GSTACK · FOUNDATION + PBL EXTENSION

Codex
+ Claude

코드를 시키기 전에,
개발팀을 설계한다.

Codex와 Claude Code에 gstack을 설치하고, 문제 탐색부터
기획·개발·리뷰·QA·출시·회고까지 하나의 프로젝트로 연결하는 실습형 강의록

THINK → PLAN → REVIEW → TEST

수강 대상 · Git 기초 학습자 / 실습 환경 · Codex 또는 Claude Code

01 / COURSE MAP

강의 목표와 180분 운영안

이 강의의 목표는 명령어 암기가 아니라, AI에게 역할과 검증 절차를 부여하는 방법을 익히는 것입니다.

Codex 또는 Claude Code를 설치하고 로그인한다.

선택한 개발 환경에 gstack을 설치하고 상태를 확인한다.

office-hours, plan-eng-review, review, qa-only 스킬을 실행한다.

AI 출력의 사실과 가정을 구분하고 Git diff와 실제 동작으로 검증한다.

00-15	오리엔테이션	gstack과 실행 환경의 관계 이해
15-30	환경 점검	Git, Bun, 계정과 터미널 확인
30-55	Codex	설치, 로그인, 첫 읽기 작업
55-80	Claude Code	설치, 로그인, 첫 읽기 작업
80-90	gstack	host별 설치와 상태 확인
90-100	휴식	설치 문제 보완
100-145	문제와 계획	office-hours와 plan-eng-review
145-165	리뷰와 QA	review와 qa-only 비교
165-180	비교와 회고	도구별 결과와 PBL 적용 정리

02 / CONCEPT

gstack은 무엇인가

gstack은 AI 코딩 에이전트에 제품, 기술, 리뷰, QA, 보안, 출시 역할을 부여하는 스킬 기반 개발 워크플로입니다. 새로운 언어나 프레임워크가 아니며, AI 모델 자체도 아닙니다.

Think	제품 탐색	누구의 어떤 문제인가?	office-hours
Plan	기술 계획	어떻게 만들고 검증할까?	plan-eng-review
Review	코드 검토	변경에 어떤 위험이 있는가?	review
Test	사용자 검증	실제로 어떤 상황에서 실패하는가?	qa / qa-only
Ship	출시	테스트와 문서가 준비됐는가?	ship
Reflect	회고	다음 작업에 남길 학습은?	retro / learn

역할 분리가 필요한 이유

CEO 관점은 이 기능이 정말 필요한지 묻습니다.

엔지니어 관점은 데이터 흐름과 실패 경로를 묻습니다.

리뷰어 관점은 변경 코드의 위험과 누락을 찾습니다.

QA 관점은 실제 사용자가 재현할 수 있는 실패를 찾습니다.

핵심 원칙

AI가 더 많이 작업하게 만드는 것이 아니라, 서로 다른 관점으로 작업을 검토하게 만드는 것이 gstack의 핵심입니다.

03 / BEFORE CLASS

수업 전 준비와 공통 환경

강사는 수업 전에 계정 접근 여부와 설치 시간을 확인해야 합니다. 학습자는 Codex 또는 Claude Code 중 하나를 선택하며, 두 환경을 모두 사용할 수 있다면 비교 실습까지 수행합니다.

필수 확인

- Git
- Bun 1.4.2 이상
- 인터넷 연결
- 터미널
- Codex 또는 Claude Code 계정
- ZIP 압축 해제 기능

공통 명령

```
git --version
bun --version
```

macOS	네이티브 터미널	설치 후 새 터미널 열기
Linux	Bash 또는 Zsh	브라우저 도구의 시스템 의존성 확인
Windows	WSL2	Codex, Claude, gstack을 같은 환경에 설치

왜 ZIP으로 배포하는가

- 원격 실습 저장소를 미리 만들거나 공개할 필요가 없습니다.
- 학습자가 단계별 자료를 필요한 시점에 받을 수 있습니다.
- 압축을 푼 뒤 로컬 Git 저장소를 만들어 변경 이력을 확인할 수 있습니다.

03A / WINDOWS BEGINNER

처음 보는 용어부터 이해하기

Windows 학습자는 Windows 네이티브 설치를 기본 과정으로 사용합니다. WSL2는 Linux 환경이 필요한 경우에만 선택합니다.

터미널	컴퓨터에 글자로 명령을 전달하는 창
PowerShell	Windows에 기본으로 포함된 터미널
명령어	PowerShell에 붙여넣고 Enter를 눌러 실행하는 문장
PATH	Windows가 codex, bun 같은 프로그램을 찾는 위치 목록
WSL	Windows 안에서 Ubuntu Linux 환경을 실행하는 선택 기능
host	gstack 스킬을 설치할 AI 도구. 예: codex, claude

명령을 입력하는 방법

Windows 키를 누르고 PowerShell을 검색해 실행합니다.

