

THE portfolio - Long Story version

Introduction

- Name: Jiung Hahm
- Email: amoretspero (at) gmail (dot) com
 - Alternative: amoretspero (at) snu (dot) ac (dot) kr
- Github: <https://github.com/amoretspero>
- Leetcode: <https://leetcode.com/amoretspero>

Education

University

- 서울대학교 공과대학 컴퓨터공학부 (Undergraduate)
 - Enrollment: Mar. 2013.
 - (Expected) Graduation: Feb. 2022. (Bachelor's)
 - GPA: 3.69/4.3 (as of Aug. 2021.)

Military Service

- 산업기능요원 복무만료
 - Industrial Technical Personnel, Released from call on 19 Sep. 2020

Work Experience

(주)오너클랜

- 회사 주요 업무: 오픈마켓에서 활동하는 소규모 판매자들을 위한 도매 중개업
- 기간: 2017.08 - 2020.09
- 최종 직급: 기술연구소 / 대리

Internal / Open API 서비스 및 서버 개발

사내 여러 서비스를 개발하기 위한 기반이 되는 사내 API와 함께 기존 웹서비스 사용자들 중 API를 사용하고자 하는 사용자들에게 제공하는 Open API 서비스 개발을 진행했습니다.

회사가 제공하는 서비스의 특성상 내부에서 사용하는 기능의 subset이 외부 사용자가 사용하는 기능들이었기 때문에 사내 API 개발을 먼저 진행했습니다. 그 후, 더 세밀한 계정 및 권한 컨트롤, Quota 제한 기능, 더 많은 로드를 견딜 수 있는 구성을 추가해 Open API로 서비스를 오픈하는 과정을 거쳤습니다.

이 과정에서

- API 서비스에 대한 요구 사항의 정의
- API schema의 정의
- 서버 코드 구현
- Gitlab CI/CD 기반의 CI/CD 사용
- API 서비스 배포 및 유지관리

에 이르는 모든 부분을 직접 진행함으로써 대규모의 서비스라고 할 수는 없지만 당시 회사에서 경험할 수 있는 스케일에서의 API 서비스를 시작부터 끝까지 직접 경험해 볼 수 있었습니다. 구체적인 기술 사항에 대한 설명은 다음과 같습니다.

- Skills

- Server Runtime: [Node.js](#)
 - 회사가 제공하는 기존 웹 서비스는 PHP를 기반으로 구현되어 있었으나, 기존 코드에 덧붙일 경우에 증가할 수 있는 기술부채와 신규 백엔드로의 전환을 고려하여 Node.js를 선택했습니다.
 - Node.js에서의 웹 서버 프레임워크는 [Express.js](#)를 사용했습니다.
- Programming Language: [TypeScript](#)
 - 개발 시작 당시 표준이었던 ES2017 표준에 따른 JavaScript를 사용할 수도 있었으나 사내에서 서비스 개발의 편의를 위해 더 강한 type system을 가진 언어의 도입에 대한 요구가 있었습니다. 물론 VSCode등 에디터의 도움을 받을 수도 있었지만 코드 자체에서 더 명시적으로 typing을 지원하는 것은 훨씬 더 많은 편리함을 줄 것이라는 의견이 있었습니다. 개인적으로는 기존에 관심이 있던 Functional Programming Language들에서 지원하는 매우 강력한 type system에 매력을 느끼고 있었습니다. 결과적으로 JavaScript의 편리함과, Functional Programming Language보다는 못하지만 JavaScript에 비해서는 비교적 강력한 type system을 갖춘 TypeScript를 개발 언어로 결정하게 되었습니다. 이후 사내에서 소속 팀이 개발한 모든 JavaScript로 배포된 서비스는 모두 TypeScript로 소스코드가 작성되었습니다.
- API type: [GraphQL](#)
 - REST 방식의 API 서비스 대신 새롭게 시작하는 서비스에 당시에 어느 정도 안정화된 버전이 배포되기 시작한 GraphQL을 써보고 싶은 생각도 있었습니다. 또한 서비스의 특성상, 특정 entity에 대해 많은 정보를 구조적으로 제공하는 것이 API 서비스를 사용하는 입장에서도 편리했는데, 이 부분에서 GraphQL의 구조화된 쿼리와 응답이 매력적일 것으로 판단해 GraphQL을 선택하게 되었습니다.
 - Express.js와 함께, GraphQL API server를 구성하기 위해 [Apollo Server](#)를 사용했습니다.
- CI/CD: [Gitlab CI/CD](#)
 - 개발 초기에는 CI 없이 진행했지만 사내 서비스의 시작과 함께 배포까지 편하게 관리하려면 CI/CD를 구축해놓는 것이 편리하다고 판단해서 사내 코드 저장소로 사용한 Gitlab의 CI/CD 기능을 활용했습니다.
 - 대부분의 개발을 혼자 진행하고, 테스트 코드를 많이 작성하지는 못하는 등 CI의 이점을 받기는 어려웠지만 사내 테스트 및 배포 과정에서 CD 프로세스의 도움은 충분히 받을 수 있었습니다.
- Cloud Service Provider: [Google Cloud Platform](#)
 - GCP Kubernetes Engine를 사용해 프로토타입 개발 및 초기 서비스 운영 후, 네트워크 트래픽 등의 문제로 사내 서버로 이전해서 운영했습니다.

