10주차

1. 한학기 스케줄

- 기능별 스케줄 대비 진행 현황표

기능	예정일정	진행현황(프론트)	진행현황(백엔드)	비고
로그인/회원가입	4/15~4/21	완료	완료	- 메일 인증
팀 스페이스 CRUD, 나가기	4/22~4/28	완료	완료	- 삭제, 나가기는 생성자만 가능
팀 스페이스 초대 토큰 발급/수락	4/29~5/5	완료	완료	- 메일로 초대 링크 전송
카테고리 CRUD	4/22~4/28	완료	완료	
투두리스트 CRUD&담당자 지정	4/29~5/5	**진행 중**	완료	- 담당자 지정 기능 추가
파일 업로드/검증	4/29~5/5	**진행 예정**	완료	- 유효성 검사 기능 추가
파일 자동 변환	5/6~5/12	진행 예정	완료	- 변환 상태 확인 API 추가
마이페이지	5/13~5/19	진행 예정	완료	
알림 시스템	5/13~5/19	진행 예정	진행 예정	

API 명세서(백엔드 진행 완료된 것들)

페이지	Aa 기능	메서드	API Path
메인페이지	팀 스페이스 목록 조회(옆에 동그랗게,, 이미지랑, 팀명)	GET	/teams/users/{userId}/teams
	팀스페이스 별 이번주 투두 조회(내가 담당인 것만)	GET	/todos/me/weekly?userId=?

페이지	Aa 기능	메서드	API Path
팀스페이스	팀 스페이스 목록별 투두 조회(전체)	GET	/todos/team/{teamId}/by-category
~~팀원 아미지도 조화 되게끔 수정~~	팀원 조회	GET	/teams/{teamId}/members
	팀 스페이스 목록별 투두 조회(하나)	GET	/todos/team/{teamId}/category/{catId}
	팀 스페이스 내 이번주 할일 목록 조회	GET	/todos/team/5/weekly
마이페이지	이름 변경	PUT	/users/me/name?userId=?
이전 비밀번호랑 같을 경우 비밀번호 변경 안되게끔 설정	비밀번호 변경	POST	/auth/reset-password
	프로필 사진 변경	POST	/users/me/image?userId=2
	이메일 변경용 인증 코드 전송	POST	/users/me/change/send-code
	이메일 변경용 코드 검증	POST	/sers/me/change/verify-code
	이메일 변경	PUT	/users/me?userId=2

- 개인별 스케줄 대비 진행 현황표

일정	**강지윤**	**김예빈**	**이규나**	**신화주**
5/6~5/12	~~- 팀원 추가 화면 구성 + 전체 api 연결~~ ~~- 팀스페이스 삭제 / 수정 / 나가기 페이지 페이지 제작 + 기능 구현 및 api 연결~~	- 알림 관련 API - ~~마이페이지 API~~ - ~~[메인페이지] 팀스페이스 별 이번주 투두 조회 (내가 담당인 것만)~~ ~~- [팀 스페이스]~~	~~팀스페이스~~ ~~- 목록 관리하기~~ ~~버튼 기능 넣기~~ ~~- 팀 스페이스 내 목록 추가(팝업)~~ ~~- 목록 api 연결~~ ~~목록~~ ~~- 목록 수정하기~~	~~- Convert API 연동~~ ~~- 상태 저장~~ - 변환 처리 로직 ~~- 제출 완료 처리~~

		~~내 이번주 할일 목록 조회~~ ~~- [팀 스페이스] 내 TODO 조회(목록별~~ ~~)~~ ~~- 팀원 조회 시 이미지 반환되게끔 수정~~ ~~- 팀 스페이스 목록 조회~~	~~(팝업)~~ ~~- 투두리스트 보이기~~ ~~- 목록 내 투두리스트 추가하기(팝업)~~ ~~- 투두리스트 api 연결~~	
5/13 ~ 5/19	- 마이 페이지 화면 구현 - 마이 페이지 개인정보 변경 세부 화면 구현 및 api 연결 - 팀스페이스 프로필 이미지 변경 기능 추가 -팀 초대 수락 시 미회원가입 유저 경우 팀스페이스로 연결되도록 구현 -마우스 오버 시 안내 문구(메인+팀스페이스) -알림 api연	- 알림 관련 API - API 리팩토링	~~API 연결 시 발생한 오류 해결하기~~ 팀스페이스 - default 목록 및 투두리스트가 보이도록 하기 투두리스트 - 투두리스트 마우스 오버 시 수정 및 삭제 팝업 뜨도록 버튼 넣기 - 파일 업로드 및 형식 지정 기능 추가 - 파일 업로드 시 상태 시각화 - 각 할 일 담당자 설정 기능 추가하기	- 파일 삭제 - 변환 처리 로직 - 투두리스트 로직 정리 - 파일 로직 정리 - 제출 결과 저장 - API 리팩토링

- 기존 스케줄에서 수정된 부분
  - 프론트엔드
  - 백엔드

- 알림 관련 API 완성 못하여서 이번주에 진행할 예정 ← 프론트와 연결작업하다보니 API 수정할 것들이 생겨서 수정 먼저하느라 일정이 밀렸음

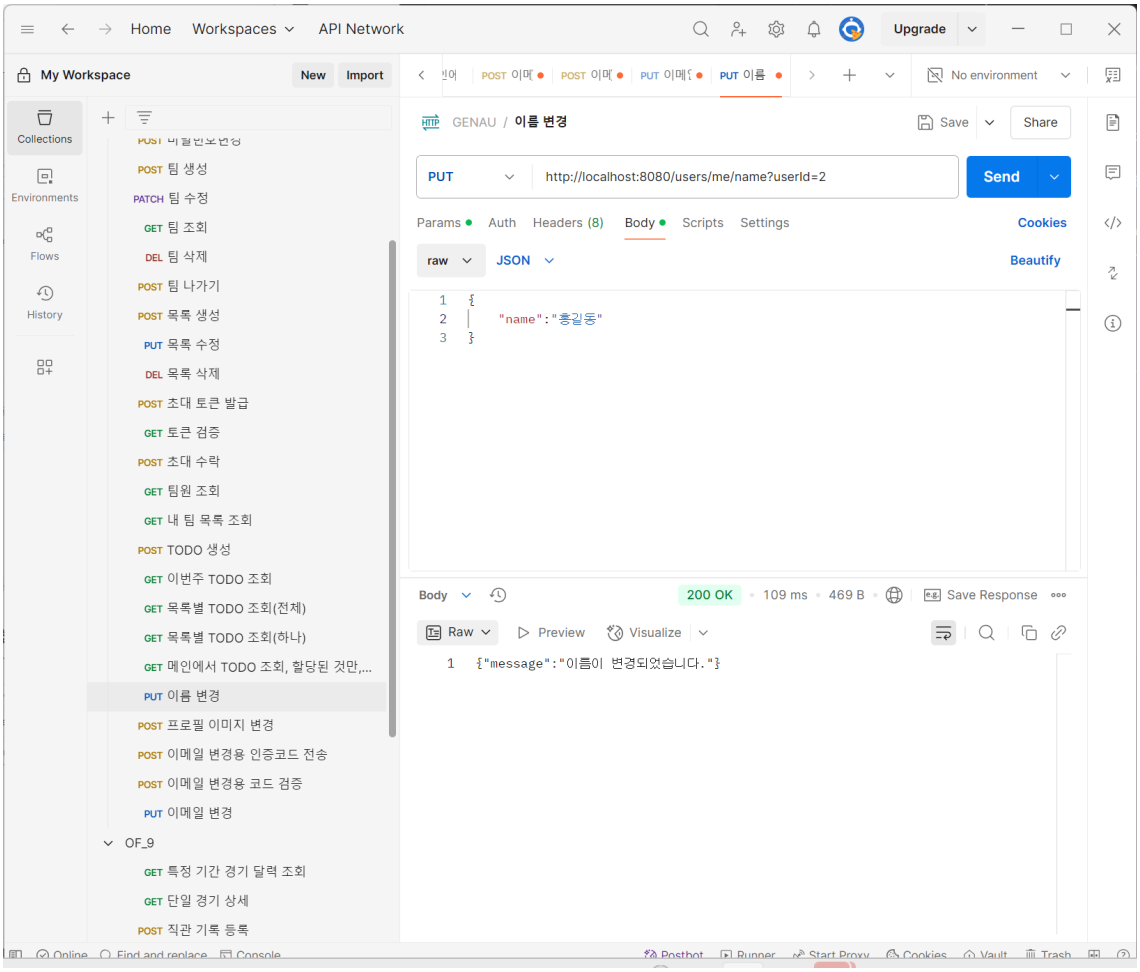
---

2. 지난 한 주간 진행 내용(개인별)