화면의 PS C:₩Users₩사용자이름> 표시는 입력하지 않습니다.

강의록의 명령 상자 안쪽만 복사해 붙여넣습니다.

명령을 붙여넣은 뒤 Enter를 누르고 완료될 때까지 기다립니다.

설치가 끝나면 PowerShell을 닫고 새로 열어야 PATH가 반영될 수 있습니다.

환경 선택

Windows 네이티브	처음 설치하고 PowerShell을 사용하는 경우	기본 과정
WSL2 Ubuntu	Linux 환경 통일이 필요하거나 네이티브가 차단된 경우	선택 과정

Windows 필수 도구 설치

1. PowerShell 열기

Windows 키를 누르고 PowerShell을 검색해 실행합니다. 관리자 권한은 WSL 설치 때만 필요하며, 일반 설치는 먼저 일반 PowerShell로 시도합니다.

2. Git과 Node.js

```
winget install --id Git.Git -e
winget install OpenJS.NodeJS.LTS
```

설치가 끝나면 PowerShell을 닫고 새로 엽니다.

3. Bun

```
powershell -c "irm bun.sh/install.ps1|iex"
```

4. 설치 결과 확인

```
git --version
node --version
npm --version
bun --version
```

예상 결과

각 명령 아래에 숫자로 된 버전이 표시됩니다.

'명령을 찾을 수 없습니다'가 나오면 PowerShell을 완전히 닫고 다시 엽니다.

Bun은 gstack 설치에 필요하고 Node.js는 Windows 도구 실행과 프로젝트 실습에 사용됩니다.

03C / WINDOWS NATIVE 2

Codex와 Claude Code 설치

Codex 설치

```
powershell -ExecutionPolicy ByPass -c "irm https://chatgpt.com/codex/install.ps1 | iex"

codex --version
codex
```

처음 codex를 실행하면 화면 안내에 따라 ChatGPT 계정 등의 제공되는 인증 방식으로 로그인합니다.

Claude Code 설치

```
winget install Anthropic.ClaudeCode

claude --version
claude doctor
claude
```

Claude Code 사용이 가능한 계정이 필요합니다. Codex만 실습한다면 Claude Code 설치는 건너뛰어도 됩니다.

한 번에 점검

```
git --version
node --version
npm --version
bun --version
codex --version
claude --version
```

성공 기준

선택한 AI 도구의 버전 번호가 표시됩니다.

codex 또는 claude를 실행하면 대화 입력 화면이 열립니다.

각 도구에서 로그인 절차가 완료됩니다.

Windows에 gstack 등록하기

1. gstack 내려받기

```
git clone --single-branch --depth 1 https://github.com/garrytan/gstack.git "$HOME₩gstack"
cd "$HOME₩gstack"
```

2. 사용할 AI 도구에 등록

```
.₩setup.ps1 --host codex
.₩setup.ps1 --host claude
.₩setup.ps1 --status
```

한 도구만 사용하는 경우 해당 host 명령만 실행합니다.

스크립트 실행이 차단될 때

```
powershell -ExecutionPolicy Bypass -File .₩setup.ps1 --host codex
powershell -ExecutionPolicy Bypass -File .₩setup.ps1 --host claude
```

3. 새 세션에서 확인

Codex와 Claude Code를 실행 중이었다면 모두 종료합니다.

새 PowerShell을 열고 프로젝트 폴더에서 codex 또는 claude를 실행합니다.

Codex에서는 gstack-office-hours 스킬을 사용해 달라고 요청합니다.

Claude Code에서는 /office-hours를 실행합니다.

03E / ZIP PRACTICE

Windows에서 ZIP 실습 시작하기

1. ZIP 압축 풀기

강의 페이지에서 단계별 ZIP을 다운로드합니다.

다운로드한 파일을 마우스 오른쪽 버튼으로 누릅니다.

'모두 추출' 또는 '압축 풀기'를 선택합니다.

압축을 푼 폴더 안의 project 폴더를 엽니다.

2. 현재 폴더에서 PowerShell 열기

파일 탐색기 주소 표시줄을 클릭하고 powershell을 입력한 뒤 Enter를 누릅니다. 열린 창의 경로가 project 폴더인지 확인합니다.

