Anhang I

In diesem Dokument wird der gesamte umgesetzte Code bereitgestellt. Wurde die Datei nicht erstellt, sondern darauf aufgebaut, wird nur der Ausschnitt mit der Eigenleistung aufgezeigt.

Backend

**src/main/java/com/generali/ovweb/controller/CustomerController.java**

**(Nur Ausschnitt)**

|  |
| --- |
| (...) public class CustomerController {  (...)  @PostMapping(value = "/import", consumes = MediaType.*MULTIPART\_FORM\_DATA\_VALUE*) @Operation(  summary = "Import Customers from a .xls or .xlsx file",  responses = {  @ApiResponse(responseCode = "200", description = "Successfully imported customers"),  @ApiResponse(responseCode = "400", description = "Invalid file format")  }) public ImportResult importCustomers(@RequestParam("file") MultipartFile file) {  try {  return importService.importData(file);  } catch (Exception e) {  throw new ResponseStatusException(HttpStatus.*BAD\_REQUEST*, e.getMessage());  } }  (...) |

**src/main/java/com/generali/ovweb/model/Customer.java**

**(Nur Ausschnitt)**

|  |
| --- |
| (...)  public class Customer {  (...)  @Column(nullable = false) private String partnerNumber;  (...) |

**src/main/java/com/generali/ovweb/persistence/CustomerRepository.java**

**(Nur Ausschnitt)**

|  |
| --- |
| (...)  public interface CustomerRepository extends JpaRepository<Customer, Long> {  (...)  Optional<Customer> findByLastNameAndFirstNameAndBirthdate(  String lastName, String firstName, LocalDate birthdate);  (...) |

**src/main/java/com/generali/ovweb/persistence/CustomerRepository.java**

**(Nur Ausschnitt)**

|  |
| --- |
| (...)  public class CustomerService {  (...)  Address newAddress = customer.getAddress(); if (newAddress != null) {  existingCustomer.setAddress(newAddress); }  (...) |