Open API까지 서비스를 시작한 후, 운영 시 일평균 2M requests, 초당 최대 150-200 requests 정도가 처리되었습니다. 사내에서 API 서비스 전용으로 할당받은 서버를 1개 사용했고, Node.js의 클러스터 기능과 PM2 패키지를 사용해서 서버 리소스를 최대한 효율적으로 사용할 수 있도록 구성했습니다. 또한 비교적 단순하면서 API call 수가 많은 부분은 Node.js Worker를 사용해서 별도의 thread pool이 처리하도록 했습니다. API request 수는 적지만 하나의 request가 많은 overhead를 갖는 경우에 다른 API request 처리에 병목을 발생시키지 않도록 하는 데 도움이 되었습니다.

이와 함께 Quota 제한을 위해 [Redis](#) 기반의 사용량 제한 기능을 추가해서 비정상적이거나 서버가 감당할 수 없는 트래픽을 막고, 소수의 사용자에 의해 리소스가 점유되지 않도록 했습니다.

API 서비스 오픈 후 회사 서비스 사용자들 중 Open API를 사용해서 개발할 수 있는 여력이 되는 사용자들이 API 서비스를 적극적으로 사용하도록 유도한 결과, 웹사이트의 불필요한 스크랩 트래픽을 80% 이상 감소시켜 웹서비스 퀄리티 유지에도 도움이 되었습니다.

회사 주요 서비스 리뉴얼 작업 - 백엔드

PHP로 구현되어있던 기존 웹 서비스에 새롭게 추가해야 하는 기능을 적용시키는 동시에 핵심 서비스 코드를 모두 새로 구현하는 리뉴얼을 진행했습니다. 크게 데이터 마이그레이션, 신규 API 서버 개발, 신규 프론트엔드 개발 부분으로 나누어 작업했고, 데이터 마이그레이션 일부와 신규 API 서버 개발을 담당했습니다.

각 부분에서 담당한 역할과 기술 사항은 다음과 같습니다.

