MÓDULO 02: DISEÑO DE SISTEMAS ML

Del Problema de Negocio a la Arquitectura Técnica

Guía MLOps v5.0: Senior Edition | DuqueOM | Noviembre 2025

MÓDULO 02: Diseño de Sistemas ML

Del Problema de Negocio a la Arquitectura Técnica

"Un arquitecto Senior no dibuja casas bonitas; diseña sistemas que sobreviven a terremotos, a cambios de requisitos y a desarrolladores que se van."

Duración	Teoría	Práctica
5-6 horas	40%	60%

ADR de Inicio: ¿Por Qué Diseñar Antes de Codificar?

```
╔═══════════════════════════════════════════════════════════════╗
║  ADR-002: Diseño Obligatorio Antes del Código                 ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  CONTEXTO:                                                    ║
║  El 73% de proyectos ML que fallan lo hacen por problemas de DISEÑO,  ║
║  no de algoritmos (Sculley et al., "Hidden Technical Debt in ML Systems"). ║
║                                                               ║
║  DECISIÓN:                                                    ║
║  Todo proyecto debe completar: ML Canvas + Diagrama de Arquitectura + ║
║  ADRs para decisiones técnicas clave ANTES de escribir código.  ║
║                                                               ║
║  CONSECUENCIAS:                                               ║
║  (+) Alineamiento stakeholders-equipo desde el inicio         ║
║  (+) Documentación de trade-offs para futuros desarrolladores ║
║  (+) Menor retrabajo por requisitos mal entendidos            ║
║  (-) Añade 1-2 semanas al inicio del proyecto                 ║
╚═══════════════════════════════════════════════════════════════╝
```

Lo Que Lograrás en Este Módulo

1. **Traducir** problemas de negocio a problemas de ML con métricas claras
2. **Completar** un ML Canvas profesional
3. **Diseñar** arquitectura usando el modelo C4
4. **Documentar** decisiones técnicas con ADRs
5. **Crear** un diagrama de flujo de datos

2.1 Traducción Negocio → ML (El Arte del Senior)

El Anti-Patrón: "Tengo Datos, Voy a Hacer ML"

```
╔═══════════════════════════════════════════════════════════════╗
║                    ⚠ EL ERROR DEL JUNIOR                      ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  JUNIOR:                                                      ║
║  "Tengo datos de clientes → Voy a probar XGBoost → Algo saldrá"  ║
║                                                               ║
║  PROBLEMA:                                                    ║
║  • No sabe qué métrica optimizar (¿accuracy? ¿F1? ¿costo de negocio?)  ║
║  • No sabe si el modelo genera valor                          ║
║  • No puede priorizar features porque no entiende el negocio  ║
║  • Cuando el proyecto "termina", nadie lo usa                 ║
╠═══════════════════════════════════════════════════════════════╣
║                    EL ENFOQUE DEL SENIOR                      ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  SENIOR:                                                      ║
║  "El banco pierde $2M/año por churn → Predecir top 10% de riesgo →  ║
║   Campaña de retención → ROI esperado $400K → Métricas:       ║
║   Precision@10% > 50%, AUC > 0.85, Latencia < 100ms"          ║
║                                                               ║
║  VENTAJAS:                                                    ║
║  • Métrica clara conectada a $$$                              ║
║  • Sabe cuándo el modelo es "suficientemente bueno"           ║
║  • Puede justificar inversión en infraestructura              ║
║  • El proyecto genera valor medible                           ║
╚═══════════════════════════════════════════════════════════════╝
```

Framework de Traducción: Negocio → ML → Sistema

```mermaid
flowchart LR
    subgraph Negocio[" NEGOCIO"]
        A[Problema de Negocio]
        B[KPIs de Éxito]
        C[Restricciones]
    end

    subgraph ML[" ML"]
        D[Tipo de Problema]
        E[Métricas Técnicas]
        F[Features Candidatas]
    end

    subgraph Sistema["⚙ SISTEMA"]
        G[Requisitos No-Funcionales]
        H[Arquitectura]
        I[Stack Tecnológico]
    end

    A --> D
    B --> E
    C --> G
    D --> H
    E --> H
    F --> H
    G --> I
```

