```java
package exam;

import exam.db.CustomerDAO;
import exam.exceptions.*;
import exam.health.DatabaseHealthCheck;
import exam.resources.CustomerResource;
import exam.resources.CustomerViewResource;
import io.dropwizard.core.Application;
import io.dropwizard.jdbi3.JdbiFactory;
import io.dropwizard.core.setup.Bootstrap;
import io.dropwizard.core.setup.Environment;
import io.dropwizard.views.common.ViewBundle;
import org.jdbi.v3.core.Jdbi;

public class CRMApplication extends Application<CRMConfiguration> {

    public static void main(final String[] args) throws Exception {
        new CRMApplication().run(args);
    }

    @Override
    public String getName() {
        return "CRM";
    }

    @Override
    public void initialize(final Bootstrap<CRMConfiguration> bootstrap) {
        bootstrap.addBundle(new ViewBundle<>());
    }

    @Override
    public void run(final CRMConfiguration configuration,
                    final Environment environment) {
        final JdbiFactory factory = new JdbiFactory();
        final Jdbi jdbi = factory.build(environment, configuration.getDataSourceFactory(), "po
stgresql");

        final DatabaseHealthCheck databaseHealthCheck = new DatabaseHealthCheck(jdbi);
        environment.healthChecks().register("database", databaseHealthCheck);

        final CustomerDAO customerDAO = jdbi.onDemand(CustomerDAO.class);
        customerDAO.createTable();

        final CustomerResource customerResource = new CustomerResource(customerDAO);
        environment.jersey().register(customerResource);

        final CustomerViewResource customerViewResource = new CustomerViewResource(customerDAO
, environment.getValidator());
        environment.jersey().register(customerViewResource);

        // Register Exception Mappers
        environment.jersey().register(ValidationExceptionMapper.class);
        environment.jersey().register(JsonProcessingExceptionMapper.class);
        environment.jersey().register(IllegalArgumentExceptionMapper.class);
        environment.jersey().register(WebApplicationExceptionMapper.class);
        environment.jersey().register(RuntimeExceptionMapper.class);
        environment.jersey().register(new ProblemDetailHtmlMessageBodyWriter());
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import exam.api.error.ValidationError;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.MultivaluedMap;
import jakarta.ws.rs.ext.MessageBodyWriter;
import jakarta.ws.rs.ext.Provider;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.annotation.Annotation;
import java.lang.reflect.Type;
import java.nio.charset.StandardCharsets;
import java.time.format.DateTimeFormatter;
import java.util.Map;

@Provider
@Produces(MediaType.TEXT_HTML)
public class ProblemDetailHtmlMessageBodyWriter implements MessageBodyWriter<ProblemDetail> {

    private static final DateTimeFormatter DATE_TIME_FORMATTER = DateTimeFormatter.ofPattern("
yyyy-MM-dd HH:mm:ss");

    @Override
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations, Medi
aType mediaType) {
        return ProblemDetail.class.isAssignableFrom(type);
    }

    @Override
    public void writeTo(ProblemDetail problemDetail,
                        Class<?> type,
                        Type genericType,
                        Annotation[] annotations,
                        MediaType mediaType,
                        MultivaluedMap<String, Object> httpHeaders,
                        OutputStream entityStream) throws IOException {

        String html = buildHtml(problemDetail);
        entityStream.write(html.getBytes(StandardCharsets.UTF_8));
    }

    private String buildHtml(ProblemDetail problemDetail) {
        StringBuilder sb = new StringBuilder(2048);
        sb.append("<!DOCTYPE html>\n")
          .append("<html lang=\"de\">\n")
          .append("<head>\n")
          .append("    <meta charset=\"UTF-8\">\n")
          .append("    <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0
\">\n")
          .append("    <title>").append(escape(problemDetail.getTitle())).append("</title>\n")
          .append("    <script src=\"https://cdn.tailwindcss.com\"></script>\n")
          .append("</head>\n")
          .append("<body class=\"bg-gray-50 min-h-screen\">\n")
          .append("    <div class=\"max-w-4xl mx-auto px-6 py-10\">\n")
          .append("        <div class=\"bg-white border border-red-200 rounded-xl shadow-lg p-
8\">\n")
          .append("            <h1 class=\"text-2xl font-bold text-red-600 mb-4\">")
          .append(escape(problemDetail.getTitle()))
          .append("</h1>\n")
          .append("            <p class=\"text-gray-700 mb-2\"><strong>Status:</strong> ").app
```

```
end(problemDetail.getStatus()).append("</p>\n")
          .append("            <p class=\"text-gray-700 mb-2\"><strong>Detail:</strong> ")
          .append(escape(problemDetail.getDetail()))
          .append("</p>\n")
          .append("            <p class=\"text-gray-500 text-sm\"><strong>Instance:</strong> "
)
          .append(escape(problemDetail.getInstance()))
          .append("</p>\n");

        if (problemDetail.getTimestamp() != null) {
            sb.append("            <p class=\"text-gray-500 text-sm\"><strong>Zeitpunkt:</stro
ng> ")
              .append(DATE_TIME_FORMATTER.format(problemDetail.getTimestamp()))
              .append("</p>\n");
        }

        if (problemDetail.getValidationErrors() != null && !problemDetail.getValidationErrors(
).isEmpty()) {
            sb.append("            <div class=\"mt-6\">\n")
              .append("                <h2 class=\"text-lg font-semibold text-yellow-700 mb-2\
">Validierungsfehler</h2>\n")
              .append("                <ul class=\"list-disc list-inside space-y-2 text-gray-7
00\">\n");
            for (ValidationError error : problemDetail.getValidationErrors()) {
                sb.append("                    <li><strong>")
                  .append(escape(error.getField()))
                  .append(":</strong> ")
                  .append(escape(error.getMessage()))
                  .append(optionalRejectedValue(error.getRejectedValue()))
                  .append("</li>\n");
            }
            sb.append("                </ul>\n")
              .append("            </div>\n");
        }

        if (problemDetail.getExtensions() != null && !problemDetail.getExtensions().isEmpty())
 {
            sb.append("            <div class=\"mt-6\">\n")
              .append("                <h2 class=\"text-lg font-semibold text-gray-800 mb-2\">
ZusÃ¤tzliche Informationen</h2>\n")
              .append("                <div class=\"bg-gray-100 rounded-lg p-4 text-sm text-gr
ay-700 space-y-2\">\n");
            for (Map.Entry<String, Object> entry : problemDetail.getExtensions().entrySet()) {
                sb.append("                    <div><strong>")
                  .append(escape(entry.getKey()))
                  .append(":</strong> ")
                  .append(escape(String.valueOf(entry.getValue())))
                  .append("</div>\n");
            }
            sb.append("                </div>\n")
              .append("            </div>\n");
        }

        sb.append("            <div class=\"mt-6 flex space-x-3\">\n")
          .append("                <a href=\"/ui\" class=\"bg-blue-600 hover:bg-blue-700 text-
white px-4 py-2 rounded-lg\">Zur Startseite</a>\n")
