# URL Shortener

## (Midterm Exam)

**PREPARED FOR**

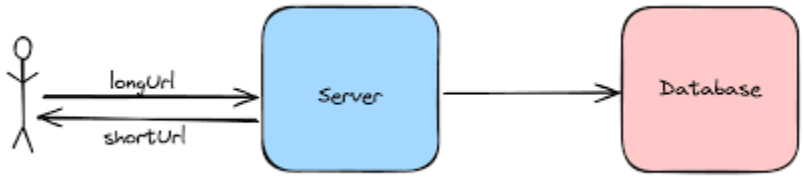
Software Engineering Course AUT

**Date**

Dec 2025

# مستند پروژه میان‌ترم عملی

## عنوان پروژه: سامانه‌ی کوتاه‌کننده‌ی لینک (URL Shortener)




### ۱. مقدمه

در این تمرین میان‌ترم، هدف آن است که دانشجویان با استفاده از مفاهیم آموخته‌شده در مباحث **طراحی و پیاده‌سازی سامانه‌های بک‌اند**، اقدام به پیاده‌سازی یک پروژه‌ی کاربردی با رویکرد حرفه‌ای نمایند.
پروژه‌ی حاضر با موضوع **کوتاه‌سازی آدرس‌های اینترنتی (URL Shortener)** طراحی شده و دانشجویان موظف‌اند آن را با استفاده از ابزارها و فناوری‌های زیر پیاده‌سازی کنند:

- **زبان برنامه‌نویسی:** Python
- **فریم‌ورک:** FastAPI
- **ORM:** SQLAlchemy
- **پایگاه‌داده:** PostgreSQL
- **مدیریت مایگریشن:** Alembic
- **مدیریت وابستگی‌ها:** Poetry
- **کنترل نسخه و همکاری تیمی:** Git و GitHub
- **ابزار تست API:** Postman

این پروژه تیمی بوده و انتظار می‌رود تمامی اعضای گروه نسبت به کلیه‌ی اجزای پروژه شناخت و تسلط کافی داشته باشند.

لطفاً **توضیحات ارائه‌شده در تاپیک مربوط به پروژه در گروه درس را حتماً به‌صورت کامل و دقیق مطالعه کنید** تا در روند انجام میان‌ترم هیچ نکته‌ای از قلم نیفتد.

## ۲. هدف پروژه

هدف از طراحی این پروژه، تمرین مفاهیم اصلی توسعه‌ی بک‌اند، از جمله طراحی معماری لایه‌ای، مدیریت داده‌ها، کار با پایگاه‌داده رابطه‌ای، کنترل خطا و مستندسازی API های سامانه است.

## ۳. معرفی سامانه‌ی کوتاه‌کننده‌ی لینک

**URL Shortener** سرویسی است که یک آدرس اینترنتی طولانی را دریافت کرده و در ازای آن یک آدرس کوتاه و قابل‌استفاده در اختیار کاربر قرار می‌دهد.
در هنگام استفاده، کاربر با مراجعه به آدرس کوتاه، به آدرس اصلی (Original URL) هدایت (Redirect) می‌شود.
کاربرد این سامانه‌ها در به‌اشتراک‌گذاری لینک‌ها، مدیریت ترافیک و تحلیل بازدیدکنندگان بسیار رایج است.

## ۴. الزامات فنی و معماری سامانه

### ۴.۱. معماری لایه‌ای

ساختار پروژه باید به‌صورت لایه‌ای طراحی شود و شامل بخش‌های زیر باشد:

- **Controller / Router:** مسئول تعریف مسیرها (Endpoints) و کنترل درخواست‌ها و پاسخ‌ها
- **Service:** شامل منطق تجاری (Business Logic) سامانه
- **Repository:** مسئول تعامل با پایگاه‌داده از طریق SQLAlchemy
- **Model:** تعریف مدل‌های داده‌ای و جداول پایگاه‌داده

رعایت این ساختار **الزامی** بوده و در ارزیابی پروژه لحاظ خواهد شد.