Tabla de Traducción (Ejemplos)

Problema de Negocio	Tipo ML	Métrica Negocio	Métrica Técnica	Requisito Sistema
Reducir abandono de clientes	Clasificación Binaria	$ retenido/año	AUC-ROC, Precision@K	Batch diario o API < 100ms
Estimar precio de vehículos	Regresión	% error en valuación	RMSE, MAPE	API síncrona < 200ms
Detectar fraude en tarjetas	Anomaly Detection	$ fraude evitado	Precision, Recall	Streaming < 50ms
Recomendar productos	Ranking/RecSys	Lift en ventas	NDCG@10, MAP	API < 100ms, cold-start handling
Predecir demanda	Series Temporales	% reducción stockout	MAPE, Bias	Batch semanal

Ejercicio 2.1: Traduce Tu Problema

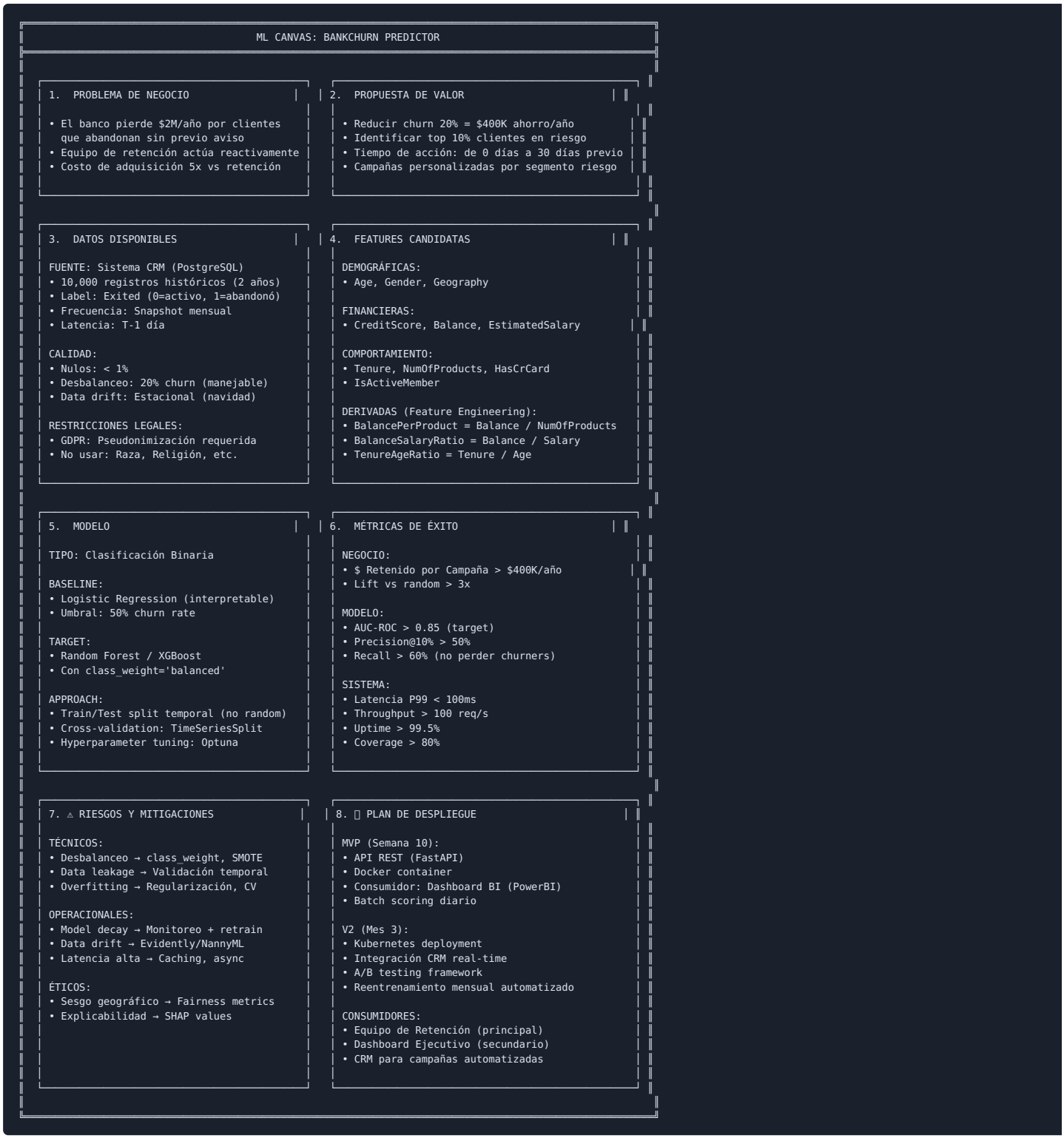
Para el proyecto que elegiste (BankChurn, CarVision, TelecomAI o propio):