          .append("                <a href=\"javascript:history.back()\" class=\"bg-gray-200 h
over:bg-gray-300 text-gray-800 px-4 py-2 rounded-lg\">ZurÃ¼ck</a>\n")
          .append("            </div>\n")
          .append("        </div>\n")
          .append("    </div>\n")
          .append("</body>\n")
          .append("</html>\n");
```

```
        return sb.toString();
    }

    private String optionalRejectedValue(Object value) {
        if (value == null) {
            return "";
        }
        return " <span class=\"text-gray-500\">(Wert: " + escape(String.valueOf(value)) + ")</
span>";
    }

    private String escape(String input) {
        if (input == null) {
            return "";
        }
        return input
                .replace("&", "&amp;")
                .replace("<", "&lt;")
                .replace(">", "&gt;")
                .replace("\"", "&quot;")
                .replace("'", "&#39;");
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import exam.api.error.ValidationError;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.validation.ConstraintViolation;
import jakarta.validation.ConstraintViolationException;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.ExceptionMapper;
import jakarta.ws.rs.ext.Provider;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;
import java.util.stream.Collectors;

@Provider
public class ValidationExceptionMapper implements ExceptionMapper<ConstraintViolationException
> {

    private static final Logger logger = LoggerFactory.getLogger(ValidationExceptionMapper.cla
ss);

        @Context
        private HttpServletRequest request;

        @Context
        private HttpHeaders httpHeaders;


    @Override
    public Response toResponse(ConstraintViolationException exception) {
        logger.warn("Validation error occurred: {}", exception.getMessage());

        List<ValidationError> validationErrors = exception.getConstraintViolations().stream()
                .map(this::createValidationError)
                .collect(Collectors.toList());

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/constraint-violation",
                "Constraint Violation",
                Response.Status.BAD_REQUEST.getStatusCode(),
                "One or more constraints were violated",
                request != null ? request.getRequestURI() : "unknown"
        );
        problemDetail.setValidationErrors(validationErrors);

        return ProblemDetailResponseBuilder.build(request, httpHeaders, Response.Status.BAD_RE
QUEST, problemDetail);
    }

    private ValidationError createValidationError(ConstraintViolation<?> violation) {
        String fieldName = getFieldName(violation);
        return new ValidationError(
                fieldName,
                violation.getInvalidValue(),
                violation.getMessage()
        );
    }

    private String getFieldName(ConstraintViolation<?> violation) {
```

```
        String propertyPath = violation.getPropertyPath().toString();
        return propertyPath.contains(".") ?
                propertyPath.substring(propertyPath.lastIndexOf('.') + 1) :
                propertyPath;
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.ExceptionMapper;
import jakarta.ws.rs.ext.Provider;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;

@Provider
public class RuntimeExceptionMapper implements ExceptionMapper<RuntimeException> {

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

    private static final int STACK_TRACE_PREVIEW_LIMIT = 10;

    @Context
    private HttpServletRequest request;

    @Context
    private HttpHeaders httpHeaders;

    @Override
    public Response toResponse(RuntimeException exception) {
        logger.error("Unexpected runtime error occurred", exception);

        String instance = request != null ? request.getRequestURI() : "unknown";
        String detail = exception.getMessage() != null ? exception.getMessage() : "An unexpect
ed error occurred";

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/internal-error",
                "Internal Server Error",
                Response.Status.INTERNAL_SERVER_ERROR.getStatusCode(),
                detail,
                instance
        );
        problemDetail.setExtensions(buildExtensions(exception));

        return ProblemDetailResponseBuilder.build(request, httpHeaders, Response.Status.INTERN
AL_SERVER_ERROR, problemDetail);
    }

    private Map<String, Object> buildExtensions(RuntimeException exception) {
        Map<String, Object> extensions = new LinkedHashMap<>();
        extensions.put("exceptionClass", exception.getClass().getName());
        extensions.put("rootCause", getRootCauseDescription(exception));
        extensions.put("stackTrace", formatStackTrace(exception));
        return extensions;
    }

    private String getRootCauseDescription(Throwable throwable) {
        Throwable root = throwable;
        while (root.getCause() != null && root.getCause() != root) {
```

```
            root = root.getCause();
        }
        String message = root.getMessage();
        return message != null ? root.getClass().getName() + ": " + message : root.getClass().
getName();
    }

    private String formatStackTrace(Throwable throwable) {
        return Arrays.stream(throwable.getStackTrace())
                .limit(STACK_TRACE_PREVIEW_LIMIT)
                .map(StackTraceElement::toString)
                .collect(Collectors.joining("\n"));
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;

import java.util.List;

public final class ProblemDetailResponseBuilder {

    private static final MediaType DEFAULT_MEDIA_TYPE = MediaType.APPLICATION_JSON_TYPE;

    private ProblemDetailResponseBuilder() {}

    public static Response build(HttpServletRequest request,
                                 HttpHeaders headers,
                                 Response.StatusType status,
                                 ProblemDetail problemDetail) {
        MediaType mediaType = negotiate(request, headers);
        return Response.status(status)
                .type(mediaType)
                .entity(problemDetail)
                .build();
    }

    private static MediaType negotiate(HttpServletRequest request, HttpHeaders headers) {
        if (isUiRequest(request)) {
            return MediaType.TEXT_HTML_TYPE;
        }

        if (headers != null) {
            List<MediaType> acceptableMediaTypes = headers.getAcceptableMediaTypes();
            for (MediaType mediaType : acceptableMediaTypes) {
                if (mediaType.isCompatible(MediaType.TEXT_HTML_TYPE) || containsHtml(mediaType
)) {
                    return MediaType.TEXT_HTML_TYPE;
                }
                if (mediaType.isCompatible(MediaType.APPLICATION_JSON_TYPE)) {
                    return MediaType.APPLICATION_JSON_TYPE;
                }
            }
        }

        return DEFAULT_MEDIA_TYPE;
    }

    private static boolean containsHtml(MediaType mediaType) {
        return mediaType.toString().toLowerCase().contains("html");
    }

    private static boolean isUiRequest(HttpServletRequest request) {
        if (request == null) {
            return false;
        }
        String uri = request.getRequestURI();
        return uri != null && uri.startsWith("/ui");
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.ws.rs.WebApplicationException;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.ExceptionMapper;