در این پروژه **الزامی است** که بین لایه‌ها از **Constructor Injection** برای تزریق وابستگی‌ها استفاده شود.
هدف از این الزام این است که:

- وابستگی‌ها (repositories, services, clients) **سخت‌کُد** نشوند،
- تست‌پذیری (testability) بالا برود،
- لایه‌ها از هم **loosely coupled** بمانند،
- و معماری پروژه شکل تمیزتری پیدا کند.

در واقع هیچ سرویسی نباید داخل خودش برود و مستقیم `SomeRepository()` بسازد؛ بلکه ریپوزیتوری باید **از بیرون** به آن **تزریق** شود.

**ارتباط بین لایه‌ها باید از طریق Constructor Injection انجام شود.**

- کنترلر به سرویس وابسته است
- سرویس به ریپوزیتوری وابسته است ← ریپوزیتوری باید از طریق سازنده به سرویس تزریق شود.
- هیچ لایه‌ای نباید در داخل خودش نمونه‌ی لایه‌ی زیرین را به‌صورت مستقیم بسازد.

### ۴.۲. رعایت اصول RESTful Naming

در طراحی مسیرهای API باید **اصول نام‌گذاری RESTful** رعایت شود. از جمله:

- استفاده از **اسم جمع (plural nouns)** برای مسیرها (مثلاً `urls/` نه `url/` یا `getUrl/`).
- استفاده از **افعال HTTP** برای بیان نوع عملیات (مثلاً `POST` برای ایجاد، `GET` برای دریافت، `DELETE` برای حذف).
- عدم استفاده از افعال در مسیر (مثلاً `create/` یا `delete/` مجاز نیست).
- مسیرها باید معنای مشخص، خوانا و سازگار با استانداردهای REST داشته باشند.
- پاسخ‌ها باید از الگوی مشخص با فیلد `status` و داده‌ی خروجی تبعیت کنند.

### ۴.۳. سایر الزامات فنی

- پاسخ‌ها باید دارای ساختار JSON یکسان باشند.
- استفاده از کدهای وضعیت HTTP مناسب در همه‌ی روت‌ها الزامی است.
- هندلینگ خطا باید در تمام بخش‌ها پیاده‌سازی شود.
- استفاده از Type Hint و اعتبارسنجی ورودی‌ها الزامی است.
- ماژول‌ها باید نام‌گذاری منظم، معنادار و مطابق استانداردهای PEP8 (کانونشن داده شده) داشته باشند.

### ۴.۴. پایگاه‌داده و مایگریشن

پایگاه‌داده مورد استفاده **PostgreSQL** است و باید با ابزار **Alembic** مایگریشن‌ها ایجاد و اجرا شوند.
جدول اصلی سامانه شامل اطلاعات لینک‌های کوتاه‌شده است. انتخاب دقیق فیلدها بر عهده‌ی دانشجو است، اما حداقل فیلدهای زیر پیشنهاد می‌شود:

- شناسه (id)
- آدرس اصلی (original_url)
- کد کوتاه‌شده (short_code)
- تاریخ ایجاد (created_at)

تمامی دانشجویان **موظف هستند فقط و فقط** از روش زیر برای بالا آوردن پایگاه‌داده استفاده کنند:

**اجرای PostgreSQL با دستور docker run** (استفاده از Docker Compose یا نصب مستقیم PostgreSQL روی سیستم *مجاز نیست*).

- اتصال به دیتابیس از طریق **SQLAlchemy**.
- ایجاد و مدیریت جداول تنها از طریق **Alembic Migration** (نوشتن، اجرا و اعمال مایگریشن‌ها).

هرگونه راه‌اندازی دیتابیس با روش‌های دیگر (Docker Compose، نصب لوکال PostgreSQL، یا اجرای جداول بدون Migration) **مردود محسوب می‌شود** و باعث **کسر نمره** خواهد شد.

لطفاً پیش از شروع کار، محیط دیتابیس را دقیقاً طبق داک پروژه راه‌اندازی کنید.