3. 로컬 Git 기준점 만들기

```
git init
git add .
git commit -m "실습 시작 상태"
git status
```

Git 이름 오류가 나올 때

```
git config user.name "student"
git config user.email "student@example.local"
git commit -m "실습 시작 상태"
```

이 설정은 현재 실습 폴더 안에서만 사용됩니다.

4. AI 도구 시작

```
# Codex
codex

# Claude Code
claude
```

03F / WSL2 OPTION

WSL2 Ubuntu 선택 과정

WSL은 Windows 안에서 Ubuntu Linux 환경을 실행하는 기능입니다. gstack의 필수 조건은 아니며, Linux 개발 환경으로 통일할 때 선택합니다.

1. 관리자 PowerShell

```
wsl --install
```

설치가 끝나면 재부팅합니다. 시작 메뉴에서 Ubuntu를 실행해 사용자 이름과 비밀번호를 만듭니다. 비밀번호를 입력할 때 화면에 문자가 보이지 않는 것은 정상입니다.

2. Ubuntu 안에서 필수 도구

```
sudo apt update
sudo apt install -y git curl unzip
curl -fsSL https://bun.com/install | bash
source ~/.bashrc
```

3. AI 도구와 gstack도 Ubuntu 안에 설치

```
curl -fsSL https://chatgpt.com/codex/install.sh | sh
curl -fsSL https://claude.ai/install.sh | bash
git clone --single-branch --depth 1 https://github.com/garrytan/gstack.git ~/gstack
cd ~/gstack
./setup --host codex
./setup --host claude
./setup --status
```

환경을 섞지 않기

> Codex만 Windows에 설치하고 gstack은 WSL에 설치하면 서로 다른 홈 폴더를 사용합니다. 모두 Windows에 설치하거나 모두 WSL2 Ubuntu에 설치해야 합니다.

03G / TROUBLESHOOTING

초급자 설치 오류 해결표

codex, claude, bun 명령을 찾을 수 없음	PowerShell을 완전히 닫고 새로 연 뒤 버전을 다시 확인
setup.ps1 실행이 차단됨	ExecutionPolicy Bypass 명령으로 해당 파일만 실행
Git 커밋에서 이름과 이메일을 요구함	현재 실습 폴더에 student 이름과 로컬 이메일 설정
gstack 스킬이 보이지 않음	AI 도구를 재시작하고 setup.ps1 --status 확인
Windows 보안 기능이 바이너리를 차단함	보안 설정을 끄지 말고 강사에게 알린 뒤 WSL2 경로 사용
Claude Code 로그인이 불가능함	계정에서 Claude Code 사용 권한이 있는지 확인하거나 Codex 트랙 선택
현재 폴더를 모르겠음	PowerShell에서 pwd를 실행하고 project 폴더인지 확인

강사에게 보여줄 정보

```
pwd
git --version
bun --version
codex --version
claude --version
```

비밀번호, 인증 코드, API 키는 화면 공유나 제출물에 포함하지 않습니다.

Codex 설치와 첫 실행

Codex는 터미널의 로컬 프로젝트를 읽고 파일 수정과 명령 실행을 수행합니다. 첫 실행 시 ChatGPT 계정 등 제공되는 인증 방식으로 로그인합니다.

macOS · Linux · WSL

```
curl -fsSL https://chatgpt.com/codex/install.sh | sh
```

npm 대안

```
npm install -g @openai/codex
```

설치 확인과 로그인

```
codex --version
codex
```

첫 읽기 작업

```
이 프로젝트의 목적, 주요 파일, 실행 방법을 설명해 줘.
아직 파일은 수정하지 마.
```

gstack 호출 방식

Codex에서는 스킬 이름과 원하는 제한을 자연어로 명확히 지정합니다.

```
gstack-office-hours 스킬을 사용해 이 프로젝트를 검토해 줘.
아직 코드를 수정하지 마.
```

확인 포인트

- 현재 프로젝트를 인식하는가?
- 실제 파일을 근거로 설명하는가?
- 수정 금지 요청을 지켰는가?
- 작업 후 git status에 예상하지 않은 변경이 없는가?

04A / CODEX PROMPT INSTALL

Codex에게 gstack 설치를 요청하기

Codex CLI 자체 설치와 로그인은 먼저 완료해야 합니다. 이후 Codex는 현재 환경을 점검하고 gstack 설치 명령을 제안·실행할 수 있습니다. 파일 쓰기나 네트워크 접근은 승인 정책에 따라 사용자 확인을 거칩니다.

