15 Portfolio Projects

That Get Forward Deployed Engineers Hired

Real-world project briefs with business scenarios, tech stacks, architecture flows, and interview talking points

Why Projects Matter More Than Resumes for FDE Roles

Forward Deployed Engineers are hired to solve real customer problems with working technology. That is why FDE hiring looks different from many traditional software engineering roles. A strong resume can get you noticed, but it is the projects you have built, deployed and can explain that convince a hiring manager you can be trusted in front of a customer.

Most candidates show a list of tools: Python, LangChain, Docker, AWS. Employers already assume you can learn tools. What they want to see is whether you can take an unclear business problem, turn it into a clear plan, build something that works, and show that it made a difference. A good FDE portfolio tells that story again and again.

What hiring managers look for in an FDE portfolio

- **A real business problem**, not a tutorial clone. The project should start with "a team was losing time because…".
- **Working integrations** between systems such as APIs, databases and third-party tools.
- **Deployment**: a live link or a clear explanation of how it runs in production.
- **Clear communication**: a README and a short demo that a non-technical person could follow.
- **Evidence of judgement**: trade-offs you made, risks you managed, and what you would do next.

How to Use This Guide

You do not need to build all 15 projects. A focused portfolio of **three or four well-documented projects** is far stronger than ten half-finished ones. Pick at least one project from each tier so your portfolio shows a progression from integration skills, to applied AI, to production-grade deployment.

Each project brief follows the same structure, so you can compare them quickly:

- **Business scenario**: the customer problem you are solving.
- **What you'll build** and an **architecture flow** showing how the pieces connect.
- **Tech stack**: suggested tools, which you can swap for equivalents you already know.
- **Core features** and a **stretch goal** for going further.
- **What it proves to employers** and an **interview talking point** you can adapt.

Time estimates assume part-time effort of roughly 10 hours a week. If your background is already strong in one area, you will move faster.

Suggested starting combinations

If your background is...	Start with these projects	Why
Software engineering	2, 6, 13, 15	Builds on coding strength and adds AI, deployment and customer work.
Data science / ML	5, 9, 12, 14	Turns modelling skills into business-facing, explainable solutions.
Cloud / DevOps	4, 7, 11, 13	Adds AI and customer-facing workflows to your infrastructure skills.

New graduate / career switcher	1, 3, 6, 15	Covers the fundamentals, then proves you can run a full engagement.

Project Selection Matrix

Use this matrix to see all 15 projects at a glance and choose the mix that fills your skill gaps. The five skill areas match the core FDE capabilities: **Code** (programming), **API** (system integration), **AI** (machine learning and LLMs), **Cloud** (deployment) and **Customer** (business problem-solving).

#	Project	Tier	Industry	Core skills	Est. time
1	Multi-Source Data Sync Tool	Foundation	Retail	Code, API	1–2 weeks
2	Webhook-Driven Order Alert System	Foundation	E-commerce	API, Cloud	1 week
3	Automated Invoice Data Extractor	Foundation	Finance	Code, AI	1–2 weeks
4	Internal KPI Reporting Dashboard	Foundation	Operations	Code, API, Customer	1–2 weeks
5	Customer Feedback Classifier	Foundation	SaaS	AI, API	1 week
6	Policy Document RAG Assistant	Intermediate	HR	AI, Cloud	2 weeks
7	AI Customer Support Copilot	Intermediate	Telecom	AI, API, Customer	2–3 weeks

8	Clinical Admin Workflow Automator	Intermediate	Healthcare	AI, API, Customer	2–3 weeks
9	Contract Clause Extractor and Risk Flagger	Intermediate	Legal	AI	2 weeks
10	Sales Call Summarizer and CRM Updater	Intermediate	B2B Sales	AI, API	2 weeks
11	Multi-Agent Procurement Assistant	Advanced	Manufacturing	AI, API, Cloud	3–4 weeks
12	Shipment Delay Predictor and Rerouting Advisor	Advanced	Logistics	AI, Code, Cloud	3 weeks
13	Containerized LLM App with CI/CD and Monitoring	Advanced	Any industry	Cloud, Code	2–3 weeks
14	Fraud Alert Triage Agent	Advanced	Banking	AI, API, Customer	3 weeks
15	End-to-End Customer Deployment Simulation	Advanced	Capstone	All five	4–6 weeks

Tier 1: Foundation Projects (1–5)

Programming, APIs and integration

These projects build the base every FDE needs: reading data from different systems, connecting them reliably, and adding a first layer of AI. They are small enough to finish in one or two weeks, but each one solves a problem that real businesses pay to fix. If you are new to the field, start here.

