상세구현내용

메인 페이지 및 화면 전체 구조

- 로고를 클릭하면 메인 페이지로 이동, 가장 상단 우측에 로그인, 모달창과 로그인 시 로그아웃 모달창 및 마이페이지 진입점을 둠
- 메인 이미지는 사이트 전반의 톤 앤 매너를 유지하면서 동시에 사이트 분위기를 조성하는 일러스트로 배치
- 메인 페이지에 사이트의 전반적인 컨텐츠가 노출될 수 있도록 구성하되 단조롭지 않도록 다양하게 배치
- 컨텐츠는 최신순으로 정해진 개수만큼 데이터베이스에서 가져와서 띄움
- 스크롤이 지나치게 길어지지 않도록 이벤트와 공지사항은 한 블록에 배치하되 대신에 다양한 이벤트 포스터를 보여줄 수 있도록 무한 자동 재생 슬라이드로 여러 개의 포스터를 보여줌. 리액트 라이브러리인 React-Slick으로 구현하였으며 사진 밑의 점이나 드래그, 호버시 보여지는 화살표 클릭으로 다른 이미지를 직접 조종하여 볼 수 있음
- 공지사항 또한 최신순으로 노출하되 +를 클릭하면 공지사항 페이지로 이동
- 메인 페이지가 지나치게 번잡하지 않도록 클릭하여 이동하기를 최소화함
- 푸터는 메인 컨텐츠 부분이 짧아도 허공에 뜨지 않고 하단에 위치하도록 위치시킴.

메인 메뉴바 및 세부 메뉴바(토글 메뉴)

실제 구현된 화면(커뮤니티 버튼 활성화 – 관련 페이지가 띄워진 상태)

메인 메뉴바

```
export default function TopMenus() {
  const topMenusRef = useRef(null);
  const { setTopMenuPos } = useContext(TopMenuPosContext);
  useEffect(() => {
    if (topMenusRef?.current) {
      const topMenuPos = topMenusRef.current.getBoundingClientRect().x;
      setTopMenuPos(topMenuPos);
    }
  }, [topMenusRef, topMenusRef?.current?.getBoundingClientRect().x]);

  const { visibleAll, updateVisibleAll } = useContext(ToggleContext);
  useContext(ToggleContext);
  const { visible, pointInTop, pointInMiddle } = visibleAll;
  const { recActivity, totalActivity, community, customerCenter } =
    menuContents;

  useEffect(() => {
    if (pointInTop || pointInMiddle) {
      updateVisibleAll((draft) => {
        draft.visible = true;
      });
    } else {
      updateVisibleAll((draft) => {
        draft.visible = false;
      });
    }
  }, [pointInTop, pointInMiddle]);

  return (
    <ul
      ref={topMenusRef}
      className="menuListUl"
      onPointerOver={() => {
        updateVisibleAll((draft) => {
          draft.pointInTop = true;
        });
      }}
      onPointerOut={() => {
        updateVisibleAll((draft) => {
          draft.pointInTop = false;
        });
      }}
    >
```

```
      <TopMenu menuNo={1} menu={recActivity} />
      <TopMenu menuNo={2} menu={totalActivity} />
      <TopMenu menuNo={3} menu={community} />
      <TopMenu menuNo={4} menu={customerCenter} />
    </ul>
  );
}
```

토글 세부 메뉴바

```
export default function SelectMenuBox() {
  const { topMenuPos } = useContext(TopMenuPosContext);

  const { visibleAll, updateVisibleAll } = useContext(ToggleContext);
  const { visible, pointInTop, pointInMiddle } = visibleAll;
  const { recActivity, totalActivity, community, customerCenter } = detailMenu;

  useEffect(() => {
    if (pointInMiddle || pointInTop) {
      updateVisibleAll((draft) => {
        draft.visible = true;
      });
    } else {
      updateVisibleAll((draft) => {
        draft.visible = false;
      });
    }
  }, [pointInMiddle, pointInTop]);

  return (
    visible && (
      <div
        className="selectMenuBox"
        onPointerOver={() => {
          updateVisibleAll((draft) => {
            draft.pointInMiddle = true;
          });
        }}
        onPointerOut={() => {
          updateVisibleAll((draft) => {
            draft.pointInMiddle = false;
          });
        }}
      >
        <div className="MenuSetBoxList" style={{ left: topMenuPos }}>
          <MenuSetBox detailMenu={recActivity} />
```

```
          <MenuSetBox detailMenu={totalActivity} />
          <MenuSetBox detailMenu={community} />
          <MenuSetBox detailMenu={customerCenter} />
        </div>
      </div>
    )
  );
}
```