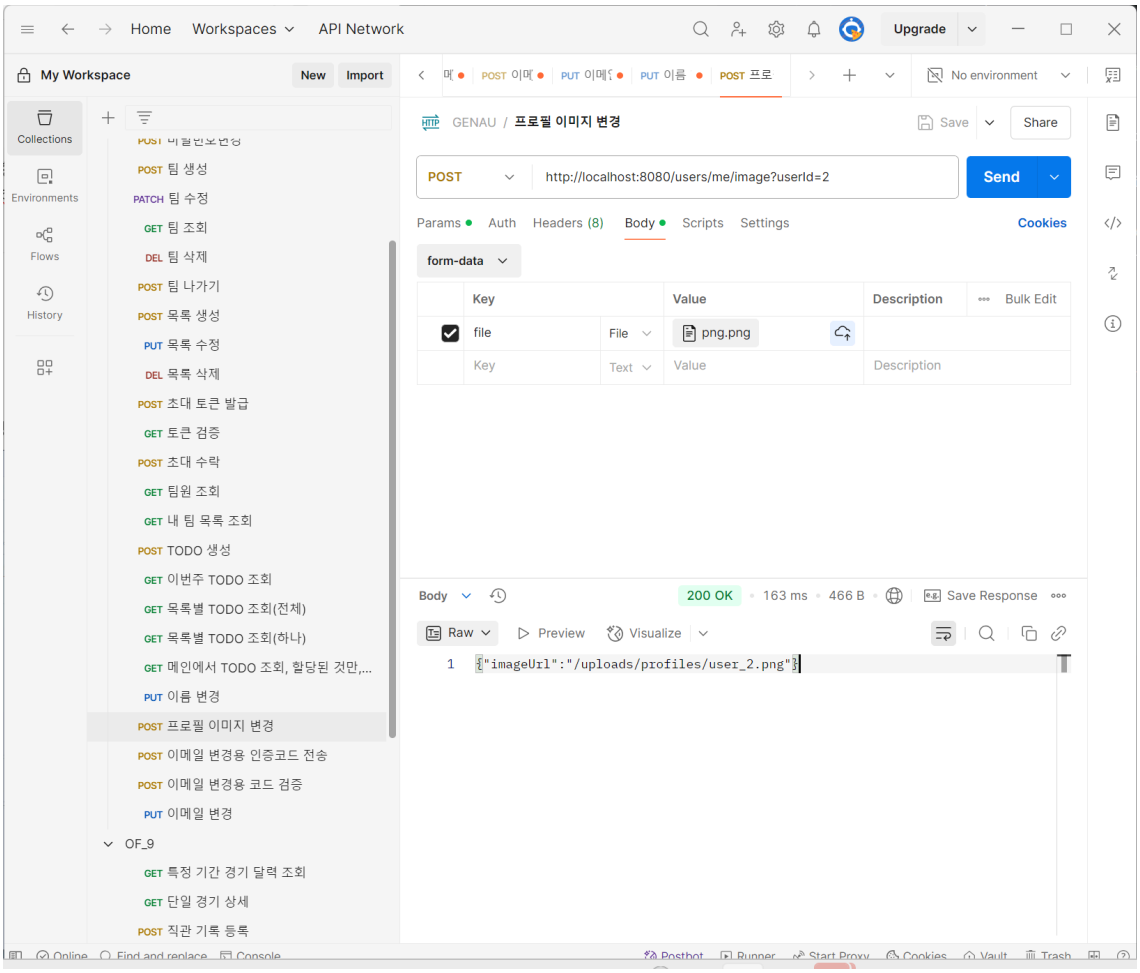
1. 김예빈
    a. 개발 요약

- 마이페이지 API

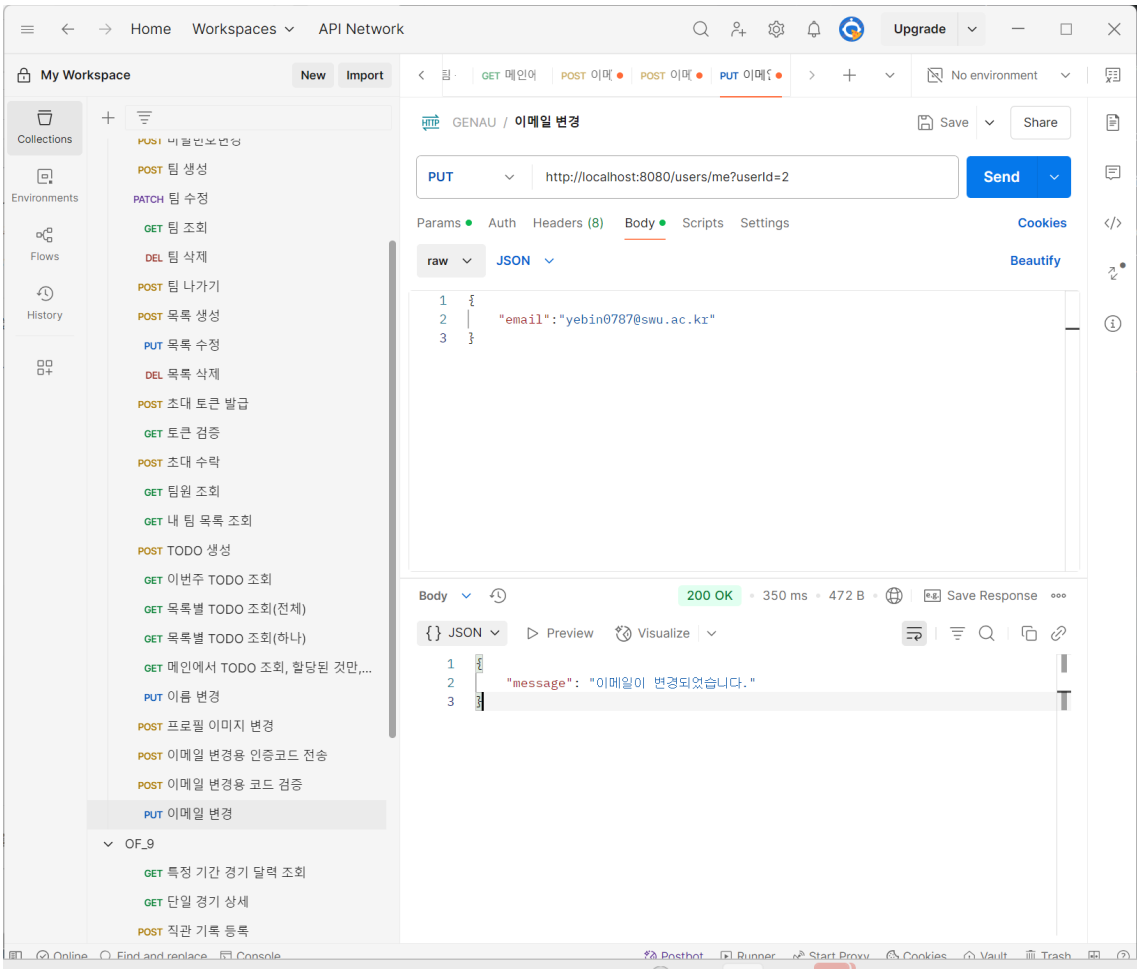
: 이름 변경, 프로필 이미지 변경,비밀번호 변경, 이메일 변경 관련 한 API 설계