import jakarta.ws.rs.ext.Provider;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Provider
public class WebApplicationExceptionMapper implements ExceptionMapper<WebApplicationException>
 {

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

    @Context
    private HttpServletRequest request;

    @Context
    private HttpHeaders httpHeaders;

    @Override
    public Response toResponse(WebApplicationException exception) {
        Response response = exception.getResponse();
        int status = response.getStatus();

        if (status >= 500) {
            logger.error("Web application error occurred: {}", exception.getMessage(), excepti
on);
        } else if (status >= 400) {
            logger.warn("Client error occurred: {}", exception.getMessage());
        }

        String type = getTypeForStatus(status);
        String title = getTitleForStatus(status);
        String detail = exception.getMessage() != null ? exception.getMessage() : getDefaultDe
tailForStatus(status);

        ProblemDetail problemDetail = new ProblemDetail(
                type,
                title,
                status,
                detail,
        request != null ? request.getRequestURI() : "unknown"
        );

        return ProblemDetailResponseBuilder.build(request, httpHeaders, response.getStatusInfo
(), problemDetail);
    }

    private String getTypeForStatus(int status) {
        switch (status) {
            case 400: return "https://problems.customer-crm.com/bad-request";
            case 401: return "https://problems.customer-crm.com/unauthorized";
            case 403: return "https://problems.customer-crm.com/forbidden";
            case 404: return "https://problems.customer-crm.com/not-found";
            case 405: return "https://problems.customer-crm.com/method-not-allowed";
            case 406: return "https://problems.customer-crm.com/not-acceptable";
```

```
            case 409: return "https://problems.customer-crm.com/conflict";
            case 415: return "https://problems.customer-crm.com/unsupported-media-type";
            case 500: return "https://problems.customer-crm.com/internal-error";
            default: return "https://problems.customer-crm.com/http-error";
        }
    }

    private String getTitleForStatus(int status) {
        switch (status) {
            case 400: return "Bad Request";
            case 401: return "Unauthorized";
            case 403: return "Forbidden";
            case 404: return "Not Found";
            case 405: return "Method Not Allowed";
            case 406: return "Not Acceptable";
            case 409: return "Conflict";
            case 415: return "Unsupported Media Type";
            case 500: return "Internal Server Error";
            default: return "HTTP Error";
        }
    }

    private String getDefaultDetailForStatus(int status) {
        switch (status) {
            case 400: return "The request was malformed or invalid";
            case 401: return "Authentication is required to access this resource";
            case 403: return "Access to this resource is forbidden";
            case 404: return "The requested resource was not found";
            case 405: return "The HTTP method is not allowed for this resource";
            case 406: return "The requested media type is not acceptable";
            case 409: return "The request conflicts with the current state of the resource";
            case 415: return "The media type is not supported";
            case 500: return "An internal server error occurred";
            default: return "An HTTP error occurred";
        }
    }
}
```

```java
package exam.exceptions;

import com.fasterxml.jackson.core.JsonProcessingException;
import exam.api.error.ProblemDetail;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.ExceptionMapper;
import jakarta.ws.rs.ext.Provider;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Provider
public class JsonProcessingExceptionMapper implements ExceptionMapper<JsonProcessingException>
 {

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

    @Context
    private HttpServletRequest request;

    @Context
    private HttpHeaders httpHeaders;


    @Override
    public Response toResponse(JsonProcessingException exception) {
        logger.warn("JSON processing error occurred: {}", exception.getMessage());

    String instance = request != null ? request.getRequestURI() : "unknown";

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/malformed-json",
                "Malformed JSON",
                Response.Status.BAD_REQUEST.getStatusCode(),
                "Request body contains malformed JSON: " + exception.getOriginalMessage(),
        instance
        );

    return ProblemDetailResponseBuilder.build(request, httpHeaders, Response.Status.BAD_REQUES
T, problemDetail);
    }
}
```

```java
package exam.exceptions;

import exam.api.error.ProblemDetail;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.ext.ExceptionMapper;
import jakarta.ws.rs.ext.Provider;
import jakarta.ws.rs.core.Context;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Provider
public class IllegalArgumentExceptionMapper implements ExceptionMapper<IllegalArgumentExceptio
n> {

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

    @Context
    private HttpServletRequest request;

    @Context
    private HttpHeaders httpHeaders;

    @Override
    public Response toResponse(IllegalArgumentException exception) {
        logger.warn("Illegal argument error occurred: {}", exception.getMessage());

    String instance = request != null ? request.getRequestURI() : "unknown";

        ProblemDetail problemDetail = new ProblemDetail(
                "https://problems.customer-crm.com/illegal-argument",
                "Illegal Argument",
                Response.Status.BAD_REQUEST.getStatusCode(),
                exception.getMessage(),
        instance
        );

    return ProblemDetailResponseBuilder.build(request, httpHeaders, Response.Status.BAD_REQUES
T, problemDetail);
    }
}
```

```java
package exam.health;

import com.codahale.metrics.health.HealthCheck;
import org.jdbi.v3.core.Jdbi;

/**
 * A health check for the database connection.
 * It executes a simple query to ensure the connection is alive and valid.
 */
public class DatabaseHealthCheck extends HealthCheck {
    private final Jdbi jdbi;

    public DatabaseHealthCheck(Jdbi jdbi) {
        this.jdbi = jdbi;
    }

    @Override
    protected Result check() throws Exception {
        try {
            jdbi.withHandle(handle -> handle.execute("SELECT 1"));
            return Result.healthy("Database connection is healthy.");
        } catch (Exception e) {
            return Result.unhealthy("Cannot connect to the database: " + e.getMessage());
        }
    }
}
```

```java
package exam.resources;

import exam.api.Customer;
import exam.db.CustomerDAO;
import jakarta.validation.Valid;
import jakarta.validation.constraints.NotNull;
import jakarta.ws.rs.*;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.core.UriBuilder;

import java.net.URI;
import java.util.List;

@Path("/customers")
@Produces(MediaType.APPLICATION_JSON)
public class CustomerResource {

    private final CustomerDAO customerDAO;

    public CustomerResource(CustomerDAO customerDAO) {
        this.customerDAO = customerDAO;
    }

    @GET