상태값을 관리하는 Context

```
export const ToggleContext = createContext({});

const initialVisible = {
  visible: false,
  pointInTop: false,
  pointInMiddle: false,
};

export function ToggleMenu({ children }) {
  const [visibleAll, updateVisibleAll] = useImmer(initialVisible);
  const hoverStyle = 'toggleMenu';

  return (
    <ToggleContext.Provider
      value={{
        visibleAll,
        updateVisibleAll,
        hoverStyle,
      }}
    >
      {children}
    </ToggleContext.Provider>
  );
}
```

- 별개의 컴포넌트로 있는 메인 메뉴바와 토글 세부 메뉴바의 상태를 공유하기 위해 Context를 활용하여 PointerOver 및 PointerOut 상태 공유
- 화면의 가로 사이즈가 줄어들 때 각 메뉴의 크기가 동적으로 줄어듦. 이때 메인 메뉴와 세부 메뉴의 위치가 엇갈리지 않게 하기 위해 메인 메뉴의 가로 위치를 공유하게 하도록 함.
- 토글 메뉴바는 메인 메뉴바 뿐만 아니라 토글 메뉴바 위에 포인터가 있을 때에도 띄워진 상태로 유지할 수 있도록 두 상태 값을 모두 공유함. 즉, 메인 메뉴바든 세부 메뉴바든 어느 곳 하나라도 PointerOver 상태면 토글 메뉴바가 띄워진 상태로 유지됨.
- 메인 메뉴의 경우 NavLink를 활용하여 해당 버튼이 클릭 또는 특정 URL 범위일 때 활성화 스타일이 적용되도록 함.
- 세부 메뉴의 경우 클릭하기 좀더 용이하도록 Hover시에만 활성화 스타일이 적용되도록 함.
- 토글 메뉴와 메인 이미지를 제외한 헤더 부분은 스크롤해도 보여질 수 있도록 상단에 고정.

공통 기능 – 페이징 처리 서비스

```java
@Service
@RequiredArgsConstructor
public class OffSetBasedPaginationUtilsImpl implements OffSetBasedPaginationUtils {
    private final int pageBundleNo = 5;

    @Override
    public int findStartPostNo(int totalCount, int perPagePostCount, int requestPageNo) {
        return ((requestPageNo - 1) * perPagePostCount) + 1;
    }

    @Override
    public int findEndPostNo(int totalCount, int perPagePostCount, int requestPageNo) {
        int startPostNo = findStartPostNo(totalCount, perPagePostCount, requestPageNo);

        if (requestPageNo < findFinalEndPageNo(totalCount, perPagePostCount)) {
            return (startPostNo + perPagePostCount) - 1;
        }

        return startPostNo + findFinalEndPagePostNo(totalCount, perPagePostCount);
    }

    @Override
    public int findFinalEndPageNo(int totalCount, int perPagePostCount) {
        return (totalCount % perPagePostCount) == 0 ? (totalCount / perPagePostCount) :
((totalCount / perPagePostCount) + 1);
    }

    @Override
    public int findFinalEndPagePostNo(int totalCount, int perPagePostCount) {
        return (totalCount % perPagePostCount) == 0 ? perPagePostCount : (totalCount %
perPagePostCount);
    }
}
```

- 사이트는 특성상 전반적으로 오프셋 기반의 페이지네이션 처리가 적절하다고 판단하여 주로 오프셋 기반으로 구현함.
- 페이징 처리는 사이트 전반에 걸쳐서 계속 필요한 기능이기 때문에 공통 기능으로 만들어서 팀원들과 공유함.
- 메서드는 첫 번째 게시물과 마지막 게시물의 순번을 찾는 것을 목표로 하여 만듦. (findFinalEndPageNo()와 findFinalEndPagePostNo()는 각각 마지막 페이지 번호와 마지막 페이지의 게시물이 몇 번째인지 찾는 것으로 마지막 게시물 순번을 찾기 위함)
- 이 메서드들은 LIMIT 쿼리를 제공하지 않는 오라클에서 필요한 게시물만큼 보내기 위해서 기준점이 될 첫 번째 게시물과 마지막 게시물의 번호를 찾는데 사용됨.

