

제작기간: 2023-02-07 ~ 2023.02.17
버전:v0.0.1

이와이 – 운동 기록 페이지 작성 기여

01 기획 의도

02 소프트웨어 구성

03 목업 디자인

04 Back - End

05 Front - End

Part 1

기획 의도



기획 의도

운동에 관심이 생겨 관련 API를 찾아보다가 아이폰과 스토어에 있는 피트니스 앱을 알게 되었습니다. 처음에는 피트니스 앱을 구현하는 API를 만들려고 구상했다가 곰곰이 생각해보니 사용자들이 사용하는 피트니스 앱은 이미 수많은 앱들이 좋은 퀄리티로 나와있었고, 그렇다면 트레이너를 위한 회원 관리 시스템을 설계해보자. 하고 만들게 되었습니다.

Part 2

소프트웨어 구성

소프트웨어 구성



- Java
- Spring Boot
- JPA
- Postgres



- Javascript
- vue.js
- Nuxt



Part 3

목업 디자인

목업 디자인

회원 정보

유저 ID	이름	성별	키	몸무게	운동 수행 능력	BMI 지수	비만도	마지막 방문	하루 목표치
1	홍길동	남자	180.1	74.3	초급자	22.91	정상	2023-02-14	120(분)
2									
3									
4									
5									
6									
7									
8									
9									
10									

회원 평균
사용자 전체 평균 페이지

신규 등록
정보 등록 페이지

등록하기
메인 페이지

수정
정보 수정 페이지

< 1 2 3 4 >

회원 등록

이름: 홍길동
성별: 남
키: 180.1
몸무게: 74.3
운동 수행 능력: 초급자
하루 운동 목표치(분): 120

정보 등록
등록 완료 페이지 1

등록 취소
페이지

등록 취소
페이지

등록
취소

메인으로
메인 페이지

회원 정보 페이지

회원 정보 등록 페이지

목업 디자인

현재 등록된 회원 수는 명입니다.

등록된 회원의 평균 키는 cm입니다.

등록된 회원의 평균 몸무게는 kg입니다.

등록된 초심자 수는 명입니다.

등록된 중급자 수는 명입니다.

등록된 상급자 수는 명입니다.

[돌아가기](#)

[사용자 정보
페이지](#)

전체 회원 통계 페이지

회원 정보 수정

체중

내음을 입력해주세요

운동능력

초심자

중급자

고급자

[수정 카드
페이지](#)

[수정 완료
페이지 or 수정
실패 페이지](#)

[취소](#)

[수정](#)

회원 정보 수정 페이지

목업 디자인

운동 기록							오늘의 운동 기록			
기록	회원 이름	오늘의 운동 기록 검색 페이이지	운동 이름	중량 및 횟수	날짜	운동 시작 시간	운동 종료 시간	상태	운동 기록 등록 페이이지	메인 페이지
									기록 등록	돌아가기
1	김길동	가슴	벤치 프레스	5세트/10회/50kg	2023-02-14	10:00:00	11:15:00	운동 중	운동 종료 페이이지	운동 종료 페이이지
2									운동 종료 페이이지	운동 종료 페이이지
3									운동 종료 페이이지	운동 종료 페이이지
4									운동 종료 페이이지	운동 종료 페이이지
5									운동 종료 페이이지	운동 종료 페이이지
6									운동 종료 페이이지	운동 종료 페이이지
7									운동 종료 페이이지	운동 종료 페이이지
8									운동 종료 페이이지	운동 종료 페이이지
9									운동 종료 페이이지	운동 종료 페이이지
10									운동 종료 페이이지	운동 종료 페이이지

운동 기록 페이이지

회원 ID	오늘의 운동 부위	운동 이름	중량(kg)	세트	횟수
1	운동 부위 ▶ 가슴 등	벤치 프레스	50	5	10

운동 등록 완료 페이이지1

운동 기록 관리 페이이지

등록 돌아가기

운동 기록 등록 페이이지

A photograph of a person from behind, standing in a doorway. The person is wearing a dark red, long-sleeved shirt. The background is a bright, overexposed interior space with vertical architectural elements. The overall composition is moody and minimalist.

Part 4

Back-End

Back - End (Entity)

```

@Entity
@Getters
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false, length = 20)
    private String userName;
    @Enumerated(value = EnumType.STRING)
    @Column(nullable = false)
    private Gender gender;
    @Column(nullable = false)
    private Double userHeight;
    @Column(nullable = false)
    private Double userWeight;
    @Column(nullable = false)
    private LocalDateTime lastConnect;
    @Enumerated(value = EnumType.STRING)
    @Column(nullable = false)
    private FitnessAbility fitnessAbility;
    @Column(nullable = false)
    private Integer dailyGoalMinute;

    public void putUserInfo(UserInfoUpdateRequest request) {
        this.userWeight = request.getUserWeight();
        this.fitnessAbility = request.getFitnessAbility();
        this.dailyGoalMinute = request.getDailyGoalMinute();
    }
    public void putLastConnect() {
        this.lastConnect = LocalDateTime.now();
    }

    private UserInfo(UserInfoBuilder builder) {
        this.userName = builder.userName;
        this.gender = builder.gender;
        this.userHeight = builder.userHeight;
        this.userWeight = builder.userWeight;
        this.lastConnect = builder.lastConnect;
        this.fitnessAbility = builder.fitnessAbility;
        this.dailyGoalMinute = builder.dailyGoalMinute;
    }

    public static class UserInfoBuilder implements CommonModelBuilder<UserInfo> {

        private final String userName;
        private final Gender gender;
        private final Double userHeight;
        private final Double userWeight;
        private final LocalDateTime lastConnect;
        private final FitnessAbility fitnessAbility;
        private final Integer dailyGoalMinute;

        public UserInfoBuilder(UserInfoRequest request) {
            this.userName = request.getUserName();
            this.gender = request.getGender();
            this.userHeight = request.getUserHeight();
            this.userWeight = request.getUserWeight();
            this.lastConnect = LocalDateTime.now();
            this.fitnessAbility = request.getFitnessAbility();
            this.dailyGoalMinute = request.getDailyGoalMinute();
        }

        @Override
        public UserInfo build() {
            return new UserInfo(this);
        }
    }
}

```

회원 정보 Entity

회원 정보를 기록하기 위한 Entity입니다.
회원 이름, 성별, 키, 몸무게, 마지막 방문일, 운동 수행 능력, 오늘의 목표치를 설정했습니다.