1. 실습 폴더에서 Codex 시작

```
# Windows PowerShell
mkdir "$HOME₩gstack-class" -ErrorAction SilentlyContinue
cd "$HOME₩gstack-class"
codex

# macOS · Linux · WSL
mkdir -p ~/gstack-class
cd ~/gstack-class
codex
```

2. 환경 점검 프롬프트

```
gstack 설치를 준비하고 있어.
현재 운영체제와 셸, Git, Bun, Codex 버전과 기존 gstack 설치 여부를
점검해 줘. 아직 설치하거나 파일을 수정하지 말고, 확인한 사실,
부족한 항목, 설치 경로와 실행할 명령 계획만 보여 줘.
```

점검 결과 예시

환경	Windows 11 / PowerShell 7
확인됨	Git 2.x, Bun 1.x, Codex 로그인
부족함	gstack 미설치
계획	공식 저장소를 사용자 홈/gstack에 받은 뒤 Codex host 등록

왜 점검과 설치를 나누는가

잘못된 운영체제 명령, 기존 폴더 덮어쓰기, 예상하지 않은 설치 경로를 실행 전에 발견하기 위해서입니다.

04B / CODEX PROMPT INSTALL

승인하며 설치하고 첫 스킬 실행하기

gstack 설치 프롬프트

```
방금 점검한 결과를 기준으로 gstack을 Codex host에 설치해 줘.

1. https://github.com/garrytan/gstack 공식 저장소만 사용해.
2. Windows면 $HOME₩gstack과 setup.ps1을,
macOS·Linux·WSL이면 ~/gstack과 ./setup을 사용해.
3. 기존 폴더가 있으면 덮어쓰거나 삭제하지 말고 멈춰서 알려 줘.
4. 파괴적인 명령은 실행하지 마.
5. 네트워크나 홈 폴더 쓰기는 먼저 승인을 요청해.
6. --host codex 등록과 --status 확인까지 실행해.
7. 실행 명령, 설치 경로, 상태를 요약해.
```

승인 창 체크

명령이 git clone 또는 gstack setup인가?

저장소 주소가 garrytan/gstack 공식 주소인가?

대상이 내 홈 폴더의 gstack 경로인가?

삭제, 강제 초기화, 예상 밖 시스템 변경이 없는가?

첫 사용 프롬프트

```
Codex를 새로 시작한 뒤:

gstack-office-hours 스킬을 사용할 수 있는지 확인해 줘.
사용 가능하면 이 프로젝트의 대상 사용자, 사용자 문제,
가장 작은 핵심 기능, 검증되지 않은 가정을 정리해 줘.
아직 파일을 수정하지 마.
```

예상 완료 결과

```
설치 경로: C:₩Users₩student₩gstack
등록 host: codex
상태: installed / skills linked
다음 단계: Codex를 재시작하고 gstack-office-hours 실행
```

비밀번호, 인증 코드, API 키는 프롬프트나 제출물에 넣지 않습니다.

Claude Code 설치와 첫 실행

Claude Code는 gstack의 정식 지원 환경입니다. 주요 gstack 스킬을 슬래시 명령으로 직접 호출할 수 있습니다.

Homebrew

```
brew install --cask claude-code
```

npm 대안

```
npm install -g @anthropic-ai/claude-code
```

설치 확인과 로그인

```
claude --version
claude doctor
claude
```

첫 읽기 작업

```
이 프로젝트의 목적, 주요 파일, 실행 방법을 설명해 줘.
아직 파일은 수정하지 마.
```

gstack 호출 방식

```
/office-hours
```

명령 실행 후 프로젝트 아이디어와 검토할 범위를 입력합니다.

계정 확인

수업 전 Claude Code 사용이 가능한 계정인지 확인합니다. 무료 Claude.ai 플랜만으로는 Claude Code에 접근할 수 없습니다.

호스트별 gstack 설치

gstack 저장소를 한 번 내려받은 뒤 원하는 실행 환경에 스킬을 등록합니다. 설치 후에는 Codex 또는 Claude Code를 새 세션으로 다시 시작합니다.

1. 공식 저장소 내려받기

```
git clone --single-branch --depth 1 ₩
+ https://github.com/garrytan/gstack.git ~/gstack
cd ~/gstack
```

2. 선택한 host 설치

```
# Codex만 설치
./setup --host codex

# Claude Code만 설치
./setup --host claude
```

3. 설치 상태 확인

```
./setup --status
```

Host	codex 또는 claude
Tier	Claude는 full, Codex는 experimental
Version	설치된 gstack 버전
Path	스킬이 등록된 실제 경로

문제 해결

스킬이 보이지 않으면 실행 중인 세션을 종료하고 다시 시작합니다.