## ۵. یوزر استوری‌ها (User Stories)

در این بخش، نیازمندی‌های سامانه به‌صورت **یوزر استوری** (User Story) تعریف شده‌اند تا رفتار و انتظارات سیستم از دید کاربر مشخص شوند.
هر یوزر استوری بیانگر یک **قابلیت اصلی (Feature)** از سامانه‌ی کوتاه‌کننده‌ی لینک است که باید به‌صورت **End-to-End** پیاده‌سازی شود.

در مجموع، این پروژه شامل چهار قابلیت اصلی است که در قالب چهار مسیر (Route) ارائه شده‌اند:

1. ایجاد لینک کوتاه (POST)
2. ریدایرکت به آدرس اصلی (GET)
3. مشاهده‌ی همه‌ی لینک‌های کوتاه‌شده (GET)

4. حذف لینک کوتاه‌شده (DELETE)

هر دانشجو موظف است دو مورد از این قابلیت‌ها را به‌صورت کامل (از **Controller تا Repository**) پیاده‌سازی نماید.
ساختار هر یوزر استوری شامل **شرح هدف، سناریوی کاربر، و معیارهای پذیرش (Acceptance Criteria)** است تا پیاده‌سازی هر بخش به‌صورت دقیق و قابل ارزیابی انجام شود.

## User Story – ایجاد لینک کوتاه (POST)

**من به‌عنوانِ** یک کاربر سامانه **می‌خواهم** بتوانم یک آدرس اینترنتی بلند (original URL) را به سامانه ارسال کنم **تا بتوانم** یک لینک کوتاه دریافت کنم و آن را برای اشتراک‌گذاری آسان‌تر استفاده نمایم.

**Acceptance Criteria (AC):**

- کاربر بتواند با ارسال درخواست `POST /urls` و فیلد `original_url` در بدنه، لینک کوتاه ایجاد کند.
- سیستم باید اعتبار آدرس ورودی را بررسی کند (خالی یا فرمت نامعتبر نباشد).
- در صورت معتبر بودن، سامانه باید یک `short_code` یکتا تولید کند و در پایگاه‌داده ذخیره نماید.
- پاسخ باید شامل فیلد `status = success` و اطلاعات لینک ایجاد شده باشد.
- در صورت خطا (ورودی نامعتبر یا خطای داخلی)، پاسخ باید شامل `status = failure` و پیام مناسب باشد.
- از کد وضعیت HTTP متناسب استفاده شود (`201` برای موفقیت، `400` برای ورودی نامعتبر، `500` برای خطای سرور).

---

## User Story – دریافت و ریدایرکت به آدرس اصلی (GET /u/{code})

**من به‌عنوانِ** یک کاربر سامانه **می‌خواهم** با وارد کردن لینک کوتاه (short URL) بتوانم به آدرس اصلی هدایت شوم **تا بتوانم** به‌صورت مستقیم از لینک کوتاه برای دسترسی به محتوای مورد نظر استفاده کنم.

**Acceptance Criteria (AC):**

- کاربر بتواند با ارسال درخواست

```
GET /u/{code}
```

به سامانه، از طریق لینک کوتاه به آدرس اصلی ریدایرکت شود.

- سیستم باید کد کوتاه را از مسیر دریافت کرده و در پایگاه‌داده جست‌وجو کند.
- در صورت یافتن لینک، کاربر باید با وضعیت HTTP `302` یا `307` به `original_url` هدایت شود.
در صورت عدم وجود لینک، پاسخ باید شامل `status = failure` و پیام `"URL not found"` باشد.
- از کدهای وضعیت مناسب استفاده شود (`302` برای ریدایرکت موفق، `404` برای لینک ناموجود، `500` برای خطای سرور).

---

## User Story – مشاهده‌ی همه‌ی لینک‌های کوتاه‌شده (GET /urls)

**من به‌عنوانِ** کاربر یا مدیر سامانه **می‌خواهم** بتوانم فهرست تمامی لینک‌های کوتاه‌شده را مشاهده کنم **تا بتوانم** وضعیت و جزئیات آن‌ها را برای بررسی، تست یا گزارش مشاهده نمایم.

