

포트폴리오





How 보다 Why 가 익숙한 개발자, 전영빈입니다


멋있어 보이는 기술보다 실제로 필요한 기술을 찾고 적용하는 것을 좋아합니다.

새로운 기술에도 관심이 많지만 기술의 동작 원리와 이유를 깊이 있게 공부하는 것을 더 좋아하는 편입니다.

팀 구성원들의 작업 효율을 높이는 것에 보람을 느낍니다. 팀의 원활한 개발을 위해 라이브 환경과 유사한 테스트 채널과 로그 관제 시스템을 구성한 경험이 있습니다.

 binkorea2@gmail.com

 [GitHub](#) /  [Blog](#)

 +82 10-9037-9122

Education

삼성 청년 SW 아카데미 10기

(2023.07 ~ 2024.06)

Spring Boot와 Vue.js를 기반으로 전반적인 웹 개발 과정을 학습하고 총 3번의 팀 프로젝트를 진행했습니다.

강릉원주대학교 컴퓨터공학과

(2017.03 ~ 2023.02)

운영체제, 네트워크, 데이터베이스 등 기본적인 컴퓨터과학 이론에 대해 학습했습니다.
강릉원주대학교 부속 해양센서네트워크시스템 기술연구센터에서 학부 연구생으로 활동하며 수중 환경에서 주로 사용하는 다중 매체 통신에 대해 연구했습니다.

Award

SSAFY 자율 주제 프로젝트

2023.05

"개인화된 습관 관리 서비스, 바른 생활"으로 SSAFY 자율 주제 프로젝트에서 우수상을 수상했습니다.

SSAFY 4차 산업 기술 프로젝트

2023.04

"해양 생태계를 배경으로 한 웹소켓 기반 멀티 플레이 게임, Lost Marine"으로 메타버스 및 게임 주제 프로젝트에서 우수상을 수상했습니다.

강릉원주대학교 컴퓨터공학과 프로그래밍 경진대회

2022.12

2022년 강릉원주대학교 컴퓨터공학과에서 주최한 프로그래밍 경진대회에서 최우수상을 수상했습니다.

한국정보기술학회 추계대학생 논문경진대회

2021.11

"데이터마이닝을 통한 직장인 퇴사 분석"을 주제로 2021년 한국정보기술학회 추계대학생 논문경진대회에서 우수논문상을 수상했습니다.

Skill

Backend

Java / Spring Boot
Spring Data JPA
Spring Security
TypeScript / Node.js

Infra

AWS EC2 / RDS / S3
Docker / Docker Compose
Jenkins
Nginx
ELK

Tool

IntelliJ IDEA
VS Code
Vim

Collaborations

Git
Jira / Confluence
Gerrit

Database

MySQL
Redis

Project

바른생활

- 프로젝트 개요
- 사람들과 경쟁하고 비교하는 삶에서 벗어나 자신의 습관을 기록하고 유지하며 집중할 수 있도록 도와주는 해빗 트래커 웹 애플리케이션
 - 어떤 습관을 만들어야 할 지 모를 사용자들을 위한 습관 추천 서비스
 - 달성한 습관을 깃허브 잔디와 같은 스트릭을 통해 확인하고 빠진 스트릭을 보충할 수 있는 리커버리 기능
 - 한 달 간의 습관 달성 기록을 정리하여 확인할 수 있는 리캡 서비스

기간 2024.04 ~ 2024.05 (6주)

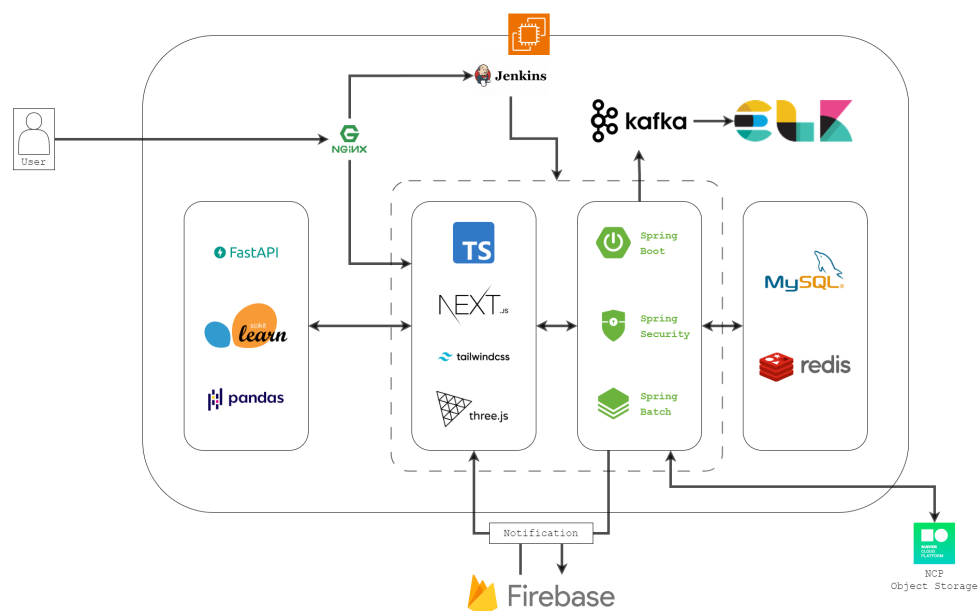
참여인원 6명 (FE 3, BE 3)

기술 스택 Spring Boot, Spring Data JPA, Spring Security, Mysql, Redis
Docker, Jenkins, Nginx, NCP Object Storage

GitHub <https://github.com/iamyoungbin/BareunSanghwal>

서비스 링크 <https://bareun.life>

아키텍처



파사드 패턴을 적용하여 스트릭 서비스 구현

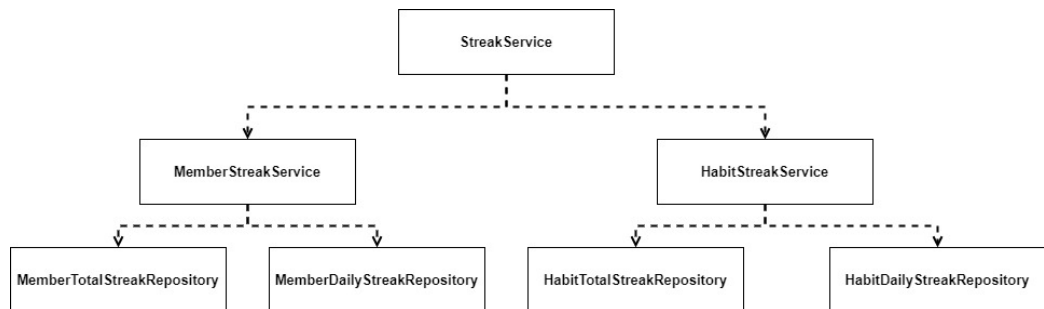
유저가 얼마나 오래 습관을 달성했는가를 기록하는 스트릭 기능 구현을 위해 다음과 같은 복잡한 요구사항이 있었습니다.

- 멤버가 설정한 각 습관에 대해 그 습관이 현재 몇 일, 최대 몇일 연속하여 달성되었는가에 대한 정보를 제공할 것.
- 멤버가 한 개라도 습관을 달성한 날짜가 현재 몇 일, 최대 몇 일 연속하여 달성되었는가에 대한 정보를 제공할 것.
- 멤버의 스트릭이 달성되지 않은 날에 리커버리를 사용하면 해당 날짜에 습관을 달성한 것으로 간주하고, 이후 날짜의 스트릭 정보를 그에 맞게 갱신할 것.

스트릭 도메인에 연관된 서비스가 많아 습관 달성과 같은 이벤트가 일어났을 때, 다른 도메인에서 스트릭 관련 인터페이스를 일일이 호출해야 하는 상황이 발생했습니다.

이를 해결하기 위해 스트릭 도메인에 **파사드 패턴**을 적용하여 어떤 레포지토리를 의존하는지에 따른 두 개의 저수준 인터페이스와 이를 통합하는 하나의 고수준 인터페이스로 구성했습니다.

이러한 구조를 통해 다른 도메인에서 스트릭 도메인을 참조하더라도 하나의 인터페이스만 참조하게 되므로 한 번의 호출만으로 해당 도메인의 기능을 사용할 수 있게 되었습니다. 또한, 더 이상 저수준 인터페이스의 변화에 다른 도메인들이 영향받지 않게 되므로 객체 지향 설계의 중요 원칙 중 하나인 **OCP**를 지킬 수 있었습니다.



프론트엔드 빌드 최적화

Docker Cache와 Multi-Staging를 통해 프론트엔드 빌드 시간을 6분 30초에서 2분 30초로 단축했습니다.

초기에 최적화를 적용하지 않고 단순히 빌드했을 때, 6분 30초라는 시간이 측정되었고 체감적으로 너무 긴 시간이라고 생각했습니다. 이 때문에 어떻게 하면 기존의 과정을 축소 혹은 생략시킬 수 있을지 고민했습니다. 그러던 와중 next.js에서 .next/cache를 이용해 빌드 과정을 최적화한다는 내용을 알게 되었습니다.

이를 적용하여 맨 처음 빌드 시에 생성된 .next/cache 파일을 시스템에 보관해두었다가 이후 **Docker Cache**를 통해 빌드 시에 이를 가져와 재활용했습니다. 그 결과 next.js 빌드 과정에서 1분에 달하는 시간을 단축시킬 수 있었습니다.

또한, 빌드 시에 프론트엔드 이미지 파일이 6.04GB로 매우 거대하게 생성되었습니다. 실제 서비스 운용 시에 필요한 .next 나 node_modules등의 정적 파일만이 아닌 전체 코드가 이미지에 포함되었기 때문이라고 생각했습니다.

이를 해결하기 위해 우선 **Multi-Staging**을 통해 도커 파일을 의존성 설치를 위한 스테이지, 프론트엔드 빌드 스테이지, 서비스 실행 스테이지로 나누었습니다. 또한, 서비스 실행 스테이지에서 실제 실행에 필요한 정적 파일만을 포함시켜 도커 이미지 생성 시에 필요 없는 소스가 포함되지 않도록 했습니다. 그 결과 프론트엔드 도커 이미지를 2.01GB까지 줄일 수 있었습니다.

		Declarative: Tool Install	Checkout	Before Build	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 24s)		109ms	2s	1s	2min 19s	809ms
#169 5월 21일 11:03	1 commit	116ms	1s	1s	2min 34s	808ms
#168 5월 21일 10:07	3 commits	118ms	2s	1s	2min 39s	804ms
#167 5월 21일 00:32	1 commit	107ms	1s	1s	2min 9s	779ms

Lost Marine

- 프로젝트 개요
- 해양 생태계 보호와 멸종 위기 생물 조명을 주제로 한 웹소켓 기반 서버이별 멀티 플레이 게임
 - 멸종위기 해양생물 캐릭터를 조작하여 바다 속을 탐험
 - 다양한 플레이어를 캐릭터와 진화 기능을 통해 폭 넓은 사용자 경험 제공
 - 다른 플레이어와의 전투 및 중립 아이템을 통한 경쟁과 전략적 요소 추가

기간 2024.02 ~ 2024.04 (6주)

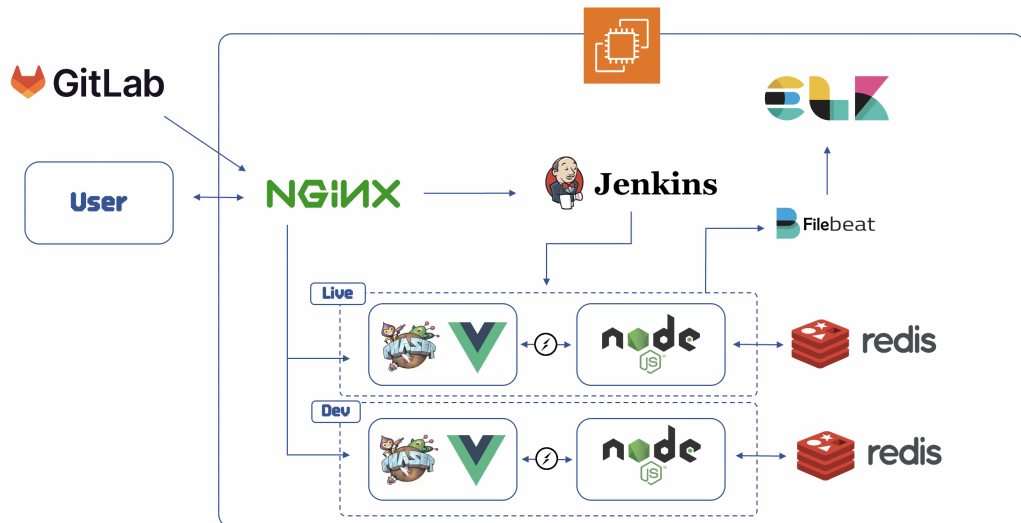
참여인원 6명 (FE 3, BE 3)

기술 스택 Typescript, Node.js, Express, Socket.IO, Redis
Docker, Jenkins, Nginx, ELK, Filebeat

GitHub <https://github.com/lost-marine/lost-marine-server>

서비스 링크 <https://lostmarine-io.site/>

아키텍처



담당업무

지형지물 관리 및 액터 스폰 위치 구현

스폰이 가능한 액터인 플레이어와 플랑크톤의 스폰 이벤트 발생 시에 해당 위치에 다른 액터가 존재하면 안된다는 제약사항이 있었습니다. 플레이어의 스폰 위치가 1000여 개, 플랑크톤의 스폰 위치가 2000여 개로 스폰 이벤트 발생 시마다 최악의 경우 약 3000번의 반복문을 통해 스폰 위치를 찾아야 하는 상황이 발생했습니다.

이를 해결하기 위해 **R-Tree** 기반의 영역 데이터 관리가 가능한 라이브러리인 **RBush**를 이용했습니다. R-Tree는 B-Tree에서 다차원 검색이 가능하도록 확장된 형태의 자료구조로, "특정 좌표에서 N만큼의 거리 이내에 존재하는 노드가 있는 지"와 같은 질의에 빠르게 답을 줄 수 있습니다.

해당 자료구조를 통해 스폰 위치 탐색의 시간 복잡도를 $O(N)$ 에서

$O(\log N)$ 까지 줄일 수 있었습니다.

라이브 채널과 테스트 채널을 분리하여 인프라 구성

소켓 정보를 통해 사용자를 기억하고 매 주기마다 사용자에게 게임 환경 정보를 보내는 Stateful한 서버에서 안정적으로 개발하기 위해 라이브 환경과 유사한 테스트 채널이 반드시 필요하다고 느꼈습니다.

이를 위해 서브 도메인을 사용해 실제 서비스를 제공하는 **Live채널**과 테스트를 위한 **Dev채널**로 분리하고, Dev채널에 **CI/CD Pipeline**을 구축하여 통합된 코드를 즉시 테스트할 수 있도록 구성했습니다.

또한, Dev채널에서 테스트가 완료된 버전을 릴리즈 버전 단위로 묶어 Live채널에 배포했습니다.

Apr 2
iamyoungbin
v1.1.0
a9a6ed0
Compare

ver 1.1.0

Feature

- 대시 기능을 추가했습니다.
- 다른 위치 간에 순간이동을 할 수 있는 포탈을 추가했습니다.
- 이로인 효과를 얻을 수 있는 각종 아이템을 추가했습니다.
- 게임 도움말을 추가했습니다.
- 백그라운드 음악을 끄고 켤 수 있게 수정했습니다.

Assets



ELK를 사용한 로그 관제 시스템 구성

웹소켓을 사용한 Stateful한 서버 구성에서 중요한 점은 하나의 호스트에서 문제가 발생하더라도 다른 호스트들은 정상적으로 게임을 플레이하는 것입니다. 서버에 문제가 발생하는 것과 동시에 서버가 멈추지 않기 때문에 그만큼 에러의 발견이 늦고, 에러를 즉각적으로 확인할 수 있는 시스템이 중요하다고 생각했습니다.

따라서, **Filebeat**와 **ELK**를 사용하여 라이브 서버에서 발생한 로그를 수집하고 관리할 수 있는 로그 관제 시스템을 구성하여 서버의 상태를 즉각적으로 확인할 수 있도록 했습니다.

또한, Trigger와 Webhook을 이용하여 에러 로그 발생 시에 팀에서 사용하는 협업 툴인 Mattermost에 알람과 에러 내용을 전달하여 서버 장애에 즉각적으로 반응할 수 있도록 했습니다.

다중 매체 통신에서의 전송 속도 제어 프로토콜 설계 및 구현

- 프로젝트 개요
- 다중 매체 통신에서 통신 매체의 전송 속도가 지속적으로 변화함에 따라 통신 속도가 다른 노드 간에 패킷 손실이 발생할 수 있음
 - 이를 방지하기 위해 노드의 메시지 큐 포화도에 따라 주변 노드의 전송 속도를 제어할 수 있는 프로토콜을 설계 및 구현

기간 2022.04 ~ 2022.06 (10주)

참여인원 5명

스택 C, STM32F429

GitHub <https://github.com/iamyoungbin/Design-and-implementation-of-transmission-control-protocol-in-multimedia-communications>

담당업무

흐름 제어를 위한 프로토콜 동작 과정 설계

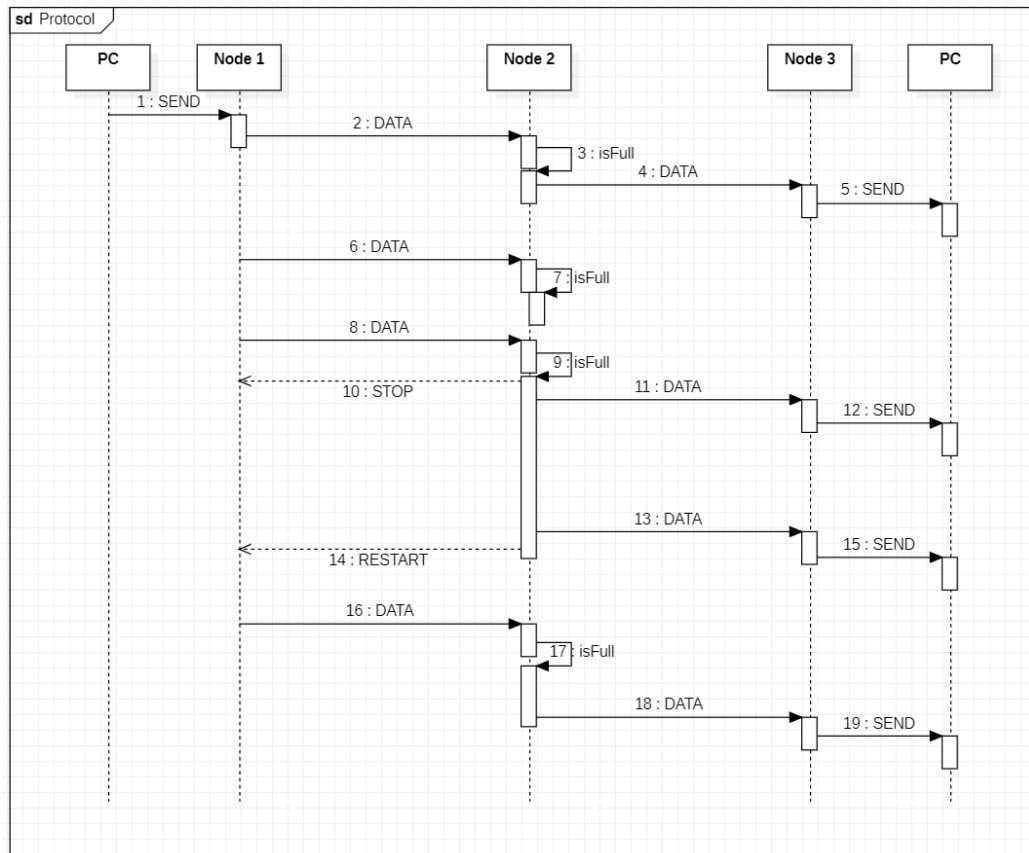
메세지 큐의 포화도에 따라 주변 노드의 패킷 전송을 제한하기 위한 프로토콜 동작 과정을 설계했습니다.

우선 각 노드는 패킷을 수신하고 메세지 타입을 확인하여 **데이터 메세지**인지 **제어 메세지**인지 확인합니다.

메세지 타입이 데이터 타입이라면 패킷으로부터 페이로드를 읽어들이 메세지 큐에 저장합니다. 이 때, 만약 메세지 큐가 70% 이상 포화 상태라면 송신 노드에게 패킷을 그만 보낼 것을 부탁하는 STOP 메세지를 보냅니다. 이후, 주기적으로 큐의 상태를 확인하고. 메세지 큐가 비게 되면 STOP 메세지를 보낸 노드들에게 다시 송신을 허락하는 RESTART 메세지를 보냅니다.

메세지 타입이 STOP 메세지일 경우 발신지 노드로의 데이터 전송을 RESTART 메세지를 받을 때까지 중단합니다. 이후, RESTART 메세지를 받을 경우 송신지 주소로의 데이터 전송을 재개합니다.

이와 같은 과정을 통해 노드 전송 속도 차이로 인한 패킷 손실을 방지할 수 있었습니다. 또한, 프로젝트에서는 메세지 큐의 임계값을 70% 정도로 넉넉하게 잡았지만, 이를 구할 수 있는 알고리즘을 적용하여 더 효율적으로 동작할 수 있을 것으로 예상합니다.



기능 구현을 위한 패킷 설계

시퀀스 차트를 통해 프로토콜의 동작 과정을 설계한 이후, 이를 구현할 수 있도록 패킷에 필요한 필드를 추가했습니다.

- **Length** : 패킷의 전체 길이를 나타내는 필드로 한 패킷의 길이가 최대 10byte부터 32byte이므로 필드 크기를 1byte로 정의했습니다. 패킷 수신 시 패킷의 전체 길이를 파악 하기 위해 Length 필드를 제일 앞에 두어 가장 먼저 읽을 수 있게 배치했습니다.
- **Mss_Type** : 메시지 타입을 나타내는 필드입니다. 프로토콜에서 필요로 하는 메시지는 세 종류의 메시지 타입으로 한 개의 데이터 타입 메시지와 두 개의 제어 메시지 타입으로 총 세 종류의 메시지입니다.
- **SRC_Address, DEST_Address** : 발신지와 송신지 주소를 나타내는 필드입니다. IPv4에서 IP 주소는 32bit로 표현되므로 4byte를 할당했습니다.
- **Payload** : 데이터를 저장하는 필드입니다. 구현 시 사용될 최대 문자열의 길이가 최대 22자이므로 이를 고려하여 메모리 22byte를 할당했습니다.

Length 1 byte	Mss_Type 1 byte	SRC_Address 4 byte	DEST_Address 4 byte	Payload 0 ~ 22 byte
------------------	--------------------	-----------------------	------------------------	------------------------

데이터마이닝을 통한 직장인 퇴사 분석

- 프로젝트 개요
- 신입 사원의 1년 이상 퇴사율이 20%에 달한다는 것을 알게 되었고 어떤 요인이 가장 밀접한 영향을 갖는지 알아보기 위해 프로젝트 진행
 - 분류 모델을 검증하고 평가하는 역할을 수행
 - 해당 연구를 수행하며 게재한 논문에서 제3저자로 기재

기간 2021.10 ~ 2021.11 (5주)

참여인원 6명

스택 Python, Pandas, scikit-learn

논문 링크 <https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE10664701>

담당업무

오차행렬과 ROC 곡선을 통한 모델 예측 성능 평가

하이퍼 파라미터 튜닝 이후 모델의 정확도가 79%로 측정되었고, 이를 실제 모델의 예측 성능이라고 판단할 수 있을지 의문을 가졌습니다. 프로젝트에서 사용한 데이터셋은 참과 거짓의 비율이 76:24인 불균형 데이터셋이기 때문에 모델이 그저 참이라고 예측하지만 해도 76%의 정확도를 가지기 때문입니다.

이를 보완하기 위해 오차 행렬을 통해 모델의 성능을 다시 평가했습니다. 오차 행렬은 모델의 예측 유형과 빈도를 보여주는 지표입니다. 평가 결과, 양성으로 예측된 값 중 실제로 양성 값의 비율인 **정밀도(Precision)**는 **0.968**이고, 전체 양성 값 중 실제로 양성으로 예측한 값의 비율인 **재현율(Recall)**은 **1.0**이었습니다. 따라서, 분류기가 1년 이상 근무하는 사람에 대해서 100% 확률로 긍정 예측하고, 해당 긍정 예측에 대해 96.8%의 정확도를 가짐을 알 수 있었습니다.

위의 평가 지표를 통해 **ROC 곡선**을 만들어 모델의 성능을 시각적으로 확인할 수 있도록 했습니다. 또한, 불균형 데이터셋에서 성능을 평가하기 위해 주로 사용하는 **F1 스코어**가 **0.98**임을 확인하여 모델이 충분히 높은 성능을 가졌다고 평가할 수 있었습니다.

```
[[26231  0]
 [ 857 7336]]
```