Project 1: Multi-Source Data Sync Tool

Industry: Retail | **Tier:** 1 – Foundation | **Est. time:** 1–2 weeks | **Skills:** Code, API

Business scenario

A mid-sized retail chain keeps customer records in three places: a CRM used by marketing, a Google Sheet maintained by store managers, and a PostgreSQL loyalty database. Over time these records drift apart. Marketing sends offers to customers who have already unsubscribed, and store staff cannot see recent online purchases when a customer walks in.

What you'll build

A Python service that pulls customer records from each source through its API, matches duplicates using rules such as email and phone number, resolves conflicts with a clear policy (for example, "most recently updated wins"), and writes one clean record back to every system on a schedule.

> **Architecture flow**
>
> CRM API + Sheets API + PostgreSQL → Python sync engine → Matching & conflict rules → Unified record → Write-back + sync log

Tech stack

Layer	Suggested tools
Language	Python 3
Integration	REST APIs (HubSpot free tier or a mock CRM), Google Sheets API
Data	PostgreSQL, SQLAlchemy
Scheduling	cron or APScheduler
Testing	pytest with sample messy data

Core features

- **Pagination and rate-limit handling** so the sync works on thousands of records, not just a demo set.
- **Fuzzy matching** on name and phone to catch duplicates like "Priya S." and "Priya Sharma".
- **Conflict-resolution log** that records which value won and why.
- **Dry-run mode** that shows every change before anything is written.
- **Retry with exponential backoff** for failed API calls.

Stretch goal

Add a small review page where a business user approves uncertain matches before they are merged. This adds a human-in-the-loop step, which is common in real FDE deployments.

> **What it proves to employers**
>
> You can work with messy enterprise data, connect systems that were never designed to talk to each other, and build something safe enough to run against production data.

> **Interview talking point**
>
> "The store had three versions of the truth. I built a sync engine with a dry-run mode so the operations team could see every change before it happened. That is what got them to trust it."

Project 2: Webhook-Driven Order Alert System

Industry: E-commerce | **Tier:** 1 – Foundation | **Est. time:** 1 week | **Skills:** API, Cloud

Business scenario

An online store's operations team only learns about failed payments, high-value orders and stock-outs when customers complain. They want real-

time alerts inside the tools they already use, without logging into another dashboard.

What you'll build

A lightweight backend that receives webhooks from the store and payment platforms, applies simple business rules, and routes each alert to the right Slack channel or email address.

> **Architecture flow**
>
> Store / payment webhook → FastAPI endpoint → Signature check → Rules engine → Slack / Email → Event log

Tech stack

Layer	Suggested tools
Backend	Python, FastAPI
Event sources	Stripe or Shopify webhooks in test mode (or a mock sender)
Notifications	Slack incoming webhooks, SendGrid email
Storage	SQLite or PostgreSQL for the event log
Hosting	Render, Railway or AWS Lambda

Core features

- **HMAC signature verification** so only genuine events are processed.
- **Configurable rules**, for example: order value above $5,000 → #vip-orders channel; payment failed twice → email the finance lead.
- **Idempotency** so the same event never triggers two alerts.
- **Retries** when Slack or email delivery fails.
- **Daily digest** summarising the previous day's alerts.

Stretch goal

Move the rules into a YAML file or a small settings page so the operations team can change thresholds without asking a developer.

What it proves to employers

You understand event-driven integration, basic API security, and designing systems that non-technical users can control.

Interview talking point

"I designed the rules so the operations lead could change thresholds herself. Reducing her dependency on engineering was part of the deliverable."

Project 3: Automated Invoice Data Extractor

Industry: Finance | **Tier:** 1 – Foundation | **Est. time:** 1–2 weeks | **Skills:** Code, AI

Business scenario

A finance team manually types more than 400 vendor invoices a month into its accounting system. Each invoice takes a few minutes, and small typing errors lead to wrong payments and painful reconciliations at month-end.

What you'll build

A pipeline that collects PDF invoices from a folder or inbox, extracts key fields (vendor, invoice number, date, line items, tax and total), validates them, and outputs clean JSON ready for the accounting system.