공통 기능 – 이미지 업로더

인터페이스 – 이미지를 업로드하기 위한 주요 유틸 정의

```
public interface StoreRequestImagesUtils {
    public static String getNewDirPathAsPerPostType(String requestPostType, int postId) {
        String imgSrc;
        switch(requestPostType) {
            case "totalActivity": imgSrc = "total_activity_img/activity_" + postId + "/";
                break;
            case "recActivity": imgSrc = "recommendation_img/activity/activity_" + postId + "/";
                break;
            case "recTown": imgSrc = "recommendation_img/town/town_" + postId + "/";
                break;
            case "recTownReport": imgSrc = "recommendation_img/town_report/town_report_" + postId +
"/";
                break;
            case "event": imgSrc = "event_img/event_" + postId + "/";
                break;
            case "notice": imgSrc = "notice_img/notice_" + postId + "/";
                break;
            default: throw new IllegalArgumentException("Illegal Post Type: " + requestPostType);
        }
        return imgSrc;
    }

    public static String getNewFileNameAsPerPostType(String requestPostType, int postId, int imgNo,
String imgExtension) {
        String imgName;
        switch(requestPostType) {
            case "event": imgName = "event" + postId + "_img" + imgNo + imgExtension;
                break;
            case "totalActivity": imgName = "activity" + postId + "_img" + imgNo + imgExtension;;
                break;
            case "recActivity": imgName = "rec_a_img" + postId + "_" + imgNo + imgExtension;;
                break;
            case "recTown": imgName = "rec_town_img" + postId + "_" + imgNo + imgExtension;;
                break;
            case "recTownReport": imgName = "town_report" + postId + "_img" + imgNo +
imgExtension;;
                break;
            case "notice": imgName = "notice_img_" + imgNo + imgExtension;;
                break;
            default: throw new IllegalArgumentException("잘못된 유형: " + requestPostType);
        }
        return imgName;
    }

    public static void makeNewDir(String rootPath, String imgSrc) throws IOException {
```

```
        Files.createDirectories(Paths.get(rootPath + imgSrc));
    }

    public static List<String> getFullPathsList(String imgSrc, String...imgNames) {
        List<String> imgFullPaths = new ArrayList<String>();
        imgFullPaths.add(imgSrc + imgNames[0]);
        imgFullPaths.add((imgNames.length >= 2) && (imgNames[1] != null) ? (imgSrc + imgNames[1]) :
null);
        imgFullPaths.add((imgNames.length >= 3) && (imgNames[2] != null) ? (imgSrc + imgNames[2]) :
null);
        imgFullPaths.add((imgNames.length >= 4) && (imgNames[3] != null) ? (imgSrc + imgNames[3]) :
null);
        imgFullPaths.add((imgNames.length == 5) && (imgNames[4] != null) ? (imgSrc + imgNames[4]) :
null);
        return imgFullPaths;
    }

    public void saveImageFiles(List<MultipartFile> imgFiles, List<String> fullPathsList, String
rootPath) throws IOException;
    public void saveImageFKAndThumbnail(ImageInfoUploadMarker imageInfoUploadMarker, int imgId, int
postId);
    public void saveImagePaths(ImageInfoUploadMarker imageInfoUploader, int imgId, List<String>
imgFullPaths);
}
```

구현 클래스 – 이미지를 업로드(서버에 저장)하기 위한 기능 구현

```
@Service
@RequiredArgsConstructor
public class StoreRequestImagesUtilsImpl implements StoreRequestImagesUtils {
    @Override
    public void saveImageFiles(List<MultipartFile> imgFiles, List<String> fullPathsList, String
rootPath) throws IOException {
        for (int i = 0; i < imgFiles.size(); i++) {
            MultipartFile requestFile = imgFiles.get(i);

            File storedFile = new File(rootPath + fullPathsList.get(i));
            requestFile.transferTo(storedFile);
        };
    }

    @Override
    public void saveImageFKAndThumbnail(ImageInfoUploadMarker imageInfoUploadMarker, int imgId, int
postId) {
        imageInfoUploadMarker.requestUpdateFK(imgId, postId);
    }

    @Override
    public void saveImagePaths(ImageInfoUploadMarker imageInfoUploader, int imgId, List<String>
imgFullPaths) {
```

```
        imageInfoUploader.requestInsertToDB(imgId, imgFullPaths);
    }
}
```

마커 인터페이스 – 공통 서비스인 이미지 업로더를 사용하기 위해 구현해야 할 인터페이스

```
public interface ImageInfoUploadMarker {
    public int getPostCurrentIdNo();
    public int getImgCurrentIdNo();
    public void requestInsertToDB(int imgId, List<String> pathList);
    public void requestUpdateFK(int imgId, int postId);
}
```