이름 변경 api 설계 및 postman으로 검증

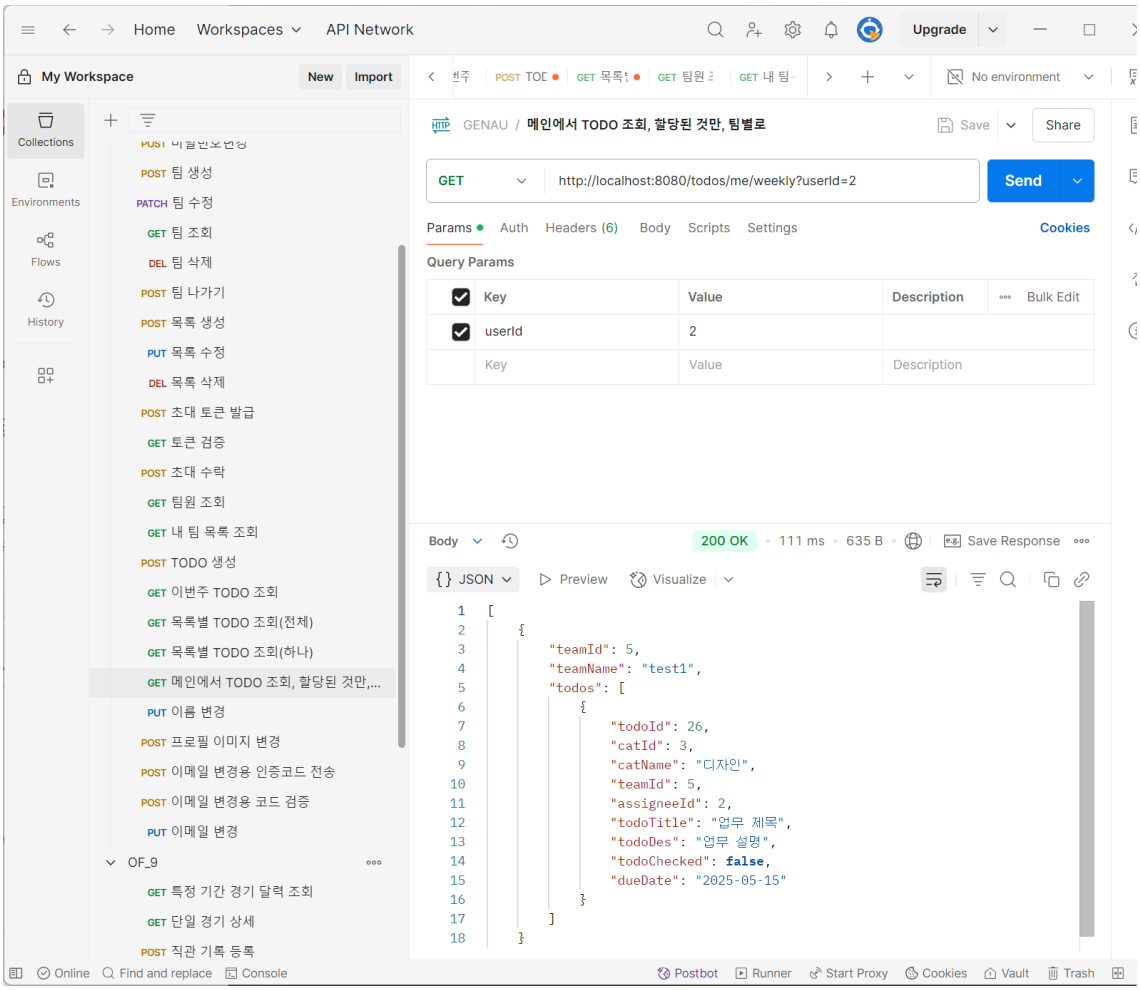


프로필 이미지 변경 api 설계 및 postman으로 검증



이메일 변경 관련 API 설계 및 postman으로 검증

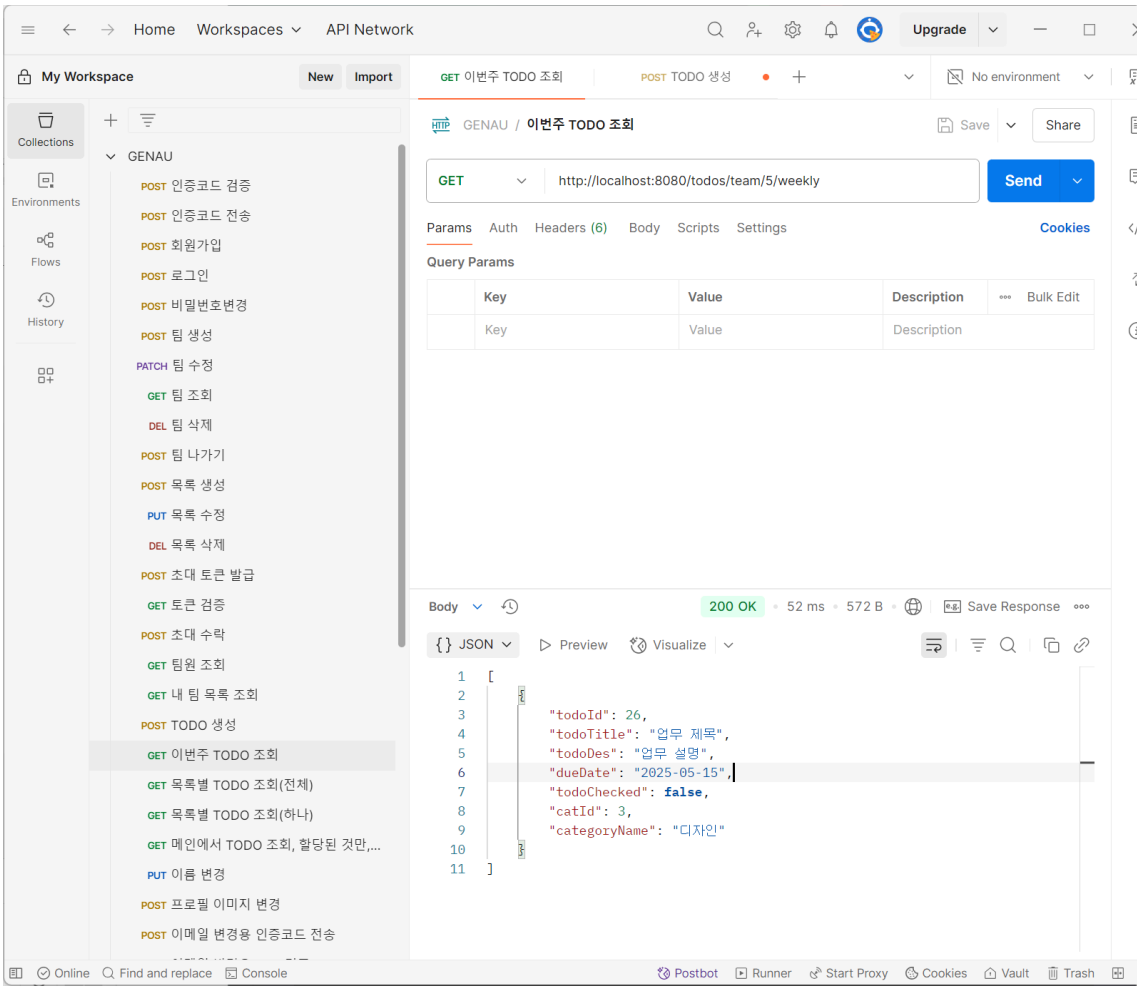
- [메인페이지] 팀스페이스 별 이번주 투두 조회 (내가 담당인 것만)