    public List<Customer> getAllCustomers() {
        return customerDAO.findAll();
    }

    @GET
    @Path("/{id}")
    public Response getCustomer(@PathParam("id") Long id) {
        return customerDAO.findById(id)
                .map(customer -> Response.ok(customer).build())
                .orElse(Response.status(Response.Status.NOT_FOUND).build());
    }

    @POST
    public Response createCustomer(@NotNull @Valid Customer customer) {
        customer.setId(0);
        Customer newCustomer = customerDAO.save(customer);
        URI location = UriBuilder.fromResource(CustomerResource.class).path("{id}").build(newC
ustomer.getId());
        return Response.created(location).entity(newCustomer).build();
    }

    @PUT
    @Path("/{id}")
    public Response updateCustomer(@PathParam("id") Long id, @NotNull @Valid Customer customer
) {
        return customerDAO.findById(id)
                .map(existingCustomer -> {
                    customer.setId(id);
                    Customer updatedCustomer = customerDAO.save(customer);
                    return Response.ok(updatedCustomer).build();
                })
                .orElse(Response.status(Response.Status.NOT_FOUND).build());
    }

    @DELETE
    @Path("/{id}")
    public Response deleteCustomer(@PathParam("id") Long id) {
        if (customerDAO.deleteById(id) > 0) {
```

```
            return Response.noContent().build();
        } else {
            return Response.status(Response.Status.NOT_FOUND).build();
        }
    }
}
```

```java
package exam.resources;

import exam.api.Customer;
import exam.db.CustomerDAO;
import exam.views.HomeView;
import exam.views.TestValidationView;
import exam.views.customers.CustomerDetailView;
import exam.views.customers.CustomerFormView;
import exam.views.customers.CustomerListView;
import exam.views.forms.CustomerForm;
import jakarta.validation.ConstraintViolation;
import jakarta.validation.Validator;
import jakarta.ws.rs.BeanParam;
import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.FormParam;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.PathParam;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import jakarta.ws.rs.core.UriBuilder;

import java.time.LocalDate;
import java.time.format.DateTimeParseException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Objects;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

@Path("/ui")
@Produces(MediaType.TEXT_HTML)
public class CustomerViewResource {

    private final CustomerDAO customerDAO;
    private final Validator validator;

    private static final Map<String, String> CUSTOMER_TYPE_OPTIONS =
            Collections.unmodifiableMap(createCustomerTypeOptions());
    private static final Map<String, String> STATUS_OPTIONS =
            Collections.unmodifiableMap(createStatusOptions());
    private static final Map<String, String> CONTACT_METHOD_OPTIONS =
            Collections.unmodifiableMap(createContactMethodOptions());

    public CustomerViewResource(CustomerDAO customerDAO, Validator validator) {
        this.customerDAO = customerDAO;
        this.validator = validator;
    }

    @GET
    public HomeView home() {
        return new HomeView(customerDAO.count());
    }

    @GET
```

```
    @Path("/test-validation")
    public TestValidationView testValidation() {
        return new TestValidationView();
    }

    @GET
    @Path("/customers")
    public CustomerListView listCustomers(@QueryParam("message") String message) {
        return new CustomerListView(customerDAO.findAll(), message);
    }

    @GET
    @Path("/customers/new")
    public CustomerFormView newCustomerForm() {
        CustomerForm form = new CustomerForm();
        form.setStatus(Customer.Status.LEAD.name());
        form.setCustomerType(Customer.CustomerType.SOLE_PROPRIETORSHIP.name());
        form.setWantsToBeContactedBy(new ArrayList<>());
        return buildFormView(form, false, Collections.emptyMap(), Collections.emptyList());
    }

    @GET
    @Path("/customers/{id}")
    public Response viewCustomer(@PathParam("id") long id) {
        return customerDAO.findById(id)
                .map(customer -> Response.ok(new CustomerDetailView(customer, CONTACT_METHOD_O
PTIONS)).build())
                .orElseGet(() -> redirectToList("Kunde nicht gefunden"));
    }

    @GET
    @Path("/customers/{id}/edit")
    public Response editCustomerForm(@PathParam("id") long id) {
        Optional<Customer> existing = customerDAO.findById(id);
        if (existing.isEmpty()) {
            return redirectToList("Kunde nicht gefunden");
        }
        CustomerForm form = new CustomerForm();
        form.populateFromCustomer(existing.get());
        return Response.ok(buildFormView(form, true, Collections.emptyMap(), Collections.empty
List())).build();
    }

    @POST
    @Path("/customers")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public Response createCustomer(@BeanParam CustomerForm form) {
        Map<String, List<String>> fieldErrors = new LinkedHashMap<>();
        List<String> globalErrors = new ArrayList<>();

        Customer customer = buildCustomerFromForm(form, fieldErrors, globalErrors);
        customer.setId(0);

        addViolations(fieldErrors, validator.validate(customer));

        if (!fieldErrors.isEmpty() || !globalErrors.isEmpty()) {
            return Response.status(Response.Status.BAD_REQUEST)
                    .entity(buildFormView(form, false, fieldErrors, globalErrors))
                    .build();
        }

        Customer saved = customerDAO.save(customer);
        return redirectToList("Kunde â\200\236" + saved.getName() + "â\200\234 erfolgreich ers
```

```
tellt");
    }

    @POST
    @Path("/customers/{id}")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public Response updateCustomer(@PathParam("id") long id, @BeanParam CustomerForm form) {
        Optional<Customer> existing = customerDAO.findById(id);
        if (existing.isEmpty()) {
            return redirectToList("Kunde nicht gefunden");
        }

        form.setId(id);

        Map<String, List<String>> fieldErrors = new LinkedHashMap<>();
        List<String> globalErrors = new ArrayList<>();

        Customer customer = buildCustomerFromForm(form, fieldErrors, globalErrors);
        customer.setId(id);

        addViolations(fieldErrors, validator.validate(customer));

        if (!fieldErrors.isEmpty() || !globalErrors.isEmpty()) {
            return Response.status(Response.Status.BAD_REQUEST)
                    .entity(buildFormView(form, true, fieldErrors, globalErrors))
                    .build();
        }

        customerDAO.save(customer);
        return redirectToList("Kunde â\200\236" + customer.getName() + "â\200\234 wurde aktual
isiert");
    }

    @POST
    @Path("/customers/{id}/delete")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public Response deleteCustomer(@PathParam("id") long id) {
        Optional<Customer> existing = customerDAO.findById(id);
        if (existing.isEmpty()) {
            return redirectToList("Kunde nicht gefunden");
        }

        customerDAO.deleteById(id);
        return redirectToList("Kunde â\200\236" + existing.get().getName() + "â\200\234 erfolg
reich gelÃ¶scht");
    }

    @POST
    @Path("/test-force-error")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void forceError(@FormParam("errorType") String errorType) {