템플릿 서비스 – 이미지 업로드 서비스를 더 단순하게 사용하게 하기 위한 템플릿 메서드

```
@Service
@RequiredArgsConstructor
public class StoreRequestImagesServiceImpl implements StoreRequestImagesService {
    private final StoreRequestImagesUtils storeRequestImagesUtils;

    @Value("#{utilProperties['image.root.path']}")
    private String rootPath;

    @Override
    public void storeRequestImages(String requestPostType, ImageInfoUploadMarker uploader,
List<MultipartFile> imgFiles) throws IOException {
        int postId = uploader.getPostCurrentIdNo();
        int imgId = uploader.getImgCurrentIdNo();

        String imgSrc = StoreRequestImagesUtils.getNewDirPathAsPerPostType(requestPostType,
postId);
        StoreRequestImagesUtils.makeNewDir(rootPath, imgSrc);

        List<String> imgNames = new ArrayList<String>(5);
        for (int i = 0; i < imgFiles.size(); i++) {
            String imgOriginName = imgFiles.get(i).getOriginalFilename();
            String imgExtension = (imgOriginName != null) ?
imgOriginName.substring(imgOriginName.lastIndexOf(".")) : null;

            imgNames.add(StoreRequestImagesUtils.getNewFileNameAsPerPostType(requestPostType,
postId, (i + 1), imgExtension));
        }
        List<String> imgFullPaths = StoreRequestImagesUtils.getFullPathsList(imgSrc,
imgNames.toArray(imgNames.toArray(new String[0])));

        storeRequestImagesUtils.saveImageFiles(imgFiles, imgFullPaths, rootPath);
        storeRequestImagesUtils.saveImageFKAndThumbnail(uploader, imgId, postId);
        storeRequestImagesUtils.saveImagePaths(uploader, imgId, imgFullPaths);
    }
}
```

- 이미지 업로드 기능은 사이트 여러 지점에서 필요한 중복된 기능이므로 한 번 구현으로 편리하게 사용하면서 코드를 줄이도록 공통 기능으로 개발함.
- StoreRequestImagesUtils 인터페이스는 기본적인 서비스 기능의 구조를 작성하고자 하였지만 이미지 저장시 생성될 폴더 및 파일 이름의 규칙은 static으로 선언하여 쉽게 사용할 수 있도록 함. 폴더 생성과 이미지 파일 이름 또한 static으로 선언.
- 폴더는 있으면 해당 폴더를 이용하되 없으면 새로운 폴더를 생성하는 것이 목적이므로 Files.createDirectories()를 활용.
- 구현 클래스인 StoreRequestImagesUtilsImpl 클래스는 이미지 파일을 저장하고 이미지 코드와 이미지 경로를 데이터베이스의 해당 테이블에 저장할 수 있도록 하는 이미지 업로드를 위한 일련의 과정을 정의함. 따라서 이미지 업로더를 사용하기 위해서는 카테고리별로 달라지는 테이블명과 컬럼명 등을 적절하게 적용할 수 있도록 ImageInfoUploadMarker라는 인터페이스를 구현하도록 함.
- ImageInfoUploadMarker 인터페이스에서 구현해야 할 내용은 먼저 이미지 및 파일 이름이 적절하게 생성되도록 하게 하기 위해 현재 게시글 번호와 이미지 번호를 조회하는 것. 이후 게시글 테이블에는 이미지 코드가, 이미지 테이블에는 저장된 이미지의 경로가 저장될 수 있도록 해야 함.
- 이미지 업로더를 좀 더 간편하게 사용하게 하기 위해 콜백함수까지 사용하였음에도 사용을 위해 작성해야 할 코드가 많아서 다시 템플릿 역할을 하는 서비스인 StoreRequestImagesService를 정의하고 템플릿 메서드를 작성.
- 이미지는 테이블에 업로드 됨에 따라 대표 이미지(썸네일)가 자동으로 동기화 되는 트리거가 작동하므로 이를 문제없이 동작하게 하기 위해 적절한 순서로 동작하게 정의함.

공통 기능 – 이미지 리소스 반환

```
@Service
public class GetOneImgFromPathServiceImpl implements GetOneImgFromPathService {
    @Value("#{utilProperties['image.root.path']}")
    private String imageRootPath;

    @Override
    public ResponseEntity<byte[]> getOneImgFromPath(String imagePath) {
        ResponseEntity<byte[]> image = null;

        try {
            String srcFileName = URLDecoder.decode(imageRootPath + imagePath,
StandardCharsets.UTF_8);
            File file = new File(srcFileName);
            HttpHeaders headers = new HttpHeaders();

            headers.add("Content-Type", Files.probeContentType(file.toPath()));

            image = new ResponseEntity<byte[]>(FileCopyUtils.copyToByteArray(file), headers,
HttpStatus.OK);

        } catch (Exception e) {
            return new ResponseEntity<byte[]>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return image;
    }

    @Override
    public ResponseEntity<UrlResource> getOneImgResourceFromPath(String imagePath) throws
IOException {
```

```
        String srcFileName = URLDecoder.decode(imageRootPath + imagePath, StandardCharsets.UTF_8);
        MediaType mediaType =
MediaType.parseMediaType(Files.probeContentType(Paths.get(srcFileName)));
        UrlResource resource = new UrlResource("file:" + srcFileName);

        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_TYPE, mediaType.toString())
                .body(resource);
    }
}
```