: 담당자 확인하여 팀 스페이스 별 이번주 나의 할일 목록을 조회할 수 있게끔함

- [팀 스페이스] 이번주 할일 목록 조회

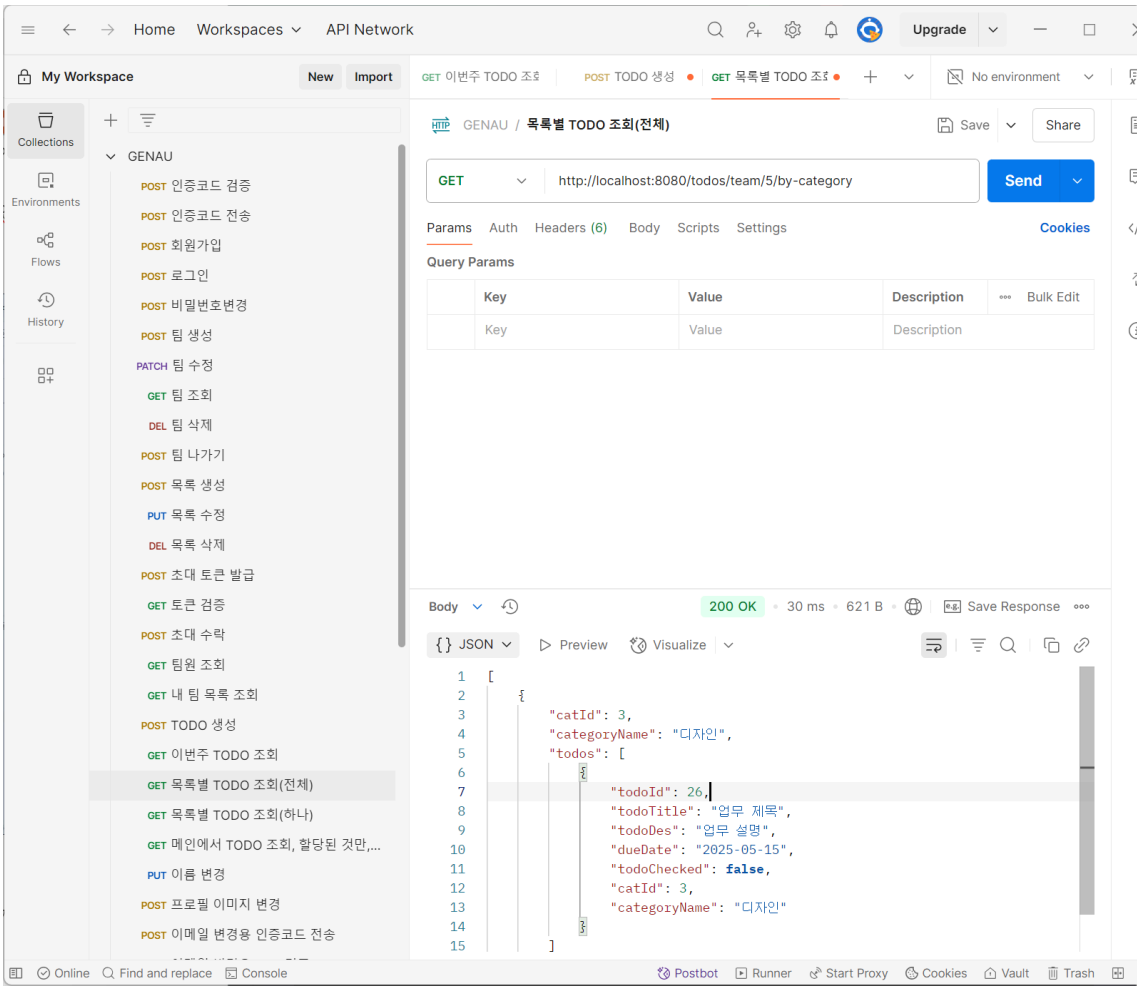
: 카테고리 상관없이 마감기한이 이번주인 할 일 목록을 조회함



-

[팀 스페이스] 내 TODO 조회(목록별)

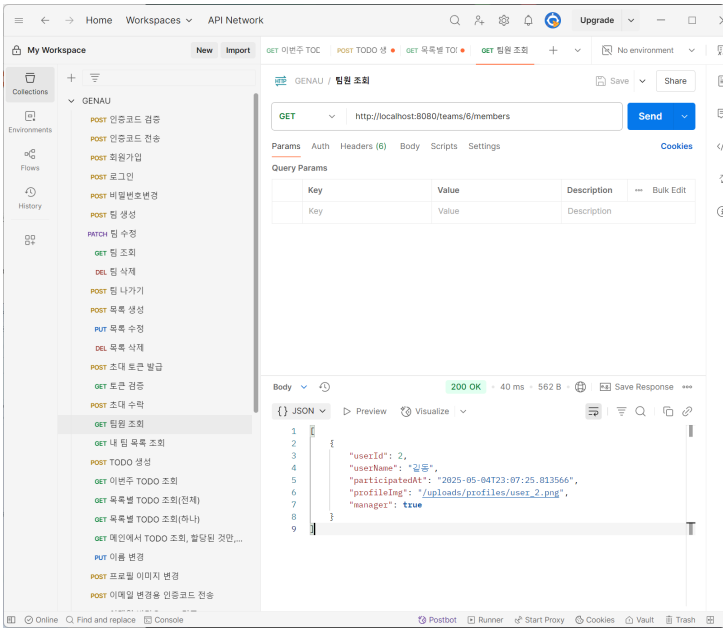
: 목록별로 TODO 조회하게끔 설계(전체 목록, 카테고리 id값넣어주면 개별로 보여주는 등)



-

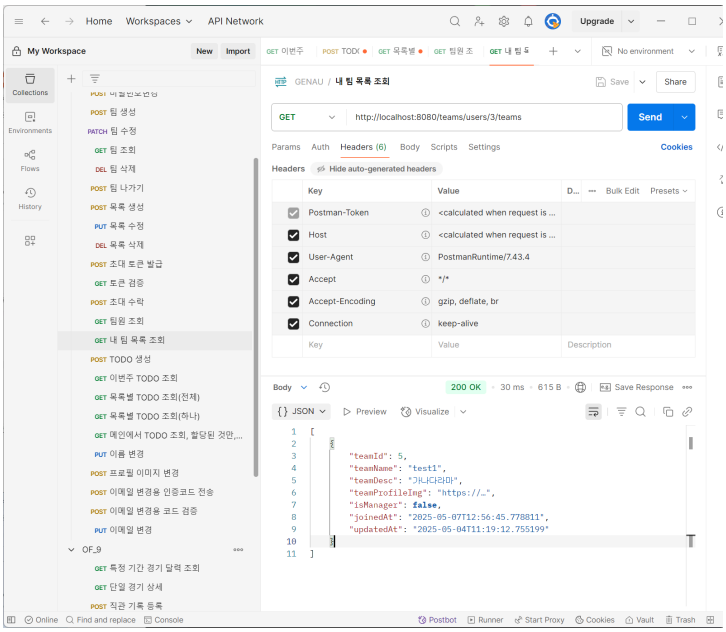
팀원 조회 시 이미지 반환되게끔 수정

: 팀원 조회 시 id값만 반환했었으나, 이미지 값 반환하게끔 수정



- **팀 스페이스 목록 조회**

: 메인페이지에서 팀스페이스 목록 조회 할 수 있게끔 API 설계



2. 이규나

    a. 개발 요약

목록 관리하기 버튼(아코디언 형태)

: 각 목록에 대하여 목록 이름 변경하기, 목록 삭제하기 및 목록 추가하기 기능을 구현함.

목록 추가하기 팝업창 생성

: “목록 추가하기” 클릭 시에 목록을 추가하기 위해 이름을 입력 받는 팝업창을 구현함.

목록 API 연결

: 각 목록이 생성된 팀스페이스 내에서만 보이도록 API 연결 및 수정·보완함.

목록 이름 변경하기 팝업창 생성

: 목록 관리하기 버튼 내 “목록 이름 변경하기” 클릭 시, 목록 이름을 변경하고 DB에서도 해당 부분이 반영될 수 있도록 API 연결 및 기능을 생성함.

