/internal-error",
                "Internal Server Error",
                HttpStatus.INTERNAL_SERVER_ERROR.value(),
                "An unexpected error occurred",
                request.getDescription(false).replace("uri=", "")
        );

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(problemDetail);
    }

    private String getFieldName(ConstraintViolation<?> violation) {
        String propertyPath = violation.getPropertyPath().toString();
        return propertyPath.contains(".") ?
               propertyPath.substring(propertyPath.lastIndexOf('.') + 1) :
               propertyPath;
    }
}
```

```java
package com.customer.controller.advice;

import com.customer.dto.error.ValidationError;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.http.HttpStatus;
import org.springframework.ui.Model;
import org.springframework.validation.BindException;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.servlet.http.HttpServletRequest;
import javax.validation.ConstraintViolation;
import javax.validation.ConstraintViolationException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@ControllerAdvice(basePackages = "com.customer.controller")
public class WebExceptionHandler {

    private static final Logger logger = LoggerFactory.getLogger(WebExceptionHandler.class);

    @Value("${app.debug.enabled:false}")
    private boolean debugEnabled;

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ModelAndView handleValidationException(MethodArgumentNotValidException ex,
                                                  HttpServletRequest request) {
        logger.warn("Web validation error occurred: {}", ex.getMessage());

        List<ValidationError> validationErrors = ex.getBindingResult().getFieldErrors().stream
()
                .map(error -> new ValidationError(
                        error.getField(),
                        error.getRejectedValue(),
                        error.getDefaultMessage()))
                .collect(Collectors.toList());

        // Add global errors
        ex.getBindingResult().getGlobalErrors().forEach(error ->
                validationErrors.add(new ValidationError(
                        error.getObjectName(),
                        null,
                        error.getDefaultMessage())));

        ModelAndView mav = createErrorModelAndView("Validierungsfehler",
                "Die Eingabe war fÃ¼r ein oder mehrere Felder ungÃ¼ltig",
                HttpStatus.BAD_REQUEST, request);

        mav.addObject("validationErrors", validationErrors);

        if (debugEnabled) {
            addDebugInfo(mav, ex, request);
```

```
        }

        return mav;
    }

    @ExceptionHandler(ConstraintViolationException.class)
    public ModelAndView handleConstraintViolationException(ConstraintViolationException ex,
                                                           HttpServletRequest request) {
        logger.warn("Web constraint violation occurred: {}", ex.getMessage());

        List<ValidationError> validationErrors = ex.getConstraintViolations().stream()
                .map(violation -> new ValidationError(
                        getFieldName(violation),
                        violation.getInvalidValue(),
                        violation.getMessage()))
                .collect(Collectors.toList());

        ModelAndView mav = createErrorModelAndView("Verletzung von EinschrÃ¤nkungen",
                "Eine oder mehrere EinschrÃ¤nkungen wurden verletzt",
                HttpStatus.BAD_REQUEST, request);

        mav.addObject("validationErrors", validationErrors);

        if (debugEnabled) {
            addDebugInfo(mav, ex, request);
        }

        return mav;
    }

    @ExceptionHandler(DataIntegrityViolationException.class)
    public ModelAndView handleDataIntegrityViolationException(DataIntegrityViolationException
ex,
                                                              HttpServletRequest request) {
        logger.error("Web data integrity violation: {}", ex.getMessage());

        ModelAndView mav = createErrorModelAndView("Datenbank-IntegritÃ¤tsfehler",
                "Eine DatenbankeinschrÃ¤nkung wurde verletzt",
                HttpStatus.CONFLICT, request);

        if (debugEnabled) {
            addDebugInfo(mav, ex, request);
        }

        return mav;
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ModelAndView handleIllegalArgumentException(IllegalArgumentException ex,
                                                       HttpServletRequest request) {
        logger.warn("Web illegal argument: {}", ex.getMessage());