**src/main/java/com/generali/ovweb/service/ExcelProcessingService.java**

|  |
| --- |
| package com.generali.ovweb.service;  import com.generali.ovweb.model.Address; import com.generali.ovweb.model.Customer; import com.generali.ovweb.model.enums.Gender; import com.generali.ovweb.model.enums.Salutation; import com.generali.ovweb.persistence.UserRepository; import java.io.IOException; import java.time.LocalDate; import java.util.\*; import lombok.extern.slf4j.Slf4j; import org.apache.poi.hssf.usermodel.HSSFWorkbook; import org.apache.poi.ss.usermodel.\*; import org.apache.poi.xssf.usermodel.XSSFWorkbook; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service; import org.springframework.web.multipart.MultipartFile;  @Slf4j @Service public class ExcelProcessingService {   private final UserRepository userRepository;   @Autowired  public ExcelProcessingService(UserRepository userRepository) {  this.userRepository = userRepository;  }   Workbook getWorkbookFromMultipartFile(MultipartFile file) throws IOException {  String contentType = file.getContentType();  if ("application/vnd.ms-excel".equals(contentType)) {  // .xls  return new HSSFWorkbook(file.getInputStream());  } else if ("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"  .equals(contentType)) {  // .xlsx  return new XSSFWorkbook(file.getInputStream());  } else {  throw new IllegalArgumentException("Das Dateiformat wird nicht unterstützt.");  }  }   void validateWorkbookStructure(Workbook workbook) {  Sheet customersSheet = workbook.getSheet("Customers");  Sheet addressesSheet = workbook.getSheet("Addresses");   if (customersSheet == null) {  throw new IllegalArgumentException("Customers sheet not found in the workbook.");  }  if (addressesSheet == null) {  throw new IllegalArgumentException("Addresses sheet not found in the workbook.");  }   validateSheetStructure(  customersSheet,  Arrays.*asList*("PartnerNo", "Name", "Vorname", "Geburtstag", "eMail", "Telefon"));  validateSheetStructure(  addressesSheet,  Arrays.*asList*("PartnerNo", "Strasse", "Hausnummer", "Land", "Plz", "Ortschaft"));  }   void validateSheetStructure(Sheet sheet, List<String> expectedColumnNames) {  *log*.atInfo()  .setMessage("Validiere Struktur für Sheet")  .addKeyValue("Sheet", sheet.getSheetName())  .log();  Row headerRow = sheet.getRow(0);  List<String> actualColumnNames = new ArrayList<>();  headerRow.forEach(  cell -> {  if (cell.getCellType() == CellType.*STRING*) {  actualColumnNames.add(cell.getStringCellValue());  }  });   for (String expectedColumnName : expectedColumnNames) {  if (!actualColumnNames.contains(expectedColumnName)) {  *log*.atError()  .setMessage("Fehlende erwartete Spalte")  .addKeyValue("Spalte", expectedColumnName)  .addKeyValue("Sheet", sheet.getSheetName())  .log();  throw new IllegalArgumentException("Fehlende erwartete Spalte: " + expectedColumnName);  }  }  }   void validateAddresses(List<Customer> importedCustomers, Map<String, Address> addressMap) {  Set<String> customerPartnerNumbers = new HashSet<>();   if (importedCustomers == null) {  throw new IllegalArgumentException("Die Liste der importierten Kunden ist leer");  }  if (addressMap == null) {  throw new IllegalArgumentException("Die Adresse Map ist leer");  }   for (Customer customer : importedCustomers) {  customerPartnerNumbers.add(customer.getPartnerNumber());  }  for (String partnerNumber : addressMap.keySet()) {  if (!customerPartnerNumbers.contains(partnerNumber)) {  String errorMessage = "Kein Kunde gefunden für Partner-Nummer: " + partnerNumber;  *log*.atError()  .setMessage("Kunden Daten Inkonsistent - Adresse gehört zu keinem Kunden")  .addKeyValue("PartnerNumber", partnerNumber)  .log();  throw new IllegalStateException(errorMessage);  }  }  }   Map<String, Address> collectAddresses(Sheet addressesSheet) {  Map<String, Address> addressMap = new HashMap<>();  for (Row row : addressesSheet) {  if (row.getRowNum() == 0) continue;  if (row.getCell(0) == null || row.getCell(0).getCellType() == CellType.*BLANK*) {  break;  }  String partnerNumber = row.getCell(0).getStringCellValue();  Address address = parseAddress(row);  addressMap.put(partnerNumber, address);  }  return addressMap;  }   Address parseAddress(Row row) {  try {  Address address = new Address();  *log*.atInfo()  .setMessage("Parsing address")  .addKeyValue("PartnerNumber", row.getCell(0).getStringCellValue())  .log();  address.setStreetName(row.getCell(1).getStringCellValue());  Cell houseNumberCell = row.getCell(2);  if (houseNumberCell != null) {  String houseNumber;  if (houseNumberCell.getCellType() == CellType.*STRING*) {  houseNumber = houseNumberCell.getStringCellValue();  } else if (houseNumberCell.getCellType() == CellType.*NUMERIC*) {  houseNumber = String.*valueOf*(houseNumberCell.getNumericCellValue());  } else {  *log*.atError()  .setMessage("Ungültiger Zelltyp für Hausnummer")  .addKeyValue("Zeile", row.getRowNum())  .addKeyValue("Hausnummer", houseNumberCell.getCellType())  .log();  throw new IllegalArgumentException("Ungültiger Zelltyp für Hausnummer");  }  address.setHouseNumber(houseNumber);  } else {  *log*.atWarn()  .setMessage("Postleitzahl-Zelle ist null")  .addKeyValue("Zeile", row.getRowNum())  .log();  }  address.setCountry(row.getCell(3).getStringCellValue());  Cell plzCell = row.getCell(4);  if (plzCell != null) {  int plz;  if (plzCell.getCellType() == CellType.*STRING*) {  plz = Integer.*parseInt*(plzCell.getStringCellValue());  } else if (plzCell.getCellType() == CellType.*NUMERIC*) {  plz = (int) plzCell.getNumericCellValue();  } else {  *log*.atError()  .setMessage("Ungültiger Zelltyp für Postleitzahl")  .addKeyValue("Zeile", row.getRowNum())  .addKeyValue("Postleitzahl", plzCell.getCellType())  .log();  throw new IllegalArgumentException("Ungültiger Zelltyp für Postleitzahl");  }  address.setPlz(plz);  } else {  *log*.atWarn()  .setMessage("Postleitzahl-Zelle ist null")  .addKeyValue("Zeile", row.getRowNum())  .log();  }  address.setCity(row.getCell(5).getStringCellValue());  return address;  } catch (IllegalStateException | NumberFormatException | NullPointerException e) {  *log*.atError()  .setMessage("Fehler beim Parsen der Adresse")  .addKeyValue("Zeile", row.getRowNum())  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw e;  }  }   List<Customer> collectCustomers(Sheet customersSheet, Map<String, Address> addressMap) {  List<Customer> importedCustomers = new ArrayList<>();  for (Row row : customersSheet) {  if (row.getRowNum() == 0) continue;  if (row.getCell(0) == null || row.getCell(0).getCellType() == CellType.*BLANK*) {  break;  }  String partnerNumber = row.getCell(0).getStringCellValue();  Address address = addressMap.get(partnerNumber);  if (address != null) {  Customer customer = parseCustomer(row, address);  importedCustomers.add(customer);  }  }  return importedCustomers;  }   Customer parseCustomer(Row row, Address address) {  try {  Customer customer = new Customer();  *log*.atInfo()  .setMessage("Parsing Customer")  .addKeyValue("PartnerNumber", row.getCell(0).getStringCellValue())  .log();  customer.setPartnerNumber(row.getCell(0).getStringCellValue());  customer.setLastName(row.getCell(1).getStringCellValue());  customer.setFirstName(row.getCell(2).getStringCellValue());  LocalDate birthdate = row.getCell(3).getLocalDateTimeCellValue().toLocalDate();  customer.setBirthdate(birthdate);  customer.setEmail(row.getCell(4).getStringCellValue());  customer.setPhoneNumber(row.getCell(5).getStringCellValue());  customer.setAddress(address);   // Hardcode für Gender und Salutation  customer.setGender(Gender.*DIVERSE*);  customer.setSalutation(Salutation.*MX*);   // Hardcode für Demo Owner = Test Users  if (userRepository.findById(1L).isPresent()) {  customer.setOwner(userRepository.findById(1L).get());  }   return customer;  } catch (IllegalStateException | NumberFormatException e) {  *log*.atError()  .setMessage("Fehler beim Parsen des Kunden")  .addKeyValue("Zeile", row.getRowNum())  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw e;  }  } } |

**src/main/java/com/generali/ovweb/service/ImportService.java**

|  |
| --- |
| package com.generali.ovweb.service;  import com.generali.ovweb.model.Address; import com.generali.ovweb.model.Customer; import com.generali.ovweb.model.dto.ImportResult; import com.generali.ovweb.model.dto.UpdateCustomer; import com.generali.ovweb.persistence.CustomerRepository; import io.micrometer.tracing.annotation.NewSpan; import java.io.IOException; import java.util.\*; import lombok.extern.slf4j.Slf4j; import org.apache.poi.ss.usermodel.\*; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transactional; import org.springframework.web.multipart.MultipartFile;  @Slf4j @Service public class ImportService {   private final CustomerRepository customerRepository;  private final CustomerService customerService;  private final ExcelProcessingService excelProcessingService;   private int addedCustomersCount = 0;  private int updatedCustomersCount = 0;   @Autowired  public ImportService(  CustomerService customerService,  CustomerRepository customerRepository,  ExcelProcessingService excelProcessingService) {  this.customerRepository = customerRepository;  this.customerService = customerService;  this.excelProcessingService = excelProcessingService;  }   @NewSpan(name = "import customers")  @Transactional(rollbackFor = Exception.class)  public ImportResult importData(MultipartFile file) {  *log*.atInfo()  .setMessage("Importvorgang gestartet")  .addKeyValue("Dateiname", file.getOriginalFilename())  .log();  ImportResult importResult = new ImportResult();  List<String> errors = new ArrayList<>();   addedCustomersCount = 0;  updatedCustomersCount = 0;   try (Workbook workbook = excelProcessingService.getWorkbookFromMultipartFile(file)) {  excelProcessingService.validateWorkbookStructure(workbook);  Sheet customersSheet = workbook.getSheet("Customers");  Sheet addressesSheet = workbook.getSheet("Addresses");   Map<String, Address> addressMap = excelProcessingService.collectAddresses(addressesSheet);  List<Customer> potentialImportedCustomers =  excelProcessingService.collectCustomers(customersSheet, addressMap);   excelProcessingService.validateAddresses(potentialImportedCustomers, addressMap);   for (Customer customer : potentialImportedCustomers) {  saveOrUpdateCustomer(customer);  }   importResult.setAddedCount(addedCustomersCount);  importResult.setUpdatedCount(updatedCustomersCount);  importResult.setErrors(errors);  importResult.setImportedCustomers(potentialImportedCustomers);   } catch (IOException e) {  *log*.atError()  .setMessage("Lesefehler der Excel-Datei")  .addKeyValue("Dateiname", file.getOriginalFilename())  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw new IllegalStateException(e.getMessage());  } catch (NullPointerException e) {  *log*.atError()  .setMessage("Ein erwartetes Sheet wurde nicht gefunden")  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw new NullPointerException(e.getMessage());  } catch (IllegalStateException | NumberFormatException e) {  *log*.atError()  .setMessage("Fehler beim Parsen der Daten")  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw new IllegalStateException(e.getMessage());  } catch (Exception e) {  errors.add(e.getMessage());  *log*.atError()  .setMessage("Ein unerwarteter Fehler ist aufgetreten")  .addKeyValue("Fehlermeldung", e.getMessage())  .log();  throw new IllegalStateException(e.getMessage());  }  *log*.atInfo()  .setMessage("Import abgeschlossen")  .addKeyValue("Hinzugefügt", addedCustomersCount)  .addKeyValue("Aktualisiert", updatedCustomersCount)  .addKeyValue("Fehler", errors)  .log();  return importResult;  }   private void saveOrUpdateCustomer(Customer customer) {  Optional<Customer> existingCustomerOpt =  customerRepository.findByLastNameAndFirstNameAndBirthdate(  customer.getLastName(), customer.getFirstName(), customer.getBirthdate());   if (existingCustomerOpt.isPresent()) {  *log*.atInfo()  .setMessage("Aktualisiere existierenden Kunden")  .addKeyValue("partnerNumber", customer.getPartnerNumber())  .addKeyValue("existingCustomerId", existingCustomerOpt.get().getId())  .log();  Customer existingCustomer = existingCustomerOpt.get();  updateExistingCustomer(existingCustomer, customer);  updatedCustomersCount++;  } else {  *log*.atInfo()  .setMessage("Füge neuen Kunden hinzu")  .addKeyValue("partnerNumber", customer.getPartnerNumber())  .log();  customerRepository.save(customer);  addedCustomersCount++;  }  }   private void updateExistingCustomer(Customer existingCustomer, Customer newCustomer) {  UpdateCustomer updateCustomerData = new UpdateCustomer();  updateCustomerData.setCustomer(newCustomer);  updateCustomerData.setUpdaterId("Import-Wizard");   customerService.updateCustomer(existingCustomer.getId(), updateCustomerData);  } } |

**src/main/java/com/generali/ovweb/model/dto/ImportResult.java**

|  |
| --- |
| package com.generali.ovweb.model.dto;  import com.generali.ovweb.model.Customer; import java.util.List; import lombok.AllArgsConstructor; import lombok.Getter; import lombok.NoArgsConstructor; import lombok.Setter;  @Setter @Getter @NoArgsConstructor @AllArgsConstructor public class ImportResult {   private int addedCount;  private int updatedCount;  private List<String> errors;  private List<Customer> importedCustomers; } |

**pom.xml**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  (...)  <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi</artifactId>  <version>5.2.5</version> </dependency> <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi-ooxml</artifactId>  <version>5.2.5</version> </dependency>  (...) |

Datenbank (Flyway)

**src/main/resources/db/migration/V1\_0\_10\_\_add\_partner\_number\_to\_customertabel.sql**

|  |
| --- |
| ALTER TABLE if exists customer  ADD COLUMN partner\_number VARCHAR(255); |

Tests

**src/test/java/com/generali/ovweb/service/ExcelProcessingServiceTest.java**

|  |
| --- |
| package com.generali.ovweb.service;  import static org.junit.Assert.\*; import static org.junit.jupiter.api.Assertions.*assertDoesNotThrow*;  import com.generali.ovweb.model.Address; import com.generali.ovweb.model.Customer; import com.generali.ovweb.persistence.UserRepository;   import java.util.\*;  import org.apache.poi.ss.usermodel.\*; import org.apache.poi.xssf.usermodel.XSSFWorkbook; import org.junit.jupiter.api.Test; import org.mockito.Mock; import org.mockito.Mockito; import org.springframework.web.multipart.MultipartFile;  class ExcelProcessingServiceTest {   @Mock UserRepository userRepository;   ExcelProcessingService excelProcessingService = new ExcelProcessingService(userRepository);   @Test  void getWorkbookFromMultipartFile\_withUnsupportedFileType\_throwsException() {  MultipartFile file = Mockito.*mock*(MultipartFile.class);  Mockito.*when*(file.getContentType()).thenReturn("unsupported/type");   *assertThrows*(  IllegalArgumentException.class,  () -> {  excelProcessingService.getWorkbookFromMultipartFile(file);  });  }   @Test  void validateWorkbookStructure\_withValidWorkbook\_NoExceptionThrown() {  Workbook workbook = new XSSFWorkbook();   Sheet customerSheet = workbook.createSheet("Customers");  Row customerRow = customerSheet.createRow(0);  String[] customerHeaders = {"PartnerNo", "Name", "Vorname", "Geburtstag", "eMail", "Telefon"};  for (int i = 0; i < customerHeaders.length; i++) {  Cell cell = customerRow.createCell(i);  cell.setCellValue(customerHeaders[i]);  }   Sheet addressSheet = workbook.createSheet("Addresses");  Row addressRow = addressSheet.createRow(0);  String[] addressHeaders = {"PartnerNo", "Strasse", "Hausnummer", "Land", "Plz", "Ortschaft"};  for (int i = 0; i < addressHeaders.length; i++) {  Cell cell = addressRow.createCell(i);  cell.setCellValue(addressHeaders[i]);  }   *assertDoesNotThrow*(() -> excelProcessingService.validateWorkbookStructure(workbook));  }   @Test  void validateWorkbookStructure\_withMissingSheets\_ExceptionThrown() {  Workbook workbook = new XSSFWorkbook();   *assertThrows*(  IllegalArgumentException.class,  () -> excelProcessingService.validateWorkbookStructure(workbook));  }   @Test  void validateAddresses\_withCustomersPresent\_noExceptionThrown() {  List<Customer> customers = new ArrayList<>();  Map<String, Address> addresses = new HashMap<>();   Customer customer = new Customer();  customer.setPartnerNumber("123");  customers.add(customer);   Address address = new Address();  addresses.put("123", address);   *assertDoesNotThrow*(() -> excelProcessingService.validateAddresses(customers, addresses));  }   @Test  void validateAddresses\_withMissingCustomer\_ThrowsExceptionWithCorrectContains() {  List<Customer> customers = new ArrayList<>();  Map<String, Address> addresses = new HashMap<>();   Address address = new Address();  addresses.put("123", address);   IllegalStateException thrown =  *assertThrows*(  IllegalStateException.class,  () -> {  excelProcessingService.validateAddresses(customers, addresses);  });   *assertTrue*(  thrown.getMessage().contains("Kein Kunde gefunden für Partner-Nummer: 123"));  }   @Test  void validateAddresses\_withNullImportedCustomersList\_ThrowsIllegalArgumentException() {  *assertThrows*(  IllegalArgumentException.class,  () -> excelProcessingService.validateAddresses(null, new HashMap<>()));  }   @Test  void validateAddresses\_withNullAddressMap\_ThrowsIllegalArgumentException() {  *assertThrows*(  IllegalArgumentException.class,  () -> excelProcessingService.validateAddresses(new ArrayList<>(), null));  }   @Test  void validateAddresses\_withBothArgumentsNull\_ThrowsIllegalArgumentException() {  *assertThrows*(  IllegalArgumentException.class, () -> excelProcessingService.validateAddresses(null, null));  }   @Test  void collectAddresses\_withValidData\_noExceptionThrown() {  Sheet sheet = Mockito.*mock*(Sheet.class);  Row headerRow = Mockito.*mock*(Row.class);  Row dataRow = Mockito.*mock*(Row.class);  Cell partnerNumberCell = Mockito.*mock*(Cell.class);   Mockito.*when*(sheet.iterator()).thenReturn(Arrays.*asList*(headerRow, dataRow).iterator());  Mockito.*when*(dataRow.getRowNum()).thenReturn(1);  Mockito.*when*(dataRow.getCell(0)).thenReturn(partnerNumberCell);  Mockito.*when*(partnerNumberCell.getCellType()).thenReturn(CellType.*STRING*);  Mockito.*when*(partnerNumberCell.getStringCellValue()).thenReturn("123");   Address expectedAddress = new Address();  ExcelProcessingService service = Mockito.*spy*(new ExcelProcessingService(userRepository));  Mockito.*doReturn*(expectedAddress).when(service).parseAddress(dataRow);   Map<String, Address> result = service.collectAddresses(sheet);   *assertNotNull*(result);  *assertTrue*(result.containsKey("123"));  *assertEquals*(expectedAddress, result.get("123"));  }   @Test  void collectAddresses\_withInvalidData\_throwsException() {  Sheet sheet = Mockito.*mock*(Sheet.class);  Row headerRow = Mockito.*mock*(Row.class);  Row invalidDataRow = Mockito.*mock*(Row.class);  Cell invalidCell = Mockito.*mock*(Cell.class);   Mockito.*when*(sheet.iterator()).thenReturn(Arrays.*asList*(headerRow, invalidDataRow).iterator());  Mockito.*when*(invalidDataRow.getRowNum()).thenReturn(1);  Mockito.*when*(invalidDataRow.getCell(0)).thenReturn(invalidCell);  Mockito.*when*(invalidCell.getCellType()).thenReturn(CellType.*BLANK*);   ExcelProcessingService service = new ExcelProcessingService(userRepository);   Map<String, Address> result = service.collectAddresses(sheet);   *assertTrue*(result.isEmpty());  }   @Test  void parseAddress\_WithValidRow\_throwsNoException() {  Row mockRow = Mockito.*mock*(Row.class);  Cell mockPartnerNumberCell = Mockito.*mock*(Cell.class);  Cell mockStreetNameCell = Mockito.*mock*(Cell.class);  Cell mockHouseNumberCell = Mockito.*mock*(Cell.class);  Cell mockCountryCell = Mockito.*mock*(Cell.class);  Cell mockPlzCell = Mockito.*mock*(Cell.class);  Cell mockCityCell = Mockito.*mock*(Cell.class);   Mockito.*when*(mockRow.getCell(0)).thenReturn(mockPartnerNumberCell);  Mockito.*when*(mockRow.getCell(1)).thenReturn(mockStreetNameCell);  Mockito.*when*(mockRow.getCell(2)).thenReturn(mockHouseNumberCell);  Mockito.*when*(mockRow.getCell(3)).thenReturn(mockCountryCell);  Mockito.*when*(mockRow.getCell(4)).thenReturn(mockPlzCell);  Mockito.*when*(mockRow.getCell(5)).thenReturn(mockCityCell);   Mockito.*when*(mockPartnerNumberCell.getStringCellValue()).thenReturn("123");  Mockito.*when*(mockStreetNameCell.getStringCellValue()).thenReturn("Main Street");  Mockito.*when*(mockHouseNumberCell.getStringCellValue()).thenReturn("42");  Mockito.*when*(mockHouseNumberCell.getCellType()).thenReturn(CellType.*STRING*);  Mockito.*when*(mockCountryCell.getStringCellValue()).thenReturn("Country");  Mockito.*when*(mockPlzCell.getStringCellValue()).thenReturn("1234");  Mockito.*when*(mockPlzCell.getCellType()).thenReturn(CellType.*STRING*);  Mockito.*when*(mockCityCell.getStringCellValue()).thenReturn("City");   Address result = excelProcessingService.parseAddress(mockRow);   *assertNotNull*(result);  *assertEquals*("Main Street", result.getStreetName());  *assertEquals*("42", result.getHouseNumber());  *assertEquals*("Country", result.getCountry());  *assertEquals*(1234, result.getPlz());  *assertEquals*("City", result.getCity());  } } |

**src/test/java/com/generali/ovweb/service/ImportServiceTest.java**

|  |
| --- |
| package com.generali.ovweb.service;  import static org.junit.jupiter.api.Assertions.\*; import static org.mockito.ArgumentMatchers.*any*; import static org.mockito.Mockito.\*;  import com.generali.ovweb.model.Customer; import com.generali.ovweb.model.dto.ImportResult; import com.generali.ovweb.persistence.CustomerRepository; import java.io.ByteArrayInputStream; import java.io.IOException; import java.util.Collections; import java.util.Optional; import org.apache.poi.hssf.usermodel.HSSFWorkbook; import org.apache.poi.ss.usermodel.Workbook; import org.junit.jupiter.api.Test; import org.springframework.mock.web.MockMultipartFile; import org.springframework.web.multipart.MultipartFile;  class ImportServiceTest {   @Test  void importData\_withValidFileOnlyCustomer\_ReturnsSuccessfulImportResult() throws IOException {   CustomerService mockCustomerService = *mock*(CustomerService.class);  CustomerRepository mockCustomerRepository = *mock*(CustomerRepository.class);  ExcelProcessingService mockExcelProcessingService = *mock*(ExcelProcessingService.class);   Workbook mockWorkbook = new HSSFWorkbook();  *when*(mockExcelProcessingService.getWorkbookFromMultipartFile(*any*(MultipartFile.class)))  .thenReturn(mockWorkbook);   *when*(mockCustomerRepository.findByLastNameAndFirstNameAndBirthdate(  *anyString*(), *anyString*(), *any*()))  .thenReturn(Optional.*empty*());   *when*(mockExcelProcessingService.collectAddresses(*any*())).thenReturn(Collections.*emptyMap*());  *when*(mockExcelProcessingService.collectCustomers(*any*(), *any*()))  .thenReturn(Collections.*singletonList*(new Customer()));   ImportService importService =  new ImportService(mockCustomerService, mockCustomerRepository, mockExcelProcessingService);   byte[] content = {};  MultipartFile mockFile =  new MockMultipartFile(  "file", "test.xls", "application/vnd.ms-excel", new ByteArrayInputStream(content));   ImportResult result = importService.importData(mockFile);   *assertNotNull*(result);  *assertFalse*(result.getImportedCustomers().isEmpty());  *assertTrue*(result.getErrors().isEmpty());   *verify*(mockCustomerRepository, *atLeastOnce*()).save(*any*(Customer.class));  } } |

Frontend

**components/dropdownSuggestions/useDropdownSuggestions.tsx**

|  |
| --- |
| import { useState } from 'react';  import { useSession } from "next-auth/react";  }; |

**components/CreateCustomerForm.tsx**

**(Nur Ausschnitt)**

|  |
| --- |
| (...)  import DropdownSuggestions from './dropdownSuggestions/DropdownSuggestions';  import { useDropdownSuggestions } from './dropdownSuggestions/useDropdownSuggestions';  (...)   civilStatus: z.string().nonempty('Zivilstand ist erforderlich').min(2).max(50),      occupation: z.string().nonempty('Beruf ist erforderlich').min(2).max(50),      employmentRelationship: z.string().nonempty('Anstellungsverhältnis ist erforderlich').min(2).max(50),      employmentPosition: z.string().nonempty('Funktion ist erforderlich').min(2).max(50),      residencePermit: z.string().nonempty('Aufenthaltsbewilligung ist erforderlich').min(2).max(50), |

**frontend/src/app/import/page.tsx**

|  |
| --- |
| 'use client'  import FileUploader from "@/components/FileUploader";  export default function ImportPage() {  return (  <div className="container mx-auto py-10">  <main className="mt-10">  <div className="bg-white p-8">  <h1 className="text-2xl font-bold mb-6">Willkommen beim Kundenimport-Wizard</h1>  <p className="mb-8">Mit diesem Assistenten können Sie Ihre Kundendaten schnell und einfach in das  OVWEB übertragen.</p>   <ol className="list-none mb-8 pl-0">  {["<strong>Datei hochladen:</strong> Wählen Sie Ihre .xls-Importdatei aus und laden Sie sie hoch",  "<strong>Validierung:</strong> Der Wizard überprüft das Format und die Konsistenz Ihrer Daten.",  "<strong>Importvorgang:</strong> Nach erfolgreicher Validierung werden Ihre Daten in das OVWEB importiert.",  "<strong>Abschluss:</strong> Sie erhalten eine Zusammenfassung der importierten Daten."].map((item, index) => (  <li key={index} className="flex items-center mb-4 pt-4 border-t-2 border-gray-200">  <span className="text-3xl font-light mr-4">{index + 1}</span>  <div className="text-base" dangerouslySetInnerHTML={{\_\_html: item}}></div>  </li>  ))}  </ol>  </div>   <FileUploader />   </main>  </div>  ) } |

**frontend/src/components/header.tsx**

**(Nur Ausschnitt)**

|  |
| --- |
| export function Header() {  (...)  <NavigationMenuItem className="relative">  <Link  href="/import"  legacyBehavior  passHref  >  <NavigationMenuLink  className={cn(  *navigationMenuTriggerStyle*(),  'relative'  )}  >  IMPORT  <div className="h-1 empty:bg-[#f2644a] w-full absolute bottom-0 left-0"></div>  </NavigationMenuLink>  </Link> </NavigationMenuItem>  (...) |

**frontend/src/components/ErrorPage.tsx**

|  |
| --- |
| import React from 'react'; import { FontAwesomeIcon } from "@fortawesome/react-fontawesome"; import { *faQuestion* } from "@fortawesome/free-solid-svg-icons";  interface ErrorPageProps {  onClick: () => void; }  const ErrorPage: React.FC<ErrorPageProps> = ({ onClick }) => {  return (  <div className="flex items-center justify-center bg-gray-100 w-full">  <div className="w-full p-6">  <div className="bg-white p-6 rounded-lg shadow-lg mx-auto max-w-4xl">  <div className="flex flex-col items-center">  <FontAwesomeIcon icon={*faQuestion*} size="2xl" style={{color: "#C5281C"}}/>  <p className="text-lg text-gray-800 mb-2">Ein Fehler ist aufgetreten...</p>  <p className="text-gray-600 mb-4">Bitte nehmen Sie kontakt mit dem Generali-Support-Team  auf.</p>  <button  onClick={onClick}  className="mt-2 bg-red-500 text-white py-2 px-4 rounded hover:bg-red-700 transition duration-300 ease-in-out"  >  Zurück  </button>  </div>  </div>  </div>  </div>  ); };  export default ErrorPage; |

**frontend/src/components/ErrorPage.stories.tsx**

|  |
| --- |
| import type { Meta, StoryObj } from '@storybook/react'; import ErrorPage from "@/components/ErrorPage";   const *meta*: Meta<typeof ErrorPage> = {  title: 'Components/ErrorPage',  component: ErrorPage, };  export default *meta*;  export const *Default*: StoryObj<typeof ErrorPage> = { };  export const *ErrorState*: StoryObj<typeof ErrorPage> = { };  export const *SuccessState*: StoryObj<typeof ErrorPage> = { }; |

**frontend/src/components/FileInput.tsx**

|  |
| --- |
| import React, { ChangeEvent } from 'react';  interface FileInputProps {  onChange: (e: ChangeEvent<HTMLInputElement>) => void; }  const FileInput: React.FC<FileInputProps> = ({ onChange }) => {  return (  <>  <input  type="file"  className="hidden"  id="file-upload"  onChange={onChange}  accept=".xls,.xlsx"  />  <label  htmlFor="file-upload"  className="flex flex-col items-center justify-center cursor-pointer"  >  <span className="text-gray-500 font-semibold">Klicken Sie um Ihre exportierte Excel-Kundendaten-Datei auszuwählen</span>  <button  className="mt-4 px-4 py-2 bg-red-500 text-white rounded hover:bg-red-600"  onClick={() => *document*.getElementById('file-upload')?.click()}  >  Datei auswählen (.xls)  </button>  </label>  </>  ); };  export default FileInput; |

**frontend/src/components/FileInput.stories.tsx**

|  |
| --- |
| import type { Meta, StoryObj } from '@storybook/react'; import FileUploader from "@/components/FileUploader";   const *meta*: Meta<typeof FileUploader> = {  title: 'Components/ErrorPage',  component: FileUploader, };  export default *meta*;  export const *Default*: StoryObj<typeof FileUploader> = { };  export const *ErrorState*: StoryObj<typeof FileUploader> = { };  export const *SuccessState*: StoryObj<typeof FileUploader> = { }; |

**frontend/src/components/FileUploader.tsx**

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'; import { ImportResult} from "@it-apprentices/ovweb"; import SummaryComponent from "@/components/Summary"; import ErrorPage from "@/components/ErrorPage"; import FileInput from "@/components/FileInput"; import LoadingAnimation from "@/components/LoadingAnimation"; import {uploadFile} from '@/actions'   const FileUploader: React.FC = () => {  const [error, setError] = useState(false);  const [isLoading, setIsLoading] = useState(false);  const [importResult, setImportResult] = useState<ImportResult | null>(null);    const handleFileChange = async (e: ChangeEvent<HTMLInputElement>) => {  const selectedFile = e.target.files?.[0]  if (selectedFile) {  const extension = selectedFile.name.split('.').pop()?.toLowerCase()  if (extension === 'xls' || extension === 'xlsx') {  setIsLoading(true)  try {  const formData = new *FormData*()  formData.append('file', selectedFile)  const response = await uploadFile(formData)  if(response){  setImportResult(response)  setError(false)  }  } catch (error) {  setError(true)  } finally {  setIsLoading(false)  }  } else {  alert('Bitte wählen Sie eine Datei mit der Erweiterung .xls oder .xlsx aus.')  }  }  }   const handleReset = () => {  setError(false);  setIsLoading(false);  setImportResult(null);  };   return (  <div className="border-2 border-dashed border-red-300 p-6 rounded-md mb-6">  {error ? (  <ErrorPage onClick={() => setError(false)} />  ): importResult && importResult.addedCount !== undefined && importResult.updatedCount !== undefined ? (  <SummaryComponent addedCount={importResult.addedCount} updatedCount={importResult.updatedCount} onReset={handleReset} />  ): !isLoading ? (  <FileInput onChange={handleFileChange} />  ) : isLoading ? (  <LoadingAnimation />  ) : (  <LoadingAnimation />  )}  </div>  ); };  export default FileUploader; |

**frontend/src/components/FileUploader.stories.tsx**

|  |
| --- |
| import type { Meta, StoryObj } from '@storybook/react'; import FileUploader from './FileUploader';   const *meta*: Meta<typeof FileUploader> = {  title: 'Components/FileUploader',  component: FileUploader, };  export default *meta*;  export const *Default*: StoryObj<typeof FileUploader> = { };  export const *ErrorState*: StoryObj<typeof FileUploader> = {  args: {  error: true,  }, };  export const *SuccessState*: StoryObj<typeof FileUploader> = {  args: {  importResult: { addedCount: 1, updatedCount: 2 },  }, }; |

**frontend/src/components/LoadingAnimation.tsx**

|  |
| --- |
| import React from 'react'; import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'; import {*faArrowRight*, *faDatabase*, *faFileExcel*} from "@fortawesome/free-solid-svg-icons";  const LoadingAnimation: React.FC = () => {  return (  <div  className="mt-4 flex flex-col justify-center items-center">  <div className="flex justify-center space-x-2">  <FontAwesomeIcon icon={*faFileExcel*} size="4x" style={{color: "#C5281C"}}/>  <FontAwesomeIcon icon={*faArrowRight*} fade size="4x" style={{ color: "#6d6e71", animationDelay: '1s'}}/>  <FontAwesomeIcon icon={*faDatabase*} size="4x" style={{color: "#C5281C"}}/>  </div>  <span  className="text-xl text-red-500 text-center mt-2">Excel wird verarbeitet...</span>  </div>  ); };  export default LoadingAnimation; |

**frontend/src/components/LoadingAnimation.stories.tsx**

|  |
| --- |
| import { Meta, StoryObj } from '@storybook/react'; import LoadingAnimation from './LoadingAnimation';  const *meta*: Meta<typeof LoadingAnimation> = {  title: 'Components/LoadingAnimation',  component: LoadingAnimation, };  export default *meta*;  export const *Default*: StoryObj<typeof LoadingAnimation> = { }; |