- 리액트를 사용하여 클라이언트 사이드에서 렌더링이 되게 하였으므로 서버 및 데이터베이스에 저장된 이미지를 사용하기 위해서는 별도로 리소스를 반환해주는 작업이 필요. 이에 따라 이미지 경로를 받아 byte[] 또는 UrlResource 타입으로 반환하도록 하는 서비스 구현.
- HEADER를 정의해야 하므로 HEADER를 포함한 ResponseEntity 타입으로 반환하도록 하여 이미지 반환이 필요할 때 좀 더 간편하게 사용할 수 있도록 함.
- 이미지 경로는 데이터베이스에 저장된 경로를 활용하되 다른 필요한 경로는 properties에 정의하여 경로 관리가 좀 더 용이하게 하도록 함.

공통 기능 – 이미지를 Base64로 변환

```
@Service
@RequiredArgsConstructor
public class ImgPathToBase64ServiceImpl implements ImgPathToBase64Service {
    @Value("#{utilProperties['image.root.path']}")
    private String rootPath;

    @Override
    public String convertBase64(MultipartFile file) throws IOException {
        String fileName =
StringUtils.cleanPath(Objects.requireNonNull(file.getOriginalFilename()));

        BufferedImage image = ImageIO.read(file.getInputStream());
        try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {
            ImageIO.write(image, fileName.substring(fileName.lastIndexOf(".") + 1), baos);
            return Base64.getEncoder().encodeToString(baos.toByteArray());
        }
    }

    @Override
    public String convertBase64(String src) throws IOException {
        String fullPath = StringUtils.cleanPath(rootPath + src);
        String extension = fullPath.substring(fullPath.lastIndexOf(".") + 1);

        BufferedImage image = ImageIO.read(new File(fullPath));
        try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {
            ImageIO.write(image, extension, baos);
            return "" +
```

```
Base64.getEncoder().encodeToString(baos.toByteArray());
        }
    }
}
```

- 이미지를 UrlResource나 byte[]로 변환해서 반환할 수도 있지만 Base64로 반환하면 문자열 형태로 관리하기가 더 쉽다고 판단하여 Base64로 변환하는 서비스도 추가로 제작하게 됨.
- 주로 게시글 상세 조회 페이지에서 이미지를 여러 개 반환할 경우 List<String> 형태로 반환하게 함.
- 클라이언트가 보낸 MultipartFile이나 경로에서 이미지를 추출하여 변환하는 것이 주요 기능.

공통 기능 – 네트워크 통신

```
import { API_BASE_URL } from './api-config';

async function networkCall(api, method, requestBody, isTextResult) {
  let options = {
    headers: new Headers({
      'Content-Type': 'application/json',
      credentials: 'include',
    }),
    url: API_BASE_URL + api,
    method: method,
  };

  if (requestBody) {
    options.body = JSON.stringify(requestBody);
  }

  return fetch(options.url, options)
    .then((response) => {
      if (response.ok) {
        return isTextResult ? response.text() : response.json();
      } else {
        return isTextResult ? response.text() : response.json();
      }
    })
    .catch((error) => {
      console.log('http error');
      console.log(error);
    });
}
                                                    …생략…

async function networkImgCall(api, method, requestBody) {
  let options = {
    headers: new Headers({
```

```
      credentials: 'include',
    }),
    url: API_BASE_URL + api,
    method: method,
  };

  if (requestBody) {
    options.body = JSON.stringify(requestBody);
  }

  return fetch(options.url, options)
    .then((response) => {
      if (response.ok) {
        return response.blob();
      } else {
        console.log(response.status);
      }
    })
    .then((data) => {
      const imgUrl = URL.createObjectURL(data);
      URL.revokeObjectURL(data);
      return imgUrl;
    })
    .catch((error) => {
      console.log('http error');
      console.log(error);
    });
}
                                                    …생략…

async function networkPOSTImgCall(api, method, formRef, isTextResult) {
  const formData = new FormData();

  const requestFiles = new FormData(formRef).getAll('files');
  requestFiles.forEach((file) => formData.append('files', file));

  const requestBody = new FormData(formRef);
  requestBody.delete('files');
  formData.append(
    'data',
    new Blob([JSON.stringify(Object.fromEntries(requestBody.entries()))], {
      type: 'application/json',
    })
  );
```

```
  let options = {
    headers: new Headers({
      credentials: 'include',
    }),
    url: API_BASE_URL + api,
    method: method,
    body: formData,
  };

  return fetch(options.url, options)
    .then((response) => {
      if (response.status >= 200 && response.status < 300) {
        return isTextResult ? response.text() : response.json();
      } else {
        console.log(response.status);
        return isTextResult ? response.text() : response.json();
      }
    })
    .catch((error) => {
      console.log('http error');
      console.log(error);
    });
}
```

...생략...