        String normalized = errorType == null ? "generic" : errorType.toLowerCase(Locale.ROOT)
;
        switch (normalized) {
            case "illegal-argument":
                throw new IllegalArgumentException("Absichtlich ausgelÃ¶ste IllegalArgumentExc
eption fÃ¼r Testzwecke");
            case "null-pointer":
                throw new NullPointerException("Absichtlich ausgelÃ¶ste NullPointerException f
Ã¼r Testzwecke");
            case "generic":
            default:
                throw new RuntimeException("Absichtlich ausgelÃ¶ste RuntimeException fÃ¼r Test
```

```
zwecke");
        }
    }

    private CustomerFormView buildFormView(CustomerForm form,
                                           boolean edit,
                                           Map<String, List<String>> fieldErrors,
                                           List<String> globalErrors) {
        return new CustomerFormView(
                form,
                edit,
                fieldErrors,
                globalErrors,
                CUSTOMER_TYPE_OPTIONS,
                STATUS_OPTIONS,
                CONTACT_METHOD_OPTIONS
        );
    }

    private Customer buildCustomerFromForm(CustomerForm form,
                                           Map<String, List<String>> fieldErrors,
                                           List<String> globalErrors) {
        Customer customer = new Customer();
        customer.setName(trimToNull(form.getName()));
        customer.setContactPerson(trimToNull(form.getContactPerson()));
        customer.setAddress(trim(form.getAddress()));
        customer.setEmail(trimToNull(form.getEmail()));
        customer.setPhone(trimToNull(form.getPhone()));
        customer.setIndustry(trim(form.getIndustry()));

        String typeValue = trimToNull(form.getCustomerType());
        if (typeValue == null) {
            addFieldError(fieldErrors, "customerType", "Kundentyp ist erforderlich");
        } else {
            try {
                customer.setCustomerType(Customer.CustomerType.valueOf(typeValue));
            } catch (IllegalArgumentException ex) {
                addFieldError(fieldErrors, "customerType", "UngÃ¼ltiger Kundentyp");
            }
        }

        String statusValue = trimToNull(form.getStatus());
        if (statusValue == null) {
            addFieldError(fieldErrors, "status", "Status ist erforderlich");
        } else {
            try {
                customer.setStatus(Customer.Status.valueOf(statusValue));
            } catch (IllegalArgumentException ex) {
                addFieldError(fieldErrors, "status", "UngÃ¼ltiger Status");
            }
        }

        String lastContact = trim(form.getLastContactDate());
        if (lastContact != null && !lastContact.isEmpty()) {
            try {
                customer.setLastContactDate(LocalDate.parse(lastContact));
            } catch (DateTimeParseException ex) {
                addFieldError(fieldErrors, "lastContactDate", "UngÃ¼ltiges Datum (Format: yyyy
-MM-dd)");
            }
        } else {
            customer.setLastContactDate(null);
        }
```

```
        List<String> contactMethods = new ArrayList<>(form.getWantsToBeContactedBy().stream()
                .map(CustomerViewResource::trimToNull)
                .filter(Objects::nonNull)
                .collect(Collectors.toList()));

        if (contactMethods.size() > 10) {
            addFieldError(fieldErrors, "wantsToBeContactedBy", "Es sind maximal 10 KontaktprÃ¤
ferenzen erlaubt");
        }

        for (String method : contactMethods) {
            if (!CONTACT_METHOD_OPTIONS.containsKey(method)) {
                addFieldError(fieldErrors, "wantsToBeContactedBy", "Unbekannte KontaktprÃ¤fere
nz: " + method);
            }
        }

        customer.setWantsToBeContactedBy(contactMethods);

        if (customer.getPhone() == null) {
            addFieldError(fieldErrors, "phone", "Telefonnummer ist erforderlich");
        }

        return customer;
    }

    private void addViolations(Map<String, List<String>> fieldErrors,
                               Set<ConstraintViolation<Customer>> violations) {
        for (ConstraintViolation<Customer> violation : violations) {
            addFieldError(fieldErrors,
                    simplifyPropertyPath(violation.getPropertyPath().toString()),
                    violation.getMessage());
        }
    }

    private static void addFieldError(Map<String, List<String>> fieldErrors, String field, Str
ing message) {
        fieldErrors.computeIfAbsent(field, key -> new ArrayList<>()).add(message);
    }

    private static String simplifyPropertyPath(String path) {
        String simplified = path;
        if (simplified.contains(".")) {
            simplified = simplified.substring(simplified.lastIndexOf('.') + 1);
        }
        if (simplified.contains("[")) {
            simplified = simplified.substring(0, simplified.indexOf('['));
        }
        return simplified;
    }

    private Response redirectToList(String message) {
        UriBuilder builder = UriBuilder.fromPath("/ui/customers");
        if (message != null && !message.isBlank()) {
            builder.queryParam("message", message);
        }
        return Response.seeOther(builder.build()).build();
    }

    private static Map<String, String> createCustomerTypeOptions() {
        Map<String, String> options = new LinkedHashMap<>();
        options.put(Customer.CustomerType.SOLE_PROPRIETORSHIP.name(), "Einzelunternehmen");
```

```
        options.put(Customer.CustomerType.LIMITED_LIABILITY_COMPANY.name(), "Limited Liability
 Company (LLC)");
        options.put(Customer.CustomerType.CORPORATION.name(), "Aktiengesellschaft");
        options.put(Customer.CustomerType.NON_PROFIT_ORGANIZATION.name(), "GemeinnÃ¼tzige Orga
nisation");
        return options;
    }

    private static Map<String, String> createStatusOptions() {
        Map<String, String> options = new LinkedHashMap<>();
        for (Customer.Status status : Customer.Status.values()) {
            options.put(status.name(), status.name().substring(0, 1) + status.name().substring
(1).toLowerCase(Locale.GERMAN));
        }
        return options;
    }

    private static Map<String, String> createContactMethodOptions() {
        Map<String, String> options = new LinkedHashMap<>();
        options.put("EMAIL", "E-Mail");
        options.put("PHONE", "Telefon");
        options.put("VISIT", "Vor-Ort-Termin");
        options.put("VIDEO_CALL", "Video-Call");
        options.put("CHAT", "Live-Chat");
        return options;
    }

    private static String trim(String value) {
        return value == null ? null : value.trim();
    }

    private static String trimToNull(String value) {
        String trimmed = trim(value);
        return (trimmed == null || trimmed.isEmpty()) ? null : trimmed;
    }
}
```

```java
package exam.api;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.validation.constraints.*;

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

/**
 * Represents a customer in the CRM system.
 * This class is used for API requests and responses.
 */
public class Customer {

    @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")
    @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 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 Status status;

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