- Skills

- Server Runtime: Node.js

- 신규 API 서버는 Node.js 기반으로 구현했습니다. 리뉴얼 프로젝트를 시작하기 전 이미 Open API가 Node.js 기반으로 개발되어 성공적으로 사용되고 있었기 때문에 개발 시간을 조금이라도 단축할 수 있었기 때문입니다.
- Programming Language: TypeScript
 - Open API를 구현할 때 TypeScript를 선택한 이유와 마찬가지로 큰 규모의 서비스일수록 vanilla JS보다 강력한 type system의 장점을 살릴 수 있다고 판단해 TypeScript를 선택했습니다.
- API type: GraphQL
 - Open API 서비스와 마찬가지로 expressjs를 기반으로 apollo server를 사용해 GraphQL API server를 구현했습니다. 선택한 이유는 마찬가지로 구조화된 쿼리 및 응답에서 얻을 수 있는 이점이었습니다. 이미 구조화된 데이터를 프론트엔드에서 사용할 때도 편리하고, 필요한 데이터를 API 서버에 요청할 때도 필요한 데이터 구조대로 쿼리할 수 있는 이점이 있었으며 프론트엔드 개발 담당자로부터 빠르게 개발하는 데 많은 도움이 되었다는 피드백도 받을 수 있었습니다.
- Database: MySQL
 - Database Engine은 기존에 사용하던 MySQL을 사용하되 데이터베이스 스키마는 처음부터 새로 구축했습니다.
- CI/CD: Gitlab CI
 - 개발 과정에서 여러 인원이 동시에 작업하면서 TypeScript type 공유 등을 위해 Lerna를 활용해 monorepo 방식으로 작업했습니다. 이 과정에서 작업 효율을 높이기 위해 Gitlab에서 제공하는 CI를 사용했습니다.
- 서비스 요구사항 분석 내용에 따른 DB 스키마 재구축
 - TypeORM 패키지를 이용한 스키마를 정의하는 작업에 참여했고,
 - 리뉴얼에 필요한 DB migration 코드 작성 일부를 담당했습니다.
- Node.js 및 TypeScript, GraphQL을 기반으로 한 API 서버 구현
 - GraphQL API type 정의 및 resolver 구현 등 GraphQL을 기반으로 한 API 서버에서 구현해야 하는 대부분을 담당했습니다.
 - API 서버에서 DB 데이터를 가져오는 부분은 TypeORM에서 제공하는 기능을 통해 해결하려고 했으나, 개발을 진행하면서 속도가 지나치게 느리다는 단점을 발견했습니다. 이를 해결하기 위해 상대적으로 DB data write/read operation time의 영향이 적은 데이터 수정 및 삭제 작업은 그대로 TypeORM을 사용하고, 빠른 응답이 필요한 데이터 읽기와 관련된 부분에서는 TypeORM 대신 node.js에서 사용할 수 있는 mysql driver(node-mysql)을 이용했습니다. 하지만 이 경우 직접 작성해야 하는 코드가 늘어나는 단점이 있었기 때문에 이 문제를 해결하기 위해 직접 GraphQL query에 대한 TypeScript decorator 기반의 query-format wrapper를 구현하고 프로젝트에서 사용했습니다. 이 wrapper를 사용한 결과 몇 개의 decorator를 추가하고 특수한 경우들에 대해서만 DB 관련 코드를 작성하면 되었고, 결과적으로 GraphQL API 서비스 개발시 data fetch/format 과정의 코드를 절반 이상 감소시킬 수 있었습니다.
 - query-format wrapper example:
 - DB schema code:

```

@Entity()
export class Feed {
  @PrimaryGeneratedColumn({ name: "key" })
  public key: number;

  @Column({ nullable: true })
  public registeredAdminKey?: number | null;

  @ManyToOne(() => Admin, { nullable: true, onDelete: "CASCADE" })
  @JoinColumn({ name: "registeredAdminKey" })
  public registeredAdmin?: Admin | null;

  @Column({ type: "enum", enum: FeedResourceType })
  public resourceType: FeedResourceType;

  @Column({ type: "simple-array" })
  public resourceKeys: number[];

  @Column({ type: "text" })
  public content: string;

  @Column({ type: "enum", enum: FeedSeverity })
  public severity: FeedSeverity;

  @Column({ nullable: true, type: "enum", enum: FeedEndpoint })
}

```

```

public endpoint?: FeedEndpoint | null;

@ValidateIf((obj, val) => val !== undefined && val !== null)
@ArrayMinSize(1)
@Column({ nullable: true, type: "simple-array" })
public audienceKeys?: number[] | null;

@CreateDateColumn(CreatedAtColumnOption)
public createdAt: Date;

@Column(UpdatedAtColumnOption)
public checkedAt: Date;

public constructor(data?: IFeedCreateOpt) {
    if (data) {
        this.registeredAdminKey = data.registeredAdminKey;
        this.resourceType = data.resourceType;
        this.resourceKeys = data.resourceKeys;
        this.content = data.content;
        this.severity = data.severity;
        this.audienceKeys = data.audienceKeys;
        this.endpoint = data.endpoint;
    }
}
}

```

■ GraphQL entity code:

```

@DbEntity(model.Feed) // This is for query-format wrapper to know DB model's `Feed` is connected to thi
@ObjectType({ description: "Feed", implements: Node })
export class Feed extends SingleKeyNode {
    @Field({ description: "The time when this feed is created." })
    public createdAt!: Date;

    @Field(() => GraphQLTimestamp, { nullable: true, description: "The time when this feed is checked." })
    public checkedAt?: Date | null;

    @Field(() => model.FeedResourceType, { description: "Type of resource." })
    public resourceType!: model.FeedResourceType;

    @Field(() => String, { description: "Feed content." })
    public content!: string;

    @Field(() => model.FeedSeverity, { description: "Severity of feed." })
    public severity!: model.FeedSeverity;

    @Field(() => model.FeedEndpoint, { nullable: true, description: "If endpoint is specified, target e
    public endpoint?: model.FeedEndpoint | null;

    public row: any;

    @DbColumn({ selects: ["audienceKeys"] }) // This is for query-format wrapper to query specific colu
    @Field(() => [ID], { nullable: true, description: "If targeted to specific users, targets' IDs." })
    public audienceIds(): string[] | null {
        return this.row.audienceKeys === null ? null : (this.row.audienceKeys as string).split(",")
            .filter((rk) => rk && !/\s*\$/.test(rk))
            .map((rk) => toGlobalId("User", [rk]));
    }

    @DbColumn({ selects: ["resourceKeys", "resourceType"] }) // This is for query-format wrapper to que
    @Field(() => [ID], { description: "If related resource exist, related resources' IDs." })
    public resourceIds(): string[] {
        return (this.row.resourceKeys as string).split(",")
            .filter((rk) => rk && !/\s*\$/.test(rk))
            .map((rk) => toGlobalId(this.row.resourceType, [rk]));
    }
}