1. **Problema de Negocio**: ¿Qué duele? ¿Cuánto cuesta?
2. **Tipo de ML**: ¿Clasificación, regresión, clustering, etc.?
3. **Métrica de Negocio**: ¿Cómo se mide el éxito en $$$?
4. **Métrica Técnica**: ¿Qué optimizamos? (AUC, RMSE, etc.)
5. **Requisito de Sistema**: ¿Batch o real-time? ¿Latencia?

2.2 ML Canvas: El Blueprint del Proyecto

¿Qué es el ML Canvas?

El **ML Canvas** es un framework de 1 página que captura todas las decisiones clave de un proyecto ML. Es como el Business Model Canvas pero para sistemas d
ML.



Template Vacío para Tu Proyecto

```
# ML CANVAS: [NOMBRE DEL PROYECTO]

## 1. Problema de Negocio
- ¿Qué duele?
- ¿Cuánto cuesta el problema actual?
- ¿Quién sufre?

## 2. Propuesta de Valor
- ¿Cómo ML alivia el dolor?
- ¿Cuál es el ROI esperado?
- ¿Qué decisiones habilita?

## 3. Datos Disponibles
- Fuente:
- Volumen:
- Frecuencia:
- Calidad (nulos, duplicados):
- Restricciones legales:

## 4. Features Candidatas
- Demográficas:
- Transaccionales:
- Comportamiento:
- Derivadas (feature engineering):

## 5. Modelo
- Tipo de problema:
- Baseline:
- Target:
- Approach de validación:

## 6. Métricas de Éxito
- Negocio:
- Modelo:
- Sistema:

## 7. ⚠ Riesgos y Mitigaciones
- Técnicos:
- Operacionales:
- Éticos:

## 8. Plan de Despliegue
- MVP:
- V2:
- Consumidores:
```

2.3 Arquitectura con el Modelo C4

¿Qué es C4?

El **modelo C4** (Context, Container, Component, Code) es un framework para documentar arquitectura de software en 4 niveles de zoom.

```
╔═══════════════════════════════════════════════════════════════╗
║                    MODELO C4: 4 NIVELES DE ZOOM               ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║   NIVEL 1: CONTEXTO (System Context)                          ║
║   • Vista de pájaro: El sistema y sus usuarios/sistemas externos
║   • Audiencia: Todos (stakeholders, devs, ops)                ║
║   • Pregunta: "¿Qué es esto y quién lo usa?"                   ║
║                                                               ║
║   NIVEL 2: CONTENEDORES (Container)                           ║
║   • Zoom in: Aplicaciones, bases de datos, servicios          ║
║   • Audiencia: Arquitectos, tech leads                        ║
║   • Pregunta: "¿Qué partes tiene el sistema?"                 ║
║                                                               ║
║   NIVEL 3: COMPONENTES (Component)                            ║
║   • Zoom in++: Módulos dentro de cada contenedor              ║
║   • Audiencia: Desarrolladores                                ║
║   • Pregunta: "¿Cómo está organizado internamente?"           ║
║                                                               ║
║   NIVEL 4: CÓDIGO (Code)                                      ║
║   • Máximo zoom: Clases, funciones                            ║
║   • Audiencia: Desarrolladores (el que va a implementar)      ║
║   • Nota: Usualmente se genera desde el código, no se dibuja  ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

Nivel 1: Contexto del Sistema BankChurn