Architecture flow

PDF upload / inbox → Text extraction (pdfplumber or OCR) → LLM field extraction → Validation rules → JSON → Database / accounting API

Tech stack

Layer	Suggested tools
Parsing	pdfplumber for digital PDFs, Tesseract OCR for scans
AI	OpenAI or Claude API with structured (JSON) output
Validation	Pydantic models
Storage	PostgreSQL
Review UI	Streamlit

Core features

- Handles both **digital and scanned** invoices.
- **Schema-enforced extraction** so every output has the same fields.
- **Validation rules**: line items must add up to the total, dates must be valid, and duplicate invoice numbers are flagged.
- **Low-confidence flag** that sends uncertain invoices to a human reviewer.
- **Export** to CSV or JSON for import into accounting software.

Stretch goal

Hand-label 50 sample invoices and report field-level accuracy (for example, how often the total and vendor name were extracted correctly).

> **What it proves to employers**
>
> You can combine traditional tools with LLMs and you build validation around AI output instead of trusting it blindly.

> **Interview talking point**
>
> "Extraction was the easy part. The validation layer, which checks that line items add up to the total, is what made it usable for the finance team."

Project 4: Internal KPI Reporting Dashboard

Industry: Operations | **Tier:** 1 – Foundation | **Est. time:** 1–2 weeks | **Skills:** Code, API, Customer

Business scenario

An operations manager spends every Monday morning copying numbers from four different tools into a slide deck for leadership. The numbers are often out of date by the time the meeting starts, and different teams calculate the same metric differently.

What you'll build

An automated pipeline that pulls data from each tool's API, calculates the agreed KPIs, publishes a live dashboard, and emails a weekly summary to leadership.

> **Architecture flow**
>
> Tool APIs (CRM, helpdesk, sheets) → ETL script → KPI calculations → Dashboard + scheduled weekly email

Tech stack

Layer	Suggested tools
Processing	Python, pandas
Storage	PostgreSQL or DuckDB
Dashboard	Streamlit or Metabase
Scheduling	GitHub Actions cron
Delivery	Email with PDF snapshot

Core features

- **Written KPI definitions** agreed with the business owner (for example, what exactly counts as an "active customer").
- **Week-over-week trends** for every metric.
- **Threshold highlighting** so problems stand out in red.
- **Automated weekly email** with a PDF snapshot.
- **Data freshness indicator** showing when each source last updated.

Stretch goal

Add a short LLM-generated summary at the top of the email explaining what changed this week and why it might matter.

> **What it proves to employers**
>
> You can turn a manual business routine into reliable automation and you know that agreeing on definitions is as important as writing code.

> **Interview talking point**
>
> "The hardest part was getting two teams to agree on what an active customer meant. Once we wrote it down, the code took two days."

Project 5: Customer Feedback Classifier

Industry: SaaS | **Tier:** 1 – Foundation | **Est. time:** 1 week | **Skills:** AI, API

Business scenario

A SaaS company receives around 2,000 pieces of feedback a month from app reviews, support tickets and NPS surveys. The product team cannot read all of it, so recurring issues stay hidden until they become serious.

What you'll build

A classifier that tags each piece of feedback by sentiment, topic (billing, bugs, onboarding, feature request) and urgency, then rolls the results into a weekly trend report.

> **Architecture flow**
>
> Feedback sources (CSV / API) → Cleaning → LLM classification with fixed taxonomy → Database → Weekly trend report

Tech stack

Layer	Suggested tools
Language	Python
AI	LLM API with few-shot prompting
Storage	PostgreSQL
Reporting	Streamlit charts or a scheduled HTML report

Core features

- **Fixed topic list** agreed with the product team so results stay consistent week to week.
- **Few-shot prompting** with real examples for each category.
- **Batch processing with cost tracking** so you know the price per 1,000 items.
- **Trend charts** by topic and sentiment.
- **Top representative quotes** for each topic, so product managers hear the customer's voice.

Stretch goal

Compare the LLM against a simple keyword-matching baseline on a labelled sample, and report accuracy and cost for both.

What it proves to employers

You use LLMs practically, think about cost, and evaluate results instead of assuming the AI is right.

Interview talking point

"The LLM beat my keyword baseline on topic accuracy, but it cost money per item. I showed both numbers and let the product team choose."

Tier 2: Intermediate Projects (6–10)

AI, LLMs and RAG in real workflows

Tier 2 is where you show you can apply AI to a specific business process. Each project combines LLMs or RAG with integrations, human review and measurement. These are the projects most likely to come up in an FDE interview, because they mirror the work FDEs do in their first months on a customer site.