Setter 대신 빌더(builder) 패턴을 사용해 가독성과 유연성을 높이고 유지 보수를 쉽게 하였습니다.
또한, Service에서 Setter를 이용해 정보를 저장했을 때 일어날 수 있는 실수를 방지하는 것도 목적 중 하나입니다.

몸무게, 운동 수행 능력, 오늘의 목표치는 회원의 상태에 따라 달라 질 수 있기 때문에 수정 기능을 추가하였고,

마지막 방문일은 회원이 매번 방문할 때마다 갱신 되기 때문에 수정 기능을 추가하였습니다.

Back-End (Entity)

```

@Entity
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class FitnessHistory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "userInfoId", nullable = false)
    private UserInfo userInfo;
    @Column(nullable = false)
    private LocalDate todayDate;
    @Enumerated(value = EnumType.STRING)
    @Column(nullable = false)
    private ExercisePart exercisePart;
    @Column(nullable = false, length = 30)
    private String exerciseName;
    @Column(nullable = false)
    private Integer exerciseWeight;
    @Column(nullable = false)
    private Integer exerciseSetValue;
    @Column(nullable = false)
    private Integer exerciseValue;
    @Column(nullable = false)
    private LocalTime startFitness;
    @Column(nullable = false)
    private LocalTime finishFitness;
    @Column(nullable = false)
    private Boolean isFinished;

    public void putFinishFitness() {
        this.finishFitness = LocalTime.now();
        this.isFinished = true;
    }

    private FitnessHistory(FitnessHistoryBuilder builder) {
        this.userInfo = builder.userInfo;
        this.todayDate = builder.todayDate;
        this.exercisePart = builder.exercisePart;
        this.exerciseName = builder.exerciseName;
        this.exerciseWeight = builder.exerciseWeight;
        this.exerciseSetValue = builder.exerciseSetValue;
        this.exerciseValue = builder.exerciseValue;
        this.startFitness = builder.startFitness;
        this.finishFitness = builder.finishFitness;
        this.isFinished = builder.isFinished;
    }
}

```

```

public static class FitnessHistoryBuilder implements CommonModelBuilder<FitnessHistory> {

    private final UserInfo userInfo;
    private final LocalDate todayDate;
    private final ExercisePart exercisePart;
    private final String exerciseName;
    private final Integer exerciseWeight;
    private final Integer exerciseSetValue;
    private final Integer exerciseValue;
    private final LocalTime startFitness;
    private final LocalTime finishFitness;
    private final Boolean isFinished;

    public FitnessHistoryBuilder(UserInfo userInfo, FitnessHistoryRequest request) {
        this.userInfo = userInfo;
        this.todayDate = LocalDate.now();
        this.exercisePart = request.getExercisePart();
        this.exerciseName = request.getExerciseName();
        this.exerciseWeight = request.getExerciseWeight();
        this.exerciseSetValue = request.getExerciseSetValue();
        this.exerciseValue = request.getExerciseValue();
        this.startFitness = LocalTime.now();
        this.finishFitness = this.startFitness;
        this.isFinished = false;
    }

    @Override
    public FitnessHistory build() {
        return new FitnessHistory(this);
    }
}

```

운동 기록 Entity

회원의 운동 정보를 기록하기 위한 Entity입니다.
회원의 ID를 FK로 설정하였고, 날짜, 운동 부위, 운동 명,
중량 및 횟수, 운동 시작 시간과 종료 시간을 설정했습니다.

운동이 끝났을 때 운동을 종료함과 동시에 운동 종료 시간을 수정하는 기능을 넣었습니다.

Back-End (Controller)

```

@Api(tags = "등록된 회원 전체 통계")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/stat")
public class UserInfoStatController {
    private final UserInfoStatService userInfoStatService;
    @ApiOperation(value = "회원 전체 통계 불러오기")
    @GetMapping("/user")
    public SingleResult<UserInfoStatResponse> getStat() {
        return ResponseService.getSingleResult(userInfoStatService.getUserStat());
    }
}

```

```

{
    "isSuccess": true,
    "code": 0,
    "msg": "성공하였습니다.",
    "data": {
        "totalUserCount": 2,
        "averageUserHeight": 172.5,
        "averageUserWeight": 60,
        "beginnerUserCount": 1,
        "intermediateUserCount": 0,
        "seniorUserCount": 1
    }
}

```

전체 회원 통계 Controller

등록된 회원들의 통계 Controller입니다.

Swagger 기능인 Api태그를 추가해 Swagger에서 기능의 가독성을 높였습니다.

또한 ResponseService를 통해 기능이 정상적으로 작동하는지에러 코드와 메시지가 출력될 수 있게 하였습니다.

Back - End (Controller)

```

@Api(tags = "회원 정보")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/user-info")
public class UserInfoController {
    private final UserInfoService userInfoService;
    @ApiOperation(value = "회원 정보 등록")
    @PostMapping("/data")
    public CommonResult setUserInfo(@RequestBody @Valid UserInfoRequest request) {
        userInfoService.setUserInfo(request);

        return ResponseService.getSuccessResult();
    }
    @ApiOperation(value = "회원 정보 리스트")
    @GetMapping("/all")
    public ListResult<UserInfoItem> getUserInfo() {
        return ResponseService.getListResult(userInfoService.getUserInfo(), true);
    }
    @ApiOperation(value = "회원 상세 정보")
    @GetMapping("/detail/{id}")
    public SingleResult<UserInfoResponse> getUserInfoById(@PathVariable long id) {
        return ResponseService.getSingleResult(userInfoService.getUserInfoById(id));
    }

    @ApiOperation(value = "회원 정보 수정")
    @PutMapping("/update/id/{id}")
    public CommonResult putUserInfo(@PathVariable long id, @RequestBody @Valid UserInfoUpdateRequest
request) {
        userInfoService.putUserInfo(id, request);

        return ResponseService.getSuccessResult();
    }
    @ApiOperation(value = "마지막 방문일 수정")
    @PutMapping("/date-update/id/{id}")
    public CommonResult putLastConnect(@PathVariable long id) {
        userInfoService.putLastConnect(id);

        return ResponseService.getSuccessResult();
    }
}