```
C4Context
    title Sistema BankChurn - Diagrama de Contexto

    Person(retention_team, "Equipo Retención", "Actúa sobre predicciones para retener clientes")
    Person(data_team, "Data Team", "Monitorea y mejora modelos")

    System(bankchurn, "BankChurn Predictor", "Predice probabilidad de churn y genera scores de riesgo")

    System_Ext(crm, "Sistema CRM", "Fuente de datos de clientes")
    System_Ext(bi, "Dashboard BI", "Visualiza scores y métricas")
    System_Ext(campaign, "Sistema Campañas", "Ejecuta acciones de retención")

    Rel(retention_team, bankchurn, "Consulta scores", "API/Dashboard")
    Rel(data_team, bankchurn, "Monitorea y retrains", "MLflow/Grafana")
    Rel(crm, bankchurn, "Envía datos clientes", "Batch/API")
    Rel(bankchurn, bi, "Envía predicciones", "API")
    Rel(bankchurn, campaign, "Envía lista de riesgo", "API/Webhook")
```

Nivel 2: Contenedores

```
C4Container
    title Sistema BankChurn - Diagrama de Contenedores

    Person(user, "Usuario", "Equipo Retención / Data Team")

    Container_Boundary(bankchurn, "BankChurn Predictor") {
        Container(api, "API REST", "FastAPI", "Expone endpoints de predicción y health")
        Container(model, "ML Pipeline", "Sklearn", "Pipeline entrenado: preprocessor + model")
        Container(mlflow, "MLflow", "Python", "Tracking de experimentos y model registry")
        ContainerDb(db, "Model Storage", "S3/Local", "Artefactos .pkl")
    }

    System_Ext(crm, "CRM", "PostgreSQL")
    System_Ext(prometheus, "Prometheus", "Métricas")
    System_Ext(grafana, "Grafana", "Dashboards")

    Rel(user, api, "HTTP/JSON", "REST")
    Rel(api, model, "Loads", "joblib")
    Rel(model, db, "Reads", "S3/File")
    Rel(mlflow, db, "Writes", "Artifacts")
    Rel(crm, api, "Batch data", "CSV/API")
    Rel(api, prometheus, "Exposes /metrics", "HTTP")
    Rel(prometheus, grafana, "Scrapes", "PromQL")
```

Nivel 3: Componentes (API Container)

```
C4Component
    title API REST - Diagrama de Componentes

    Container_Boundary(api, "API REST (FastAPI)") {
        Component(routes, "Routes", "fastapi.APIRouter", "Define endpoints: /predict, /health, /metrics")
        Component(schemas, "Schemas", "Pydantic", "Valida requests y responses")
        Component(inference, "Inference Service", "Python Class", "Carga modelo y ejecuta predicción")
        Component(middleware, "Middleware", "FastAPI", "Logging, timing, error handling")
        Component(config, "Config", "Pydantic Settings", "Lee variables de entorno")
    }

    ContainerDb(model_store, "Model Store", "S3")
    Container(prometheus, "Prometheus Client", "prometheus_client")

    Rel(routes, schemas, "Uses", "Type validation")
    Rel(routes, inference, "Calls", "predict()")
    Rel(inference, model_store, "Loads", "joblib.load()")
    Rel(middleware, prometheus, "Exposes", "Counters/Histograms")
    Rel(routes, config, "Reads", "Settings")
```

2.4 Diagrama de Flujo de Datos

Training Pipeline