- 클라이언트와 서버를 연결하는 네트워크 통신은 비교적 비슷한 기능을 수행하면서도 길고 복잡한 코드를 갖고 있기 때문에 공통 기능으로 정의하여 공유해서 사용.
- 위의 코드 이외에도 상황에 따라서 적절하게 사용할 수 있도록 하는 다양한 코드들이 있음.
- 주로 서버에 API 요청으로 받은 텍스트, 객체, 이미지 등을 받아서 반환하는 것과 클라이언트의 데이터를 서버로 보내서 등록/수정/삭제 등을 요청하는 것으로 나뉨.

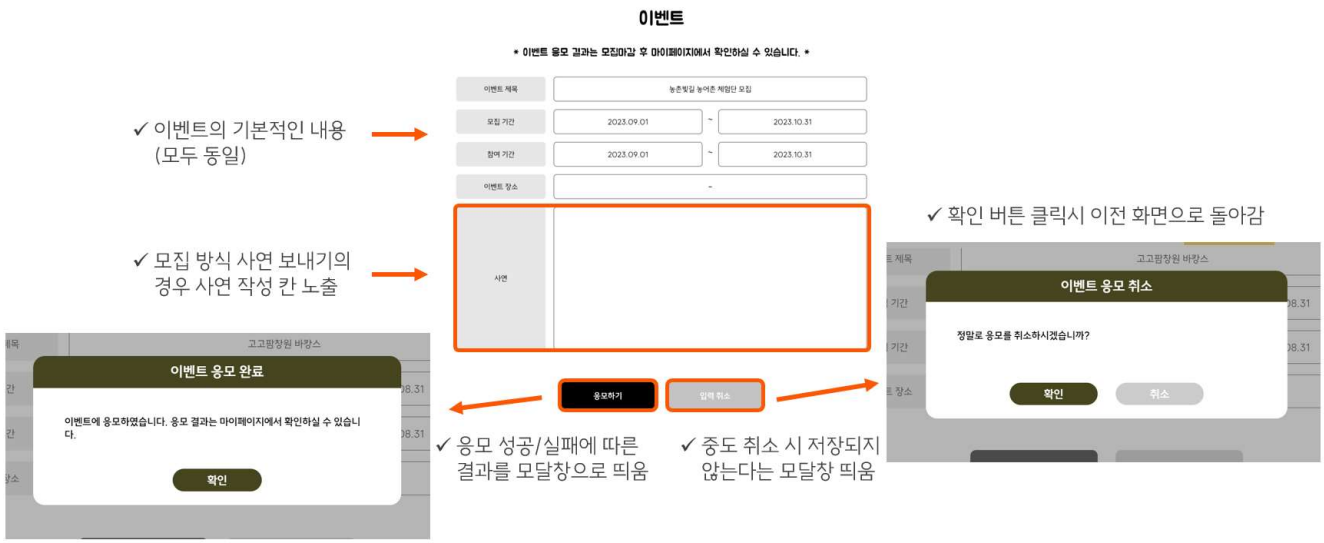
이벤트 – 카드형 목록 조회, 줄글형 목록 조회

- 타이틀 및 목록 조회 타입을 선택할 수 있는 네비게이션 바와 마감 여부 필터링은 이벤트 목록 조회 페이지의 RootLayout으로 두어 공통적으로 보여지도록 함.
- 카드형 또는 줄글형에 해당하는 이미지를 클릭했을 때 적절한 UI 형태로 목록 조회가 되도록 라우팅 함.
- 이벤트 모집 방식에 따라서 응모하기 버튼이 다른 텍스트와 이미지로 보여지도록 함. 버튼을 클릭하면 이벤트 참여/응모하기 페이지로 이동함.
- 카드형, 줄글형, 마감여부에 따른 게시글 개수에 따라서 적절하게 페이지 처리가 되도록 함.
- 카드형이나 줄글형 모두 카드를 클릭했을 때 해당 게시글의 상세 조회 페이지로 이동. 다만 찜하기에 해당하는 하트를 클릭했을 때에는 찜하기만 활성화/비활성화됨.