Project 6: Policy Document RAG Assistant

Industry: HR | **Tier:** 2 – Intermediate | **Est. time:** 2 weeks | **Skills:** AI, Cloud

Business scenario

The HR team at a 1,500-person company answers the same questions every week about leave, reimbursements and benefits. The answers exist, but they are spread across 40 policy PDFs that employees rarely read.

What you'll build

A retrieval-augmented generation (RAG) assistant that answers employee questions using only the official policy documents, and shows the exact document and section each answer came from.

> **Architecture flow**
>
> Policy PDFs → Chunking → Embeddings → Vector database → Retriever → LLM answer with citations → Chat UI

Tech stack

Layer	Suggested tools
Framework	LangChain or LlamaIndex
Embeddings	OpenAI or open-source embedding model
Vector store	Chroma, pgvector or Pinecone
LLM	OpenAI or Claude API
Interface	Streamlit or Next.js

Core features

- **Citations** that name the document and section behind every answer.
- **"I don't know" behaviour** when the policies do not cover the question.
- **Chunk-size experiments** documented in the README.
- **Thumbs up / down feedback** on every answer.
- **Re-indexing** when a policy document is updated.

Stretch goal

Build an evaluation set of 30 real-sounding questions with correct answers. Measure retrieval accuracy and answer correctness before and after tuning.

What it proves to employers

You can build RAG end to end, keep answers grounded in source material, and measure quality with an evaluation set.

Interview talking point

"I made the assistant say "I don't know" when a policy didn't cover something. For HR, a confident wrong answer is worse than no answer."

Project 7: AI Customer Support Copilot

Industry: Telecom | **Tier:** 2 – Intermediate | **Est. time:** 2–3 weeks | **Skills:** AI, API, Customer

Business scenario

Support agents at a telecom provider handle more than 60 tickets per shift. For each one they search old tickets and the knowledge base to find the right fix, which slows responses and leads to inconsistent answers.

What you'll build

A copilot that reads each new ticket, finds similar resolved tickets and relevant knowledge-base articles, and drafts a reply. The agent reviews, edits and sends it.

Architecture flow

New ticket → Similar-ticket search + KB search → LLM draft reply → Agent review / edit → Send → Edit feedback logged

Tech stack

Layer	Suggested tools
Backend	Python, FastAPI
Helpdesk	Zendesk or Freshdesk API (or a mock helpdesk)
Retrieval	Vector database for tickets and KB articles
LLM	OpenAI or Claude API
UI	Simple React sidebar

Core features

- **Draft replies** that follow the company's tone guidelines.
- **Source links** showing which tickets and articles the draft used.
- **Edit tracking** that records how much the agent changed each draft.
- **Escalation detection** for angry customers or legal threats.
- **PII masking** so account numbers and phone numbers never reach the LLM.

Stretch goal

Use the edit data to build a quality metric: the less agents need to change a draft, the better the copilot is performing.

> **What it proves to employers**
>
> You design human-in-the-loop systems, protect sensitive data, and measure real productivity impact.

> **Interview talking point**
>
> "I tracked how much agents edited each draft. That gave us a real quality metric instead of guessing whether the AI was helping."

Project 8: Clinical Admin Workflow Automator

Industry: Healthcare | **Tier:** 2 – Intermediate | **Est. time:** 2–3 weeks | **Skills:** AI, API, Customer

Business scenario

Front-desk staff at a multi-specialty clinic spend hours every day processing referral letters, intake forms and appointment requests that arrive by email and scanned fax. Data is retyped into the scheduling system, and urgent referrals can sit unnoticed.

What you'll build

A workflow that classifies each incoming document, extracts the relevant fields, pre-fills the scheduling system, and sends anything uncertain to a staff review queue.

> **Architecture flow**
>
> Email / fax inbox → Document classifier → Field extraction → Validation → Scheduling system (mock API) → Exception queue

Tech stack

Layer	Suggested tools
Orchestration	n8n or Python
Parsing	OCR for scanned documents
AI	LLM for classification and extraction
Systems	Mock EHR / scheduling API
Storage	PostgreSQL with audit logging

Core features

- **Document classification** into referral, intake form or appointment request.
- **Field extraction** for patient name, referring doctor, reason and urgency.
- **Exception queue** for low-confidence documents.
- **Audit log** recording every automated action.
- **Synthetic data only**, with no real patient information.