Windows에서는 설치한 셸과 실행 셸이 같은지 확인합니다.

gstack 폴더에서 ./setup --status를 다시 실행합니다.

07 / PRACTICE 1

ZIP 실습 자료와 로컬 Git 기준점

각 단계의 ZIP을 내려받아 압축을 풉니다. 원격 저장소가 없어도 로컬 Git을 사용하면 AI가 바꾼 내용을 정확히 확인할 수 있습니다.

프로젝트 기준점 만들기

```
cd 다운로드한-실습폴더/project
git init
git add .
git commit -m "실습 시작 상태"
git status
```

샘플 앱 실행

```
python3 -m http.server 8000
```

브라우저에서 http://localhost:8000을 엽니다.

첫 스킬 실행 - Codex

```
gstack-office-hours 스킬을 사용해 이 프로젝트를 검토해 줘.
대상 사용자, 사용자 문제, 핵심 가치, 최소 기능, 검증할 가정을 정리해 줘.
아직 코드를 수정하지 마.
```

첫 스킬 실행 - Claude Code

```
/office-hours
```

이어서 같은 검토 범위와 수정 금지 조건을 입력합니다.

실습 결과

- AI가 제안한 대상 사용자
- 가장 작은 핵심 기능
- 검증되지 않은 가정
- AI가 잘못 이해한 내용
- 사람이 결정해야 할 내용

08 / PRACTICE 2

기술 계획 검토

문제 정의 결과를 실제 구현 가능한 기술 계획으로 바꿉니다. 이 단계에서도 구현은 하지 않습니다.

Codex

```
gstack-plan-eng-review 스킬을 사용해 기술 계획을 검토해 줘.
변경 구성요소, 데이터 흐름, 예외 상황, 테스트 항목,
기술적 위험과 구현 순서를 포함해 줘. 아직 구현하지 마.
```

Claude Code

```
/plan-eng-review
```

검토 기준

프로젝트 근거	실제 존재하는 파일과 구조를 사용했는가?
가정	존재하지 않는 기능이나 데이터를 가정하지 않았는가?
실패 경로	입력 오류, 저장 실패, 중복 동작을 포함했는가?
테스트	완료 여부를 판단할 수 있는 검증 항목이 있는가?
범위	수업 시간 안에 수행 가능한가?

색상 분류 활동

초록: 파일에서 확인한 사실

노랑: 추가 확인이 필요한 가정

빨강: 잘못됐거나 과도한 범위

09 / PRACTICE 3

코드 리뷰와 QA

review는 코드 변경의 위험을 보고, qa-only는 실제 사용자 흐름에서 재현 가능한 문제를 찾습니다.

작업 전 확인

```
git status
git diff
```

Codex 리뷰

```
gstack-review 스킬을 사용해 프로젝트를 검토해 줘.
잠재 버그, 예외 처리, 테스트 누락, 불필요한 복잡성을 확인해 줘.
수정하지 말고 심각도와 근거만 보고해 줘.
```

Claude Code 리뷰

```
/review
```

리뷰 스킬은 일부 문제를 자동 수정할 수 있으므로, 이번 실습에서는 수정하지 말고 보고만 하도록 명시합니다.

QA 범위

- 정상 입력
- 빈 문자열과 공백
- 매우 긴 입력
- 빠른 연속 클릭
- 새로고침
- 작은 화면
- 오류 발생 시 안내

QA 스킬

```
# Codex
gstack-qa-only 스킬을 사용해 위 범위를 테스트해 줘.

# Claude Code
/qa-only
```

작업 후 확인

```
git status
git diff
```

10 / LIFECYCLE MAP

기획부터 QA까지 이어지는 전체 과정

앞의 3시간 과정은 설치와 첫 스킬 경험을 위한 기초 모듈입니다. 이후 PBL에서는 각 단계의 산출물을 다음 단계의 입력으로 넘겨 한 프로젝트를 완성합니다.

문제 탐색	office-hours	누구의 문제인가	문제 정의와 가정
명세	spec	어디까지 만들 것인가	시나리오와 인수 조건
계획	plan-ceo / design / eng	어떤 접근을 승인할 것인가	제품·디자인·기술 계획
개발	기본 에이전트 + Git	어떤 작은 변경을 반영할까	diff, 실행 결과, 커밋
검증	review / investigate / qa	어떤 결함을 고칠까	리뷰와 QA 증거
출시	cso / benchmark / ship	남은 위험을 수용할까	문서, PR, 회고

수정 권한 규칙

- 문제 탐색·명세·계획은 수정 금지
- 개발부터 승인한 범위만 수정
- 리뷰와 qa-only는 보고 전용
- qa는 승인한 결함만 수정
- ship과 배포는 브랜치·테스트·권한 확인 후 실행

문제를 찾고 완료 기준을 고정하기

실습 시나리오

Focus List 앱에 빈 입력 방지, 중복 방지, 긴 제목 처리 기능을 추가합니다. 기능을 바로 만들지 않고 사용자 문제와 검증 기준을 먼저 작성합니다.