        ModelAndView mav = createErrorModelAndView("UngÃ¼ltige Anfrage",
                ex.getMessage(),
                HttpStatus.BAD_REQUEST, request);

        if (debugEnabled) {
            addDebugInfo(mav, ex, request);
        }

        return mav;
    }
```

```
    @ExceptionHandler(Exception.class)
    public ModelAndView handleGenericException(Exception ex, HttpServletRequest request) {
        logger.error("Unexpected web error occurred", ex);

        ModelAndView mav = createErrorModelAndView("Interner Serverfehler",
                "Es ist ein unerwarteter Fehler aufgetreten",
                HttpStatus.INTERNAL_SERVER_ERROR, request);

        if (debugEnabled) {
            addDebugInfo(mav, ex, request);
        }

        return mav;
    }

    private ModelAndView createErrorModelAndView(String title, String message,
                                                 HttpStatus status, HttpServletRequest request)
{
        ModelAndView mav = new ModelAndView("error/error");
        mav.addObject("errorTitle", title);
        mav.addObject("errorMessage", message);
        mav.addObject("errorStatus", status.value());
        mav.addObject("timestamp", LocalDateTime.now().format(DateTimeFormatter.ISO_LOCAL_DATE
_TIME));
        mav.addObject("path", request.getRequestURI());
        mav.addObject("debugEnabled", debugEnabled);
        return mav;
    }

    private void addDebugInfo(ModelAndView mav, Exception ex, HttpServletRequest request) {
        Map<String, Object> debugInfo = new HashMap<>();
        debugInfo.put("exceptionClass", ex.getClass().getSimpleName());
        debugInfo.put("exceptionMessage", ex.getMessage());
        debugInfo.put("requestMethod", request.getMethod());
        debugInfo.put("requestURL", request.getRequestURL().toString());
        debugInfo.put("userAgent", request.getHeader("User-Agent"));
        debugInfo.put("remoteAddress", request.getRemoteAddr());

        // Add stack trace (first 10 elements for brevity)
        StackTraceElement[] stackTrace = ex.getStackTrace();
        List<String> stackTraceLines = new ArrayList<>();
        for (int i = 0; i < Math.min(10, stackTrace.length); i++) {
            stackTraceLines.add(stackTrace[i].toString());
        }
        debugInfo.put("stackTrace", stackTraceLines);

        // Add request parameters
        Map<String, String[]> parameterMap = request.getParameterMap();
        if (!parameterMap.isEmpty()) {
            debugInfo.put("requestParameters", parameterMap);
        }

        mav.addObject("debugInfo", debugInfo);
    }

    private String getFieldName(ConstraintViolation<?> violation) {
        String propertyPath = violation.getPropertyPath().toString();
        return propertyPath.contains(".") ?
               propertyPath.substring(propertyPath.lastIndexOf('.') + 1) :
               propertyPath;
    }
}
```

```java
package com.customer.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class TestController {

    @GetMapping("/test-validation")
    public String testValidation() {
        return "test-validation";
    }

    @PostMapping("/test-force-error")
    public String forceError(@RequestParam(required = false) String errorType) {
        switch (errorType != null ? errorType : "generic") {
            case "illegal-argument":
                throw new IllegalArgumentException("Dies ist eine Test-IllegalArgumentExceptio
n zur Demonstration von Debug-Informationen");
            case "null-pointer":
                throw new NullPointerException("Dies ist eine Test-NullPointerException zur De
monstration von Debug-Informationen");
            case "validation":
                throw new IllegalArgumentException("Erzwungener Validierungsfehler zur Demonst
ration von Debug-Informationen");
            default:
                throw new RuntimeException("Dies ist eine generische Testausnahme zur Demonstr
ation von Debug-Informationen");
        }
    }
}
```

```java
package com.customer.controller;

import com.customer.persistence.repo.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class SimpleController {

    @Value("${spring.application.name}")
    String appName;