```
flowchart LR
    subgraph Data[" DATA"]
        RAW[(Raw Data<br/>data/raw/)]
        PROC[(Processed<br/>data/processed/)]
    end

    subgraph Training[" TRAINING"]
        LOAD[Load Data]
        FE[Feature<br/>Engineering]
        SPLIT[Train/Test<br/>Split]
        TRAIN[Train Model]
        EVAL[Evaluate]
    end

    subgraph Artifacts[" ARTIFACTS"]
        MODEL[(Pipeline.pkl<br/>models/)]
        METRICS[(Metrics<br/>mlruns/)]
    end

    subgraph Tracking[" TRACKING"]
        MLFLOW[MLflow<br/>Server]
    end

    RAW --> LOAD
    LOAD --> FE
    FE --> PROC
    FE --> SPLIT
    SPLIT --> TRAIN
    TRAIN --> EVAL
    EVAL --> MODEL
    EVAL --> METRICS
    METRICS --> MLFLOW
    MODEL --> MLFLOW

    style Data fill:#e1f5fe
    style Training fill:#fff3e0
    style Artifacts fill:#e8f5e9
    style Tracking fill:#f3e5f5
```

Inference Pipeline

```
flowchart LR
    subgraph Client[" CLIENT"]
        REQ[JSON Request]
        RESP[JSON Response]
    end

    subgraph API[" API"]
        VALID[Validate<br/>Pydantic]
        INFER[Inference<br/>Service]
        FORMAT[Format<br/>Response]
    end

    subgraph Model[" MODEL"]
        LOAD[Load<br/>Pipeline]
        PREDICT[Predict<br/>Proba]
    end

    subgraph Monitor[" MONITOR"]
        LOGS[Structured<br/>Logs]
        METRICS[Prometheus<br/>Metrics]
    end

    REQ --> VALID
    VALID --> INFER
    INFER --> LOAD
    LOAD --> PREDICT
    PREDICT --> FORMAT
    FORMAT --> RESP

    INFER --> LOGS
    INFER --> METRICS

    style Client fill:#e3f2fd
    style API fill:#fff8e1
    style Model fill:#e8f5e9
    style Monitor fill:#fce4ec
```

2.5 Architecture Decision Records (ADRs)

¿Qué es un ADR?

Un **ADR** documenta una decisión arquitectónica importante: el contexto, la decisión tomada, y las consecuencias.

Template ADR

```
# ADR-XXX: [Título de la Decisión]

## Estado
[Propuesto | Aceptado | Deprecado | Superseded por ADR-YYY]

## Contexto
¿Cuál es el problema que estamos tratando de resolver?
¿Qué restricciones tenemos?

## Decisión
¿Qué decidimos hacer?

## Consecuencias

### Positivas
-

### Negativas
-

### Neutras
-

## Alternativas Consideradas
| Alternativa | Pros | Contras | Razón de Rechazo |
| ----------- | ---- | ------- | ---------------- |
|             |      |         |                  |

## Referencias
- Links a documentación, papers, etc.
```

ADRs Ejemplo para BankChurn

ADR-003: FastAPI sobre Flask

```
# ADR-003: Usar FastAPI para la API REST

## Estado
Aceptado

## Contexto
Necesitamos exponer el modelo como API REST. Las opciones principales son:
- Flask (maduro, amplia adopción)
- FastAPI (moderno, async, tipado)
- Django REST (batteries-included, pero pesado)

## Decisión
Usaremos **FastAPI** para la API REST.

## Consecuencias

### Positivas
- Validación automática con Pydantic (ya usamos para config)
- Documentación OpenAPI generada automáticamente
- Soporte nativo async (mejor performance bajo carga)
- Type hints forzados (consistente con nuestro código)
- Mejor performance que Flask (Starlette + Uvicorn)

### Negativas
- Menos tutoriales/recursos que Flask (aunque creciendo rápidamente)
- Requiere entender async/await para features avanzados
- Algunos desarrolladores pueden no estar familiarizados

### Neutras
- Similar curva de aprendizaje inicial que Flask

## Alternativas Consideradas
| Alternativa | Pros | Contras | Razón de Rechazo |
| ----------- | ---- | ------- | ---------------- |
| Flask | Maduro, muchos recursos | Sin async, sin tipos, docs manual | Performance y DX inferior |
| Django REST | Batteries-included | Muy pesado para API simple, ORM no necesario | Overkill para nuestro caso |