목록 추가 시 팀 스페이스 내에 목록을 보이는 사각형(TodoBoard) 렌더링

할 일(투두리스트) API 연결

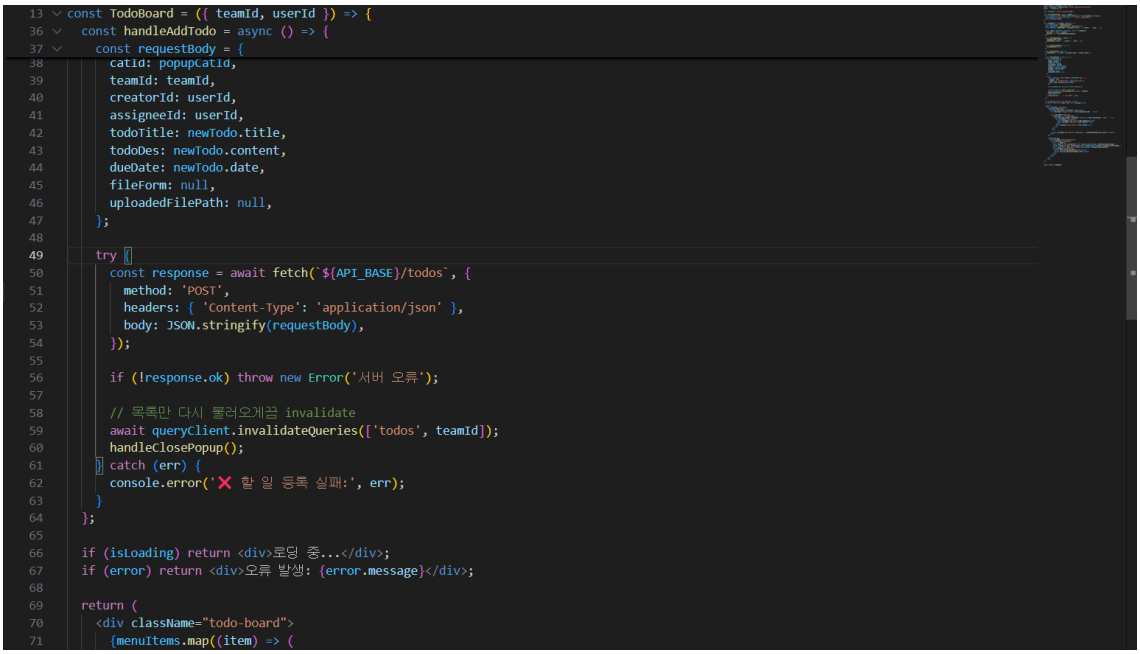
할 일 추가하기 팝업창 생성

: TodoBoard 하단의 + 버튼 클릭 시에 할 일을 추가할 수 있는 팝업창을 구현함

b. 코딩 자료

    a. TodoBoard.jsx

```jsx
import React, { useState } from 'react';
import { useQuery, useQueryClient } from '@tanstack/react-query';
import './TodoBoard.css';

const API_BASE = 'http://localhost:8080';

const fetchTodosByTeam = async (teamId) => {
  const response = await fetch(`${API_BASE}/todos/team/${teamId}/by-category`);
  if (!response.ok) throw new Error('할 일 목록 불러오기 실패');
  return response.json();
};

const TodoBoard = ({ teamId, userId }) => {
  const queryClient = useQueryClient();
  const [popupCatId, setPopupCatId] = useState(null);
  const [newTodo, setNewTodo] = useState({ title: '', content: '', date: '' });

  const { data: menuItems, isLoading, error } = useQuery({
    queryKey: ['todos', teamId],
    queryFn: () => fetchTodosByTeam(teamId),
  });

  const handleOpenPopup = (catId) => {
    setPopupCatId(catId);
    setNewTodo({ title: '', content: '', date: '' });
  };

  const handleClosePopup = () => {
    setPopupCatId(null);
  };

  const handleChange = (e) => {
    setNewTodo({ ...newTodo, [e.target.name]: e.target.value });
  };

  const handleAddTodo = async () => {
    const requestBody = {
```



```jsx
const TodoBoard = ({ teamId, userId }) => {
  const handleAddTodo = async () => {
    const requestBody = {
      catId: popupCatId,
      teamId: teamId,
      creatorId: userId,
      assigneeId: userId,
      todoTitle: newTodo.title,
      todoDes: newTodo.content,
      dueDate: newTodo.date,
      fileForm: null,
      uploadedFilePath: null,
    };

    try {
      const response = await fetch(`${API_BASE}/todos`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(requestBody),
      });

      if (!response.ok) throw new Error('서버 오류');

      // 목록만 다시 불러오게끔 invalidate
      await queryClient.invalidateQueries(['todos', teamId]);
      handleClosePopup();
    } catch (err) {
      console.error('❌ 할 일 등록 실패:', err);
    }
  };

  if (isLoading) return <div>로딩 중...</div>;
  if (error) return <div>오류 발생: {error.message}</div>;

  return (
    <div className="todo-board">
      {menuItems.map((item) => (
```

```jsx

return (
  <div className="todo-board">
    {menuItems.map((item) => (
      <div key={item.catId} className="category-box">
        <h3 className="category-title">{item.categoryName}의 할 일</h3>

        <div className="todo-list">
          {(item.todos || []).map((todo) => (
            <div key={todo.todoId} className={`todo-card ${todo.todoChecked ? 'done' : ''}`}>
              <div className="todo-header">
                <span className="todo-title">{todo.todoTitle}</span>
                <span className="todo-date">{todo.dueDate}</span>
              </div>
              <div className="todo-content">{todo.todoDes}</div>
            </div>
          ))}
        </div>

        <button className="add-todo-btn" onClick={() => handleOpenPopup(item.catId)}>+</button>
      </div>
    ))}

    {popupCatId && (
      <div className="todo-popup-overlay">
        <div className="todo-popup">
          <h3>할 일 추가</h3>
          <input name="title" placeholder="제목" value={newTodo.title} onChange={handleChange} />
          <textarea name="content" placeholder="내용" value={newTodo.content} onChange={handleChange} />
          <input name="date" type="date" value={newTodo.date} onChange={handleChange} />
          <div className="popup-buttons">
            <button onClick={handleAddTodo}>등록</button>
            <button onClick={handleClosePopup}>취소</button>
          </div>
        </div>
      </div>
```

b. Managebtn.jsx

```jsx
import React, { useState, useEffect } from 'react';
import './Managebtn.css';

const API_BASE = 'http://localhost:8080';

const Managebtn = ({ teamId, menuItems, setMenuItems }) => {
  const [isOpen, setIsOpen] = useState(false);
  const [openIndex, setOpenIndex] = useState(null);
  const [isPopupOpen, setIsPopupOpen] = useState(false);
  const [isDeleteConfirmOpen, setIsDeleteConfirmOpen] = useState(false);
  const [isRenameOpen, setIsRenameOpen] = useState(false);
  const [newListName, setNewListName] = useState('');
  const [listToDelete, setListToDelete] = useState(null);
  const [listToRename, setListToRename] = useState(null);

  useEffect(() => {
    if (!teamId) return;
    fetchCategories();
  }, [teamId]);

  const fetchCategories = () => {
    fetch(`${API_BASE}/teams/${teamId}/categories`)
      .then((res) => res.json())
      .then((data) => setMenuItems(data))
      .catch((err) => console.error('❌ 목록 불러오기 실패:', err));
  };

  const handleAddList = () => {
    if (!newListName.trim()) return;

    fetch(`${API_BASE}/teams/${teamId}/categories`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ catName: newListName }),
    })
      .then((res) => res.json())
      .then(() => {
```

```jsx
      })
      .then((res) => res.json())
      .then(() => {
        setNewListName('');
        setIsPopupOpen(false);
        fetchCategories();
      })
      .catch((err) => console.error('목록 추가 실패:', err));
  };

  const handleDeleteList = () => {
    fetch(`${API_BASE}/teams/${teamId}/categories/${listToDelete.catId}`, {
      method: 'DELETE',
    })
      .then(() => {
        setIsDeleteConfirmOpen(false);
        setListToDelete(null);
        fetchCategories();
      })
      .catch((err) => console.error('삭제 실패:', err));
  };

  const handleRenameList = () => {
    if (!newListName.trim()) return;

    fetch(`${API_BASE}/categories/${listToRename.catId}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ catName: newListName }),
    })
      .then(() => {
        setIsRenameOpen(false);
        setNewListName('');
        setListToRename(null);
```

```jsx
return (
  <>
    {isOpen && <div className="accordion-overlay" onClick={() => setIsOpen(false)} />}

    <div className={`accordion-container ${isOpen ? 'show' : ''}`} onClick={(e) => e.stopPropagation()}>
      {isOpen && (
        <>
          {Array.isArray(menuItems) && menuItems.map((item, index) => (
            <div className="accordion-item" key={item.catId}>
              <div
                className="accordion-title"
                onClick={() => setOpenIndex(openIndex === index ? null : index)}
              >
                {openIndex === index ? '▼' : '▶'} {item.catName}
              </div>
              {openIndex === index && (
                <div className="accordion-content">
                  <div className="accordion-link" onClick={() => {
                    setListToRename(item);
                    setIsRenameOpen(true);
                    setNewListName(item.catName);
                  }}>
                    목록 이름 변경하기
                  </div>
                  <div className="accordion-link delete" onClick={() => {
                    setListToDelete(item);
                    setIsDeleteConfirmOpen(true);
                  }}>
                    목록 삭제하기
                  </div>
                </div>
              )}
            </div>
          ))}
          <div className="accordion-add" onClick={() => setIsPopupOpen(true)}>
            + 목록 추가하기
          </div>
        </>
      )}
    </div>

    {isPopupOpen && (
      <div className="popup-overlay">
        <div className="popup-container">
          <h3>목록 추가</h3>
          <input
            type="text"
            value={newListName}
            onChange={(e) => setNewListName(e.target.value)}
            placeholder="목록 이름을 입력하세요"
          />
          <div className="popup-buttons">
            <button onClick={handleAddList}>추가</button>
            <button onClick={() => setIsPopupOpen(false)}>취소</button>
          </div>
        </div>
      </div>
    )}

    {isDeleteConfirmOpen && (
      <div className="popup-overlay">
        <div className="popup-container">
          <h3>정말 삭제합니까?</h3>
          <div className="popup-buttons">
            <button onClick={handleDeleteList}>확인</button>
            <button onClick={() => setIsDeleteConfirmOpen(false)}>취소</button>
          </div>
        </div>
      </div>
```