    @Autowired
    CustomerRepository customerRepo;

    @GetMapping("/")
    public String homePage(Model model) {
        model.addAttribute("appName", appName);
        model.addAttribute("customerCount", customerRepo.count());
        return "home";
    }
}
```

```java
package com.customer.controller;

import com.customer.dto.CustomerRequest;
import com.customer.dto.CustomerResponse;
import com.customer.service.CustomerService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.validation.Valid;
import java.util.List;
import java.util.Optional;

@Controller
@RequestMapping("/customers")
public class CustomerWebController {

    private static final Logger logger = LoggerFactory.getLogger(CustomerWebController.class);

    @Autowired
    private CustomerService customerService;

    @GetMapping
    public String listCustomers(Model model, @RequestParam(required = false) String message) {
        List<CustomerResponse> customers = customerService.getAllCustomers();
        model.addAttribute("customers", customers);
        if (message != null) {
            model.addAttribute("message", message);
        }
        return "customers/list";
    }

    @GetMapping("/{id}")
    public String viewCustomer(@PathVariable Long id, Model model) {
        logger.debug("Viewing customer with ID: {}", id);
        Optional<CustomerResponse> customer = customerService.getCustomerById(id);
        if (customer.isPresent()) {
            model.addAttribute("customer", customer.get());
            return "customers/detail";
        } else {
            return "redirect:/customers?message=Customer not found";
        }
    }

    @GetMapping("/new")
    public String newCustomerForm(Model model) {
        logger.debug("Showing new customer form");
        model.addAttribute("customer", new CustomerRequest());
        model.addAttribute("isEdit", false);
        return "customers/form";
    }

    @PostMapping
    public String createCustomer(@Valid @ModelAttribute("customer") CustomerRequest customer,
                                 BindingResult result,
                                 Model model,
                                 RedirectAttributes redirectAttributes) {
```

```
        if (result.hasErrors()) {
            model.addAttribute("isEdit", false);
            model.addAttribute("debugErrors", result.getAllErrors());
            return "customers/form";
        }

        CustomerResponse saved = customerService.createCustomer(customer);
        redirectAttributes.addFlashAttribute("message", "Kunde erfolgreich erstellt");
        return "redirect:/customers";
    }

    @GetMapping("/{id}/edit")
    public String editCustomerForm(@PathVariable Long id, Model model) {
        Optional<CustomerResponse> customer = customerService.getCustomerById(id);
        if (customer.isPresent()) {
            model.addAttribute("customer", convertToRequest(customer.get()));
            model.addAttribute("isEdit", true);
            return "customers/form";
        } else {
            return "redirect:/customers?message=Kunde nicht gefunden";
        }
    }

    @PostMapping("/{id}")
    public String updateCustomer(@PathVariable Long id, @Valid @ModelAttribute CustomerRequest
 customer,
                                 BindingResult result, Model model, RedirectAttributes redirect
Attributes) {

        logger.debug("=== UPDATE CUSTOMER REQUEST STARTED ===");
        logger.debug("Update customer ID: {}", id);
        logger.debug("Update customer data: name='{}', contactPerson='{}', email='{}', phone='
{}'",
                     customer.getName(), customer.getContactPerson(), customer.getEmail(), cust
omer.getPhone());

        if (result.hasErrors()) {
            logger.warn("=== UPDATE VALIDATION ERRORS FOUND ===");
            logger.warn("Total field errors: {}", result.getFieldErrorCount());