**Acceptance Criteria (AC):**

- کاربر بتواند با ارسال درخواست `GET /urls`، تمامی لینک‌های موجود در سیستم را دریافت کند.
- پاسخ باید شامل `status = success` و آرایه‌ای از داده‌ها (`data`) باشد که شامل اطلاعات هر لینک است (کد، آدرس اصلی، آدرس کوتاه، زمان ایجاد).
- در صورت نبود داده، پاسخ باید آرایه‌ی خالی بازگرداند، نه خطا.
- خطاهای احتمالی باید با `status = failure` و پیام مناسب مدیریت شوند.
- از کد وضعیت مناسب استفاده شود (`OK 200` برای موفقیت، `Internal Server Error 500` در صورت خطا).
- عملیات باید در لاگ سیستم ثبت شود.

## User Story – حذف لینک کوتاه‌شده (DELETE /urls/{code})

**من به‌عنوانِ** مدیر یا کاربر سامانه **می‌خواهم** بتوانم یک لینک کوتاه‌شده را حذف کنم **تا بتوانم** لینک‌های قدیمی، آزمایشی یا نامعتبر را از سامانه پاک نمایم.

**Acceptance Criteria (AC):**

- کاربر بتواند با ارسال درخواست

```
DELETE /urls/{code}
```

لینک مورد نظر را حذف کند.

- سیستم باید کد کوتاه را در پایگاه‌داده جست‌وجو کرده و در صورت وجود، رکورد مربوطه را حذف نماید.
- در صورت موفقیت، پاسخ باید شامل `status = success` و پیام `"URL deleted successfully"` باشد.
- در صورت عدم وجود لینک، پاسخ باید شامل `status = failure` و پیام `"URL not found"` باشد.
- از کد وضعیت مناسب استفاده شود (`OK 200` یا `No Content 204` برای حذف موفق، `Not Found 404` برای لینک ناموجود، `Internal Server Error 500` در صورت خطا).

فیچر امتیازی:

### User Story (امتیازی) – زمان انقضا (TTL) برای لینک‌های کوتاه‌شده

**من به‌عنوانِ** مدیر یا توسعه‌دهنده‌ی سامانه **می‌خواهم** برای هر لینک کوتاه‌شده یک زمان انقضا (TTL) تعریف شود **تا بتوانم** لینک‌های قدیمی و منقضی‌شده را به‌صورت خودکار از سیستم حذف کنم و از نگه‌داری داده‌های غیرضروری جلوگیری نمایم.

---

### توضیح TTL

TTL یا **Time To Live** مدت‌زمانی است که داده در سامانه معتبر باقی می‌ماند.
در این پروژه، TTL مشخص می‌کند هر لینک کوتاه‌شده تا چند ساعت یا روز در سیستم فعال باشد.
پس از گذشت این زمان، لینک باید به‌صورت خودکار منقضی شده و از پایگاه‌داده حذف گردد.

TTL معمولاً از فایل پیکربندی محیطی (`env.`) خوانده می‌شود تا در محیط‌های مختلف (توسعه، تست، پروداکشن) قابل تنظیم باشد.

برای رعایت اصول حرفه‌ای، باید فایل نمونه‌ی `env.example.` نیز همراه پروژه وجود داشته باشد و مقدار پیش‌فرض TTL در آن مشخص شود.

---

### Acceptance Criteria (AC):

- مقدار TTL باید در فایل `env.` تعریف شود (مثلاً `APP_TTL_MINUTES=24`).
- در فایل `env.example.` نیز همین کلید با مقدار نمونه وجود داشته باشد تا توسعه‌دهندگان دیگر بدانند باید آن را تنظیم کنند.
- هنگام ایجاد لینک جدید، زمان ایجاد (`created_at`) باید ذخیره شود.
- سامانه باید بتواند تشخیص دهد که از زمان ایجاد لینک بیش از TTL گذشته است.
- یک **کامند (Command)** باید اضافه شود تا لینک‌های منقضی‌شده را حذف کند.
- اجرای این کامند باید تمام لینک‌هایی را که `created_at + TTL < now()` هستند حذف کند.
- این کامند باید به‌صورت زمان‌بندی‌شده (scheduler) اجرا شود.
- این قابلیت **اختیاری و امتیازی** است و انجام صحیح آن در ارزیابی نهایی نمره‌ی اضافه دارد. (توجه شود که الویت با انجام یوزر استوری‌های اصلی پروژه است.)