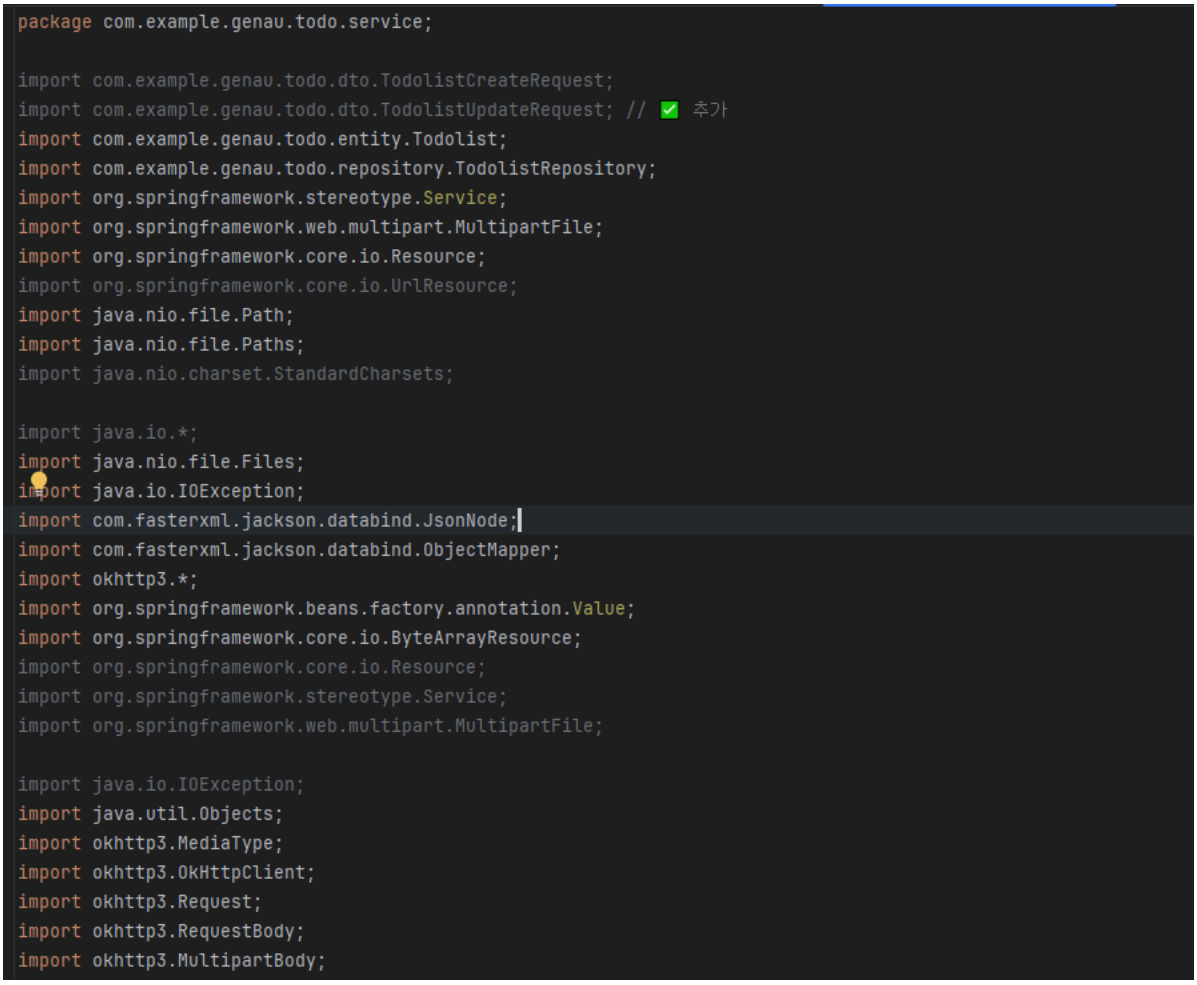
3. 신화주
    a. 개발 요약
        - 파일 변환 외부 API 연동