            // Log all field errors in detail
            for (FieldError error : result.getFieldErrors()) {
                logger.warn("UPDATE FIELD ERROR - Field: '{}', Rejected Value: '{}', Message:
'{}'",
                            error.getField(), error.getRejectedValue(), error.getDefaultMessage
());
            }

            customer.setId(id); // Ensure ID is set for edit form
            model.addAttribute("customer", customer);
            model.addAttribute("isEdit", true);
            model.addAttribute("debugErrors", result.getAllErrors());
            return "customers/form";
        }

        try {
            Optional<CustomerResponse> updatedCustomer = customerService.updateCustomer(id, cu
stomer);
            if (updatedCustomer.isPresent()) {
                logger.info("Customer updated successfully with ID: {}", id);
                redirectAttributes.addAttribute("message", "Kunde â\200\236" + updatedCustomer
.get().getName() + "â\200\234 wurde aktualisiert");
                return "redirect:/customers";
```

```
            } else {
                logger.warn("Customer with ID {} not found for update", id);
                return "redirect:/customers?message=Kunde nicht gefunden";
            }
        } catch (Exception e) {
            logger.error("Error updating customer with ID {}", id, e);
            customer.setId(id);
            model.addAttribute("customer", customer);
            model.addAttribute("isEdit", true);
            model.addAttribute("error", "Fehler beim Aktualisieren: " + e.getMessage());
            return "customers/form";
        } finally {
            logger.debug("=== UPDATE CUSTOMER REQUEST ENDED ===");
        }
    }

    @PostMapping("/{id}/delete")
    public String deleteCustomer(@PathVariable Long id, RedirectAttributes redirectAttributes)
 {
        try {
            Optional<CustomerResponse> customer = customerService.getCustomerById(id);
            if (customerService.deleteCustomer(id)) {
                redirectAttributes.addAttribute("message",
                    "Kunde" + (customer.isPresent() ? " â\200\236" + customer.get().getName()
+ "â\200\234" : "") + " erfolgreich gelÃ¶scht");
            } else {
                redirectAttributes.addAttribute("message", "Kunde nicht gefunden");
            }
        } catch (Exception e) {
            redirectAttributes.addAttribute("message", "Fehler beim LÃ¶schen: " + e.getMessage
());
        }
        return "redirect:/customers";
    }

    private CustomerRequest convertToRequest(CustomerResponse response) {
        CustomerRequest request = new CustomerRequest();
        request.setId(response.getId());
        request.setName(response.getName());
        request.setContactPerson(response.getContactPerson());
        request.setAddress(response.getAddress());
        request.setEmail(response.getEmail());
        request.setPhone(response.getPhone());
        request.setCustomerType(response.getCustomerType());
        request.setIndustry(response.getIndustry());
        request.setLastContactDate(response.getLastContactDate());
        request.setStatus(response.getStatus());
        request.setWantsToBeContactedBy(response.getWantsToBeContactedBy());
        return request;
    }
}
```

```java
package com.customer.controller;

import com.customer.dto.CustomerRequest;
import com.customer.dto.CustomerResponse;
import com.customer.service.CustomerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import javax.validation.Valid;
import javax.validation.constraints.NotNull;
import java.net.URI;
import java.util.List;

@RestController
@RequestMapping("/api/customers")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    @GetMapping
    public List<CustomerResponse> getAllCustomers() {
        return customerService.getAllCustomers();
    }

    @GetMapping("/{id}")
    public ResponseEntity<CustomerResponse> getCustomer(@PathVariable Long id) {
        return customerService.getCustomerById(id)
                .map(customer -> ResponseEntity.ok().body(customer))
                .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<CustomerResponse> createCustomer(@NotNull @Valid @RequestBody Custom
erRequest customerRequest) {
        CustomerResponse createdCustomer = customerService.createCustomer(customerRequest);

        URI location = ServletUriComponentsBuilder.fromCurrentRequest()
                .path("/{id}")
                .buildAndExpand(createdCustomer.getId())
                .toUri();

        return ResponseEntity.created(location).body(createdCustomer);
    }

    @PutMapping("/{id}")
    public ResponseEntity<CustomerResponse> updateCustomer(@PathVariable Long id,
                                                           @NotNull @Valid @RequestBody Custome
rRequest customerRequest) {
        return customerService.updateCustomer(id, customerRequest)
                .map(updatedCustomer -> ResponseEntity.ok(updatedCustomer))
                .orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteCustomer(@PathVariable Long id) {
        if (customerService.deleteCustomer(id)) {
            return ResponseEntity.noContent().build();
        } else {
            return ResponseEntity.notFound().build();
        }
```