## Referencias
- [FastAPI vs Flask Benchmark](https://fastapi.tiangolo.com/benchmarks/)
- [Why FastAPI](https://fastapi.tiangolo.com/features/)
```

ADR-004: DVC sobre Git LFS

```
# ADR-004: Usar DVC para Versionado de Datos

## Estado
Aceptado

## Contexto
Necesitamos versionar:
- Dataset de entrenamiento (CSV ~50MB)
- Modelos entrenados (PKL ~10MB)
- Posible crecimiento a GB en el futuro

## Decisión
Usaremos **DVC** (Data Version Control) con remote storage en S3/GCS.

## Consecuencias

### Positivas
- Integración nativa con Git (cada versión de datos linked a commit)
- Pipelines declarativos (dvc.yaml)
- Múltiples backends de storage (local, S3, GCS, Azure)
- Reproducibilidad con `dvc repro`
- Comunidad activa, bien documentado

### Negativas
- Curva de aprendizaje adicional
- Requiere setup de remote storage para colaboración
- Puede ser overkill para datasets muy pequeños

### Neutras
- CLI similar a Git (familiar)

## Alternativas Consideradas
| Alternativa | Pros | Contras | Razón de Rechazo |
| ----------- | ---- | ------- | ---------------- |
| Git LFS | Simple, integrado en Git | No soporta pipelines, costoso para archivos grandes | Sin reproducibilidad de pipelines |
| Delta Lake | Excelente para data lakes | Requiere Spark, overkill para nuestro caso | Complejidad innecesaria |
| DagsHub | DVC + MLflow hosted | Vendor lock-in, costo | Preferimos self-hosted |

## Referencias
- [DVC vs Git LFS](https://dvc.org/doc/user-guide/large-dataset-optimization)
```

2.6 Ejercicio Integrador: Diseña Tu Sistema

Entregables

Para tu proyecto elegido, crea los siguientes archivos en `docs/`:

1. `ML_CANVAS.md`: ML Canvas completo (usa el template)
2. `ARCHITECTURE.md`: Diagramas C4 (al menos Contexto y Contenedores)
3. `decisions/ADR-001.md`: Al menos 2 ADRs para decisiones clave

Criterios de Evaluación

Criterio	Básico (60-69)	Competente (70-84)	Destacado (85-100)
ML Canvas	Secciones incompletas	Todas las secciones, algunos detalles vagos	Completo con métricas específicas y cuantificadas
Diagramas	Solo texto descriptivo	Diagramas ASCII o básicos	Mermaid/PlantUML correctos y claros
ADRs	Sin ADRs	1 ADR básico	2+ ADRs con alternativas y trade-offs

2.7 Autoevaluación

Checklist

```
TRADUCCIÓN NEGOCIO → ML:
[ ] Puedo identificar el problema de negocio detrás de un proyecto ML
[ ] Sé calcular ROI esperado de una solución ML
[ ] Puedo elegir la métrica técnica correcta según el problema

ML CANVAS:
[ ] Puedo completar las 8 secciones del ML Canvas
[ ] Sé identificar riesgos técnicos, operacionales y éticos
[ ] Puedo definir métricas de negocio, modelo y sistema

ARQUITECTURA C4:
[ ] Entiendo los 4 niveles del modelo C4
[ ] Puedo dibujar diagramas de Contexto y Contenedores
[ ] Sé usar Mermaid para diagramas

ADRs:
[ ] Entiendo el propósito de los ADRs
[ ] Puedo documentar decisiones con alternativas y trade-offs
[ ] Sé cuándo crear un nuevo ADR vs actualizar uno existente
```

Preguntas de Reflexión

1. ¿Por qué es importante cuantificar el problema de negocio antes de empezar?
2. ¿Qué pasa si no documentas las decisiones arquitectónicas?
3. ¿Cuándo es apropiado NO usar ML para resolver un problema?

Siguiente Paso

Con el diseño completo, es hora de configurar un **entorno de desarrollo profesional**.

Ir a Módulo 03: Entornos Profesionales →

Módulo 02 completado. Ahora piensas como un arquitecto.