          : CLOUDCONVERT에서 개인 API키를 받아 파일 변환에 사용

          : 변환된 확장자로 파일 다운로드
        - 변환 상태 저장

          : 파일 변환 후 변환 상태 저장 및 확인 API 생성
        - 파일 제출 완료 처리

          : 파일 제출 로직 및 상태 처리
    b. 코딩 자료



```
package com.example.genau.todo.service;

import com.example.genau.todo.dto.TodolistCreateRequest;
import com.example.genau.todo.dto.TodolistUpdateRequest; // ✅ 추가
import com.example.genau.todo.entity.Todolist;
import com.example.genau.todo.repository.TodolistRepository;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.charset.StandardCharsets;

import java.io.*;
import java.nio.file.Files;
import java.io.IOException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import okhttp3.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.Objects;
import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.MultipartBody;
```

```java
@Service  3개 사용 위치
public class FileConvertService {

    @Value("${cloudconvert.api.key}")
    private String apiKey;

    private static final String API_URL = "https://api.cloudconvert.com/v2";  2개 사용 위치

    private final OkHttpClient httpClient = new OkHttpClient();  4개 사용 위치
    private final ObjectMapper objectMapper = new ObjectMapper();  2개 사용 위치
    private final TodolistRepository todolistRepository;  5개 사용 위치

    public FileConvertService(TodolistRepository todolistRepository) {
        this.todolistRepository = todolistRepository;
    }

    public Resource convertFile(MultipartFile file, String targetFormat, Long todoId) {  1개 사용 위치
        Todolist todo = todolistRepository.findById(todoId)
                .orElseThrow(() -> new IllegalArgumentException("Todo not found: " + todoId));

        try {
            todo.setConvertStatus("WAITING");
            todolistRepository.save(todo);
            // Step 1: Create Job
            String createJobJson = "{\n" +
                    "  \"tasks\": {\n" +
                    "    \"upload-my-file\": {\n" +
                    "      \"operation\": \"import/upload\"\n" +
                    "    },\n" +
                    "    \"convert-my-file\": {\n" +
                    "      \"operation\": \"convert\",\n" +
                    "      \"input\": \"upload-my-file\",\n" +
                    "      \"output_format\": \"" + targetFormat + "\"\n" +
                    "    },\n" +
                    "    \"export-my-file\": {\n" +
```

```
                "    \"export-my-file\": {\n" +
                "      \"operation\": \"export/url\",\n" +
                "      \"input\": \"convert-my-file\",\n" +
                "      \"inline\": true,\n" +
                "      \"archive_multiple_files\": false\n" +
                "    }\n" +
                "  }\n" +
                "}";

        Request jobRequest = new Request.Builder()
                .url(API_URL + "/jobs")
                .addHeader( name: "Authorization",  value: "Bearer " + apiKey)
                .addHeader( name: "Content-Type",  value: "application/json")
                .post(RequestBody.create(createJobJson, MediaType.parse( $this$parse: "application/json")))
                .build();

        Response jobResponse = httpClient.newCall(jobRequest).execute();
        JsonNode jobJson = objectMapper.readTree(Objects.requireNonNull(jobResponse.body()).string());
        String uploadUrl = jobJson.at( jsonPtrExpr: "/data/tasks").get(0).get("result").get("form").get("url").asText();
        String jobId = jobJson.get("data").get("id").asText();
        String taskId = jobJson.at( jsonPtrExpr: "/data/tasks").get(0).get("id").asText();

        // Step 2: Upload file to CloudConvert
        JsonNode form = jobJson.at( jsonPtrExpr: "/data/tasks").get(0).get("result").get("form");
        uploadUrl = form.get("url").asText();
        JsonNode parameters = form.get("parameters");

        MultipartBody.Builder bodyBuilder = new MultipartBody.Builder().setType(MultipartBody.FORM);

// ⚠ form parameters를 모두 추가
        parameters.fields().forEachRemaining( Entry<String, JsonNode> entry -> {
            bodyBuilder.addFormDataPart(entry.getKey(), entry.getValue().asText());
        });
```

```java
public class FileConvertService {
    public Resource convertFile(MultipartFile file, String targetFormat, Long todoId) {  1개 사용 위치
        bodyBuilder.addFormDataPart(
                name: "file",
                file.getOriginalFilename(),
                RequestBody.create(file.getBytes(), MediaType.parse( $this$parse: "application/octet-stream"))
        );


        Request uploadRequest = new Request.Builder()
                .url(uploadUrl)
                .post(bodyBuilder.build())
                .build();

        httpClient.newCall(uploadRequest).execute().close();

        // Step 3: Poll job status and get result URL
        String exportUrl = pollForExportUrl(jobId);

        // Step 4: Download result file
        Request downloadRequest = new Request.Builder().url(exportUrl).build();
        Response fileResponse = httpClient.newCall(downloadRequest).execute();
        byte[] fileBytes = Objects.requireNonNull(fileResponse.body()).bytes();


        // Step 5: Save to Local Disk
        String convertedFileName = todoId + "_converted." + targetFormat;
        Path savePath = Paths.get(System.getProperty("user.dir"),  ...more: "converted", convertedFileName);
        Files.createDirectories(savePath.getParent());
        Files.write(savePath, fileBytes);

        // Step 6: Update DB
        todo.setConvertedFileUrl(savePath.toString());
        todo.setConvertStatus("SUCCESS"); // ✅
        todo.setConvertedAt(java.time.LocalDateTime.now()); // ✅
        todolistRepository.save(todo);
```

```
        return new ByteArrayResource(fileBytes) {
            @Override  2개 사용 위치
            public String getFilename() {
                return convertedFileName;
            }

            @Override
            public long contentLength() {
                return fileBytes.length;
            }
        };

    } catch (IOException e) {
        todo.setConvertStatus("FAILED");
        todolistRepository.save(todo);
        throw new RuntimeException("파일 변환 중 오류가 발생했습니다: " + e.getMessage(), e);
    }
}


private String pollForExportUrl(String jobId) throws IOException {  1개 사용 위치
    int retries = 90;
    while (retries-- > 0) {
        System.out.println("🔄 Checking job status... (remaining retries: " + retries + ")");
        Request checkRequest = new Request.Builder()
                .url(API_URL + "/jobs/" + jobId)
                .addHeader( name: "Authorization",  value: "Bearer " + apiKey)
                .build();
```

```
            Response response = httpClient.newCall(checkRequest).execute();
            String jsonString = Objects.requireNonNull(response.body()).string();
            JsonNode root = objectMapper.readTree(jsonString);
            JsonNode tasks = root.at( jsonPtrExpr: "/data/tasks");

            for (JsonNode task : tasks) {
                String name = task.get("name").asText();
                String status = task.get("status").asText();
                System.out.println("Task name: " + name + ", Status: " + status);

                if ("export-my-file".equals(name) && "finished".equals(status)) {
                    String url = task.get("result").get("files").get(0).get("url").asText();
                    System.out.println("✅ Export file ready at: " + url);
                    return url;
                } else if ("convert-my-file".equals(name) && "error".equals(status)) {
                    System.out.println("❌ Conversion error: " + task.toString());
                    throw new RuntimeException("CloudConvert 변환 실패: " + task.toString());
                }
            }

            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException ignored) {}
        }
        throw new RuntimeException("파일 변환이 시간 내에 완료되지 않았습니다.");
    }

}
```

```
@GetMapping("/{todoId}/convert/status")  신규 *
public ResponseEntity<?> getConvertStatus(@PathVariable Long todoId) {
    try {
        Todolist todo = todolistService.getTodolist(todoId);

        // 결과 응답 구성
        return ResponseEntity.ok(
                Map.of(
                        k1: "todoId", todoId,
                        k2: "status", todo.getConvertStatus(),
                        k3: "convertedFileUrl", todo.getConvertedFileUrl(),
                        k4: "convertedAt", todo.getConvertedAt()
                )
        );
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().body(Map.of( k1: "error", e.getMessage()));
    }
}
```

4. 강지윤
    a. 개발 요약
        - 팀원 목록 조회창
            - 팀스페이스 내 오른쪽 화면에 유저 목록 창 생성
            - 목록 조회 api를 이용해 데이터베이스 내 해당 팀스페이스 멤버만 조회되도록 함
            - 팀스페이스 생성자는 최상단에 항상 표시
        - 팀원 초대 팝업
            - 팀원 초대시 초대할 링크를 전달할 이메일 주소 입력창
            - 이메일 형식 검사+ 에러 메시지
            - 토큰을 이용해 링크 유효성 검사
        - 초대 거절/수락 시 파생 화면
            - 초대 수락 화면 제작
            - 링크 만료 또는 이미 사용된 링크일 경우 화면 제작
            - 초대 수락 후 회원가입 여부에 따라 다른 화면 로직 구현
        - 팀스페이스 관리 팝업
            - 생성자만 팀 정보 수정 가능
            - 생성자는 팀 스페이스 삭제 가능
            - 팀원은 나가기만 가능
    b. 코딩 자료
    - 팀 초대 팝업

```jsx
import React, { useState } from 'react';
import './InvitePopup.css';

function InvitePopup({ teamId, teamName, onClose }) {
  const [email, setEmail] = useState('');
  const [error, setError] = useState('');
  const [alert, setAlert] = useState('');
  const [loading, setLoading] = useState(false);

  const validateEmail = (value) => {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailRegex.test(value);
  };

  const handleEmailChange = (e) => {
    const value = e.target.value;
    setEmail(value);
    setError(validateEmail(value) ? '' : '올바른 이메일 주소를 입력하세요.');
  };

  const sendInvite = async () => {
    if (!validateEmail(email)) {
      setError('올바른 이메일 주소를 입력하세요.');
      return;
    }

    setLoading(true);
    setError('');
    setAlert('');

    try {
      const response = await fetch(`${import.meta.env.VITE_API_BASE_URL}/invi
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ teamId, email: email.trim() }),
      });

```

```
      const data = await response.json();
      if (!response.ok) throw new Error(data.message || '초대 전송 실패');

      setAlert(`초대 링크가 전송되었습니다.\n(유효기간: ${new Date(data.expiresAt).toLocaleString()})`);
      setEmail('');
    } catch (err) {
      console.error('[초대 실패]', err);
      setAlert(`${err.message || '에러가 발생했습니다.'}`);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="invite-dim">
      <div className="invite-box">
        <button className="invite-close" onClick={onClose}>×</button>

        <h2 className="invite-title">
          이메일로 <span className="tn">{teamName}</span> 에 초대하기
        </h2>

        <div className="invite-input-wrapper">
          <input
            type="email"
            placeholder="example@email.com"
            className="invite-input"
            value={email}
            onChange={handleEmailChange}
          />
          {error && <p className="invite-error">{error}</p>}
        </div>
```

- 프로필 기본 화면 유틸 함수

```
const COLORS = ['#7A86D8', '#D88A7A', '#7AD8A3', '#D7D87A', '#B47AD8', '#7AD8D5', '#D87AB3'];

export function getColorForName(name) {
  let hash = 0;
  for (let i = 0; i < name.length; i++) {
    hash = name.charCodeAt(i) + ((hash << 5) - hash);
  }
  return COLORS[Math.abs(hash) % COLORS.length];
}