Stretch goal

Add role-based access and data-retention rules, and explain in the README how the design would change to meet healthcare privacy regulations.

> **What it proves to employers**
>
> You understand regulated environments, can automate real workflows, and treat safety and auditability as product features.

> **Interview talking point**
>
> "I used only synthetic patient data and logged every automated action. In healthcare, auditability is part of the product."

Project 9: Contract Clause Extractor and Risk Flagger

Industry: Legal | **Tier:** 2 – Intermediate | **Est. time:** 2 weeks | **Skills:** AI

Business scenario

An in-house legal team reviews about 50 vendor contracts a month. Most of the time goes into checking the same clauses: liability caps, termination rights, auto-renewal and data protection.

What you'll build

A tool that extracts the key clauses from each contract, compares them against the company's standard positions (its "playbook"), and flags deviations as high, medium or low risk.

> **Architecture flow**
>
> Contract PDF / DOCX → Clause segmentation → LLM clause extraction → Playbook comparison → Risk report (High / Medium / Low)

Tech stack

Layer	Suggested tools
Parsing	python-docx, pdfplumber
AI	LLM with structured output
Rules	Playbook stored as JSON or YAML
Output	HTML or PDF risk report

Core features

- **Exact quoted clause text** for every extracted item.
- **Playbook comparison**, for example: liability cap below 12 months of fees → High risk.
- **Plain-language explanation** for each risk score.
- **Side-by-side view** of the contract clause and the playbook position.
- **Reviewer override** so lawyers can correct a flag and record why.

Stretch goal

Test the tool on publicly available sample contracts and report which clauses it missed and why.

What it proves to employers

You can translate domain expertise into rules and build explainable AI that professionals can trust.

Interview talking point

"Lawyers won't trust a risk score they can't trace. Every flag shows the exact clause text and the playbook rule it broke."

Project 10: Sales Call Summarizer and CRM Updater

Industry: B2B Sales | **Tier:** 2 – Intermediate | **Est. time:** 2 weeks | **Skills:** AI, API

Business scenario

Sales reps at a B2B software company often skip CRM updates after calls. Managers have poor visibility into the pipeline, and important details such as budget and objections get lost.

What you'll build

A pipeline that transcribes call recordings, generates a structured summary, extracts next steps, budget, objections and decision-makers, and updates the CRM after the rep approves it.

Architecture flow

Call recording → Transcription (Whisper) → LLM summary + field extraction → Rep approval in Slack → CRM API update

Tech stack

Layer	Suggested tools

Transcription	Whisper or Deepgram
AI	LLM with a fixed summary template
CRM	HubSpot or Salesforce developer sandbox
Backend	FastAPI
Approval	Slack interactive message

Core features

- **Structured summary** with the same sections every time.
- **Field mapping** from the summary to CRM fields.
- **One-click approval** in Slack before anything is written.
- **Follow-up email draft** the rep can send straight away.
- **Speaker handling** for calls with several participants.

Stretch goal

Create a weekly manager digest that groups common objections across all calls.

> **What it proves to employers**
>
> You can build an end-to-end workflow that writes into a real business system and you design for user adoption.

> **Interview talking point**
>
> "Reps hated data entry, so I made approval a single click in Slack. Adoption mattered more than model quality."

Tier 3: Advanced Projects (11–15)

Agents, deployment and enterprise scale

Tier 3 projects deal with the challenges of enterprise deployment: multi-agent systems, regulated industries, production infrastructure and full customer engagements. One strong Tier 3 project can set your portfolio apart from most other candidates.

Project 11: Multi-Agent Procurement Assistant

Industry: Manufacturing | **Tier:** 3 – Advanced | **Est. time:** 3–4 weeks | **Skills:** AI, API, Cloud

Business scenario

A manufacturing company's procurement team compares vendor quotes manually across emails and spreadsheets. Approvals get stuck because nobody is sure who needs to sign off on what.

What you'll build

A multi-agent system in which each agent has a clear job: an intake agent reads purchase requests, a comparison agent evaluates vendor quotes against price history and criteria, and a routing agent sends the recommendation to the correct approver based on company policy.