```

회원 정보 User Info Controller

GET	/v1/user-info/all	회원 정보 리스트
POST	/v1/user-info/data	회원 정보 등록
PUT	/v1/user-info/date-update/id/{id}	마지막 방문일 수정
GET	/v1/user-info/detail/{id}	회원 상세 정보
PUT	/v1/user-info/update/id/{id}	회원 정보 수정

회원 정보 Controller

회원 정보 관련 Service의 Controller입니다.
 Swagger 기능인 Api태그를 추가해 Swagger에서 기능의
 가독성을 높였습니다.
 새롭게 회원을 등록하는 Create 기능과 등록된 회원 리스트를
 볼 수 있는 List형 Read 기능, 한 회원의 상세정보를 확인할 수
 있는 Read기능, 회원 정보를 수정하는 Update 기능, 회원의 마
 지막 방문일을 수정하는 Update 기능으로 이루어져있습니다.

Back - End (Controller)

```

@Api(tags = "운동 기록 관리")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/fitness/history")
public class FitnessHistoryController {
    private final FitnessHistoryService fitnessHistoryService;

    private final UserInfoService userInfoService;
    @ApiOperation(value = "기록 등록")
    @PostMapping("/data/id/{id}")
    public CommonResult setFitnessHistory(@PathVariable long id, @RequestBody @Valid
    FitnessHistoryRequest request) {
        UserInfo userInfo = userInfoService.getData(id);
        fitnessHistoryService.setFitnessHistory(userInfo, request);

        return ResponseService.getSuccessResult();
    }
    @ApiOperation(value = "기록 리스트")
    @GetMapping("/history/")
    public ListResult<FitnessHistoryItem> getHistories() {
        return ResponseService.getListResult(fitnessHistoryService.getHistories(), true);
    }
    @ApiOperation(value = "운동 종료")
    @PutMapping("/finish/id/{id}")
    public CommonResult putFinishFitness(@PathVariable long id) {
        fitnessHistoryService.putFinishFitness(id);

        return ResponseService.getSuccessResult();
    }
}

```

운동 기록 관리

POST

/v1/fitness/history/data/id/{id} 기록 등록

PUT

/v1/fitness/history/finish/id/{id} 운동 종료

GET

/v1/fitness/history/history/ 기록 리스트

운동 기록 Controller

운동 기록 관련 Service의 Controller입니다.
 Swagger 기능인 Api태그를 추가해 Swagger에서 기능의
 가독성을 높였습니다.
 회원의 운동 기록을 등록하는 Create 기능,
 전체 운동 기록을 가져오는 List형 Read 기능,
 운동을 마쳤을 때 운동 종료 시간을 새롭게 수정하는 Update
 기능으로 이루어져있습니다.

Back-End (Service)

```

@Service
@RequiredArgsConstructor
public class UserInfoStatService {
    private final UserInfoRepository userInfoRepository;

    public UserInfoStatResponse getUserStat() {
        List<UserInfo> originList = userInfoRepository.findAll();

        long totalUserCount = originList.size();
        double userHeight = 0;
        double userWeight = 0;
        long beginnerCount = 0;
        long intermediateCount = 0;
        long seniorCount = 0;

        for(UserInfo item : originList) {
            userHeight += item.getUserHeight();
            userWeight += item.getUserWeight();
            if (item.getFitnessAbility().equals(FitnessAbility-BEGINNER)) {
                beginnerCount += 1;
            } else if (item.getFitnessAbility().equals(FitnessAbility-INTERMEDIATE)) {
                intermediateCount += 1;
            } else if (item.getFitnessAbility().equals(FitnessAbility-SENIOR)) {
                seniorCount += 1;
            }
        }

        return new UserInfoStatResponse.FitnessStatResponseBuilder(
            totalUserCount,
            userHeight / totalUserCount,
            userWeight / totalUserCount,
            beginnerCount,
            intermediateCount,
            seniorCount
        ).build();
    }
}

```

전체 회원 통계 Service

전체 회원 통계 Service 입니다.

총 회원 수, 회원들의 평균 키와 몸무게, 운동 수행 능력에 따라 구분한 초심자, 중급자, 상급자의 수를 출력하는 기능입니다.

회원 정보 Repository에 저장된 기록들을 토대로 값을 가져왔고, 불러온 값은 Builder 패턴을 통해 출력하게 하였습니다.

Back-End (Service)

```

@Service
@RequiredArgsConstructor
public class UserInfoService {
    private final UserInfoRepository userInfoRepository;

    public void setUserInfo(UserInfoRequest request) {
        UserInfo addData = new UserInfo.UserInfoBuilder(request).build();

        userInfoRepository.save(addData);
    }

    public UserInfo getData(long id) {
        return userInfoRepository.findById(id).orElseThrow(CMissingDataException::new);
    }

    public ListResult<UserInfoItem> getUserInfo() {
        List<UserInfo> originList = userInfoRepository.findAll();

        List<UserInfoItem> result = new LinkedList<>();

        originList.forEach(item -> result.add(new UserInfoItem.UserInfoItemBuilder(item).build()));

        return ListConvertService.settingResult(result);
    }

    public UserInfoResponse getUserInfoById(long id) {
        UserInfo originData = userInfoRepository.findById(id).orElseThrow(CMissingDataException::new);

        return new UserInfoResponse.UserInfoResponseBuilder(originData).build();
    }

    public void putUserInfo(long id, UserInfoUpdateRequest request) {
        UserInfo originData = userInfoRepository.findById(id).orElseThrow(CMissingDataException::new);

        originData.putUserInfo(request);

        userInfoRepository.save(originData);
    }

    public void putLastConnect(long id) {
        UserInfo originData = userInfoRepository.findById(id).orElseThrow(CMissingDataException::new);

        originData.putLastConnect();

        userInfoRepository.save(originData);
    }
}

```

회원 정보 Service

회원 정보 Service입니다.