이벤트 – 상세 조회

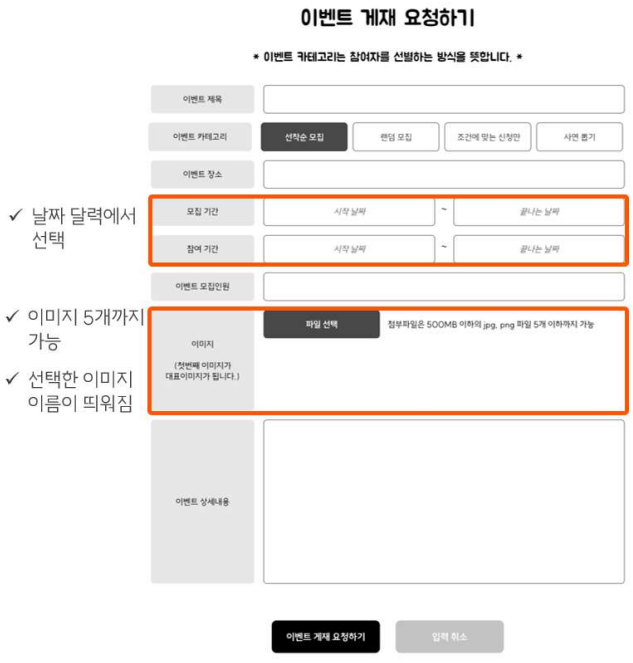
- 이벤트 상세 조회는 선택한 이벤트에 대한 모든 정보를 보여줌.
- 응모하기와 목록보기 버튼으로 페이지 이동을 보다 간편하게 함.
- 응모하기 버튼은 목록 조회에 있던 응모하기와 동일한 기능으로 좀더 쉽게 이벤트에 응모할 수 있도록 유도하고자 함.
- 이벤트는 내용에 따라서 이미지만 보여지거나 내용만 보여질수도 있음.
- 상세 조회 페이지 접근시 조회수가 카운트업됨.

이벤트 – 이벤트 참여/응모하기



– 이벤트 응모하기 버튼을 누르는 경우 이동하는 페이지.
– 서비스 이용자가 이벤트 내용을 다시 확인한 후 응모할 수 있도록 하는 역할로 이벤트 제목과 모집 기간, 참여 기간 및 이벤트 장소를 명시.
– 사연을 보내야 하는 이벤트의 경우 사연을 작성할 수 있도록 함(사연 수정하기 페이지는 마이페이지에서 접근할 수 있음)
– 응모하기는 응모에 성공하면 성공 메세지를, 실패하면 실패 메시지를 모달창으로 띄움. 확인을 누르면 목록 페이지로 이동함. 사연을 보내야 하는 이벤트의 경우 사연을 작성하지 않으면 필수 항목을 작성하지 않았다는 메시지를 모달창으로 띄움. 확인을 누르면 다시 작성 중인 페이지를 보여줌.
– 입력 취소 버튼은 입력 취소를 원하는 사용자에게 확인 메시지를 모달창으로 띄우는 역할을 함. 확인을 누를 경우 이전 페이지로 이동, 취소를 누를 경우 다시 작성 중인 페이지를 보여줌.

이벤트 – 이벤트 게재 요청하기(서비스 제공자의 이벤트 등록 요청)



- 서비스 제공자가 이벤트 게재 요청을 하기 위해 이벤트를 등록하는 페이지.
- 이벤트 카테고리는 클릭한 카테고리를 다른 스타일로 눈에 띄게 표시해줌.
- 날짜 선택의 경우 리액트 라이브러리인 DatePicker를 활용하여 날짜를 달력으로 손쉽게 선택할 수 있도록 함. 최소 시작 날짜는 현재 날짜, 끝나는 날짜는 시작 날짜 이후로 선택할 수 있도록 함.
- 모집인원은 숫자만 입력가능하며 최소 1명 이상 입력하도록 함.
- 이미지 업로드는 5개까지만 업로드 가능하도록 제한하였으며 이미지 파일 선택시 파일 선택 버튼 아래에 이미지 파일 이름이 띄워지도록 함.
- 필수 항목을 입력하지 않고 제출할 경우 필수 항목을 입력하지 않았다는 안내 메시지와 함께 입력해야 하는 항목을 빨간색으로 두드러지게 표현함(입력시 다시 원상태로 바뀜)
- 이벤트 게재 요청하기를 클릭할 경우 유효한 값인지 체크한 후 모든 값이 유효하면 서버에 제출. 서버에 제출한 후 성공했다는 메세지를 받으면 성공 안내 모달창을, 그렇지 않을 경우 실패 안내 모달창을 띄움.