---

## ۶. نکات فنی (Technical Notes)

1. کلیه‌ی پاسخ‌ها باید دارای فیلد `status` با مقدار `success` یا `failure` باشند.
2. استفاده از **کد وضعیت مناسب (HTTP Status Code)** الزامی است.
3. مدیریت خطا (Error Handling) باید برای هر چهار مسیر به‌صورت اصولی انجام شود.

### ۶.۱. تولید کد کوتاه (Short Code Generation)

برای هر آدرس اصلی (**Original URL**) که در سامانه ثبت می‌شود، باید یک **کد کوتاه (short code)** تولید گردد. این کد در انتهای URL سرویس قرار می‌گیرد و لینک کوتاه‌شده را تشکیل می‌دهد، مانند:

```
https://short.ly/abc123
http://localhost:8000/u/X9pQa2
```

### ۶.۲. یکتا بودن (Uniqueness)

هر کد کوتاه باید **یکتا** باشد؛ یعنی دو URL مختلف نباید کد یکسانی دریافت کنند.
یکتا بودن می‌تواند به یکی از روش‌های زیر تضمین شود:

1. **بررسی در پایگاه‌داده (Check in DB):**
   پس از تولید کد، در جدول بررسی شود که وجود ندارد. در صورت تکرار، مجدداً تولید شود.

2. **استفاده از شناسه‌ی رکورد (ID → Base62):**
   ابتدا رکورد در دیتابیس ذخیره شود و سپس مقدار `id` آن به فرمت Base62 تبدیل گردد.
   در این روش، یکتایی ذاتاً تضمین می‌شود.

در هر دو حالت باید روی ستون `short_code` در پایگاه‌داده **Unique Index** تعریف شود تا در صورت تولید هم‌زمان کدهای مشابه، پایگاه‌داده مانع ثبت تکراری شود.

---

### ۶.۳. کاراکترهای مجاز (Allowed Characters)

برای کوتاهی و خوانایی بهتر، کاراکترهای کد کوتاه باید از مجموعه‌ی زیر انتخاب شوند:

- حروف کوچک: `a-z`
- حروف بزرگ: `A-Z`
- اعداد: `0-9`

این مجموعه برابر با **Base62** است که شامل ۶۲ کاراکتر می‌شود.
یک کد ۶ کاراکتری از این مجموعه، حدود `56,800,235,584 = 6^62` حالت منحصربه‌فرد دارد که برای این پروژه کاملاً کافی است.

### ۶.۴. روش‌های تولید کد (Code Generation Methods)

دانشجو باید یکی از روش‌های زیر را انتخاب و در فایل `README.md` توضیح دهد.
رعایت یکتایی در سطح پایگاه‌داده **الزامی است**.

### ۱. روش تصادفی (Random Generation)

در هر ایجاد لینک، ۶ کاراکتر تصادفی از مجموعه‌ی Base62 انتخاب می‌شود.
در صورت تکرار در دیتابیس، کد مجدداً تولید می‌گردد.
روش ساده و مناسب برای تمرین‌های آموزشی.

**نمونه کد:**

```python
import random
import string

ALPHABET = string.ascii_letters + string.digits  # a-zA-Z0-9

def generate_short_code(length: int = 6) -> str:
    return ''.join(random.choices(ALPHABET, k=length))
```

### ۲. روش مبتنی بر شناسه‌ی دیتابیس (ID → Base62)

در این روش ابتدا رکورد URL در دیتابیس ذخیره می‌شود و سپس شناسه‌ی آن (`id`) به Base62 تبدیل می‌گردد.
این روش یونیک و قابل بازیابی است.