문제 탐색 - Codex

```
gstack-office-hours 스킬을 사용해 대상 사용자, 현재 대안,
핵심 문제, 최소 검증 기능, 검증되지 않은 가정을 정리해 줘.
아직 파일을 수정하지 마.
```

문제 탐색 - Claude Code

```
/office-hours
```

명세 - Codex

```
gstack-spec 스킬을 사용해 빈 입력·중복 항목·긴 제목 처리의
정상/예외 시나리오와 인수 조건을 작성해 줘.
아직 구현하지 마.
```

명세 - Claude Code

```
/spec
```

통과 조건

- 문제 정의가 기능명이 아니라 사용자의 불편을 설명한다.
- 포함 범위와 제외 범위가 모두 있다.
- 제3자가 같은 기준으로 성공과 실패를 판단할 수 있다.

12 / PLAN

제품·디자인·기술 계획 잠그기

제품	gstack-plan-ceo-review	/plan-ceo-review	무엇을 빼야 하는가?
디자인	gstack-plan-design-review	/plan-design-review	모든 상태가 보이는가?
기술	gstack-plan-eng-review	/plan-eng-review	데이터와 실패가 어떻게 흐르는가?
연속 검토	gstack-autoplan	/autoplan	세 관점이 충돌하지 않는가?

기술 계획 요청

```
gstack-plan-eng-review 스킬을 사용해 현재 명세를 검토해 줘.
변경 파일, 데이터 흐름, 실패 경로, 테스트 항목, 구현 순서를
포함하고 아직 구현하지 마.
```

사람이 확인할 것

- 제안한 파일과 함수가 실제로 존재하는가?
- 오류 상태와 정상 상태를 모두 다루는가?
- 수업/프로젝트 시간 안에 가능한 범위인가?
- 각 단계에 되돌리기와 검증 방법이 있는가?

13 / BUILD

작게 구현하고 Git으로 통제하기

구현은 gstack 스킬을 억지로 호출하는 단계가 아닙니다. 승인한 계획을 기본 AI 에이전트에게 작은 단위로 맡기고, gstack은 앞뒤의 검토와 검증을 강화합니다.

한 기능만 요청

```
승인한 명세와 기술 계획을 기준으로 먼저 빈 입력 방지만 구현해 줘.
기존 구조를 유지하고 변경 후 실행 가능한 검증을 수행해 줘.
다른 기능은 아직 건드리지 마.
```

변경 확인과 기능별 커밋

```
git status
git diff
# 브라우저에서 직접 동작 확인
git add .
git commit -m "feat: prevent blank tasks"
```

반복 순서

빈 입력 방지 구현 → 검증 → 커밋

중복 항목 방지 구현 → 검증 → 커밋

긴 제목 처리 구현 → 반응형 확인 → 커밋

중지 조건

명세 밖 파일이 바뀌거나 검증 명령이 실패하면 커밋하지 않습니다. diff를 확인하고 원인을 설명한 뒤 계획을 다시 승인합니다.

14 / REVIEW & INVESTIGATE

변경 위험과 근본 원인 찾기

코드 리뷰

```
# Codex
gstack-review 스킬을 사용해 현재 diff를 검토해 줘.
심각도, 파일 근거, 재현 방법만 보고하고 수정하지 마.

# Claude Code
/review
```

근본 원인 조사

```
# Codex
gstack-investigate 스킬을 사용해 중복 항목이 생기는 원인을 조사해 줘.
재현과 증거를 먼저 제시하고 승인 전에는 수정하지 마.

# Claude Code
/investigate
```

리뷰	심각도, 파일 위치, 실패 조건, 영향이 연결됨
조사	재현 → 증거 → 가설 → 확인된 원인 순서
수정	승인한 원인만 최소 범위로 변경
재검증	같은 절차로 실패가 사라지고 회귀가 없음

사람의 체크포인트

AI가 지적한 모든 항목을 고치지 않습니다. 실제 근거와 사용자 영향이 있는 결함만 승인하고, 스타일 취향과 범위 확장은 분리합니다.