> **Architecture flow**
>
> Purchase request → Intake agent → Quote comparison agent → Policy check → Approval routing agent → Mock ERP + audit trail

Tech stack

Layer	Suggested tools
Agents	LangGraph or CrewAI
AI	LLM with tool calling

Storage	PostgreSQL
Systems	Mock ERP API
Approvals	Slack or email

Core features

- **Tool-calling agents** with clearly separated responsibilities.
- **Comparison table** of vendors on price, delivery time and past reliability.
- **Policy rules**, for example: purchases above $25,000 need two approvers.
- **Full audit trail** of every agent decision.
- **Fallback to a human** whenever an agent is uncertain.

Stretch goal

Track cost and latency for every agent run and add guardrails, such as a maximum number of steps per request.

> **What it proves to employers**
>
> You can orchestrate agents and, just as importantly, you know where AI should recommend and where humans must decide.

> **Interview talking point**
>
> "Agents could recommend, but only humans could approve purchases. Defining that boundary was the main design decision."

Project 12: Shipment Delay Predictor and Rerouting Advisor

Industry: Logistics | **Tier:** 3 – Advanced | **Est. time:** 3 weeks | **Skills:** AI, Code, Cloud

Business scenario

A logistics provider only learns about shipment delays after customers complain. Dispatchers want an early warning and practical rerouting options so they can act before a delivery is missed.

What you'll build

A machine learning model that predicts the delay risk of each shipment using historical performance, carrier data and weather, combined with an LLM advisor that explains the risk and suggests alternatives to dispatchers.

> **Architecture flow**
>
> Shipment data + weather API + carrier history → Feature pipeline → Delay prediction model → Risk dashboard → LLM rerouting suggestions

Tech stack

Layer	Suggested tools
Modelling	pandas, scikit-learn or XGBoost
Data	Public supply chain datasets (e.g., Kaggle) + a weather API
AI	LLM for explanations and suggestions
Dashboard	Streamlit

Core features

- **Delay risk score** for each active shipment.
- **Top contributing factors** explained in plain language (for example, "carrier on-time rate on this route has dropped this month").
- **Regular refresh** to simulate near real-time logistics monitoring.
- **Alert threshold** that notifies dispatchers of high-risk shipments.

- **Rerouting suggestions** with the reasoning behind each one.

Stretch goal

Simulate the cost impact of acting on predictions versus doing nothing, and present it as a simple ROI estimate.

> **What it proves to employers**
>
> You can combine classic machine learning with LLMs, apply AI to logistics optimisation, and connect technical output to business value.

> **Interview talking point**
>
> "The model flagged risk, but the value came from turning that into a dispatcher action with a clear reason attached."

Project 13: Containerized LLM App with CI/CD and Monitoring

Industry: Any industry | **Tier:** 3 – Advanced | **Est. time:** 2–3 weeks | **Skills:** Cloud, Code

Business scenario

Many AI prototypes never reach production because nobody packaged, deployed or monitored them properly. Employers want FDEs who can take a working demo and make it reliable for real users.

What you'll build

Take one of your earlier projects (Project 6 works well) and make it production-ready: containerised, deployed through a CI/CD pipeline, and monitored for errors, latency, cost and answer quality.

> **Architecture flow**
>
> GitHub push → CI (tests + evals) → Docker build → Container registry → Cloud deploy → Logs, metrics, alerts

Tech stack

Layer	Suggested tools
Containers	Docker (multi-stage builds)
CI/CD	GitHub Actions
Hosting	AWS ECS, Google Cloud Run or Azure Container Apps
Infrastructure	Terraform (optional)
Monitoring	Cloud-native monitoring or Prometheus + Grafana; Langfuse for LLM tracing

Core features

- **Multi-stage Dockerfile** for small, secure images.
- **Secrets management** with no API keys in the code.
- **Automated evaluation gate** that fails the build if answer quality drops.
- **Dashboards** for latency, cost per request and error rate.
- **One-command rollback** to the previous version.

Stretch goal

Define the infrastructure in Terraform and set up a blue-green deployment so new versions go live without downtime.

> **What it proves to employers**
>
> You can take AI from notebook to production, which is where many candidates stop and where FDEs are expected to deliver.

Interview talking point

"My CI pipeline runs an evaluation set on every commit. If answer quality drops, the deployment is blocked."

Project 14: Fraud Alert Triage Agent

Industry: Banking | **Tier:** 3 – Advanced | **Est. time:** 3 weeks | **Skills:** AI, API, Customer

Business scenario

Fraud analysts at a bank face hundreds of alerts every day. Most are false positives, so genuine fraud cases wait in the same queue as harmless ones.