```
tedBy;

    // Helper field for database operations (not exposed in JSON)
    @JsonIgnore
    private String wantsToBeContactedByString;

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

    public enum CustomerType {
        SOLE_PROPRIETORSHIP,
        LIMITED_LIABILITY_COMPANY,
        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
    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 CustomerType getCustomerType() { return customerType; }
    public void setCustomerType(CustomerType customerType) { this.customerType = customerType;
 }

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

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

    // Helper methods for database operations
    public String getWantsToBeContactedByString() { return wantsToBeContactedByString; }
    public void setWantsToBeContactedByString(String wantsToBeContactedByString) { this.wantsT
oBeContactedByString = wantsToBeContactedByString; }

    @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);
    }

    @Override
    public String toString() {
        return "Customer{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", contactPerson='" + contactPerson + '\'' +
                ", customerType=" + customerType +
                ", status=" + status +
                '}';
    }
}
```

```java
package exam.api.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 exam.api.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 exam;

import com.fasterxml.jackson.annotation.JsonProperty;
import io.dropwizard.core.Configuration;
import io.dropwizard.db.DataSourceFactory;

/**
 * The configuration class for the CRM application.
 * Includes database settings for PostgreSQL.
 */
public class CRMConfiguration extends Configuration {
    @JsonProperty
    private DataSourceFactory database = new DataSourceFactory();

    public DataSourceFactory getDataSourceFactory() {
        return database;
    }
}
```