```

```
}
```

- GraphQL resolver code:

```
@Resolver(Feed)
export class FeedResolver {
    private service: FeedService;

    public constructor() {
        this.service = new FeedService();
    }

    // #region FieldResolvers

    @Visibility(
        [Endpoint.Admin],
        FieldResolver(() => Admin, {
            complexity: fieldResolverComplexityEstimator(1),
            description: "Information about administrator who issued this feed, if available.",
            nullable: true,
        }),
    )
    public async registeredAdmin(
        @Ctx() ctx: IContext,
        @Info() info: GraphQLResolveInfo,
        @Root() feed: Feed,
    ): Promise<Admin | null> {
        if (feed.row.registeredAdminKey === null) {
            return null;
        }

        const r = await Q(ctx, info, { key: feed.row.registeredAdminKey }); // This single call will take care of parsing resolve-info, generating the right query and so on
        if (r.queryResult.registeredAdmin !== null) {
            return entityBuilder(r.queryResult.registeredAdmin, Admin); // This call will wrap plain-but-formatted object into a GraphQL-compliant one
        }

        return null;
    }

    // #endregion

    // #region Queries

    @Visibility(
        [Endpoint.Admin],
        Query(() => Feed, {
            nullable: true,
        }),
    )
    public async feed(
        @Ctx() ctx: IContext,
        @Info() info: GraphQLResolveInfo,
        @Arg("id", () => ID, { description: "ID of feed to fetch data." })
        id: string,
    ): Promise<Feed | null> {
        const r = await Q(ctx, info); // This single call will take care of parsing resolve-info, generating the right query and so on
        if (!r.queryResult.feed) {
            return null;
        }
        return entityBuilder(r.queryResult.feed, Feed); // This call will wrap plain-but-formatted object into a GraphQL-compliant one
    }

    // #endregion
}
```

이미지 제공 전용 서버 개발 및 CDN 연동

이 프로젝트를 진행하기 전에는 사용자가 이미지를 어떤 용도로든 업로드하게 되면, 실제로 사용될 이미지의 크기에 맞춰 미리 서버에서 변환한 후에 원본 이미지를 보관하지 않는 방식으로 처리했습니다. 그러나 원본 정보의 손실 및 미리 변환하는 overhead에서 나오는 사용자 경험 저하, 지원해야 하는 사이즈가 커질 경우에 대비하기 어려운 점 등의 문제가 있었습니다. 따라서 이를 해결하기 위해 원본 이미지를 보관함과 더불어 요청에 맞게 이미지를 resizing 및 transform 후 제공하는 서비스를 만들었습니다. 또한 이와 함께 트래픽 비용 절감을 위해 요청 가능한 이미지의 사이즈를 미리 정해 해당 이미지들이 caching이 가능하도록 한 다음 CDN까지 연결해서 실제 이미지 전용 서버로 들어오는 트래픽과 비용의 감소, 전체 서비스 단위에서의 비용 감소를 달성할 수 있었습니다.