What you'll build

A triage agent that enriches each alert with account history, transaction patterns and previous cases, summarises the evidence, and recommends a priority. The analyst always makes the final decision.

Architecture flow

Fraud alert → Data enrichment (account, transactions, prior cases) → Agent analysis → Priority + evidence summary → Analyst decision → Feedback loop

Tech stack

Layer	Suggested tools
Data	Synthetic transaction data (e.g., PaySim)
AI	LLM with tool calling
Storage	PostgreSQL

Interface	Analyst review screen
Compliance	Audit logging

Core features

- **Evidence summary** for each alert in a consistent format.
- **Priority score with reasoning** the analyst can check.
- **Decision capture** recording what the analyst decided and why.
- **Complete audit trail** for regulators.
- **No autonomous actions**: the agent never blocks a card or account itself.

Stretch goal

Run simulated analyst sessions and compare average time-to-decision with and without the agent.

> **What it proves to employers**
>
> You can apply AI in high-stakes, regulated settings with explainability and compliance built in.

> **Interview talking point**
>
> "The agent never blocks a card on its own. It saves analysts time by assembling the evidence, and every recommendation is logged for audit."

Project 15: End-to-End Customer Deployment Simulation

Industry: Capstone (your choice) | **Tier:** 3 – Advanced | **Est. time:** 4–6 weeks | **Skills:** All five

Business scenario

This project is different from the others. Instead of building a single app, you run a complete FDE engagement for a real local business, a nonprofit or a detailed fictional client, from the first conversation to a measured result.

What you'll build

A full engagement package: discovery notes, a requirements document, a working and deployed solution, user training material, and an impact report showing what changed.

> **Architecture flow**
>
> Discovery → Requirements & success metrics → MVP build with weekly demos → Deployment + training → Impact report

Tech stack

Layer	Suggested tools
Discovery	Interview notes in Notion or Google Docs; workflow maps in Excalidraw
Build	Any stack from Projects 1–14 that fits the problem
Deployment	Cloud hosting from Project 13
Reporting	Before/after metrics in a short report or slide deck

Engagement phases

Week 1 Discovery

- Interview two or three stakeholders and map the current workflow.
- List pain points and estimate the time or money each one costs.

Week 2 Scoping

- Write a one-page requirements document with success metrics.
- Agree on what is out of scope, just as a real FDE would.

Weeks 3–4 Build

- Implement the minimum viable solution.
- Demo progress to your "customer" every week and adjust.

Week 5 Deploy

- Move the solution to production and train the users.
- Write a short user guide.

Week 6 Measure

- Compare before-and-after metrics.
- Write a one-page impact report with lessons learned.

Core features

- **Discovery summary** showing you asked the right questions.
- **Requirements document** with measurable success criteria.
- **Architecture diagram** of the final solution.
- **Working deployment** that real users have tried.
- **Impact report** with before-and-after numbers.

> **Example engagement**
>
> A local coaching institute spends roughly 10 hours a week answering the same parent questions on WhatsApp about fees, batch timings and results. You build an FAQ assistant grounded in the institute's documents plus a simple enquiry tracker, train the office staff to use it, and measure the hours saved after four weeks.

Stretch goal

Ask your "customer" for a short written testimonial and include it, along with the impact report, in your portfolio.

What it proves to employers

You can do the entire FDE job: understand a customer, scope the problem, build, deploy, and prove business value.

Interview talking point

"I treated a local business as my customer and ran the full cycle from discovery to impact report. It's the closest thing to a real FDE engagement you can do before getting hired."

How to Present Your Projects

A great project that is poorly explained loses most of its value. Hiring managers often spend only a few minutes on each portfolio, so your projects must communicate the problem, your approach and the result quickly. Use the three tools below for every project you publish.

1. The README Template

Your README is often the first thing a hiring manager reads. Structure it like a short case study rather than a list of installation steps. The example column uses Project 3 (Invoice Data Extractor).

Section	What to include	Example
Problem	Who has the problem and what it costs them.	A finance team manually enters 400+ invoices a month, causing errors and delays.
Solution	What you built, in one or two sentences.	A pipeline that extracts and validates invoice data automatically.