```
    }
}
```

```java
package com.customer.service;

import com.customer.dto.CustomerRequest;
import com.customer.dto.CustomerResponse;
import com.customer.persistence.model.Customer;
import com.customer.persistence.repo.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.validation.annotation.Validated;

import javax.persistence.EntityNotFoundException;
import javax.validation.Valid;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@Validated
public class CustomerService {

    @Autowired
    private CustomerRepository customerRepository;

    public List<CustomerResponse> getAllCustomers() {
        return customerRepository.findAll().stream()
                .map(CustomerResponse::new)
                .collect(Collectors.toList());
    }

    public Optional<CustomerResponse> getCustomerById(Long id) {
        if (id == null || id <= 0) {
            throw new IllegalArgumentException("Customer ID must be a positive number");
        }
        return customerRepository.findById(id)
                .map(CustomerResponse::new);
    }

    public CustomerResponse createCustomer(@Valid CustomerRequest request) {
        if (request == null) {
            throw new IllegalArgumentException("Customer request cannot be null");
        }

        Customer customer = convertToEntity(request);

        if (customer.getStatus() == null) {
            customer.setStatus(Customer.Status.LEAD);
        }

        Customer savedCustomer = customerRepository.save(customer);
        return new CustomerResponse(savedCustomer);
    }

    public Optional<CustomerResponse> updateCustomer(Long id, @Valid CustomerRequest request)
{
        if (id == null || id <= 0) {
            throw new IllegalArgumentException("Customer ID must be a positive number");
        }
        if (request == null) {
            throw new IllegalArgumentException("Customer request cannot be null");
        }

        return customerRepository.findById(id)
                .map(existingCustomer -> {
```

```
                    Customer updatedCustomer = convertToEntity(request);
                    updatedCustomer.setId(id);
                    Customer savedCustomer = customerRepository.save(updatedCustomer);
                    return new CustomerResponse(savedCustomer);
                });
    }

    public boolean deleteCustomer(Long id) {
        if (id == null || id <= 0) {
            throw new IllegalArgumentException("Customer ID must be a positive number");
        }

        return customerRepository.findById(id)
                .map(customer -> {
                    customerRepository.delete(customer);
                    return true;
                })
                .orElse(false);
    }

    private Customer convertToEntity(CustomerRequest request) {
        Customer customer = new Customer();
        customer.setName(request.getName());
        customer.setContactPerson(request.getContactPerson());
        customer.setAddress(request.getAddress());
        customer.setEmail(request.getEmail());
        customer.setPhone(request.getPhone());
        customer.setCustomerType(request.getCustomerType());
        customer.setIndustry(request.getIndustry());
        customer.setLastContactDate(request.getLastContactDate());
        customer.setStatus(request.getStatus());
        customer.setWantsToBeContactedBy(request.getWantsToBeContactedBy());
        return customer;
    }
}
```

```java
package com.customer.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;
import org.springframework.validation.beanvalidation.MethodValidationPostProcessor;

@Configuration
public class ValidationConfiguration {

    @Bean
    public LocalValidatorFactoryBean validator() {
        return new LocalValidatorFactoryBean();
    }

    @Bean
    public MethodValidationPostProcessor methodValidationPostProcessor() {
        MethodValidationPostProcessor processor = new MethodValidationPostProcessor();
        processor.setValidator(validator());
        return processor;
    }
}
```

```java
package com.customer.config;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories("com.customer.persistence.repo")
@EntityScan("com.customer.persistence.model")
@SpringBootApplication(scanBasePackages = {"com.customer"})
public class Application {

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