**نمونه کد:**

```python
ALPHABET = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

def encode_base62(num: int) -> str:
    if num == 0:
        return ALPHABET[0]
    arr = []
    base = len(ALPHABET)
    while num:
        num, rem = divmod(num, base)
        arr.append(ALPHABET[rem])
    arr.reverse()
```

```
    return ''.join(arr)
```

**۳. روش مبتنی بر هش (Hashing)**

در این روش از مقدار هش URL اصلی استفاده می‌شود (مثلاً با الگوریتم .(SHA1
۶ کاراکتر اول هش برای تولید کد کوتاه استفاده می‌شود.
در صورت تصادم، باید در دیتابیس بررسی و مجدداً تولید شود.
این روش مناسب زمانی است که بخواهیم برای URL تکراری، کد قبلی برگردد.

## ۶.۵. نکته‌ی نهایی

- انتخاب روش تولید کد به‌عهده‌ی تیم توسعه است.
- هر روش باید **در README توضیح داده شود.**
- اطمینان از یکتایی (`unique`) و اعتبار کد (`valid format`) در سطح پایگاه‌داده **الزامی** است.
- ستون `short_code` باید دارای ویژگی‌های زیر باشد:
  - نوع داده: `VARCHAR(10)`
  - `UNIQUE` و `INDEXED`

---

## ۷. نحوه‌ی کار تیمی (Team Collaboration – دو نفره)

این پروژه به‌صورت **تیمی (دو نفره)** انجام می‌شود.
هدف از همکاری، تمرین توسعه‌ی مشترک یک سرویس بک‌اند با معماری لایه‌ای، قابل نگهداری و سازگار با اصول نرم‌افزار است.
هر عضو موظف است ضمن انجام وظایف تعیین‌شده، با کل ساختار پروژه آشنا بوده و بتواند از عملکرد سامانه در جلسه‌ی ارزیابی دفاع کند. (امکان دارد به‌صورت تصادفی برای ارائه انتخاب شوید.)

| عضو | وظایف اصلی | توضیحات |
|---|---|---|

| | | |
|---|---|---|
| **عضو اول** | تنظیم Poetry و محیط اجرا، طراحی مدل داده با SQLAlchemy و Alembic، پیاده‌سازی دو مسیر از API | شامل پیاده‌سازی کامل مسیرها از Controller تا Repository |
| **عضو دوم** | تنظیم Poetry و محیط اجرا، پیاده‌سازی دو مسیر از API، مستند کرد هر ۴ API در یک کالکشن در Postman | شامل پیاده‌سازی کامل مسیرها از Controller تا Repository |

### ۷.۱. تقسیم وظایف فنی

هر عضو باید دو مسیر (Route) را **End-to-End** پیاده‌سازی کند؛
یعنی برای مسیرهای انتخابی خود، جزئیات را در  لایه‌های مربوطه شخصا پیاده‌سازی کند:
**Controller ← Service ← Repository ← Model**

### ۷.۲. همکاری و هماهنگی

- اعضا باید پیش از شروع، درباره‌ی ساختار پروژه، معماری لایه‌ای و نام‌گذاری مسیرها به توافق برسند.
- پس از تکمیل هر بخش، عملکرد آن باید توسط هر دو عضو بررسی و تست شود تا از صحت کل سیستم اطمینان حاصل گردد.
- نیازی به فرآیند رسمی Code Review نیست، اما **هر عضو باید از عملکرد کامل بخش هم‌تیمی خود مطمئن باشد.**
- در پایان پروژه، تحویل باید شامل نسخه‌ی نهایی پایدار در شاخه‌ی `main` باشد.
- در طول آزمون ریپوزیتوری گیت‌هاب شما باید Private باشد.

**نام‌گذاری و سیاست گیت (Git Policy & Naming Conventions):**

تمام نام‌گذاری‌ها، ساختار شاخه‌ها، و پیام‌های کامیت باید **طبق کانوشنی که در طول این ترم در کلاس استفاده شده است** رعایت شوند.

### ۷.۳. ترتیب شروع کار

در ابتدای پروژه، باید **پوشه‌بندی، ساختار اولیه‌ی پروژه، تنظیمات پایگاه‌داده و مایگریشن اولیه** (با Alembic) روی شاخه‌ی اصلی `main` انجام شود تا تمام اعضا بر اساس یک ساختار یکسان کار کنند.