Architecture	A diagram or flow showing how components connect.	PDF → OCR → LLM extraction → validation → database.
Tech stack	Tools used and why you chose them.	Pydantic for strict validation; pdfplumber for digital PDFs.
Results	Measured outcome or evaluation results.	Field-level accuracy on 50 hand-labelled invoices.
Demo	Live link and a 2-minute video.	Streamlit app link + Loom walkthrough.
Next steps	What you would improve with more time.	Direct integration with an accounting system API.

2. The Case-Study Format

When you talk about a project in an interview or a LinkedIn post, follow the same four-part story. It mirrors how FDEs report on real customer engagements.

1. **Customer problem**: describe the situation and what it was costing the business.
2. **Approach**: explain how you understood the problem and the key decisions you made.
3. **Build**: summarise the solution and the most interesting technical challenge.
4. **Impact**: share the result, ideally with a number, and what you learned.

3. Weak vs Strong Project Descriptions

The difference between a forgettable project and a memorable one is often just how it is described. Compare these two descriptions of the same RAG project.

> **Weak description**
>
> "Built a RAG chatbot using LangChain, OpenAI and Pinecone. Supports PDF upload and chat."

> **Strong description**
>
> "Built an HR policy assistant for a simulated 1,500-employee company. It answers leave and benefits questions from 40 policy PDFs with section-level citations and declines when a policy doesn't cover the question. I created a 30-question evaluation set, improved correct answers by tuning chunk size, and deployed it on Cloud Run with an evaluation gate in CI."

The strong version names the customer, the problem, the safety decision, the evaluation method and the deployment. It reads like the work of an FDE, not a tutorial.

Extra Presentation Tips

- **Pin your best three projects** at the top of your GitHub profile.
- **Record a two-minute demo** for each project. Show the problem first, then the solution.
- **Tailor your portfolio** to the role. For a healthcare AI company, lead with Project 8; for a fintech, lead with Project 14.
- **Write one LinkedIn post per project** using the case-study format to build visibility.

Portfolio Readiness Checklist

Before you start applying for FDE roles, go through this checklist. If you can tick every box, your portfolio is ready to support your applications.

Technical quality

☐ At least one project is **deployed and accessible** through a live link.

☐ Every repository is **clean**, with no API keys, no dead code and a clear folder structure.

☐ Each project includes an **architecture diagram** or flow.

☐ At least one project has **automated tests or an evaluation set**.

Presentation

☐ Each project has a **case-study style README** using the template in this guide.

☐ Each project has a **two-minute walkthrough video**.

☐ Each project shows a **measurable outcome** or evaluation result.

☐ You can explain every project in **under 60 seconds** using the case-study format.

Coverage

☐ Your portfolio includes **at least one project from each tier**.

☐ All five skill areas (**Code, API, AI, Cloud, Customer**) are covered at least once.

☐ At least one project includes a **customer-facing artifact**, such as a requirements document, user guide or impact report.

☐ Your top projects are **tailored to the industries** you are applying to.

Pair Your Portfolio with a Recognized Credential

GSDC Forward Deployed Engineer (FDE) Certification

Projects show what you can build. A certification shows that your knowledge of the FDE role is structured and validated. Together, they give employers two kinds of evidence: practical capability and professional commitment to the discipline.

Why combine certification with your portfolio

- **Validate your understanding** of FDE concepts, technical implementation and AI integration.
- **Build a structured foundation** in customer-focused problem-solving for enterprise AI.

- **Strengthen your professional profile** alongside the hands-on projects in this guide.
- **Demonstrate commitment** to continuous development in an emerging discipline.

GSDC
Global Skill Development Council

CERTIFIED FORWARD DEPLOYED ENGINEER

LEARN DIRECTLY FROM INDUSTRY EXPERTS, AI PRACTITIONERS, AND ENGINEERING LEADERS WHO ARE BUILDING AND DEPLOYING SCALABLE AI DRIVEN SOLUTIONS FOR ENTERPRISES WORLDWIDE.



ABOUT GSDC CERTIFICATION

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

CREATED BY EXPERTS

GSDC certifications are created and authored by world's leading experts in the field.

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

LEARNING MATERIALS

Get access to learning materials such as videos, ebooks, templates, and practice exams, which will help you clear the certification exam.

LEARNING OBJECTIVE

- Validate your Forward Deployed Engineering expertise.
- Increase your earning potential with an in-demand credential.
- Get complimentary career assistance after certification.

Enroll now with the code **LEARN20** To avail **20%** discount

Enroll Now

www.gsdcouncil.org