```java
package exam.views;

public class HomeView extends BaseView {

    private final long customerCount;

    public HomeView(long customerCount) {
        super("home.ftl", "Startseite - CRM", "/ui");
        this.customerCount = customerCount;
    }

    public long getCustomerCount() {
        return customerCount;
    }
}
```

```java
package exam.views;

public class TestValidationView extends BaseView {

    public TestValidationView() {
        super("test-validation.ftl", "Validierung testen - CRM", "/ui/test-validation");
    }
}
```

```java
package exam.views.forms;

import exam.api.Customer;
import jakarta.ws.rs.FormParam;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Optional;

public class CustomerForm {

    @FormParam("id")
    private Long id;

    @FormParam("name")
    private String name;

    @FormParam("contactPerson")
    private String contactPerson;

    @FormParam("address")
    private String address;

    @FormParam("email")
    private String email;

    @FormParam("phone")
    private String phone;

    @FormParam("customerType")
    private String customerType;

    @FormParam("industry")
    private String industry;

    @FormParam("lastContactDate")
    private String lastContactDate;

    @FormParam("status")
    private String status;

    @FormParam("wantsToBeContactedBy")
    private List<String> wantsToBeContactedBy;

    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 String getCustomerType() {
        return customerType;
    }

    public void setCustomerType(String customerType) {
        this.customerType = customerType;
    }

    public String getIndustry() {
        return industry;
    }

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

    public String getLastContactDate() {
        return lastContactDate;
    }

    public void setLastContactDate(String lastContactDate) {
        this.lastContactDate = lastContactDate;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public List<String> getWantsToBeContactedBy() {
```

```
        return wantsToBeContactedBy == null ? Collections.emptyList() : wantsToBeContactedBy;
    }

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

    public void populateFromCustomer(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() != null ? customer.getCustomerType().na
me() : null;
        this.industry = customer.getIndustry();
        this.lastContactDate = customer.getLastContactDate() != null ? customer.getLastContact
Date().toString() : null;
        this.status = customer.getStatus() != null ? customer.getStatus().name() : null;
        this.wantsToBeContactedBy = new ArrayList<>(Optional.ofNullable(customer.getWantsToBeC
ontactedBy())
                .orElse(Collections.emptyList()));
    }
}
```

```java
package exam.views.customers;

import exam.api.Customer;
import exam.views.BaseView;

import java.util.List;

public class CustomerListView extends BaseView {

    private final List<Customer> customers;
    private final String message;

    public CustomerListView(List<Customer> customers, String message) {
        super("list.ftl", "KundenÃ¼bersicht - CRM", "/ui/customers");
        this.customers = customers;
        this.message = message;
    }

    public List<Customer> getCustomers() {
        return customers;
    }

    public String getMessage() {
        return message;
    }
}
```

```java
package exam.views.customers;

import exam.views.BaseView;
import exam.views.forms.CustomerForm;

import java.util.Collections;
import java.util.List;
import java.util.Map;

public class CustomerFormView extends BaseView {

    private final CustomerForm form;
    private final boolean edit;
    private final Map<String, List<String>> fieldErrors;
    private final List<String> globalErrors;
    private final Map<String, String> customerTypeOptions;
    private final Map<String, String> statusOptions;
    private final Map<String, String> contactMethodOptions;

    public CustomerFormView(CustomerForm form,
                            boolean edit,
                            Map<String, List<String>> fieldErrors,
                            List<String> globalErrors,
                            Map<String, String> customerTypeOptions,
                            Map<String, String> statusOptions,
                            Map<String, String> contactMethodOptions) {
        super("form.ftl",
                edit ? "Kunde bearbeiten – CRM" : "Kunde anlegen – CRM",
                "/ui/customers");
        this.form = form;
        this.edit = edit;
        this.fieldErrors = fieldErrors == null ? Collections.emptyMap() : fieldErrors;
        this.globalErrors = globalErrors == null ? Collections.emptyList() : globalErrors;
        this.customerTypeOptions = customerTypeOptions;
        this.statusOptions = statusOptions;
        this.contactMethodOptions = contactMethodOptions;
    }

    public CustomerForm getForm() {
        return form;
    }

    public boolean isEdit() {
        return edit;
    }

    public Map<String, List<String>> getFieldErrors() {
        return fieldErrors;
    }

    public List<String> getGlobalErrors() {
        return globalErrors;
    }

    public Map<String, String> getCustomerTypeOptions() {
        return customerTypeOptions;
    }

    public Map<String, String> getStatusOptions() {
        return statusOptions;
    }

    public Map<String, String> getContactMethodOptions() {
```

```
        return contactMethodOptions;
    }
}
```

```java
package exam.views.customers;

import exam.api.Customer;
import exam.views.BaseView;

import java.util.Map;

public class CustomerDetailView extends BaseView {

    private final Customer customer;
    private final Map<String, String> contactMethodOptions;

    public CustomerDetailView(Customer customer, Map<String, String> contactMethodOptions) {
        super("detail.ftl", "Kundendetails - CRM", "/ui/customers");
        this.customer = customer;
        this.contactMethodOptions = contactMethodOptions;
    }

    public Customer getCustomer() {
        return customer;
    }

    public Map<String, String> getContactMethodOptions() {
        return contactMethodOptions;
    }
}
```

```java
package exam.views;

import io.dropwizard.views.common.View;

public abstract class BaseView extends View {

    private final String title;
    private final String activePath;

    protected BaseView(String templateName, String title, String activePath) {
        super(templateName);
        this.title = title;
        this.activePath = activePath;
    }

    public String getTitle() {
        return title;
    }

    public String getActivePath() {
        return activePath;
    }
}
```

```java
package exam.db;

import exam.api.Customer;
import org.jdbi.v3.sqlobject.config.RegisterRowMapper;
import org.jdbi.v3.sqlobject.customizer.Bind;
import org.jdbi.v3.sqlobject.customizer.BindBean;
import org.jdbi.v3.sqlobject.statement.SqlQuery;
import org.jdbi.v3.sqlobject.statement.SqlUpdate;

import java.util.List;
import java.util.Optional;