```

- 초대 수락 + 수락 후 화면

```jsx
import React, { useEffect, useState } from 'react';
import { useSearchParams, useNavigate } from 'react-router-dom';
import { IoClose } from 'react-icons/io5';
import './AcceptInvitation.css';

function AcceptInvitation() {
  const [params] = useSearchParams();
  const [loading, setLoading] = useState(true);
  const [teamName, setTeamName] = useState('');
  const [teamDesc, setTeamDesc] = useState('');
  const [userName, setUserName] = useState('');
  const [error, setError] = useState('');
  const [accepted, setAccepted] = useState(false);
  const [showWelcome, setShowWelcome] = useState(false);
  const [teamId, setTeamId] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    const token = params.get('token');
    if (!token) {
      setError('유효하지 않은 접근입니다.');
      setLoading(false);
      return;
    }

    const fetchValidation = async () => {
      try {
        const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}/invitations/validate?token=${token}`);
        if (!res.ok) throw new Error(`서버 오류: ${res.status}`);
        const data = await res.json();
        if (data.valid) {
          setTeamId(data.teamId);
          setUserName(data.userName || data.email);
        } else {
          throw new Error('초대 링크가 만료되었거나 이미 사용되었습니다.');
        }
```

```jsx
          const teamRes = await fetch(`${import.meta.env.VITE_API_BASE_URL}/teams/${data.teamId}`);
          if (!teamRes.ok) throw new Error('팀 정보를 불러오는 데 실패했습니다.');

          const teamData = await teamRes.json();
          setTeamName(teamData.teamName);
          setTeamDesc(teamData.teamDesc);
        } catch (err) {
          console.error('[VALIDATE ERROR]', err);
          setError(err.message || '유효성 확인 중 오류가 발생했습니다.');
        } finally {
          setLoading(false);
        }
      };

      fetchValidation();
    }, [params]);

    const handleAccept = async () => {
      const token = params.get('token');
      try {
        const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}/invitations/accept`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ token }),
        });

        if (!res.ok) {
          const errorText = await res.text();
          throw new Error(errorText || '수락에 실패했습니다.');
        }
```

```
      const result = await res.json();
      setTeamName(result.teamName);
      setTeamDesc(result.teamDesc);
      setTeamId(result.teamId);

      setAccepted(true);
      setShowWelcome(true);
    } catch (err) {
      console.error('[ACCEPT ERROR]', err);
      setError(err.message || '초대 수락 중 오류가 발생했습니다.');
    }
  };

```

```
  return (
    <div className={`accept-wrapper${accepted && showWelcome ? ' dim-background' : ''}`}>
      <div className="accept-box">
        {loading ? (
          <p className="accept-message">유효성 확인 중...</p>
        ) : error ? (
          <p className="accept-error">❌ {error}</p>
        ) : accepted && showWelcome ? (
          <div className="welcome-popup">
            <button
              className="welcome-close"
              onClick={() => {
                setShowWelcome(false);
                navigate(`/team/${teamId}`);
              }}
              aria-label="닫기"
            >
              <IoClose />
            </button>
            <h3># <strong>{teamName}</strong>에 오신 걸 환영합니다!</h3>
            <p>{teamName}의 팀스페이스입니다.</p>
            <div className="welcome-desc">{teamDesc}</div>
          </div>
        ) : (
          <>
            <h2 className="accept-title">GENAU 팀 초대</h2>
            <p className="accept-message">
              <strong>{userName}</strong>님, <strong>{teamName}</strong>에 초대되었습니다.
            </p>
            <button className="accept-btn" onClick={handleAccept}>
              초대 수락하기
            </button>
          </>
        )}
      </div>
    </div>
```

```
  );
}

export default AcceptInvitation;

```

- 팀스페이스 관리 팝업

```jsx
import React, { useState } from 'react';
import './TeamProfilePopup.css';
import { getColorForName } from '../../utils/colorUtils';

function TeamProfilePopup({ teamInfo, currentUserId, onClose, onUpdatedOrDeleted }) {
  const isManager = currentUserId === teamInfo.userId;
  const [name, setName] = useState(teamInfo.teamName);
  const [desc, setDesc] = useState(teamInfo.teamDesc);
  const [img, setImg] = useState(teamInfo.teamProfileImg);

  const handleUpdate = async () => {
    try {
      const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}/teams/${teamInfo.teamId}`, {
        method: 'PATCH',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userId: currentUserId, teamName: name, teamDesc: desc, teamProfileImg: img }),
      });

      if (res.ok) {
        const updated = await res.json();
        alert('팀 정보가 수정되었습니다.');
        onUpdatedOrDeleted('update', updated);
        onClose();
      } else {
        const text = await res.text();
        alert('수정 실패: ' + text);
      }
    } catch (err) {
      console.error('수정 중 오류:', err);
    }
  };

```

```jsx
  const handleLeave = async () => {
    if (!window.confirm('정말 이 팀에서 나가시겠습니까?')) return;
    try {
      const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}/teams/${teamInfo.teamId}/leave`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userId: currentUserId }),
      });

      if (res.ok) {
        alert('팀에서 나갔습니다.');
        onUpdatedOrDeleted('leave', teamInfo.teamId);
        onClose();
      } else {
        const text = await res.text();
        alert('실패: ' + text);
      }
    } catch (err) {
      console.error('나가기 오류:', err);
    }
  };

  const handleDelete = async () => {
    if (!window.confirm('정말 이 팀을 삭제하시겠습니까?')) return;
    try {
      const res = await fetch(`${import.meta.env.VITE_API_BASE_URL}/teams/${teamInfo.teamId}`, {
        method: 'DELETE',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userId: currentUserId }),
      });

```

```
      if (res.ok) {
        alert('팀이 삭제되었습니다.');
        onUpdatedOrDeleted('delete', teamInfo.teamId);
        onClose();
      } else {
        const text = await res.text();
        alert('삭제 실패: ' + text);
      }
    } catch (err) {
      console.error('삭제 오류:', err);
    }
  };

  const renderProfileImage = () => {
    if (img?.trim()) {
      return <img src={img} alt="profile" className="team-profile-img" />;
    } else {
      const bgColor = getColorForName(name);
      const initial = name.charAt(0).toUpperCase();
      return <div className="team-default-avatar" style={{ backgroundColor: bgColor }}>{initial}</div>;
    }
  };
```

```
  return (
    <div className="team-popup-dim">
      <div className="team-popup-box">
        <button className="popup-close" onClick={onClose}>×</button>
        {renderProfileImage()}
        {isManager ? (
          <input className="team-name-input" value={name} onChange={e => setName(e.target.value)} />
        ) : (
          <div className="team-name-read">{name}</div>
        )}
        {isManager ? (
          <textarea className="team-desc-input" value={desc} onChange={e => setDesc(e.target.value)} />
        ) : (
          <div className="team-desc-read">{desc}</div>
        )}
        {isManager ? (
          <>
            <button className="team-save-btn" onClick={handleUpdate}>수정하기</button>
            <button className="team-delete-btn" onClick={handleDelete}>팀 스페이스 삭제</button>
          </>
        ) : (
          <button className="team-leave-btn" onClick={handleLeave}>팀 스페이스 나가기</button>
        )}
      </div>
    </div>
  );
}

export default TeamProfilePopup;
```

3. 수정 내용 보고

- 프론트엔드
  - 팀원 추가 화면 디자인 수정(백그라운드 → 팝업 형태)
  - 팀원 초대 링크 클릭 시의 화면 추가
- 백엔드
  - 팀원 조회 API에 이미지도 return 하게끔 보완
  - 팀원 초대 수락하고 들어갔을 때 계정 전환되기 위해 UserId 반환하게끔 API 보완

4. 문제점 분석

- 프론트엔드
    - 팀스페이스 초기 화면에 요소가 부족하고, 각 버튼의 기능에 대한 안내가 잘 이루어지지 않아 사용자 입장에서 다소 불친절하다는 피드백을 받음
    - 이에 따라, 팀스페이스 생성 시에 default로 할 일 목록 및 할 일 카드를 생성하여 이용자가 참고할 수 있게끔 하기로 결정함.