보고 전용 QA와 수정 포함 QA

먼저 보고만 받기

```
# Codex
gstack-qa-only 스킬을 사용해 정상 입력, 빈/공백, 중복, 긴 제목,
연속 클릭, 새로고침, 모바일을 테스트해 줘. 수정하지 말고
재현 절차와 기대/실제 결과를 작성해 줘.

# Claude Code
/qa-only
```

승인한 결함만 수정하고 재검증

```
# Codex
gstack-qa 스킬을 사용해 승인한 결함만 수정하고
같은 사용자 흐름으로 재검증해 줘.

# Claude Code
/qa
```

빈/공백	항목이 추가되지 않고 이해 가능한 안내가 보임
중복	정의한 규칙에 따라 차단되며 이유가 보임
긴 제목	데스크톱과 모바일에서 레이아웃이 유지됨
연속 클릭	의도하지 않은 중복 동작이 없음
새로고침	정의한 저장 정책과 실제 동작이 일치함

보안·성능·문서로 출시 준비

보안	gstack-cso	/cso	위험, 근거, 우선순위
성능	gstack-benchmark	/benchmark	전후 측정과 회귀 여부
문서	gstack-document-release	/document-release	코드와 일치하는 문서

Focus List 점검 예

사용자 입력이 HTML이나 스크립트로 실행되지 않는가?

긴 목록에서도 입력과 렌더링이 충분히 빠른가?

README의 실행 방법·기능·제한이 실제 앱과 같은가?

중대한 미해결 위험이 있다면 출시를 멈췄는가?

선택 원칙

모든 프로젝트에서 모든 스킬을 실행할 필요는 없습니다. 사용자 데이터와 인증이 없으면 보안 범위를 줄일 수 있고, 성능 기준이 없다면 먼저 측정 가능한 기준을 정합니다.

전달하고, 배포하고, 학습 남기기

PR 준비	gstack-ship	/ship	브랜치·diff·테스트 확인
병합·배포	gstack-land-and-deploy	/land-and-deploy	실제 배포 권한과 환경 필요
배포 감시	gstack-canary	/canary	운영 URL과 기준값 필요
회고	gstack-retro / learn	/retro / /learn	사람의 판단도 함께 기록

수업에서 원격 저장소가 없을 때

ship을 실제 실행하는 대신 ZIP의 SHIP-CHECKLIST.md를 작성합니다. 무엇을 왜 바꿨는지, 어떻게 검증했는지, 남은 위험과 되돌리기 방법을 정리하면 같은 사고 과정을 연습할 수 있습니다.

최종 제출물

- 문제 정의와 인수 조건
- 제품·디자인·기술 계획
- 기능별 Git diff와 검증 결과
- 리뷰·근본 원인·QA 보고서
- 보안·성능·문서 준비표
- 출시 체크리스트와 회고

문제 탐색·명세·계획 결과 예시

> 예시는 정답지가 아니라 내 결과에 사용자, 근거, 검증 조건이 빠지지 않았는지 비교하는 기준입니다.

office-hours 예시

대상 사용자	수업과 과제를 동시에 관리하는 교육생
핵심 문제	기록한 일 중 지금 처리할 한 가지를 빠르게 구분하기 어렵다.
최소 검증 기능	3초 안에 할 일을 추가하고 완료 상태를 바꾼다.
검증할 가정	사용자는 분류 기능보다 빠른 입력을 먼저 원한다.

spec 예시

Given 입력창에 공백만 입력되어 있고 / When 추가 버튼을 누르면 / Then 항목 수는 증가하지 않으며 '내용을 입력하세요'가 표시된다.

계획 예시

변경 파일: app.js 입력 검증·중복 비교, styles.css 오류·긴 제목 상태

데이터 흐름: 입력 → trim → 빈 값 검사 → 중복 검사 → 배열 추가 → 렌더링

실패 경로: 공백, 대소문자만 다른 중복, 100자 이상 제목, 빠른 연속 클릭

구현 순서: 입력 검증 → 중복 규칙 → 화면 상태 → 회귀 테스트

구현·리뷰·QA 결과 예시

구현 후 Git

```
$ git diff --stat
app.js | 12 ++++++++++--
1 file changed, 10 insertions(+), 2 deletions(-)

$ git status --short
M app.js
```

브라우저 확인

공백 입력: 항목 0개 유지, 안내 표시

정상 입력: 항목 1개 추가

기존 완료 버튼: 정상 동작

review 예시

P1 · app.js에서 사용자 제목을 innerHTML로 삽입해 입력이 HTML로 실행될 수 있다. 악성 HTML 입력으로 재현 가능하며 textContent 사용이 필요하다.