커뮤니티(멤버) – 소통공간 카드형 목록 조회/줄글형 목록 조회

- 타이틀과 게시판 이용 수칙, 게시판 목록 조회 타입 버튼은 RootLayout으로 두고 버튼 선택에 따라 다른 타입의 목록 UI가 보여지도록 함.
- 마을 게시판에 쪽지를 붙여서 소통을 하는 듯한 재밌는 느낌을 주고자 서비스 이용자가 선택한 쪽지 타입에 따라서 다른 디자인의 쪽지 위에 게시글이 보여지도록 함. 클릭하면 해당 게시글의 상세 조회 페이지로 이동.
- 게시글을 좀 더 단정한 형태로 보고자 하는 이용자들을 위하여 줄글형 타입의 UI도 제공함. 줄글형 목록타입의 버튼을 클릭하면 줄글형 방식으로 목록을 조회할 수 있음.
- 커뮤니티는 그 특성상 SNS와 유사하게 동작하는 것이 적절하다고 판단하여 커서 기반 페이지네이션을 채택함. 게시글 등록 순서에 따라 생성되는 게시글의 ID 값을 내림차순으로 정렬하여 한 번에 보여줄 게시글 수만큼 조회하는 방식을 사용.
- 화면의 더보기 버튼을 클릭하면 다른 페이지로 이동하는 것이 아니라 정해진 게시글 수만큼 서버에서 데이터를 받아와서 게시글이 추가되는 방식. 서버와 통신하면서 최대 게시글 개수도 계속 받아오기 때문에 최대치만큼 게시글을 받아오면 더보기 버튼이 사라짐.

커뮤니티(셀러) – 마을소식 카드형 목록 조회/줄글형 목록 조회

- 서비스 제공자의 커뮤니티는 정보 전달을 주 이용방식으로 예상하기 때문에 카드형 목록 UI를 가독성이 높은 형태로 보여주고자 함.
- 페이징 처리는 서비스 이용자의 커뮤니티와 마찬가지로 커서 기반으로 함. 또한 목록 타입을 선택할 때 카드형/줄글형을 선택할 수 있는 것과, 카드 선택 시 상세 조회 페이지로 이동하는 것도 동일함.

커뮤니티(멤버/셀러) – 게시글 상세 조회

- 게시글 상세 조회는 본인 글일 경우와 다른 사람의 글일 경우(혹은 로그아웃 상태)에 따라서 다른 기능에 접근할 수 있음.
- 다른 사람의 글일 경우 목록을 보거나 좋아요 표시를 하거나 신고를 할 수 있음. 신고하기를 클릭할 경우 신고 완료를 안내하는 모달창이 띄워짐. 비회원의 경우 상세 조회는 가능하나 신고하기 및 좋아요는 불가.
- 본인 글일 경우 편집과 삭제가 가능. 편집하기는 편집 페이지로 이동하고 삭제 버튼은 바로 삭제처리 함.

커뮤니티(멤버/셀러) – 게시글 등록/편집

- 커뮤니티 글쓰기 페이지는 마을소식(서비스 제공자 커뮤니티)은 내용만 입력할 수 있도록 되어 있으며 쪽지를 선택해야 하는 소통공간(서비스 이용자 커뮤니티)의 경우 쪽지를 선택할 수 있는 슬라이드가 추가되어 있음.
- 쪽지는 기본적으로 첫 번째 쪽지가 선택되어 있고 화살표나 하단의 점을 클릭하거나 드래그하여 다른 쪽지를 볼 수도 있음. 선택된 쪽지는 테두리에 붉은 색 선으로 둘러짐.
- 글쓰기는 등록하기와 작성 취소 버튼이 있음. 등록하기 버튼 클릭시 내용이 입력되어 있지 않으면 필수 항목이 입력되지 않았다는 안내 메시지 모달창이 뜨고 값이 입력되어 있으면 등록 성공/실패 여부에 따라 적절한 메시지를 모달창에 띄움. 작성 취소시 정말 작성을 취소하겠냐는 메시지를 띄우고 확인을 누르면 이전 페이지로 이동하고 취소를 누를 경우 작성 중인 페이지에 머무름.
- 편집하기의 경우 게시글 내용을 가져와서 수정할 수 있도록 함. 특히 소통공간의 경우 선택한 쪽지 타입도 보여주되 뒤 페이지에 있는 쪽지(6-10번)를 선택했을 경우 선택한 쪽지를 잘 보여줄 수 있도록 해당 페이지를 보여주도록 함.