@RegisterRowMapper(CustomerMapper.class)
public interface CustomerDAO {

    @SqlUpdate("CREATE TABLE IF NOT EXISTS customers (" +
            "id BIGSERIAL PRIMARY KEY," +
            "name VARCHAR(100) NOT NULL," +
            "contact_person VARCHAR(100) NOT NULL," +
            "address VARCHAR(255)," +
            "email VARCHAR(100)," +
            "phone VARCHAR(20)," +
            "customer_type VARCHAR(50) NOT NULL," +
            "industry VARCHAR(100)," +
            "last_contact_date DATE," +
            "status VARCHAR(20) NOT NULL," +
            "wants_to_be_contacted_by TEXT" +
            ")")
    void createTable();

    @SqlQuery("SELECT * FROM customers ORDER BY id")
    List<Customer> findAll();

    @SqlQuery("SELECT * FROM customers WHERE id = :id")
    Optional<Customer> findById(@Bind("id") long id);

    @SqlQuery("SELECT * FROM customers WHERE name ILIKE '%' || :name || '%'")
    List<Customer> findByNameContaining(@Bind("name") String name);

    @SqlQuery("SELECT * FROM customers WHERE status = :status")
    List<Customer> findByStatus(@Bind("status") String status);

    @SqlQuery("SELECT * FROM customers WHERE customer_type = :customerType")
    List<Customer> findByCustomerType(@Bind("customerType") String customerType);

    @SqlUpdate("INSERT INTO customers (name, contact_person, address, email, phone, customer_t
ype, industry, last_contact_date, status, wants_to_be_contacted_by) " +
            "VALUES (:name, :contactPerson, :address, :email, :phone, :customerType, :industry
, :lastContactDate, :status, :wantsToBeContactedByString)")
    @org.jdbi.v3.sqlobject.statement.GetGeneratedKeys
    Customer insert(@BindBean Customer customer);

    @SqlUpdate("UPDATE customers SET " +
            "name = :name, " +
            "contact_person = :contactPerson, " +
            "address = :address, " +
            "email = :email, " +
            "phone = :phone, " +
            "customer_type = :customerType, " +
            "industry = :industry, " +
            "last_contact_date = :lastContactDate, " +
            "status = :status, " +
            "wants_to_be_contacted_by = :wantsToBeContactedByString " +
```

```
            "WHERE id = :id")
    int update(@BindBean Customer customer);

    @SqlUpdate("DELETE FROM customers WHERE id = :id")
    int deleteById(@Bind("id") long id);

    @SqlQuery("SELECT COUNT(*) FROM customers")
    long count();

    default Customer save(Customer customer) {
        if (customer == null) {
            throw new IllegalArgumentException("Customer cannot be null");
        }

        // Validate required fields
        if (customer.getName() == null || customer.getName().trim().isEmpty()) {
            throw new IllegalArgumentException("Customer name is required");
        }
        if (customer.getContactPerson() == null || customer.getContactPerson().trim().isEmpty(
)) {
            throw new IllegalArgumentException("Contact person is required");
        }
        if (customer.getCustomerType() == null) {
            throw new IllegalArgumentException("Customer type is required");
        }
        if (customer.getStatus() == null) {
            throw new IllegalArgumentException("Status is required");
        }

        // Set default status if not provided
        if (customer.getStatus() == null) {
            customer.setStatus(Customer.Status.LEAD);
        }

        // Prepare contact methods string
        if (customer.getWantsToBeContactedBy() != null && !customer.getWantsToBeContactedBy().
isEmpty()) {
            customer.setWantsToBeContactedByString(String.join(",", customer.getWantsToBeConta
ctedBy()));
        } else {
            customer.setWantsToBeContactedByString(null);
        }

        if (customer.getId() == 0) {
            return insert(customer);
        } else {
            update(customer);
            return customer;
        }
    }
}
```

```java
package exam.db;

import exam.api.Customer;
import org.jdbi.v3.core.mapper.RowMapper;
import org.jdbi.v3.core.statement.StatementContext;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Arrays;
import java.util.Collections;

public class CustomerMapper implements RowMapper<Customer> {

    @Override
    public Customer map(ResultSet rs, StatementContext ctx) throws SQLException {
        Customer customer = new Customer();
        customer.setId(rs.getLong("id"));
        customer.setName(rs.getString("name"));
        customer.setContactPerson(rs.getString("contact_person"));
        customer.setAddress(rs.getString("address"));
        customer.setEmail(rs.getString("email"));
        customer.setPhone(rs.getString("phone"));

        // Handle Enum mapping
        customer.setCustomerType(Customer.CustomerType.valueOf(rs.getString("customer_type")))
;
        customer.setStatus(Customer.Status.valueOf(rs.getString("status")));

        customer.setIndustry(rs.getString("industry"));

        // Handle nullable date
        java.sql.Date dbDate = rs.getDate("last_contact_date");
        if (dbDate != null) {
            customer.setLastContactDate(dbDate.toLocalDate());
        }

        // Handle comma-separated list
        String contactedBy = rs.getString("wants_to_be_contacted_by");
        if (contactedBy != null && !contactedBy.isEmpty()) {
            customer.setWantsToBeContactedBy(Arrays.asList(contactedBy.split(",")));
        } else {
            customer.setWantsToBeContactedBy(Collections.emptyList());
        }

        return customer;
    }
}
```