پس از آماده‌شدن این بخش، هر عضو باید **یک برنچ به نام خودش** ایجاد کند و در آن، **دو مسیر (Route)** را به‌صورت **End-to-End** از لایه‌ی Controller تا Repository پیاده‌سازی نماید.

پس از اتمام کار، عضو مربوطه باید **Pull Request** ایجاد کند و در بخش توضیحات (Description) به‌صورت خلاصه و تیتروار بنویسد چه تغییراتی در آن برنچ انجام داده است.

در پایان کار، **تمام اسکرین‌شات‌های موردنیاز از کالکشن Postman را مطابق نمونه‌ی ارائه‌شده (در** README **زیر قابل مشاهده است) در دایرکتوری مربوطه قرار دهید.** دقت کنید نام‌گذاری فایل‌ها نیز طبق الگوی مشخص‌شده انجام شود.

**فایل زیر (README)** را به‌صورت کامل در ریپوی پروژه‌ی خود قرار دهید.

نام فایل باید دقیقاً به‌صورت زیر باشد:

```
README.md
```

تمام بخش‌های داخل README (جدول تست‌ها، روش تولید کد، تسک امتیازی، کالکشن Postman و اسکرین‌شات‌ها) **باید توسط شما با اطلاعات پروژه و گروه شما تکمیل شود.**

این فایل یکی از معیارهای تصحیح است و وجود نداشتن یا ناقص بودن آن **منجر به کسر نمره** خواهد شد.

```
# 📘 README - Midterm Final Checklist

This README **must remain in your repository** and **must be fully
completed** before submitting the midterm.

---

## 1. API Test Coverage Table

Fill in the second column with the **name of the student** who implemented
and tested each API.

| # | API Endpoint / Feature | Implemented & Tested By (Student Name) |
|---|------------------------|----------------------------------------|
| 1 | Create Short Link - **POST /links** | Alice |
```

```
| 2 | Redirect to Original URL - **GET /{code}** | Bob |
| 3 | Get All Shortened Links - **GET /links** | Alice |
| 4 | Delete Short Link - **DELETE /links/{code}** | Bob |

---

## 2. Code Generation Method (Section 6.4)

Check the method you used to generate the short code:

- [ ] **1. Random Generation**
- [x] **2. ID → Base62 Conversion**
- [ ] **3. Hash-based Generation**

(Only select the one you actually implemented.)

---

## 3. Bonus User Story: TTL (Expiration Time) for Shortened Links)

If you implemented the bonus user story, mark the box and complete the
required details.

- [x] **TTL Feature Implemented**

**If checked, fill in the following information:**

- **ENV variable or config key used:**
 `LINK_TTL_SECONDS=400`

You must also ensure this key exists in `.env.example` with a sample value.


- **Location of TTL Logic (File + Function):**

Specify the exact location where TTL expiration is checked and expired
links are detected/removed.

 `services/link_service.py → delete_expired_links()`

- **How TTL cleanup is triggered:**
You must write a Command that removes expired links (created_at + TTL <
now()).
```

````
Here, write:

- Full file path of the command
- Command name / execution method
- Scheduler details
---

## 4. Postman Collection (Required)

A **Postman Collection** has been created and includes all four API routes:

- **POST /links**
- **GET /{code}**
- **GET /links**
- **DELETE /links/{code}**

### Screenshots (included in GitHub)

For each route, two screenshots have been added:

- Successful response (2xx)
- Error-handled response (4xx)

Screenshots are located in:

      ```
   /postman
   ```



### Naming Example:

```
postman/
post-links-201-success.png
post-links-400-invalid-url.png
get-code-302-redirect.png
get-code-404-not-found.png
get-links-200-success.png
delete-code-200-success.png
delete-code-404-not-found.png
````
````

```
```

Filenames must clearly show:

- Route
- HTTP status
- Success or error

---

**✔️ Make sure this README is fully completed before submission.**

نمونه README ریپازیتوری گیت‌هاب پروژه شما: