C00000



**IPA-Anhang**

**von**

**Samuel Hajnik**

2024

|  |  |
| --- | --- |
|  |  |

Inhalt

[V1\_0\_16\_\_add\_resume\_customer\_workflow\_to\_task.sql 3](#_Toc161827063)

[application.yaml 3](#_Toc161827064)

[Task.java 4](#_Toc161827065)

[TaskGetDto.java 6](#_Toc161827066)

[TaskPostDto.java 8](#_Toc161827067)

[DtoMapper.java 9](#_Toc161827068)

[TaskService.java 10](#_Toc161827069)

[EmailService.java 12](#_Toc161827070)

[TaskController.java 13](#_Toc161827071)

[TaskServiceTest.java 15](#_Toc161827072)

[EmailServiceTest.java 17](#_Toc161827073)

[OVWEBBackendApplication.java 18](#_Toc161827074)

[delete-task.ts 18](#_Toc161827075)

[get-task-details.ts 19](#_Toc161827076)

[update-task.ts 20](#_Toc161827077)

[task-table.tsx 21](#_Toc161827078)

[create-customer/[resumeTaskId]/page.tsx 24](#_Toc161827079)

[create-customer-form.tsx 24](#_Toc161827080)

[create-customer-form.stories.tsx 27](#_Toc161827081)

[create-customer-form.spec.tsx 29](#_Toc161827082)

[task-table.stories.tsx 32](#_Toc161827083)

[task-table.spec.tsx 33](#_Toc161827084)

In diesem Anhang sind nur Codestücke dokumentiert, die selbst geschrieben wurden. Um Codestücke, die nicht selbstgeschrieben worden waren, auszulassen, wird "(...)" verwendet. Falls die Datei selbst erstellt wurde, wird nichts ausgelassen.

# V1\_0\_16\_\_add\_resume\_customer\_workflow\_to\_task.sql

CREATE TABLE IF NOT EXISTS task$resume\_customer\_workflow( id BIGSERIAL PRIMARY KEY,

title VARCHAR(20),  
salutation VARCHAR(10),  
gender VARCHAR(10),  
first\_name VARCHAR(255),  
last\_name VARCHAR(255),  
birthdate DATE,  
phone\_number VARCHAR(255),  
email VARCHAR(255), address\_house\_number VARCHAR(10), address\_street\_name VARCHAR(255),  
address\_plz VARCHAR(255),  
address\_city VARCHAR(255),  
address\_country VARCHAR(255),

constraint fk\_task FOREIGN KEY (id) REFERENCES task (id)  
);  
  
ALTER TABLE task DROP CONSTRAINT task\_state\_check;  
ALTER TABLE task ADD CONSTRAINT task\_state\_check CHECK (state IN ('PENDING', 'COMPLETED', 'CANCELLED', 'RESUMED'));

# application.yaml

(...)

reminder-service:  
 start-days-after-task-creation: ${API\_REMINDER\_START\_DAYS\_AFTER\_TASK\_CREATION}

(...)

# Task.java

(...)

@Setter  
@Getter  
@NoArgsConstructor  
@Entity  
public static class ResumeCustomerWorkflow extends Task {  
  
 @Nullable private String title;  
  
 @Nullable  
 @Enumerated(EnumType.*STRING*)  
 private Salutation salutation;  
  
 @Nullable  
 @Enumerated(EnumType.*STRING*)  
 private Gender gender;  
  
 @Nullable private String firstName;  
  
 @Nullable private String lastName;  
  
 @Nullable private LocalDate birthdate;  
  
 @Nullable private String phoneNumber;  
  
 @Nullable private String email;  
  
 @Nullable private String addressHouseNumber;  
  
 @Nullable private String addressStreetName;  
  
 @Nullable private String addressPlz;  
  
 @Nullable private String addressCity;  
  
 @Nullable private String addressCountry;  
  
 @Builder  
 public ResumeCustomerWorkflow(  
 Long id,  
 String description,  
 LocalDate creationDate,  
 LocalDate dueDate,  
 User owner,  
 TaskState state,  
 String title,  
 Salutation salutation,  
 Gender gender,  
 String firstName,  
 String lastName,  
 LocalDate birthdate,  
 String phoneNumber,  
 String email,  
 String addressHouseNumber,  
 String addressStreetName,  
 String addressPlz,  
 String addressCity,  
 String addressCountry) {  
 super(id, description, creationDate, dueDate, state, owner);  
 this.title = title;  
 this.salutation = salutation;  
 this.gender = gender;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthdate = birthdate;  
 this.phoneNumber = phoneNumber;  
 this.email = email;  
 this.addressHouseNumber = addressHouseNumber;  
 this.addressStreetName = addressStreetName;  
 this.addressPlz = addressPlz;  
 this.addressCity = addressCity;  
 this.addressCountry = addressCountry;  
 }  
}  
  
@Schema(enumAsRef = true)  
public enum TaskState {  
 *PENDING*,  
 *COMPLETED*,  
 *CANCELLED*,  
 *RESUMED*}

# TaskGetDto.java

(...)

@DiscriminatorMapping(  
 value = "ResumeCustomerWorkflowGetDto",  
 schema = TaskGetDto.ResumeCustomerWorkflowGetDto.class)  
},  
oneOf = {  
 TaskGetDto.CustomerRelatedReminderTaskGetDto.class,  
 TaskGetDto.InternalReminderTaskGetDto.class,  
 TaskGetDto.ResumeCustomerWorkflowGetDto.class  
})

}

(...)

)

@JsonSubTypes.Type(  
 value = TaskGetDto.ResumeCustomerWorkflowGetDto.class,  
 name = "ResumeCustomerWorkflowGetDto")  
})

(...)

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
abstract class TaskGet {  
 private Long id;  
 private String description;  
 private LocalDate creationDate;  
 private LocalDate dueDate;  
 private Task.TaskState state;  
 private Long ownerId;  
 private boolean needsReminder;  
}

(...)

@Getter  
final class ResumeCustomerWorkflowGetDto extends TaskGet {  
 private final String title;  
 private final Salutation salutation;  
 private final Gender gender;  
 private final String firstName;  
 private final String lastName;  
 private final LocalDate birthdate;  
 private final String phoneNumber;  
 private final String email;  
 private final String addressHouseNumber;  
 private final String addressStreetName;  
 private final String addressPlz;  
 private final String addressCity;  
 private final String addressCountry;  
  
 @Builder  
 public ResumeCustomerWorkflowGetDto(  
 Long id,  
 String description,  
 LocalDate creationDate,  
 LocalDate dueDate,  
 Long ownerId,  
 Task.TaskState state,  
 String title,  
 Salutation salutation,  
 Gender gender,  
 String firstName,  
 String lastName,  
 LocalDate birthdate,  
 String phoneNumber,  
 String email,  
 String addressHouseNumber,  
 String addressStreetName,  
 String addressPlz,  
 String addressCity,  
 String addressCountry,  
 boolean needsReminder) {  
 super(id, description, creationDate, dueDate, state, ownerId, needsReminder);  
 this.title = title;  
 this.salutation = salutation;  
 this.gender = gender;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthdate = birthdate;  
 this.phoneNumber = phoneNumber;  
 this.email = email;  
 this.addressHouseNumber = addressHouseNumber;  
 this.addressStreetName = addressStreetName;  
 this.addressPlz = addressPlz;  
 this.addressCity = addressCity;  
 this.addressCountry = addressCountry;  
 }  
}

# TaskPostDto.java

(...)

@DiscriminatorMapping(  
 value = "ResumeCustomerWorkflowPostDto",  
 schema = TaskPostDto.ResumeCustomerWorkflowPostDto.class)  
},  
oneOf = {  
 TaskPostDto.CustomerRelatedReminderTaskPostDto.class,  
 TaskPostDto.InternalReminderTaskPostDto.class,  
 TaskPostDto.ResumeCustomerWorkflowPostDto.class  
})

(...)

@JsonSubTypes.Type(  
 value = TaskPostDto.ResumeCustomerWorkflowPostDto.class,  
 name = "ResumeCustomerWorkflowPostDto")  
})

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
abstract sealed class TaskPost  
 permits TaskPostDto.CustomerRelatedReminderTaskPostDto,  
 TaskPostDto.InternalReminderTaskPostDto,  
 TaskPostDto.ResumeCustomerWorkflowPostDto {  
 private String description;  
 private LocalDate dueDate;  
}

(...)

@Getter  
final class ResumeCustomerWorkflowPostDto extends TaskPost {  
 @Nullable private final String title;  
 @Nullable private final Salutation salutation;  
 @Nullable private final Gender gender;  
 @Nullable private final String firstName;  
 @Nullable private final String lastName;  
 @Nullable private final LocalDate birthdate;  
 @Nullable private final String phoneNumber;  
 @Nullable private final String email;  
 @Nullable private final String addressHouseNumber;  
 @Nullable private final String addressStreetName;  
 @Nullable private final String addressPlz;  
 @Nullable private final String addressCity;  
 @Nullable private final String addressCountry;  
  
 @Builder  
 public ResumeCustomerWorkflowPostDto(  
 String description,  
 LocalDate dueDate,  
 String title,  
 Salutation salutation,  
 Gender gender,  
 String firstName,  
 String lastName,  
 LocalDate birthdate,  
 String phoneNumber,  
 String email,  
 String addressHouseNumber,  
 String addressStreetName,  
 String addressPlz,  
 String addressCity,  
 String addressCountry) {  
 super(description, dueDate);  
 this.title = title;  
 this.salutation = salutation;  
 this.gender = gender;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthdate = birthdate;  
 this.phoneNumber = phoneNumber;  
 this.email = email;  
 this.addressHouseNumber = addressHouseNumber;  
 this.addressStreetName = addressStreetName;  
 this.addressPlz = addressPlz;  
 this.addressCity = addressCity;  
 this.addressCountry = addressCountry;  
 }  
}

# DtoMapper.java

(...)

public TaskGetDto.TaskGet mapToGetDto(Task entity, int reminderDays) {  
 TaskGetDto.TaskGet taskGetDto;  
  
 switch (entity) {  
 (...)

case Task.ResumeCustomerWorkflow e ->  
 taskGetDto =  
 TaskGetDto.ResumeCustomerWorkflowGetDto.*builder*()  
 .id(e.getId())  
 .title(e.getTitle())  
 .salutation(e.getSalutation())  
 .gender(e.getGender())  
 .firstName(e.getFirstName())  
 .lastName(e.getLastName())  
 .birthdate(e.getBirthdate())  
 .phoneNumber(e.getPhoneNumber())  
 .email(e.getEmail())  
 .addressHouseNumber(e.getAddressHouseNumber())  
 .addressStreetName(e.getAddressStreetName())  
 .addressPlz(e.getAddressPlz())  
 .addressCity(e.getAddressCity())  
 .addressCountry(e.getAddressCountry())  
 .description(e.getDescription())  
 .creationDate(e.getCreationDate())  
 .dueDate(e.getDueDate())  
 .ownerId(e.getOwner().getId())  
 .state(e.getState())  
 .build();  
  
 default -> throw new IllegalArgumentException("Unknown Task type: " + entity.getClass());  
 }  
 LocalDate reminderDate = entity.getCreationDate().plusDays(reminderDays);  
 taskGetDto.setNeedsReminder(reminderDate.isBefore(LocalDate.*now*()));  
 return taskGetDto;  
}  
  
public Task mapToEntity(TaskPostDto.TaskPost dto) {  
 return switch (dto) {  
 (...)

case TaskPostDto.ResumeCustomerWorkflowPostDto e ->  
 Task.ResumeCustomerWorkflow.*builder*()  
 .description(e.getDescription())  
 .dueDate(e.getDueDate())  
 .title(e.getTitle())  
 .salutation(e.getSalutation())  
 .gender(e.getGender())  
 .firstName(e.getFirstName())  
 .lastName(e.getLastName())  
 .birthdate(e.getBirthdate())  
 .phoneNumber(e.getPhoneNumber())  
 .email(e.getEmail())  
 .addressHouseNumber(e.getAddressHouseNumber())  
 .addressStreetName(e.getAddressStreetName())  
 .addressPlz(e.getAddressPlz())  
 .addressCity(e.getAddressCity())  
 .addressCountry(e.getAddressCountry())  
 .build();  
 };  
}

# TaskService.java

(...)

@NewSpan("updateTask")  
public Task updateTask(Long id, Task updatedTask) {  
 *log*.atInfo().addKeyValue("taskId", id).log("Updating task with id: {}", id);  
 return taskRepository  
 .findById(id)  
 .map(task -> updateTaskDetails(task, updatedTask))  
 .orElseThrow(  
 () -> {  
 *log*.atError().addKeyValue("taskId", id).log("Task not found with id: {}", id);  
 return new IllegalArgumentException("Task not found");  
 });  
}  
  
private Task updateTaskDetails(Task existingTask, Task updatedTask) {  
 if (updatedTask.getDescription() != null) {  
 existingTask.setDescription(updatedTask.getDescription());  
 }  
 if (updatedTask.getDueDate() != null) {  
 existingTask.setDueDate(updatedTask.getDueDate());  
 }  
 existingTask.setState(Task.TaskState.*RESUMED*);  
  
 if (existingTask instanceof Task.ResumeCustomerWorkflow existingWorkflow  
 && updatedTask instanceof Task.ResumeCustomerWorkflow updatedWorkflow) {  
 updateResumeCustomerWorkflowDetails(existingWorkflow, updatedWorkflow);  
 }  
 return taskRepository.save(existingTask);  
}  
  
private void updateResumeCustomerWorkflowDetails(  
 Task.ResumeCustomerWorkflow existingWorkflow, Task.ResumeCustomerWorkflow updatedWorkflow) {  
 if (updatedWorkflow.getTitle() != null) {  
 existingWorkflow.setTitle(updatedWorkflow.getTitle());  
 }  
 if (updatedWorkflow.getSalutation() != null) {  
 existingWorkflow.setSalutation(updatedWorkflow.getSalutation());  
 }  
 if (updatedWorkflow.getGender() != null) {  
 existingWorkflow.setGender(updatedWorkflow.getGender());  
 }  
 if (updatedWorkflow.getFirstName() != null) {  
 existingWorkflow.setFirstName(updatedWorkflow.getFirstName());  
 }  
 if (updatedWorkflow.getLastName() != null) {  
 existingWorkflow.setLastName(updatedWorkflow.getLastName());  
 }  
 if (updatedWorkflow.getBirthdate() != null) {  
 existingWorkflow.setBirthdate(updatedWorkflow.getBirthdate());  
 }  
 if (updatedWorkflow.getPhoneNumber() != null) {  
 existingWorkflow.setPhoneNumber(updatedWorkflow.getPhoneNumber());  
 }  
 if (updatedWorkflow.getEmail() != null) {  
 existingWorkflow.setEmail(updatedWorkflow.getEmail());  
 }  
 if (updatedWorkflow.getAddressHouseNumber() != null) {  
 existingWorkflow.setAddressHouseNumber(updatedWorkflow.getAddressHouseNumber());  
 }  
 if (updatedWorkflow.getAddressStreetName() != null) {  
 existingWorkflow.setAddressStreetName(updatedWorkflow.getAddressStreetName());  
 }  
 if (updatedWorkflow.getAddressPlz() != null) {  
 existingWorkflow.setAddressPlz(updatedWorkflow.getAddressPlz());  
 }  
 if (updatedWorkflow.getAddressCity() != null) {  
 existingWorkflow.setAddressCity(updatedWorkflow.getAddressCity());  
 }  
 if (updatedWorkflow.getAddressCountry() != null) {  
 existingWorkflow.setAddressCountry(updatedWorkflow.getAddressCountry());  
 }  
}

# EmailService.java

package com.generali.ovweb.service;  
  
@Service  
@Slf4j  
public class EmailService {  
  
 @Autowired private TaskService taskService;  
(...)

private final int reminderDays;  
  
 public EmailService(  
 (...)

@Value("${reminder-service.start-days-after-task-creation}") int reminderDays) {  
 (...)

this.reminderDays = reminderDays;  
 }  
  
(...)

@NewSpan("sendReminderEmail")  
 @Scheduled(fixedRate = 360000) // Every Hour  
 public void sendTaskReminderEmails() {  
 List<Task> tasks = taskService.getAllTasksDto();  
 LocalDate currentDate = LocalDate.*now*();  
 for (Task task : tasks) {  
 LocalDate reminderDate = task.getCreationDate().plusDays(reminderDays);  
 if (currentDate.equals(reminderDate)) {  
 String to = task.getOwner().getEmail();  
 String subject = "Task Reminder";  
 String content =  
 "This is a reminder for your task: "  
 + task.getDescription()  
 + ". "  
 + "Here is the link: "  
 + "localhost:3000/tasks";  
 try {  
 sendEmail(to, subject, content);  
 } catch (IOException ex) {  
 *log*.atError()  
 .addKeyValue("to", to)  
 .addKeyValue("subject", subject)  
 .setCause(ex)  
 .log("Failed to send reminder email: {}", ex.getMessage());  
 }  
 }  
 }  
 }  
}

# TaskController.java

package com.generali.ovweb.controller;  
  
@Slf4j  
@RestController  
@RequestMapping("/task")  
public class TaskController {  
  
(...)

public TaskGetDto.TaskGet getTaskById(@PathVariable Long id) {  
 try {  
 (...)  
 .map(task -> DtoMapper.*INSTANCE*.mapToGetDto(task, reminderDays))  
(...)

}  
  
 (...)

public List<TaskGetDto.TaskGet> getAllTasks() {  
 try {  
 *log*.atInfo().log("Getting all tasks");  
 (...)

.map(e -> DtoMapper.*INSTANCE*.mapToGetDto(e, reminderDays))  
 .toList();  
 (...) }  
  
 (...)

(...)

public TaskGetDto.TaskGet createTask(  
 @RequestBody TaskPostDto.TaskPost taskPost, @AuthenticationPrincipal Jwt principal) {  
 (...)  
 var dto = DtoMapper.*INSTANCE*.mapToGetDto(task, reminderDays);  
(...)

}  
  
 @Operation(  
 summary = "Update an existing Task",  
 responses = {  
 @ApiResponse(  
 responseCode = "200",  
 description = "Successfully updated the task",  
 content = {@Content(schema = @Schema(implementation = TaskGetDto.class))})  
 },  
 requestBody =  
 @io.swagger.v3.oas.annotations.parameters.RequestBody(  
 content =  
 @Content(  
 schema = @Schema(implementation = TaskPostDto.class),  
 examples = {  
 @ExampleObject(  
 name = "Customer reminder task sample",  
 summary = "Customer reminder task, mind the discriminator field",  
 value =  
 "{\"type\": \"CustomerRelatedReminderTaskPostDto\","  
 + "\"description\": \"Sample customer reminder\","  
 + "\"dueDate\": \"2024-03-03\","  
 + "\"relatedCustomerId\": 4"  
 + "}"),  
 @ExampleObject(  
 name = "Internal reminder task sample",  
 summary = "Internal reminder task, mind the discriminator field",  
 value =  
 "{\"type\": \"InternalReminderTaskPostDto\","  
 + "\"description\": \"Sample internal reminder\","  
 + "\"dueDate\": \"2024-03-03\","  
 + "\"relatedDepartment\": \"Underwriting\""  
 + "}")  
 })))  
 @PutMapping("/{id}")  
 public TaskGetDto.TaskGet updateTask(  
 @PathVariable Long id, @RequestBody TaskPostDto.TaskPost updatedTaskPost) {  
 try {  
 *log*.atInfo().addKeyValue("taskId", id).log("Received request to update task with id: {}", id);  
 var updatedTask = taskService.updateTask(id, DtoMapper.*INSTANCE*.mapToEntity(updatedTaskPost));  
 var dto = DtoMapper.*INSTANCE*.mapToGetDto(updatedTask, reminderDays);  
 *log*.atInfo()  
 .addKeyValue("taskId", updatedTask.getId())  
 .log("Updated task with id: {}", updatedTask.getId());  
 return dto;  
 } catch (Exception e) {  
 *log*.atError().setCause(e).log("Error while updating task with id: {}", id);  
 throw e;  
 }  
 }  
}

# TaskServiceTest.java

package com.generali.ovweb.service;  
  
import static org.junit.Assert.*assertThrows*;  
import static org.junit.Assert.*assertTrue*;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.Mockito.\*;  
import static org.mockito.Mockito.*times*;  
  
import com.generali.ovweb.model.Task;  
import com.generali.ovweb.model.User;  
import com.generali.ovweb.persistence.TaskRepository;  
import com.generali.ovweb.persistence.UserRepository;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.List;  
import java.util.Optional;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
  
public class TaskServiceTest {  
  
 @InjectMocks private TaskService taskService;  
  
 @Mock private TaskRepository taskRepository;  
  
 @Mock private UserRepository userRepository;  
  
 @BeforeEach  
 public void setup() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 public void testGetTaskById() {  
 Task task = new Task.ResumeCustomerWorkflow();  
 task.setId(1L);  
 *when*(taskRepository.findById(1L)).thenReturn(Optional.*of*(task));  
  
 Optional<Task> returnedTask = taskService.getTaskById(1L);  
  
 *assertEquals*(task, returnedTask.orElse(null));  
 }  
  
 @Test  
 public void testGetAllTasksDto() {  
 Task task1 = new Task.ResumeCustomerWorkflow();  
 task1.setId(1L);  
 Task task2 = new Task.ResumeCustomerWorkflow();  
 task2.setId(2L);  
 *when*(taskRepository.findAll()).thenReturn(Arrays.*asList*(task1, task2));  
  
 List<Task> tasks = taskService.getAllTasksDto();  
  
 *assertEquals*(2, tasks.size());  
 *assertEquals*(task1, tasks.get(0));  
 *assertEquals*(task2, tasks.get(1));  
 }  
  
 @Test  
 public void testCreateTask() {  
 Task task = new Task.ResumeCustomerWorkflow();  
 task.setId(1L);  
 User owner = new User();

*when*(userRepository.findByKeycloakId("ownerId")).thenReturn(Optional.*of*(owner));  
 *when*(taskRepository.save(*any*(Task.class))).thenReturn(task);  
  
 Task createdTask = taskService.createTask(task, "ownerId");  
  
 *assertEquals*(task, createdTask);  
 *assertEquals*(owner, createdTask.getOwner());  
 }  
  
 @Test  
 public void testDeleteTask() {  
 *doNothing*().when(taskRepository).deleteById(1L);  
  
 taskService.deleteTask(1L);  
  
 *verify*(taskRepository, *times*(1)).deleteById(1L);  
 }  
  
 @Test  
 public void testUpdateTask() {  
 Task task = new Task.ResumeCustomerWorkflow();  
 task.setId(1L);  
 *when*(taskRepository.findById(1L)).thenReturn(Optional.*of*(task));  
 *when*(taskRepository.save(*any*(Task.class))).thenReturn(task);  
  
 Task updatedTask = taskService.updateTask(1L, task);  
  
 *assertEquals*(task, updatedTask);  
 }  
  
 @Test  
 public void testGetTaskByIdNotFound() {  
 *when*(taskRepository.findById(1L)).thenReturn(Optional.*empty*());  
  
 Optional<Task> returnedTask = taskService.getTaskById(1L);  
  
 Assertions.*assertTrue*(returnedTask.isEmpty());  
 }  
  
 @Test  
 public void testCreateTaskWithInvalidOwner() {  
 Task task = new Task.ResumeCustomerWorkflow();  
 task.setId(1L);  
 *when*(userRepository.findByKeycloakId("invalidOwnerId")).thenReturn(Optional.*empty*());  
  
 *assertThrows*(  
 IllegalArgumentException.class, () -> taskService.createTask(task, "invalidOwnerId"));  
 }  
  
 @Test  
 public void testDeleteTaskNotFound() {  
 *doThrow*(new IllegalArgumentException()).when(taskRepository).deleteById(1L);  
  
 *assertThrows*(IllegalArgumentException.class, () -> taskService.deleteTask(1L));  
 }  
  
 @Test  
 public void testUpdateTaskNotFound() {  
 Task task = new Task.ResumeCustomerWorkflow();  
 task.setId(1L);  
 *when*(taskRepository.findById(1L)).thenReturn(Optional.*empty*());  
  
 *assertThrows*(IllegalArgumentException.class, () -> taskService.updateTask(1L, task));  
 }  
  
 @Test  
 public void testCreateTaskWithNull() {  
 *assertThrows*(  
 IllegalArgumentException.class, () -> taskService.createTask(null, "ownerId"));  
 }  
  
 @Test  
 public void testGetAllTasksDtoEmpty() {  
 *when*(taskRepository.findAll()).thenReturn(Collections.*emptyList*());  
  
 List<Task> tasks = taskService.getAllTasksDto();  
  
 Assertions.*assertTrue*(tasks.isEmpty());  
 }  
}

# EmailServiceTest.java

package com.generali.ovweb.service;  
  
(...)  
@Slf4j  
@SpringBootTest(classes = EmailServiceTest.EmptyTestContext.class)  
@ActiveProfiles("mailtest")  
public class EmailServiceTest {  
(...)  
 @Value("${reminder-service.start-days-after-task-creation}")  
 int reminderDays;  
  
(...)  
 @Test  
 void testSendTaskReminderEmails() throws IOException, ApiException {  
 TaskService mockTaskService = *mock*(TaskService.class);  
 Task mockTask = *mock*(Task.class);  
 *when*(mockTaskService.getAllTasksDto()).thenReturn(Collections.*singletonList*(mockTask));  
  
*when*(mockTask.getCreationDate()).thenReturn(LocalDate.*now*().minusDays(reminderDays));  
  
 User mockUser = *mock*(User.class);  
 *when*(mockUser.getEmail()).thenReturn(inbox.getEmailAddress());  
 *when*(mockTask.getOwner()).thenReturn(mockUser);  
  
 *when*(mockTask.getDescription()).thenReturn("Test task");  
  
 EmailService emailService = new EmailService(sendGridApiKey, sendGridSenderEmail, reminderDays);  
  
 ReflectionTestUtils.*setField*(emailService, "taskService", mockTaskService);  
  
 emailService.sendTaskReminderEmails();  
  
 var email =  
 waitForControllerApi.waitForLatestEmail(  
 inbox.getId(), 10000L, true, null, null, null, null);  
 *assertEquals*("Task Reminder", email.getSubject());  
 Assertions.*assertTrue*(Objects.*requireNonNull*(email.getBody()).trim().contains("Test task"));  
 }  
  
 @Configuration  
 public static class EmptyTestContext {}  
}

# OVWEBBackendApplication.java

package com.generali.ovweb;  
  
(...).EnableTransactionManagement;  
  
(...)  
@EnableScheduling  
public class OVWEBBackendApplication {  
(...)}

# delete-task.ts

'use server'  
import {ResponseError, TaskControllerApi} from '@it-apprentices/ovweb'  
import {withApi} from '@/lib/with-api'  
import {withSpan} from '@/lib/with-span'  
import {getLogger} from '@/logging/log-util'  
import {nonNullish} from '@/types/guards'  
  
export const deleteTask = async (id: number) =>  
 withSpan('deleteTask', {taskId: id}, async span => {  
 const logger = getLogger('ovweb-frontend')  
 const logContext: Record<string, unknown> = {taskId: id}  
 logger.info(logContext, 'deleting task')  
 try {  
 return withApi(TaskControllerApi).deleteTask({id})  
 } catch (error) {  
 const response = (error as ResponseError).response  
 if (nonNullish(response)) {  
 const responseStatus = response.status  
 const responseStatusText = response.statusText  
 const responseBody = await response  
 .text()  
 .catch(() => 'failed to read response body')  
 span.setAttribute('responseStatus', responseStatus)  
 span.setAttribute('responseStatusText', responseStatusText)  
 span.setAttribute('responseBody', responseBody)  
 logContext.responseStatus = responseStatus  
 logContext.responseStatusText = responseStatusText  
 logContext.responseBody = responseBody  
 }  
 logger.error(logContext, 'deleting task failed')  
 throw error  
 }  
 })

# get-task-details.ts

'use server'  
import {withSpan} from '@/lib/with-span'  
import {getLogger} from '@/logging/log-util'  
import {withApi} from '@/lib/with-api'  
import {ResponseError, TaskControllerApi} from '@it-apprentices/ovweb'  
import {nonNullish} from '@/types/guards'  
  
export const getTaskById = async (id: number) => {  
 return await withSpan('resumeTask', {requestId: id}, async span => {  
 const logger = getLogger('ovweb-frontend')  
 const logContext: Record<string, unknown> = {requestId: id}  
 logger.info(logContext, 'resuming task')  
 try {  
 return await withApi(TaskControllerApi).getTaskById({id})  
 } catch (error) {  
 const response = (error as ResponseError).response  
 if (nonNullish(response)) {  
 const responseStatus = response.status  
 const responseStatusText = response.statusText  
 const responseBody = await response  
 .text()  
 .catch(() => 'failed to read response body')  
 span.setAttribute('responseStatus', responseStatus)  
 span.setAttribute('responseStatusText', responseStatusText)  
 span.setAttribute('responseBody', responseBody)  
 logContext.responseStatus = responseStatus  
 logContext.responseStatusText = responseStatusText  
 logContext.responseBody = responseBody  
 }  
 logger.error(logContext, 'resuming task failed')  
 throw error  
 }  
 })  
}

# update-task.ts

'use server'  
import {  
 ResponseError,  
 TaskControllerApi,  
 TaskPostDto  
} from '@it-apprentices/ovweb'  
import {withApi} from '@/lib/with-api'  
import {withSpan} from '@/lib/with-span'  
import {getLogger} from '@/logging/log-util'  
import {nonNullish} from '@/types/guards'  
  
export const updateTask = async (task: TaskPostDto, id: number) => {  
 return await withSpan('updateTask', {taskId: id}, async span => {  
 const logger = getLogger('ovweb-frontend')  
 const logContext: Record<string, unknown> = {taskId: id, task}  
 logger.info(logContext, 'Updating task')  
 try {  
 const updatedTask = await withApi(TaskControllerApi).updateTask({  
 id,  
 taskPostDto: task  
 })  
 logger.info(logContext, 'Task updated')  
 return updatedTask  
 } catch (error) {  
 const response = (error as ResponseError).response  
 if (nonNullish(response)) {  
 const responseStatus = response.status  
 const responseStatusText = response.statusText  
 const responseBody = await response  
 .text()  
 .catch(() => 'failed to read response body')  
 span.setAttribute('responseStatus', responseStatus)  
 span.setAttribute('responseStatusText', responseStatusText)  
 span.setAttribute('responseBody', responseBody)  
 logContext.responseStatus = responseStatus  
 logContext.responseStatusText = responseStatusText  
 logContext.responseBody = responseBody  
 }  
 logger.error(logContext, 'Updating task failed')  
 throw error  
 }  
 })  
}

# task-table.tsx

(...)  
export function TaskTable<TData extends TaskGetDto, TValue>({  
 columns,  
 data,  
 refreshData  
}: DataTableProps<TData, TValue> & {sessionToken?: string}) {  
 const router = useRouter()  
 const [sorting, setSorting] = React.useState<SortingState>([])  
 const [columnFilters, setColumnFilters] =  
 React.useState<ColumnFiltersState>([])  
  
 const modifiedColumns: ColumnDef<TData, TValue>[] = columns.map(column => {  
 if (column.id === 'delete') {  
 return {  
 ...column,  
 cell: ({row}: {row: {original: TaskGetDto}}) => (  
 <Button  
 data-testid="delete-button"  
 onClick={async () => {  
 const id = row.*original*.id  
 if (nonNullish(id)) {  
 await handleDelete(id, refreshData)  
 }  
 }}  
 className="px-4 py-2 text-white hover:bg-red-700 rounded"  
 >  
 Delete  
 </Button>  
 )  
 }  
 }  
 if (column.id === 'resume') {  
 return {  
 ...column,  
 cell: ({row}: {row: {original: TaskGetDto}}) => (  
 <Button  
 data-testid="resume-button"  
 onClick={async () => {  
 const id = row.*original*.id  
 if (nonNullish(id)) {  
 const taskData = row.*original* if (  
 taskData.type ===  
 'ResumeCustomerWorkflowGetDto' &&  
 taskData.state !== 'RESUMED'  
 ) {  
 const task: ResumeCustomerWorkflowPostDto =  
 {  
 description: taskData.description,  
 dueDate: taskData.dueDate,  
 title: taskData.title,  
 salutation: taskData.salutation,  
 gender: taskData.gender,  
 firstName: taskData.firstName,  
 lastName: taskData.lastName,  
 birthdate: taskData.birthdate,  
 phoneNumber: taskData.phoneNumber,  
 email: taskData.email,  
 addressHouseNumber:  
 taskData.addressHouseNumber,  
 addressStreetName:  
 taskData.addressStreetName,  
 addressPlz: taskData.addressPlz,  
 addressCity: taskData.addressCity,  
 addressCountry:  
 taskData.addressCountry  
 }  
 await handleUpdate(id, task, refreshData)  
 router.push(`create-customer/${id}`)  
 }  
 }  
 }}  
 className="px-4 py-2 text-white hover:bg-red-700 rounded"  
 >  
 Resume  
 </Button>  
 )  
 }  
 }  
 return column  
 }) as ColumnDef<TData, TValue>[]  
  
 const table = useReactTable({  
(...)

}  
 })  
  
 return (  
 <div className="px-2 md:px-0">  
 <div className="flex flex-col md:flex-row items-center justify-between py-4 space-y-2 md:space-y-0">  
 <Input  
 placeholder="Search"  
 value={  
 table  
 .getColumn('description')  
 ?.getFilterValue() as string  
 }  
 onChange={event =>  
 table  
 .getColumn('description')  
 ?.setFilterValue(event.target.value)  
 }  
 className="w-full md:max-w-sm"  
 />  
 </div>  
 <div className="rounded-md border">  
 <Table>  
 <TableHeader>  
 {table.getHeaderGroups().map(headerGroup => (  
 <TableRow key={headerGroup.id}>  
 {headerGroup.headers.map(header => {  
 return (  
 <TableHead key={header.id}>  
 {header.isPlaceholder  
 ? null  
 : flexRender(  
 header.column.columnDef  
 .header,  
 header.getContext()  
 )}  
 </TableHead>  
 )  
 })}  
 </TableRow>  
 ))}  
 </TableHeader>  
 <TableBody>  
 {table.getRowModel().rows.length ? (  
 table.getRowModel().rows.map(row => (  
 <TableRow  
 key={row.id}  
 data-state={  
 row.getIsSelected() && 'selected'  
 }  
 className={  
 row.original.state === 'RESUMED'  
 ? 'bg-gray-200'  
 :row.original.needsReminder===true

? 'bg-red-200'  
 : ''  
 }  
 >  
 {row.getVisibleCells().map(cell => (  
 <TableCell key={cell.id}>  
 {flexRender(  
 cell.column.columnDef.cell,  
 cell.getContext()  
 )}  
 </TableCell>  
 ))}  
 </TableRow>  
 ))  
 ) : (  
(...)

}

# create-customer/[resumeTaskId]/page.tsx

import CreateCustomerForm from '@/components/create-customer-form'  
  
interface PageProps {  
 params: {resumeTaskId: number}  
}  
  
function Page(props: PageProps) {  
 return (  
 <CreateCustomerForm  
 resumeTaskId={props.params.*resumeTaskId*}  
 ></CreateCustomerForm>  
 )  
}  
  
export default Page

# create-customer-form.tsx

'use client'  
import {Button} from '@/components/ui/button'  
(...)  
const stepTitles = ['Kundendaten', 'Kontaktdetails', 'Adresse']  
  
export const *registerSchema* = *z*.object({  
(...)  
})  
  
export type FormModel = *z*.infer<typeof *registerSchema*>  
export const *FieldResolver* = zodResolver(*registerSchema*)  
  
export function CreateCustomerFormWithContext(props: {resumeTaskId?: number}) {  
(...)

useAsyncEffect(async () => {  
 //isValidated?  
 const resumeTaskId = props.*resumeTaskId* if (nonNullish(resumeTaskId)) {  
 const customerData = await getTaskById(resumeTaskId)  
 if (customerData.type === 'ResumeCustomerWorkflowGetDto') {  
 const formData = resumeTaskDtoToInput(customerData)  
 form.reset(formData)  
 }  
 }  
 }, [props.*resumeTaskId*, form.reset])  
  
 const {  
 formState: {errors}  
 } = form  
  
 const saveCustomer = async (data: FormModel) => {  
(...) }  
 }  
  
 try {  
(...) }  
  
 const cancelCustomerWorkflow = async (data: FormModel) => {  
 const taskPostDto: CreateTaskRequest = {  
 taskPostDto: {  
 type: 'ResumeCustomerWorkflowPostDto',  
 description: 'Customer Create Workflow',  
 dueDate: new *Date*(),  
 title: data.title,  
 salutation: data.salutation,  
 gender: data.gender,  
 firstName: data.surname,  
 lastName: data.lastname,  
 birthdate: data.birthDate,  
 phoneNumber: data.phoneNumber,  
 email: data.email,  
 addressHouseNumber: data.houseNumber,  
 addressStreetName: data.street,  
 addressPlz: data.zipcode,  
 addressCity: data.city,  
 addressCountry: 'CH'  
 }  
 }  
 try {  
 await createTask(taskPostDto)  
 toast({  
 title: 'Task wurde erfolgreich gespeichert!',  
 variant: 'success'  
 })  
 router.push('/tasks')  
 } catch (error) {  
 toast({  
 title: 'Der Task konnte aufgrund eines Serverfehlers nicht erstellt werden.',  
 variant: 'error'  
 })  
 }  
 }  
  
(...)  
 async function onCancel(data: FormModel) {  
 await cancelCustomerWorkflow(data)  
 }  
  
 (...) *console*.log('Form state after validation:', form.formState

const titleState = form.getFieldState('title')  
 const salutationState = form.getFieldState('salutation')  
 const genderState = form.getFieldState('gender')  
 const surNameState = form.getFieldState('surname')  
 const lastNameState = form.getFieldState('lastname')  
 const birthDateState = form.getFieldState('birthDate')  
  
 if (titleState.invalid) return  
 if (salutationState.invalid) return  
 if (genderState.invalid) return  
 if (surNameState.invalid) return  
 if (lastNameState.invalid) return  
 if (birthDateState.invalid) return  
  
 setFormStep(1)  
 } else if (formStep === 1) {  
 await form.trigger(['email', 'phoneNumber'])  
 const emailState = form.getFieldState('email')  
 const phoneNumberState = form.getFieldState('phoneNumber')  
  
 if (emailState.invalid) return  
 if (phoneNumberState.invalid) return  
  
 setFormStep(2)  
 }  
 }  
  
 async function handleSubmit() {  
 if (formStep === 2) {  
 // validation  
 await form.trigger(['zipcode', 'city', 'street', 'houseNumber'])  
 const zipcodeState = form.getFieldState('zipcode')  
 const cityState = form.getFieldState('city')  
 const streetState = form.getFieldState('street')  
 const houseNumber = form.getFieldState('houseNumber')  
  
 if (zipcodeState.invalid) return  
 if (cityState.invalid) return  
 if (streetState.invalid) return  
 if (houseNumber.invalid) return  
 }  
 }  
  
 function CustomerDataForm() {  
 return (  
(...)

<Button  
 type="button"  
 variant={'ghost'}  
 onClick={() =>  
 onCancel(form.getValues())  
 }  
 >  
 Später Fortfahren  
 </Button>  
 <Button  
 type="button"  
 variant={'ghost'}  
 className={cn('ml-auto', {  
 hidden: formStep === 2  
 })}  
 onClick={handleNextStep}  
 >  
 Weiter  
 <ArrowRight className="w-4 h-4 ml-2" />  
 </Button>  
 </div>  
 </Form>  
 </CardContent>  
 </div>  
 </Card>  
 </div>  
 </div>  
 )  
}  
  
export const resumeTaskDtoToInput = (  
 resumeTaskDto: ResumeCustomerWorkflowGetDto  
) => {  
 return {  
 title: resumeTaskDto.title,  
 salutation: resumeTaskDto.salutation,  
 gender: resumeTaskDto.gender,  
 surname: resumeTaskDto.firstName,  
 lastname: resumeTaskDto.lastName,  
 birthDate: resumeTaskDto.birthdate,  
 phoneNumber: resumeTaskDto.phoneNumber,  
 email: resumeTaskDto.email,  
 zipcode: resumeTaskDto.addressPlz,  
 city: resumeTaskDto.addressCity,  
 street: resumeTaskDto.addressStreetName,  
 houseNumber: resumeTaskDto.addressHouseNumber  
 }  
}  
  
export default function CreateCustomerForm(props: {resumeTaskId?: number}) {  
 const form = useForm<FormModel>({  
 resolver: FieldResolver,  
 defaultValues: {  
 title: '',  
 salutation: undefined,  
 gender: undefined,  
 surname: '',  
 lastname: '',  
 email: '',  
 phoneNumber: '',  
 zipcode: '',  
 city: '',  
 street: '',  
 houseNumber: ''  
 }  
 })  
  
 return (  
 <FormProvider {...form}>  
 <CreateCustomerFormWithContext {...props} />  
 </FormProvider>  
 )  
}

# create-customer-form.stories.tsx

import type {Meta, StoryObj} from '@storybook/react'  
import {  
 *FieldResolver*,  
 FormModel,  
 CreateCustomerFormWithContext  
} from '@/components/create-customer-form'  
import {*Gender*, *Salutation*} from '@it-apprentices/ovweb'  
import {FormProvider, useForm} from 'react-hook-form'  
import React, {FC} from 'react'  
  
import {useAsyncEffect} from 'use-async-effect'  
import {nonNullable} from 'next/dist/lib/non-nullable'  
  
const CreateCustomerFormWrapper: FC<{  
 initModel?: FormModel  
 validate?: boolean  
 resumeTaskId?: number  
}> = ({initModel, validate = false, resumeTaskId}) => {  
 const form = useForm<FormModel>({  
 resolver: *FieldResolver*,  
 mode: 'onTouched',  
 defaultValues: {  
 title: '',  
 salutation: undefined,  
 gender: undefined,  
 surname: '',  
 lastname: '',  
 email: '',  
 phoneNumber: '',  
 zipcode: '',  
 city: '',  
 street: '',  
 houseNumber: ''  
 }  
 })  
  
 useAsyncEffect(async () => {  
 if (validate) {  
 await form.trigger()  
 }  
 }, [form.trigger])  
 useAsyncEffect(async () => {  
 if (nonNullable(initModel)) {  
 await form.reset(initModel)  
 }  
 }, [form.reset])  
 return (  
 <FormProvider {...form}>  
 <CreateCustomerFormWithContext resumeTaskId={resumeTaskId} />  
 </FormProvider>  
 )  
}  
  
const *meta* = {  
 title: 'Components/CreateCustomerForm',  
 component: CreateCustomerFormWrapper  
} satisfies Meta<typeof CreateCustomerFormWrapper>  
  
export default *meta*type Story = StoryObj<typeof *meta*>  
  
export const *Empty*: Story = {  
 args: {},  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 }  
 }  
}  
  
export const *Prefilled*: Story = {  
 args: {  
 resumeTaskId: 1  
 },  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 },  
 actions: {  
 getTaskById: *Promise*.resolve({  
 type: 'ResumeCustomerWorkflowGetDto',  
 id: 1,  
 description: 'Create Customer Workflow',  
 creationDate: new *Date*(),  
 dueDate: new *Date*(),  
 state: 'PENDING',  
 title: 'Test',  
 salutation: *Salutation*.*Mr*,  
 gender: *Gender*.*Male*,  
 firstName: 'John',  
 lastName: 'Doe',  
 birthdate: new *Date*(),  
 phoneNumber: '+41111111111',  
 email: 'john.doe@example.com',  
 addressHouseNumber: '1',  
 addressStreetName: 'Test Street',  
 addressPlz: '8000',  
 addressCity: 'Test City',  
 addressCountry: 'Test Country'  
 })  
 }  
 }  
}  
  
export const *FailedValidation*: Story = {  
 args: {  
 validate: true  
 },  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 }  
 }  
}

# create-customer-form.spec.tsx

import {*fireEvent*, render, *screen*, waitFor} from '@testing-library/react'  
import '@testing-library/jest-dom'  
import CreateCustomerForm from './create-customer-form'  
import {createCustomer, createTask, getTaskById} from '@/actions'  
import {*Gender*, *Salutation*} from '@it-apprentices/ovweb'  
import {*mockData*} from '@/components/task-table.spec'  
  
const mockCustomer = {  
 title: 'Dr.',  
 salutation: *Salutation*.*Mr*,  
 gender: *Gender*.*Male*,  
 firstName: 'John',  
 lastName: 'Doe',  
 birthdate: new *Date*(),  
 phoneNumber: '+41111111111',  
 email: 'john.doe@example.com',  
 address: {  
 country: 'CH',  
 plz: '1234',  
 city: 'Test City',  
 streetName: 'Test Street',  
 houseNumber: '1'  
 }  
}  
  
const mockTask = {  
 title: 'Dr.',  
 salutation: *Salutation*.*Mr*,  
 gender: *Gender*.*Male*,  
 firstName: 'John',  
 lastName: 'Doe',  
 birthdate: new *Date*(),  
 phoneNumber: '+41111111111',  
 email: 'john.doe@example.com',  
 addressHouseNumber: 1  
}  
  
jest.mock('@/actions', () => ({  
 createCustomer: jest.fn(() => *Promise*.resolve(mockCustomer)),  
 createTask: jest.fn(() => *Promise*.resolve(mockTask)),  
 getTaskById: jest.fn(() => *Promise*.resolve())  
}))  
jest.mock('next/navigation', () => ({  
 useRouter: jest.fn(() => ({  
 push: jest.fn()  
 }))  
}))  
  
describe('CreateCustomerForm', () => {  
 it('renders the save button', async () => {  
 render(<CreateCustomerForm />)  
 await waitFor(() => {  
 expect(*screen*.getByText('Kunden erstellen')).toBeInTheDocument()  
 })  
 })  
 it('renders the "Zurück" button when form step is not 0', async () => {  
 render(<CreateCustomerForm />)  
 await waitFor(() => {  
 expect(*screen*.getByText('Zurück')).toBeInTheDocument()  
 })  
 })  
  
 it('always renders the "Später Fortfahren" button', async () => {  
 render(<CreateCustomerForm />)  
 await waitFor(() => {  
 expect(*screen*.getByText('Später Fortfahren')).toBeInTheDocument()  
 })  
 })  
  
 it('renders the "Weiter" button when form step is not 2', async () => {  
 render(<CreateCustomerForm />)  
 await waitFor(() => {  
 expect(*screen*.getByText('Weiter')).toBeInTheDocument()  
 })  
 })  
  
 it('renders form fields correctly for each form step', async () => {  
 render(<CreateCustomerForm />)  
 await waitFor(() => {  
 expect(*screen*.getByPlaceholderText('Dr./Prof.')).toBeInTheDocument()  
 expect(*screen*.getByPlaceholderText('Max')).toBeInTheDocument()  
 expect(  
 *screen*.getByPlaceholderText('Mustermann')  
 ).toBeInTheDocument()  
 expect(  
 *screen*.getByPlaceholderText('+41 78 785 98 76')  
 ).toBeInTheDocument()  
 expect(  
 *screen*.getByPlaceholderText('max.mustermann@generali.ch')  
 ).toBeInTheDocument()  
 expect(*screen*.getByPlaceholderText('Zürich')).toBeInTheDocument()  
 expect(*screen*.getByPlaceholderText('8000')).toBeInTheDocument()  
 expect(  
 *screen*.getByPlaceholderText('Bahnhofstrasse')  
 ).toBeInTheDocument()  
 expect(*screen*.getByPlaceholderText('14')).toBeInTheDocument()  
 })  
 })  
  
 it('changes form step when "Weiter" button is clicked', async () => {  
 render(<CreateCustomerForm />)  
 const weiterButton = await *screen*.findByText('Weiter')  
 *fireEvent*.click(weiterButton)  
 })  
  
 it('calls createCustomer when "Kunden erstellen" button is clicked', async () => {  
 ;(getTaskById as jest.Mock).mockReturnValue(  
 *Promise*.resolve(*mockData*[0])  
 )  
 render(<CreateCustomerForm resumeTaskId={2} />)  
 await waitFor(async () => {  
 const weiterButton = await *screen*.findByText('Weiter')  
 *fireEvent*.click(weiterButton)  
 })  
  
 await waitFor(async () => {  
 const weiterButton = await *screen*.findByText('Weiter')  
 *fireEvent*.click(weiterButton)  
 })  
 await waitFor(async () => {  
 const saveButton = await *screen*.findByText('Kunden erstellen')  
 *fireEvent*.click(saveButton)  
 })  
  
 await waitFor(async () => {  
 expect(createCustomer).toHaveBeenCalled()  
 })  
 })  
  
 it('calls createTask when "Später Fortfahren" button is clicked', async () => {  
 render(<CreateCustomerForm />)  
 const laterButton = await *screen*.findByText('Später Fortfahren')  
 *fireEvent*.click(laterButton)  
 expect(createTask).toHaveBeenCalled()  
 })  
})

# task-table.stories.tsx

import type {Meta, StoryObj} from '@storybook/react'  
import {TaskTable} from '@/components/task-table'  
import {TaskGetDto} from '@it-apprentices/ovweb'  
import {*columns*} from '@/app/tasks/columns'  
  
const *meta* = {  
 title: 'Components/TaskTable',  
 component: TaskTable  
} satisfies Meta<typeof TaskTable>  
  
export default *meta*type Story = StoryObj<typeof *meta*>  
  
const mockData: TaskGetDto[] = [  
 {  
 id: 1,  
 description: 'Test Task',  
 dueDate: new *Date*(),  
 title: 'Test',  
 salutation: 'MR',  
 gender: 'MALE',  
 firstName: 'John',  
 lastName: 'Doe',  
 birthdate: new *Date*(),  
 phoneNumber: '1234567890',  
 email: 'john.doe@example.com',  
 addressHouseNumber: '1',  
 addressStreetName: 'Test Street',  
 addressPlz: '12345',  
 addressCity: 'Test City',  
 addressCountry: 'Test Country',  
 type: 'ResumeCustomerWorkflowGetDto'  
 }  
]  
  
export const *WithTasks*: Story = {  
 args: {  
 columns: *columns*,  
 data: mockData,  
 refreshData: () => *Promise*.resolve()  
 },  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 }  
 }  
}  
  
export const *WithoutTasks*: Story = {  
 args: {  
 columns: *columns*,  
 data: [],  
 refreshData: () => *Promise*.resolve()  
 },  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 }  
 }  
}  
  
export const *ErrorOnGet*: Story = {  
 args: {  
 columns: *columns*,  
 data: mockData,  
 refreshData: () =>  
 *Promise*.reject(new *Error*('Response returned an error code'))  
 },  
 parameters: {  
 nextjs: {  
 appDirectory: true  
 }  
 }  
}

# task-table.spec.tsx

import {render, *screen*, waitFor, *fireEvent*} from '@testing-library/react'  
import '@testing-library/jest-dom'  
import {TaskTable} from '@/components/task-table'  
import {*columns*} from '@/app/tasks/columns'  
import {deleteTask, updateTask} from '@/actions'  
import {*Gender*, *Salutation*} from '@it-apprentices/ovweb'  
  
export const *mockData* = [  
 {  
 type: 'ResumeCustomerWorkflowGetDto',  
 id: 1,  
 description: 'Create Customer Workflow',  
 creationDate: new *Date*(),  
 dueDate: new *Date*(),  
 state: 'PENDING',  
 title: 'Test',  
 salutation: *Salutation*.*Mr*,  
 gender: *Gender*.*Male*,  
 firstName: 'John',  
 lastName: 'Doe',  
 birthdate: new *Date*(),  
 phoneNumber: '+41111111111',  
 email: 'john.doe@example.com',  
 addressHouseNumber: '1',  
 addressStreetName: 'Test Street',  
 addressPlz: '8000',  
 addressCity: 'Test City',  
 addressCountry: 'Test Country'  
 } as const  
]  
  
const mockDataUpdate = [  
 {  
 type: 'ResumeCustomerWorkflowGetDto',  
 id: 1,  
 description: 'Create Customer Workflow',  
 creationDate: new *Date*(),  
 dueDate: new *Date*(),  
 title: 'Test',  
 salutation: *Salutation*.*Mr*,  
 gender: *Gender*.*Male*,  
 firstName: 'Tommy'  
 }  
]  
  
jest.mock('@/actions', () => ({  
 deleteTask: jest.fn(() => *Promise*.resolve()),  
 updateTask: jest.fn(() => *Promise*.resolve(mockDataUpdate))  
}))  
jest.mock('next/navigation', () => ({  
 useRouter: jest.fn(() => ({  
 push: jest.fn()  
 }))  
}))  
  
describe('TaskTable', () => {  
 it('renders the table', async () => {  
 render(  
 <TaskTable  
 columns={*columns*}  
 data={*mockData*}  
 refreshData={jest.fn()}  
 />  
 )  
 await waitFor(() => {  
 expect(  
 *screen*.getByText('Create Customer Workflow', {exact: false})  
 ).toBeInTheDocument()  
 })  
 })  
  
 it('calls handleDelete when "Delete" button is clicked', async () => {  
 render(  
 <TaskTable  
 columns={*columns*}  
 data={*mockData*}  
 refreshData={jest.fn()}  
 />  
 )  
 await waitFor(async () => {  
 const deleteButton = *screen*.getByTestId('delete-button')  
 *fireEvent*.click(deleteButton)  
 })  
 await waitFor(() => {  
 expect(deleteTask).toHaveBeenCalled()  
 })  
 })  
  
 it('calls handleUpdate when "Resume" button is clicked', async () => {  
 render(  
 <TaskTable  
 columns={*columns*}  
 data={*mockData*}  
 refreshData={jest.fn()}  
 />  
 )  
  
 await waitFor(async () => {  
 const resumeButton = *screen*.getByTestId('resume-button')  
 *fireEvent*.click(resumeButton)  
 })  
 await waitFor(() => {  
 expect(updateTask).toHaveBeenCalled()  
 })  
 })  
})