**frontend/src/components/Summary.tsx**

|  |
| --- |
| import React from 'react'; import {FontAwesomeIcon} from "@fortawesome/react-fontawesome"; import {*faClipboardList*} from "@fortawesome/free-solid-svg-icons"; import *Link* from "next/link";  interface SummaryProps {  addedCount: number;  updatedCount: number;  onReset: () => void;  }    const SummaryComponent: React.FC<SummaryProps> = ({ addedCount, updatedCount, onReset }) => {  return (  <div className="bg-white p-6 rounded-lg shadow-lg text-center">  <span className="text-white p-2 m-8 rounded-full">  <FontAwesomeIcon icon={*faClipboardList*} size="4x" style={{color: "#C5281C"}}/>  </span>  <div className="flex items-center justify-center border-b-2 pb-4 mt-4">  <h3 className="text-lg font-semibold text-gray-700">Ihr Datenimport wurde erfolgreich  abgeschlossen.</h3>   </div>  <div className="mt-4 mx-auto w-1/2 text-left">  <table className="mx-auto">  <tbody>  <tr className="text-gray-600 border-b">  <td className="py-2">Gesamtanzahl der importierten Datensätze:</td>  <td className="p-2 text-red-700 text-xl">{addedCount + updatedCount}</td>  </tr>  <tr className="text-gray-600">  <td className="py-2">Davon neue Kundeneinträge:</td>  <td className="p-2 text-red-700 text-xl">{addedCount}</td>  </tr>  <tr className="text-gray-600">  <td className="py-2">Aktualisiert:</td>  <td className="p-2 text-red-700 text-xl">{updatedCount}</td>  </tr>  </tbody>  </table>  </div>  <div className="flex justify-between mt-6">  <button  onClick={() => {  onReset();  }}  className="text-white bg-red-500 hover:bg-red-600 font-medium rounded-lg text-sm px-5 py-2.5 mr-4"  >  Nächster Import  </button>  <Link href="/customers" passHref>  <button  className="text-white bg-red-500 hover:bg-red-600 font-medium rounded-lg text-sm px-5 py-2.5"  >  Zur Kundenübersicht  </button>  </Link>  </div>  </div>  ); };  export default SummaryComponent; |

**frontend/src/components/Summary.stories.tsx**

|  |
| --- |
| import type { Meta, StoryObj } from '@storybook/react'; import Summary from './Summary';   const *meta*: Meta<typeof Summary> = {  title: 'Components/Summary',  component: Summary, };  export default *meta*;  export const *Default*: StoryObj<typeof Summary> = {  args: {  addedCount: 1,  updatedCount: 2  }, };  export const *ErrorState*: StoryObj<typeof Summary> = { };  export const *SuccessState*: StoryObj<typeof Summary> = {  args: {  addedCount: 1,  updatedCount: 2  }, }; |

**frontend/src/actions/uploadFile.ts**

|  |
| --- |
| 'use server' import { getServerSession } from 'next-auth' import { *authOptions* } from '@/app/api/auth/[...nextauth]/auth-config' import {  CustomerControllerApi,  Configuration,  FetchAPI,  HTTPHeaders } from '@it-apprentices/ovweb' import {getLogger} from "@/logging/log-util"; import {*trace*, *context*} from '@opentelemetry/api' import {nonNullish} from '@/types/guards' import {Readable} from 'node:stream' import {FormDataEncoder} from 'form-data-encoder'     export async function uploadFile(formData: FormData) {  const tracer = *trace*.getTracer('upload-file-tracer')  const customSpan = tracer.startSpan(  'post import data span',  {},  *context*.active()  )  const currentContext = *trace*.setSpan(*context*.active(), customSpan)  return await *context*.with(  currentContext,  async span => {  const logger = getLogger('file-uploader')  try {  const session = await getServerSession(*authOptions*)  if (!session?.accessToken) {  const logContext = {  session: session,  timestamp: new *Date*().toISOString(),  }  const logger = getLogger('upload-server-action')  logger.error(logContext, 'Keine Sitzung gefunden, der Benutzer muss angemeldet sein, um Dateien hochzuladen.')  return  }   *trace*.setSpan(*context*.active(), customSpan);    const customerApi = new CustomerControllerApi(  new Configuration({  fetchApi: fetchMultipartFix,  basePath: *process*.env.WEB\_API\_FRONTEND\_URL,  accessToken: `Bearer ${session?.accessToken}`  })  )   const response = await customerApi.importCustomers({  file: formData.get('file') as File  })  const logContext = {  customersAdded: response.addedCount,  customersUpdated: response.updatedCount  }  logger.info(logContext, 'Der Upload war erfolgreich')   return response  }  catch (error) {  const file = formData.get('file') as File | null;  const logContext = {  fileName: file ? file.name : 'Dateiname nicht gefunden',  fileSize: file ? file.size : 'Dateigrösse nicht gefunden',  timestamp: new *Date*().toISOString(),  errorMessage: error instanceof *Error* ? error.message : 'Unbekannter Fehler'  }  logger.error(logContext, 'Der Upload hat einen Fehler ausgelöst')  throw error  }  finally {  span.end()  }  },  undefined,  customSpan  ) }  function isFormData(value: unknown): value is FormData {  return typeof *FormData* !== 'undefined' && value instanceof *FormData* }  const fetchMultipartFix: FetchAPI = (  input: RequestInfo | URL,  init?: RequestInit ) => {  if (nonNullish(init) && isFormData(init.body)) {  const encoder = new FormDataEncoder(init.body)  const headers = *Object*.entries(init.headers as HTTPHeaders).filter(  ([k, \_]) =>  !['content-type', 'content-length'].includes(k.toLowerCase())  )  const options = {  ...init,  duplex: 'half',  method: init.method,  headers: {  ...*Object*.fromEntries(headers),  ...encoder.headers  },  body: Readable.*from*(encoder.encode())  }   return fetch(input, options as unknown as RequestInit)  }  return fetch(input, init) } |