qa-only 예시

공백 입력	추가 안 됨	빈 항목 추가	FAIL
중복 입력	안내 후 차단	2개 추가	FAIL
긴 제목·모바일	줄바꿈 유지	버튼 밀림	FAIL
정상 입력	1개 추가	1개 추가	PASS

출시 준비·회고 결과 예시

보안·성능·문서

보안	사용자 입력 렌더링 P1 1건을 textContent로 수정하고 재검증 완료
성능	항목 100개 렌더링 18ms → 20ms, 기준 50ms 이내 PASS
문서	실행 방법, 중복 규칙, 새로고침 후 데이터가 사라지는 제한 추가
판단	중대한 미해결 위험 없음 → 출시 후보

ship 결과

변경 요약: 빈 입력·중복을 차단하고 긴 제목의 모바일 레이아웃을 보완

검증: 인수 조건 4개, QA 시나리오 8개 PASS, 보안 P1 재검증 완료

남은 제한: 데이터는 브라우저 메모리에만 있어 새로고침 후 사라짐

retro 결과

구현 전에 '중복'의 정의를 확정했더라면 재작업 1회를 줄일 수 있었다. 다음 프로젝트에서는 명세 승인표에 비교 규칙을 반드시 포함한다.

사람의 최종 확인

측정값이나 파일 근거 없이 '안전함·빠름'이라고 결론 내리지 않습니다. 다음 담당자가 변경 이유, 검증 증거, 남은 위험을 추가 질문 없이 이해할 수 있어야 합니다.

21 / COMPARISON

Codex와 Claude Code 결과 비교

동일한 gstack 스킬도 실행 모델과 도구 환경에 따라 질문, 근거, 상세 수준이 달라질 수 있습니다. 결과 길이보다 근거의 품질을 비교합니다.

gstack 호출 방식	스킬 이름을 자연어로 지정	슬래시 명령
프로젝트 근거		
가정 표시		
계획 현실성		
발견한 문제		
설명 명확성		

토론 질문

- 두 도구가 공통으로 발견한 문제는 무엇인가?
- 한 도구만 발견한 문제는 실제로 유효한가?
- 존재하지 않는 파일이나 동작을 가정했는가?
- 어떤 결정을 사람이 내려야 하는가?

주의

> **두 모델의 의견이 같아도 그것만으로 사실이 되지는 않습니다. 실제 코드, 테스트, 사용자 관찰이 최종 근거입니다.**

평가 기준과 제출물

환경 구성	20	도구와 gstack 상태를 확인했다.
스킬 선택	20	작업 목적에 맞는 역할을 선택했다.
근거 확인	20	파일과 실제 동작으로 출력을 검증했다.
품질 보고	20	재현 가능한 리뷰·QA 결과를 남겼다.
사람의 판단	20	가정과 최종 결정을 구분했다.

제출물

- 환경 점검표
- office-hours 결과 기록표
- 기술 계획 워크시트
- 코드 리뷰와 QA 보고서
- Codex와 Claude Code 비교표
- AI가 잘못 가정한 내용과 사람이 내린 결정

확인 퀴즈

- Codex host 설치 명령은 무엇인가?
- Claude Code에서 office-hours를 호출하는 명령은 무엇인가?
- 수정 없이 QA 보고서만 받고 싶을 때 어떤 스킬을 사용하는가?
- AI 작업 전후 반드시 확인할 Git 명령 두 가지는 무엇인가?

강사 운영 체크리스트

수업 전

Codex와 Claude Code 계정 접근 여부를 확인한다.

수업 네트워크에서 공식 설치 주소에 접근 가능한지 확인한다.

실습 ZIP과 PDF 다운로드를 직접 테스트한다.

설치 완료 예비 PC 또는 화면 녹화를 준비한다.

샘플 앱의 의도적 문제와 정답표는 강사용으로만 보관한다.

수업 중

학습자가 현재 작업 폴더를 pwd로 확인하게 한다.

분석 실습에는 코드 수정 금지 조건을 명시한다.

AI 작업 전후 git status와 git diff를 확인하게 한다.

AI가 제시한 파일과 함수가 실제로 존재하는지 검증하게 한다.

설치가 지연된 학습자는 짝 활동으로 실습을 계속하게 한다.

공식 문서

Codex CLI: https://developers.openai.com/codex/cli

Claude Code: https://code.claude.com/docs/en/getting-started

gstack: https://github.com/garrytan/gstack

강의의 최종 메시지

gstack은 AI에게 프로젝트를 맡기는 도구가 아니라, AI를 여러 전문 역할로 활용하면서 사람이 방향과 품질을 통제하는 개발 프로세스입니다.