Builder 패턴을 통해 신규 회원 정보를 등록하고, Builder 패턴을 통해 회원 정보 리스트를 가져와 회원 정보 Controller에게 넘겨주는 역할을 하고 있습니다.

회원 정보 수정 및 마지막 방문일 수정 기능 또한 회원 정보 Entity의 Builder 패턴을 통해 수정하는 기능을 하고 있습니다.

Back-End (Service)

```

@Service
@RequiredArgsConstructor
public class FitnessHistoryService {
    private final FitnessHistoryRepository fitnessHistoryRepository;

    public void setFitnessHistory(UserInfo userInfo, FitnessHistoryRequest request) {
        FitnessHistory addData = new FitnessHistory.FitnessHistoryBuilder(userInfo, request).build();
        fitnessHistoryRepository.save(addData);
    }

    public ListResult<FitnessHistoryItem> getHistories() {
        List<FitnessHistory> originList = fitnessHistoryRepository.findAll();

        List<FitnessHistoryItem> result = new LinkedList<>();
        originList.forEach(item -> result.add(new
FitnessHistoryItem.FitnessHistoryItemBuilder(item).build()));

        return ListConvertService.settingResult(result);
    }

    public void putFinishFitness(long id) {
        FitnessHistory originData =
fitnessHistoryRepository.findById(id).orElseThrow(CMissingDataException::new);

        originData.putFinishFitness();

        fitnessHistoryRepository.save(originData);
    }
}

```

운동 기록 Service

운동 기록 Service입니다.

Builder 패턴을 통해 회원의 운동 기록 정보를 등록하고, Builder 패턴을 통해 전체 운동 기록 리스트를 가져와 운동 기록 Controller에게 넘겨주는 역할을 하고 있습니다.

운동 종료 기능 또한 운동 기록 Entity의 Builder 패턴을 통해 수정하는 기능을 하고 있습니다.



Part 5
Front - End

Front-End (회원 정보 등록)

```
totalScore: '',
createRuleForm: {
  gender: '',
  memberName: '',
  memberHeight: '',
  memberWeight: '',
  exerciseGrade: '',
},
```

```
createRules: {
  gender: [
    { required: true, message: '성별선택은 필수입니다.', trigger: 'change' }
  ],
  memberName: [
    { required: true, message: '이름 값은 필수입니다.', trigger: 'blur' },
    { min: 2, max: 20, message: '2~20 글자 사이로 입력해주세요.', trigger: 'blur' }
  ],
  memberHeight: [
    { required: true, message: '키 값은 필수입니다.', trigger: 'blur' }
  ],
  memberWeight: [
    { required: true, message: '몸무게는 필수 입니다', trigger: 'blur' }
  ],
  exerciseGrade: [
    { required: true, message: '운동능력선택은 필수입니다.', trigger: 'change' }
  ]
}
```

폼에 등록할 데이터
준비

```
resetForm() {
  let form = this.createRuleForm
  form.gender = null
  form.memberName = null
  form.memberHeight = null
  form.memberWeight = null
  form.exerciseGrade = null
},
```

폼에 규칙을 정해 이중으로 고객에
게 받는 데이터를 검사합니다.

- 프론트에서 하지 않으면 백엔
드에서 부하가 걸릴 수 있기 때
문.

회원 등록 후 데이터를 다시
비우기 위한
resetForm() method 입니다.

회원 등록

Front-End (회원 정보 등록)

```

setData() {
  this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
  this.$store.dispatch(this.$memberConstants.DO_CREATE, this.createRuleForm)
  .then(res => {
    this.$toast.success(res.data.msg)
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
    this.createDialogVisible = false
    this.resetForm()
    this.getList()
  })
  .catch(err => {
    this.$toast.error(err.response.data.msg)
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
  })
}

doSetData() {
  let formName = 'createRuleForm'
  this.$refs[formName].validate(valid => {
    if (valid) {
      this.setData()
    } else {
      return false
    }
  })
}

```

데이터를 검사하기 위한 메서드

만약 데이터 검사를 통과했으면 API실행 메서드 실행.

API 연동

사용자에게 인지시키기 위해 로딩 창을 넣고

.then / 성공 시

성공이라는 메시지를 토스트로 나타낸 후에 로딩을 멈춥니다.

등록 모달을 끄고 다시 비우기 위해 resetForm() method를 실행합니다.

등록된 데이터를 보기 위해 getList() method를 실행합니다.

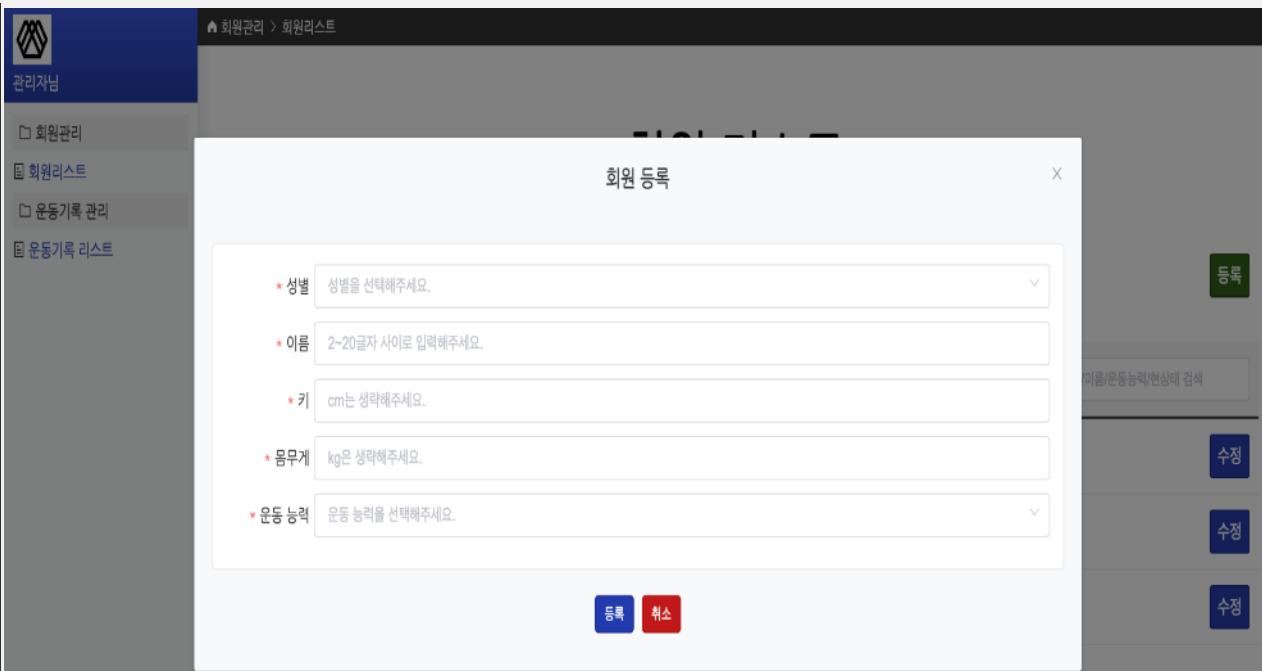
.catch / 실패 시

에러라는 메시지를 토스트로 나타낸 후에 로딩을 멈춥니다.

Front-End (회원 정보 등록)

```
<el-dialog
  title="회원 등록"
  :visible.sync="createDialogVisible"
  width="70%"

>
<el-form :model="createRuleForm" :rules="createRules" ref="createRuleForm" label-width="100px">
  <el-form-item label="성별" prop="gender">
    <el-select v-model="createRuleForm.gender" placeholder="성별을 선택해주세요.">
      <el-option label="남자" value="MAN"></el-option>
      <el-option label="여자" value="WOMAN"></el-option>
    </el-select>
  </el-form-item>
  <el-form-item label="이름" prop="memberName">
    <el-input v-model="createRuleForm.memberName" placeholder="2~20글자 사이로 입력해주세요."></el-input>
  </el-form-item>
  <el-form-item label="키" prop="memberHeight">
    <el-input v-model="createRuleForm.memberHeight" placeholder="cm는 생략해주세요."></el-input>
  </el-form-item>
  <el-form-item label="몸무게" prop="memberWeight">
    <el-input v-model="createRuleForm.memberWeight" placeholder="kg은 생략해주세요."></el-input>
  </el-form-item>
  <el-form-item label="운동 능력" prop="exerciseGrade">
    <el-select v-model="createRuleForm.exerciseGrade" placeholder="운동 능력을 선택해주세요.">
      <el-option label="초심자" value="BEGINNER"></el-option>
      <el-option label="중급자" value="MIDDLE_CLASS"></el-option>
      <el-option label="고인물" value="HIGH_CLASS"></el-option>
    </el-select>
  </el-form-item>
</el-form>
<div class="txt-center" style="margin-top: 20px;">
  <el-button type="primary" @click="doSetData()">등록</el-button>
  <el-button type="danger" @click="createDialogVisible = false">취소</el-button>
</div>
</el-dialog>
```



등록 버튼 클릭 시 dosetData() 메서드가 실행 할 수 있게 설정했습니다.
취소 버튼 클릭 시 회원등록 모달이 꺼지도록 설정했습니다.

Front-End (회원 목록)

```
search: '',
title: '회원 리스트',
createDialogVisible: false,
list: [],
totalCount: ''
```

데이터를 준비한다.

검색을 위한 search
List를 받기 위한 list
총 개수를 받기 위한 totalCount

```
created() {
  this.getList()
},
```

데이터를 미리 준비해서 나타내기
위해 created()에 getList()메서드를
넣습니다.

```
getList() {
  this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, { payload: true })
  this.$store.dispatch(this.$memberConstants.DO_LIST)
    .then((res) => {
      this.list = res.data.list
      this.totalCount = res.data.totalItemCount
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, { payload: false })
    })
    .catch((err) => {
      this.$toast.error(err.response.data.msg)
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, { payload: false })
    })
}
```

API 연동
사용자에게 인지시키기 위해 로딩 창을 넣고
.then / 성공 시 위에 만든 데이터 list에 DB에 있는 list를
넣습니다.
위에 만든 데이터 totalCount에 DB에 있는 totalCount
를 넣은 후 로딩을 멈춥니다.
.catch / 실패 시
에러라는 메시지를 토스트로 나타내고 로딩을 멈춥니다.

회원 목록 script

Front-End (회원 목록)

```

<div>
  <main-title v-bind:title="title" class="txt-center" style="margin: 50px 0" />
  <div class="txt-right mr-2">
    <el-button type="success" circle @click.native="createDialogVisible = true">등록</el-button>
  </div>
  <div style="font-size: 20px" class="mb-3" >
    <i class="el-icon-folder-opened">총 데이터 수는 {{ totalCount }}개 입니다.</i>
  </div>
  <el-table
    :data="list.filter(data => !search || data.memberName.toLowerCase().includes(search.toLowerCase())
      || data.exerciseGrade.toLowerCase().includes(search.toLowerCase())
      || data.bmiState.toLowerCase().includes(search.toLowerCase())
      || data.gender.toLowerCase().includes(search.toLowerCase()))
    >
    <thead>
      <tr>
        <th>회원ID</th>
        <th>이름</th>
        <th>성별</th>
        <th>운동능력</th>
        <th>BMI 상태</th>
        <th>등록일</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="item in list" :key="item.id">
        <td>{{ item.id }}</td>
        <td>{{ item.memberName }}</td>
        <td>{{ item.gender }}</td>
        <td>{{ item.exerciseGrade }}</td>
        <td>{{ item.bmiState }}</td>
        <td>{{ item.createDate | date }}</td>
      </tr>
    </tbody>
  </el-table>
</div>

```

등록 버튼 누르면 회원 등록 모달이 뜹니다.

검색창에 회원이름/운동능력/현 상태/ 성별을 검색하게 하기 위한 테이블입니다.

Id를 내림차순으로 정리하기 위함입니다.

Front-End (회원 리스트)

```
<el-table
  :data="list.filter(data => !search || data.memberName.toLowerCase().includes(search.toLowerCase())
    || data.exerciseGrade.toLowerCase().includes(search.toLowerCase())
    || data.bmiState.toLowerCase().includes(search.toLowerCase())
    || data.gender.toLowerCase().includes(search.toLowerCase()))
  >
  <el-table-column label="순번" prop="id" width="100" sortable>
  </el-table-column>
  <el-table-column label="성별" prop="gender" width="100">
  </el-table-column>
  <el-table-column label="이름" prop="memberName" width="130">
  </el-table-column>
  <el-table-column label="키" prop="memberHeight" width="130">
  </el-table-column>
  <el-table-column label="몸무게" prop="memberWeight" width="130">
  </el-table-column>
  <el-table-column label="운동능력" prop="exerciseGrade" width="130">
  </el-table-column>
  <el-table-column label="bmi" prop="bmi" width="130">
  </el-table-column>
  <el-table-column label="현 상태" prop="bmiState">
  </el-table-column>
  <el-table-column align="right" width="250">
    <template slot="header" slot-scope="scope">
      <el-input
        v-model="search"
        size="small"
        placeholder="성별/이름/운동능력/현상태 검색"/>
    </template>
    <template slot-scope="scope">
      <el-button type="primary" circle>수정</el-button>
    </template>
  </el-table-column>
```

회원 리스트

총 데이터 수는 5개입니다.

순번	성별	이름	키	몸무게	운동능력	bmi	현 상태	수정
5	여자	김피티	156.0cm	177.6kg	초급자	72.98	비만	<button>수정</button>
4	남자	김남자	188.0cm	67.0kg	초급자	18.96	정상	<button>수정</button>
3	여자	김여자	167.0cm	75.0kg	중급자	26.89	비만	<button>수정</button>
2	남자	김김김	177.1cm	87.0kg	초급자	27.74	비만	<button>수정</button>
1	남자	김길동	177.1cm	77.0kg	초급자	24.55	과체중	<button>수정</button>

리스트로 받아 오기 위해 테이블을 만들었습니다.

회원 리스트 실행 화면입니다.

Front-End (회원 정보 수정)

```

<script>
export default {
  asyncData({params}) {
    return {
      id: params.id
    }
  },
  validate({params}) {
    return /^[\d+$/.test(params.id)
  },
  data() {
    return {
      id: null,
      detailInfo: null,
      form: {
        userWeight: '',
        fitnessAbility: '',
        dailyGoalMinute: ''
      },
      rules: {
        userWeight: [
          {required: true, message: '이 값은 필수입니다.', trigger: 'blur'}
        ],
        fitnessAbility: [
          {required: true, message: '이 값은 필수입니다.', trigger: 'blur'}
        ],
        dailyGoalMinute: [
          {required: true, message: '이 값은 필수입니다.', trigger: 'blur'}
        ],
      }
    }
  },
  methods: {
    async getDetail() {
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
      await this.$store.dispatch(this.$fitnessAssistConstants.DO_DETAIL, {id: this.id})
      .then(res) => {
        this.detailInfo = res.data.data
        this.form.userWeight = res.data.data.userWeight
        this.form.fitnessAbility = res.data.data.fitnessAbility
        this.form.dailyGoalMinute = res.data.data.dailyGoalMinute
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
      .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
    }
  }
}

```

```

    },
    doPutData() {
      let formName = 'form'
      this.$refs[formName].validate(valid => {
        if (valid) {
          this.putData()
        } else {
          return false
        }
      })
    },
    putData() {
      let payload = {
        id: this.detailInfo.id,
        data: {
          userWeight: Number(this.form.userWeight),
          fitnessAbility: this.form.fitnessAbility,
          dailyGoalMinute: Number(this.form.dailyGoalMinute)
        }
      }
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
      this.$store.dispatch(this.$fitnessAssistConstants.DO_UPDATE, payload)
      .then(res => {
        this.$toast.success(res.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
        this.$router.replace('/user-info/list')
      })
      .catch(err => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
    },
    moveList() {
      this.$router.push(`/user-info/list`)
    },
    created() {
      this.getDetail()
    }
  }
</script>

```

회원 정보 수정 script

Front-End (회원 정보 수정)

```


<div>
  <el-form ref="form" :model="form" :rules="rules" label-width="150px">
    <el-form-item label="몸무게" prop="userWeight">
      <el-input v-model="form.userWeight" placeholder="숫자로만 적어주세요"></el-input>
    </el-form-item>
    <el-form-item label="운동 수행 능력" prop="fitnessAbility">
      <el-select v-model="form.fitnessAbility" placeholder="운동 수행 능력을 선택해주세요">
        <el-option label="초급자" value="BEGINNER"></el-option>
        <el-option label="중급자" value="INTERMEDIATE"></el-option>
        <el-option label="상급자" value="SENIOR"></el-option>
      </el-select>
    </el-form-item>
    <el-form-item label="하루 운동 목표치(분)" prop="dailyGoalMinute">
      <el-input v-model="form.dailyGoalMinute" placeholder="숫자로만 적어주세요"></el-input>
    </el-form-item>
    <div class="edit-complete">
      <el-button type="success" circle v-on:click.native="doPutData()">등록하기</el-button>
      <el-button type="success" circle v-on:click.native="moveList()">돌아가기</el-button>
    </div>
  </el-form>
</div>
</template>

```

회원 정보 수정 template

The screenshot shows a user profile editing interface. At the top, there's a navigation bar with 'HOME' and '대시보드'. Below it, three input fields are displayed:

- * 몸무게: 50 (Weight: 50)
- * 운동 수행 능력: 상급자 (Fitness Ability: Advanced)
- * 하루 운동 목표치(분): 90 (Daily Goal (minutes): 90)

At the bottom right, there are two green buttons: '등록하기' (Register) and '돌아가기' (Back).

정보 수정 페이지 입니다.
Api와 연동시켜 특정 ID의 정보를 수정할 수 있게끔 하고, 운동 수행 능력 부분을 설정된 ENUM값을 리스트 형식으로 불러올 수 있게 했습니다.

Front-End (전체 회원 통계)

```

<template>
  <div>
    <div style="font-size: 30px; margin-bottom: 40px; text-align: center; font-weight: 800;">회원 통
    </div>
    <div v-if="detailInfo != null" style="text-align: center">
      <div class="statistics-box">현재 등록된 회원 수는 <span class="in-box">{{ detailInfo.totalUserCount }}명</span> 입니다.</div>
      <div class="statistics-box">등록된 회원의 평균 키는 <span class="in-box">{{ detailInfo.averageUserHeight }}cm</span> 입니다.</div>
      <div class="statistics-box">등록된 회원의 평균 몸무게는 <span class="in-box">{{ detailInfo.averageUserWeight }}kg</span> 입니다.</div>
      <div class="statistics-box">등록된 초심자 수는 <span class="in-box">{{ detailInfo.beginnerUserCount }}명</span> 입니다.</div>
      <div class="statistics-box">등록된 중급자 수는 <span class="in-box">{{ detailInfo.intermediateUserCount }}명</span> 입니다.</div>
      <div class="statistics-box">등록된 상급자 수는 <span class="in-box">{{ detailInfo.seniorUserCount }}명</span> 입니다.</div>
    </div>
  </div>
</template>

<script>
import userStatConstants from "~/store/modules/user-stat/constants";

export default {
  data() {
    return {
      detailInfo: null,
    }
  },
  methods: {
    async getDetail() {
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
      await this.$store.dispatch(this.$userStatConstants.DO_USER_STAT)
      .then((res) => {
        this.detailInfo = res.data.data
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
      .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
    },
    created() {
      this.getDetail()
    }
  }
}
</script>

```

회원 통계

현재 등록된 회원 수는 **4명** 입니다.

등록된 회원의 평균 키는 **168.425cm** 입니다.

등록된 회원의 평균 몸무게는 **54.575kg** 입니다.

등록된 초심자 수는 **2명** 입니다.

등록된 중급자 수는 **1명** 입니다.

등록된 상급자 수는 **1명** 입니다.

전체 회원 통계 페이지입니다.

Api와 연동 시켜 박스 안에 값을 출력하도록 구현했습니다.

Front-End (운동 기록 등록)

```
<script>
no usages
export default {
  asyncData({ params }) {
    return {
      id: params.id
    }
  },
  validate({ params }) {
    return /^\d+$/.test(params.id)
  },
  data() {
    return {
      id: null,
      form: {
        part: '',
        exerciseName: '',
        exerciseWeight: '',
        exerciseSet: '',
        exerciseNumber: ''
      },
      rules: {
        part: [
          { required: true, message: '이 값은 필수입니다.', trigger: 'blur' },
        ],
        exerciseName: [
          { required: true, message: '이 값은 필수입니다.', trigger: 'blur' },
          { min:2, max:20, message:'2~20자 내로 입력해주세요.', trigger: 'blur' }
        ],
        exerciseWeight: [
          { required: true, message: '이 값은 필수입니다.', trigger: 'blur' },
        ],
        exerciseSet: [
          { required: true, message: '이 값은 필수입니다.', trigger: 'blur' },
        ],
        exerciseNumber: [
          { required: true, message: '이 값은 필수입니다.', trigger: 'blur' },
        ]
      }
    }
  }
},
```

기록 등록 시 필요한 값을 넣을 공간

Validate 기능을 넣고 form에 들어가는 값이 적합하지 않을 시 등록을 하지 않게 만들었습니다.

Front-End (운동 기록 등록)

```

<template>
  <div>
    <el-form ref="form" :model="form" :rules="rules" label-width="120px">
      <el-form-item class="custom-mtb" label="운동부위">
        <el-select v-model="form.part" placeholder="운동부위">
          <el-option label="등" value="BACK"></el-option>
          <el-option label="가슴" value="CHEST"></el-option>
          <el-option label="어깨" value="SHOULDER"></el-option>
          <el-option label="팔" value="ARM"></el-option>
          <el-option label="하체" value="LEG "></el-option>
        </el-select>
      </el-form-item>
      <el-form-item class="custom-mtb" label="운동이름">
        <el-input v-model="form.exerciseName"></el-input>
      </el-form-item>
      <el-form-item class="custom-mtb" label="중량(kg)">
        <el-input v-model="form.exerciseWeight"></el-input>
      </el-form-item>
      <el-form-item class="custom-mtb" label="세트">
        <el-input v-model="form.exerciseSet"></el-input>
      </el-form-item>
      <el-form-item class="custom-mtb" label="세트당 횟수">
        <el-input v-model="form.exerciseNumber"></el-input>
      </el-form-item>
      <el-form-item class="custom-mtb">
        <el-button type="primary" @click="onSubmit">등록</el-button>
      </el-form-item>
    </el-form>
  </div>
</template>

```

The screenshot shows the front-end interface for registering an exercise. It includes a dropdown menu for '운동부위' (Exercise部位) with options: 등 (Back), 가슴 (Chest), 어깨 (Shoulder), 팔 (Arm), and 하체 (Leg). Below it is a dropdown for '운동이름' (Exercise Name) with options: 벤치프레스 (Bench Press), 누르기 (Push-ups), and 스쿼트 (Squats). Input fields for '중량(kg)' (Weight) and '세트' (Sets) are present, along with a numeric input for '세트당 횟수' (Reps per set). A large blue button labeled '등록' (Register) is at the bottom.

운동부위 등

운동이름

중량(kg)

세트

세트당 횟수 3

등록

Enums 페이지를 가독성을 위하여
드롭 다운 메뉴로 제작하였으며,

필수 값에 대한 요청을
methods 영역에 지정하였습니다.

Front-End (운동 기록 등록)

```

methods: {
  onSubmit() {
    let payload = {
      id: this.id,
      data: {
        exerciseName: this.form.exerciseName,
        exerciseNumber: Number(this.form.exerciseNumber),
        exerciseSet: Number(this.form.exerciseSet),
        exerciseWeight: Number(this.form.exerciseWeight),
        part: this.form.part,
      }
    }
    this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: true)
    this.$store.dispatch(this.$historyConstants.DO_HISTORY_CREATE, payload)
      .then(res => {
        this.$toast.success(res.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
        this.$router.replace( to: `/history/list` )
      })
      .catch(err => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, payload: false)
      })
    }
  }
}
</script>

```

숫자로 들어가는 값은 Number()를 이용해 캐스팅합니다.

API 연동

사용자에게 인지시키기 위해 로딩 창을 넣고

.then / 성공 시

성공이라는 메시지를 토스트로 나타낸 후에 로딩을 멈추고 운동 기록 목록으로 돌아갑니다.

.catch / 실패 시

에러라는 메시지를 토스트로 나타낸 후에 로딩을 멈춥니다.

Front-End (운동 기록 목록)

```

<template>
  <div>
    <el-button type="success" circle @click.native="moveCreate()">등록</el-button>
    <el-table :data="list">

      <el-table-column
        prop="id"
        label="id"
        width="100"
      >
      </el-table-column>
      <el-table-column
        prop="memberName"
        prop="weightAndCountExercise"
        label="종량 및 횟수"
        width="100">
      </el-table-column>
      ... 중간 내용 생략
      <el-table-column
        prop="exerciseDate"
        label="날짜"
        width="150">
      </el-table-column>

      <el-table-column
        prop="totalExerciseTime"
        label="운동시간"
        width="150">
      </el-table-column>

      <el-table-column
        label="Operations"
        width="250">
        <template slot-scope="scope">
          <el-button @click="moveDetail" type="text" size="small" >상세보기</el-button>
          <el-button @click="moveEdit" type="text" size="small">수정</el-button>
          <el-button type="danger" icon="el-icon-delete" circle @click.native="delData(scope.row.id)"></el-button>
        </template>
      </el-table-column>
    </el-table>
  </div>
</template>

```

리스트 형식을 보다 편하게 볼 수 있게끔, 테이블 스타일로 구성하였습니다.

등록 버튼 클릭 시 운동 기록이 리스트에 데이터 베이스에 등록됩니다.

moveDetail -> 상세보기 페이지로 이동

moveEdit -> 수정 페이지로 이동

delData -> scope.row.id를 통해 해당 ID의 정보를 삭제

Front-End (운동 기록 목록)

```

<script>
export default {
  data() {
    return {
      list: [
        {}
      ]
    }
  },
  methods: {
    getList() {
      this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
      this.$store.dispatch(this.$exerciseConstants.DO_LIST)
      .then((res) => {
        this.list = res.data.list
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
      .catch((err) => {
        this.$toast.error(err.response.data.msg)
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
      })
    },
    delData(id) {
      if (confirm('정말 삭제하시겠습니까?')) { // javascript confirm 검색하기
        this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, true)
        this.$store.dispatch(this.$exerciseConstants.DO_DELETE, {id: id})
        .then((res) => {
          this.$toast.success(res.data.msg)
          this.getList()
          this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
        })
        .catch((err) => {
          this.$toast.error(err.response.data.msg)
          this.$store.commit(this.$customLoadingConstants.FETCH_LOADING_SHOW, false)
        })
      }
    }
  }
}

```

```

moveCreate() {
  this.$router.push(`/exercise/form`)
},
moveDetail(id) {
  let linkBaseUrl = '/exercise/detail/'
  this.$router.push(`${linkBaseUrl}${id}`)
},
moveEdit(id) {
  let linkBaseUrl = '/exercise/edit/'
  this.$router.push(`${linkBaseUrl}${id}`)
},
//methods --end
created() {
  this.getList()
},
</script>

```

데이터 영역에서,
API에서 받아온 list값을 세팅하였습니다.

Front-End (운동 기록 목록)

등록

id	회원명	회원연락처	운동부위	운동 이름	중량 및 횟수	날짜	운동시간	Operations
6	이지우	010-5555-8888	CHEST	푸쉬업	70kg/12개	2023-02-17	1시간 / 30분	상세보기 수정 삭제
7	강이진	010-4444-8888	CHEST	케틀벨	50kg/15개	2023-02-17	0시간 / 30분	상세보기 수정 삭제
8	강이진	010-4444-8888	CHEST	스쿼트	50kg/15개	2023-02-17	0시간 / 30분	상세보기 수정 삭제
9	이지우	010-5555-8888	BACK	스쿼트	50kg/12개	2023-02-16	1시간 / 30분	상세보기 수정 삭제
10	강이진	010-4444-8888	BACK	케틀벨	50kg/5개	2023-02-17	2시간 / 10분	상세보기 수정 삭제

작성이 완료될 시, 자동으로 리스트 페이지로 이동합니다.

HOME > 대시보드

등록

id	회원명	회원연락처	운동부위	운동 이름	중량 및 횟수	날짜	운동시간	Operations
6	이지우	010-5555-8888	CHEST	푸쉬업	70kg/12개	2023-02-17	1시간 / 30분	상세보기 수정 삭제

localhost:3000 내용:

정말 삭제하시겠습니까?

확인

취소

삭제 버튼을 클릭할 시, 알림을 띠우도록 했습니다.

성공하였습니다.

성공 시엔 다음과 같은 토스트를 띠우도록 설계했습니다.

i.e. of motor-activity or efficiency which has to be reckoned with, though it is bound to interfere, if it is to be a real "verisimilitude". Every true teacher must be a good teacher for instrumentality, make it inventiveness, make it what the child sees, guidance the will sees, tend to make it right.