- Skills
 - Server Runtime: Node.js
 - 빠르게 진행할수록 비용이 절감되는 프로젝트였고, 프로젝트 진행 시점에서 다수의 팀원들에게 익숙했기 때문에 Node.js를 선택했습니다.
 - Cloud Service Provider: Google Cloud Platform
 - Kubernetes Engine, Storage 등 필요한 서비스를 모두 제공하면서 가격이 다른 provider에 비해 저렴했기 때문에 선택했습니다.
 - CDN Provider: [Akamai CDN](#)(main) / GCP CDN(backup)
 - GCP CDN은 자주 일어나는 경우는 아니나 main CDN으로 사용하는 Akamai CDN 서비스가 일시적으로 서비스가 불가능할 때 회사 웹서비스의 downtime을 최소화하기 위해 backup으로 연동했습니다. 배포는 Gitlab CD를 사용해서 비상시에 최대한 빠르게 기준과 동일한 서비스를 backup CDN으로 배포할 수 있도록 설정했습니다.
 - CI/CD: Gitlab CI/CD
 - 개발 과정에서는 거의 대부분을 혼자 작업했기 때문에 CI는 크게 중요하지 않았으나 CD는 회사 서비스의 품질에 직결되는 프로젝트의 배포 안정성과 편리함을 동시에 제공해주었습니다.
- 단일 원본이미지로 여러 요청에 대응하여 URL-based dynamic resizing/transform을 제공하는 방식의 예시는 다음과 같습니다.
 - `image_name`이라는 이미지를 가로 `1920`, 세로 `1080` 으로 resizing한 후 `90` 도 회전한 다음에 `new_image.jpg`라는 이름으로 응답하기를 요청하는 경우
 - `https://cdn.example.com/image_name/resize/1920/1080/rotate/90/as/new_image.jpg`
- GCP Kubernetes Engine에서 제공하는 Cluster auto-scaling / Container pod auto-scaling를 적용해서 부하가 많으면 Cluster의 VM 수와 pod 숫자를 늘려서 대응하고 부하가 적으면 줄여서 비용을 절감하도록 했습니다.
 - Cluster와 Pod size의 경우 급격하게 로드가 증가할 때 일정 수준 이하의 response time을 확보하기 위해서 PAESSLER의 [Webserver Stress Tool](#)를 사용해 stress test를 간단하게나마 진행한 결과를 반영했습니다. 예상 가능한 최대 부하를 기준으로 테스트를 진행했을 때 Up scaling이 진행되는 중에도 response time이 허용치를 넘어가는 경우는 일정 수준 이하가 되도록 Cluster와 pod의 minimum size를 지정했습니다.

Examples:

- Original image:

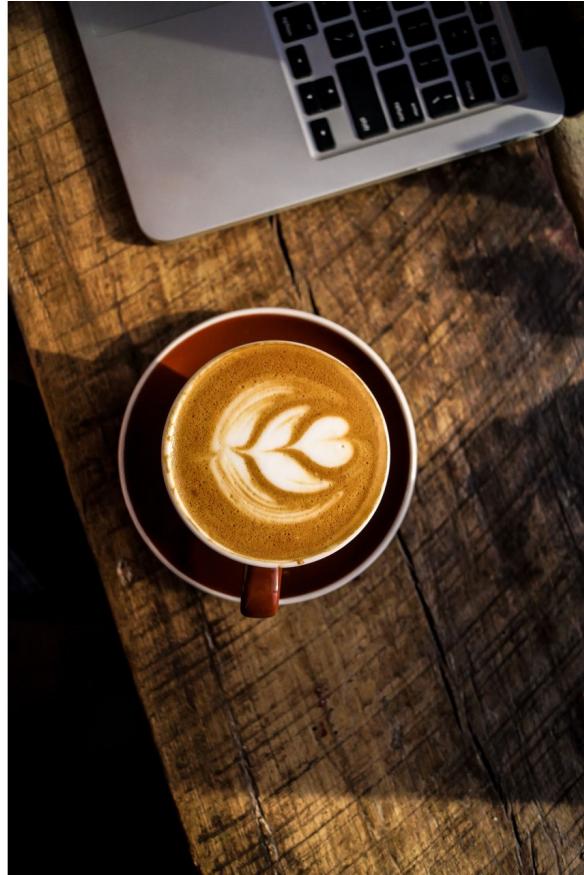


- Credit: Photo by [Nolan Issac on Unsplash](#)
- Resize to {height: 1280, width: 1920}
 - https://cdn.example.com/image_name/resize/1920/1280/as/new_image.jpg
 - Image:



- Resize to {height: 1280, width: 1920} , rotated clockwise by 90 degree. Pad if necessary
 - https://cdn.example.com/image_name/resize/1920/1280/rotate/90/as/new_image.jpg

- o Image:



- Resize to square, with `{height: 1920, width: 1920}` pad if necessary.
 - o `https://cdn.example.com/image_name/square_resize/1920/as/new_image.jpg`

- Image:



상품 유사도 확인 프로그램 개발

회사 웹서비스에서 사용자들에게 제공하는 상품들의 정보가 매우 유사하거나 동일한 내용인 경우, 사용자들이 이를 외부 오픈 마켓에 등록하는 용도로 사용할 때 페널티를 받기 쉽습니다. 또한 비슷한 종류의 상품은 하나의 그룹으로 묶어서 취급하고자 하는 사용자들의 요구도 있었습니다. 이러한 문제점과 요구사항을 해결하기 위해 웹서비스에 등록된 상품들의 상호간 유사도를 체크해서 유사도에 따라 조치를 취할 수 있도록 도와주는 프로그램을 개발한 프로젝트입니다.

- Skills

- Server Runtime: [Dotnet Core](#) (Currently evolved into .NET, the successor of .NET Framework)
 - 프로젝트의 프로토타입은 Node.js로 진행했습니다. 다만 Parallel processing이 가능한 부분들의 속도를 높이는 과정에서 Node.js보다는 Dotnet Core 위에서 C#으로 개발하는 것이 좀 더 편하게 개선이 가능해 선택했습니다.
- Programming Language: [C#](#)
 - Dotnet Core를 사용하는 이상 C# 또는 F# 정도가 선택지였으나, 빠른 개발을 위해 조금 더 익숙한 C#을 선택했습니다.
- Cloud Service Provider: Google Cloud Platform
 - 사내 서버 자원 대용으로 GCP의 Compute Engine만 사용했습니다.

개발한 프로그램은 상품 데이터 중 미리 선택된 항목들에 대해 [Min Hash](#)과 [LSH\(Locality Sensitive Hashing\)](#) algorithm을 적용해 유사도를 계산합니다. Cron으로 일정 주기마다 전체 상품에 대해 계산하도록 구현되어있고, 계산된 유사도를 바탕으로 상

품을 크게 3개의 부류로 나누게 됩니다. 매우 높은 유사도 기준을 넘어서는 경우 동일상품으로 의심하여 관리자에게 리포트하고, 이보다 유사도는 낮지만 아예 관련이 없는 정도의 유사도보다는 높은 경우 유사한 상품으로 판단하여 사용자에게 정보를 제공합니다.

(주)스카이시드

- 회사 주요 업무: 웹서비스 개발 및 마케팅 외주 / 학원업
- 기간: 2013.12 - 2017.08
- 최종 직급: 스타트업으로, 별도의 직급 체계는 없었음

2016.01 - 2017.08

- 학원 수강생 관리 시스템 제작
 - Ruby on Rails를 기반으로 제작한 웹서비스입니다. 크게 수강생 관리, 결제 이력 관리, 상담 내역 관리 등 학원에서 수강생을 관리하는 데 필요한 기능들을 제공하는데, 이 중에서 직접 개발에 참여한 부분은 다음과 같습니다.
 - 수강생 및 결제/등록 이력 DB 설계
 - 대량 문자메시지 발송 프로그램 제작 및 OpenAPI 연동
 - 혼자 연습삼아 만들었던 것들을 제외하고는 실제로 쓰기 위해 학교 외부에서 진행한 첫 프로젝트인만큼 많은 부분을 구현하지는 않았으나 여러가지 기본적인 개념을 익힐 수 있었던 프로젝트 참여였습니다.
- 자기소개서 작성 지원 웹서비스 개발
 - .NET Framework와 C#으로 개발한 프로젝트입니다. 규모는 작으나 DB 스키마 설계부터 서버 작업 및 Razor Syntax 기반으로 간단한 프론트엔드까지 직접 제작했습니다. 제공하는 기능은 다음과 같습니다.
 - 사용자의 경우 회원가입 / 자기소개서 작성 및 저장 / 자기소개서 인쇄 기능
 - 관리자의 경우 사용자가 동의한 자기소개서 열람 / 회원 관리 / 자기소개서 작성 관련 정보 편집 기능
 - 당시에 React/Vue 등을 접하지 못해 별도의 프론트엔드 프레임워크는 사용하지 않았고 ASP.NET에서 제공하는 Razor syntax를 활용해서 MVC 구조로 개발했습니다.
 - Screenshots:

The screenshot shows the homepage of the self-introduction service. It features a sidebar with navigation links like '대학/학과/전형 정보', '서울대학교', '대학 구분', '자기소개서 제출 일정', and '인재상'. The main content area has tabs for '1년 문정', '2년 문정', and '3년 문정'. Below these tabs is a large input field for a self-introduction text, with a character count indicator '147 characters / 38 words'. At the bottom of the text area are buttons for '입력있어요!' and '저장하기'.

This screenshot shows the '자기소개서 작성 시 유의사항' (Things to consider when writing a self-introduction) section. It contains several bullet points with tips for users, such as '자기소개서는 지원자 본인이 작성해야 하며, 사실에 입각하여 정직하게 지원자 자신의 능력이나 특성, 경험 등을 기술하여야 합니다.', '자기소개서에 기재된 내용에 대한 사실 확인을 요청할 경우 지원자는 적극 협조하여야 합니다.', and '자기소개서는 지원자 본인의 경험, 대학 적성, 외국어 실적 기재, 기타 부정한 사실 등의 짐증을 위해 유사도 검색을 실시하고, 해당 사실이 발견될 경우 불법적 차리되며 함께 삭제하도록 인증이 필요할 수 있습니다.'.

This screenshot shows the '대학 정보' (University Information) section. It lists three universities with their logos, names, and basic information. The first university is 서울대학교 (Seoul National University), the second is 연세대학교 (Yonsei University), and the third is 고려대학교 (Korea University). Each entry includes fields for '이름', '구분', '자기소개서 제출 시작', '자기소개서 제출 마감', and '입학처 링크'.

This screenshot shows the '유의사항' (Notes) section. It contains two bullet points: '본 페이지에서 제공하는 대학 정보는 각 대학 입학처 홈페이지를 참조하여 작성한 정보입니다.' and '기정 정책한 정보를 보기 위해서는 해당 대학 입학처를 참고하여 주시기 바랍니다.'

- 도서 원고 버전관리 웹서비스
 - .NET Framework와 C#으로 개발한 프로젝트입니다.
 - 스타트업에서 운영하던 학원에서 사용할 자체 제작 교재들, 혹은 외부 출판사와 계약하여 판매되는 도서들의 원고 작업을 할 때 작업물의 버전 관리를 도와주는 서비스입니다.
 - 대부분의 원고는 HWP 파일로 작업이 진행되었고, 컴퓨터공학 비전공자가 당연히 대다수였기 때문에 git을 통해 관리하기가 용이하지 않았습니다. 또한 원고 작성에 참여하는 사람들이 각자 자신의 파일을 들고 있으면 자신의 작업 분량에 대해서도 최신 파일과 이전 이력들을 기록하기 쉽지는 않고, 다른 사람들의 작업 내용을 쉽게 보기 가 어려웠습니다.
 - 따라서 이런 문제를 해결하기 위해 자신이 작업한 파일을 업로드만 하면 자동으로 모든 버전들이 보관되고 각자가 업로드한 파일과 도서 제작 작업 과정에 따른 이력도 쉽게 파악할 수 있도록 서비스를 제작했습니다.
- 고등학교 수학 기출문제 분류 관리 웹서비스
 - .NET Framework와 C#으로 개발한 프로젝트입니다.
 - 교재 제작과 학원 자체적으로 사용하기 위해 수능과 모의고사의 수학 기출문제들을 자체적인 기준에 따라서 분류하는 작업을 진행했는데, 이 작업 과정에서 어떤 분류에 해당하는지를 판단하는 것에만 집중할 수 있도록 나머지 과정을 최대한 줄여주는 방향으로 개발된 서비스입니다.
 - 문제 유형을 분류할 때 각 문제의 유형 정보를 가장 큰 분류부터 가장 작은 분류까지 기록하고 출처도 정확하게 기록해야 합니다. 또한 출처는 범위가 정해져 있지만, 유형 분류의 경우 작업을 진행하다보면 새롭게 유형을 만들거나 기준에 분류했던 것들을 빠르게 다시 보고 재분류해야 하는 경우도 많았습니다. 또한 유형 데이터도 사람이 기억하고 있기에 많은 데이터였고 문제의 내용도 자주 찾아보고 읽을 필요가 있었습니다.
 - 이러한 요구사항을 반영해서 사람은 최대한 문제 분류에만 집중할 수 있도록 분류 입력이나 다른 부분을 최대한 자동화하고 최대한 적은 수의 화면에서 필요한 데이터를 많이 볼 수 있도록 구성한 서비스를 제작했습니다.
- 고등학생 대상 수학 강의

2013.12 - 2016.01

- 마케팅 외주 업무 지원

Personal / Toy projects

Periodic Dataloader

- Typescript로 만든, 특정 주기마다 load function을 실행할 수 있는 dataloader입니다. GraphQL 방식의 API를 구현할 때 graphqljs에서 제공하는 dataloader에 해당 기능이 없어서 직접 사용하려고 만든 뒤 일반적으로 사용할 수 있도록 약간의 변형을 거쳐서 public package로 배포했습니다.
- 기존 Dataloader에 없던 기능 중 추가한 2개의 기능은 다음과 같습니다.
 - Periodic execution of batch load function: Load를 기다리는 key가 있을 경우에, 주어진 시간 간격마다 batch load function을 실행하는 기능입니다. 현재 [graphql-dataloader v2.0.0](#)에서는 이 기능을 `batchScheduleFn` 기능을 통해 제공합니다.
 - Unique keys only to batch load function: Load를 기다리는 key가 있을 경우에, 모든 key를 batch load function에 넘겨주는 대신 unique key들만 골라서 이들에 대해서만 batch load function을 실행하게 하는 기능입니다. 대부분의 RDBMS를 포함해 많은 경우에는 dataloader보다 아래 레이어에서 이를 충분히 효율적으로 처리할 수 있지만 그렇지 않은 레이어에 의존하는 경우 이 기능을 사용하면 불필요한 오버헤드를 줄일 수 있고 중복된 key가 매우 많은 경우에는 네트워크 사용량도 줄일 수 있는 이점을 제공합니다.
- [Github Repository](#)

FSharp Linear Algebra

- F#을 처음 접하고 간단한 라이브러리를 구현해보고 싶어 대학교 수업시간에 배운 선형대수의 개념들을 일부나마 직접 구현해 본 라이브러리입니다. 배운 내용의 복습과 F#에 조금 더 익숙해지는 데 도움이 되었습니다.
- [Github Repository](#)

Hml Equation Parser

- 한글 문서에서 수식 부분만 추출해내어 사용할 일이 있어 기존에 다른 곳에서 개발해서 오픈소스로 공개한 repository를 fork해서 기능을 더 보완했습니다.
- 기본적으로 한글에서 수식을 입력할 때 사용하는 string을 LaTeX math string으로 변환해주는 기능입니다. 자세한 내용은 아래 repository를 참조해주시면 됩니다.
- [Github Repository](#)
- Example:
 - 이미지의 아래 부분에 있는 내용을 한글 Html 파일에 입력한 후, 해당 파일을 parsing하면 위의 결과물을 얻을 수 있습니다. Html 렌더링은 KaTeX를 사용했습니다.
 - Image:

수식 입력 테스트

문제 DB에 입력할 수식을 테스트할 수 있습니다. Latex 수식은 MathJax 라이브러리를 이용해 변환됩니다.

1. 두 벡터 $\{\overrightarrow{a}\} = \left(-1, 2 \right)$, $\{\overrightarrow{b}\} = \left(2, -3 \right)$ 에 대하여 $\{\overrightarrow{a}\} + \{\overrightarrow{b}\}$ 는? [2점]

$\left(-1, -1 \right)$ $\left(-1, 1 \right)$ $\left(-1, 2 \right)$ $\left(1, -1 \right)$ $\left(1, 2 \right)$

2. $\sin \theta = \frac{1}{3}$ 일 때, $\cos(\theta + \frac{\pi}{2})$ 의 값은? [2점]

$\frac{7}{9}$ $-\frac{2}{3}$ $-\frac{5}{9}$ $-\frac{4}{9}$ $-\frac{1}{3}$

3. H_5 의 값은? [2점]

21 22 23 24 25

Convert!

Reset!

1. 두 벡터 $\vec{a} = (-1, 2)$, $\vec{b} = (2, -3)$ 에 대하여 $\vec{a} + \vec{b}$ 는? [2점]

$(-1, -1)$ $(-1, 1)$ $(-1, 2)$ $(1, -1)$ $(1, 2)$

2. $\sin \theta = \frac{1}{3}$ 일 때, $\cos\left(\theta + \frac{\pi}{2}\right)$ 의 값은? [2점]

$-\frac{7}{9}$ $-\frac{2}{3}$ $-\frac{5}{9}$ $-\frac{4}{9}$ $-\frac{1}{3}$

3. H_5 의 값은? [2점]

21 22 23 24 25

Skills

- 굵게 처리된 내용: 최근 자주 사용해서 익숙한 skill set
- 이탤릭 처리된 내용: 일에서 자주 사용하지는 않았으나 소규모 프로그램 개발 및 유지가 가능한 skill set
- 그 이외: 주로 대학교 수업 프로젝트 등에 사용했거나, 한 번쯤 다뤄봤던 정도의 skill set

Programming Languages

- **JavaScript / TypeScript**
- **Python**
- **SQL(MySQL)**
- **C#**
- **F#**
- **C/C++**
- **Java**

- PHP

Frameworks / Databases

- Node.js
- MySQL
- *Dotnet Framework / Dotnet Core*
- ArangoDB

Others

- Google Cloud Platform
- Nginx
- React
- Azure

Interests

- Programming Languages
 - TypeScript
 - F#
 - Python
- Frameworks
 - Dotnet Core
 - Node.js
 - Tensorflow
- Others
 - Functional Programming
 - React/Recoil
 - GraphQL
 - Machine Learning(incl. Deep Learning)
 - Quantum Computing

Thank you for reading this portfolio.

You can see the github version at: [Github Link](#)