

BİTİRME PROJESİ: StudyBuddy

(Pure Python – Standart Kütüphane)

Öğrenci Proje Görevi + Açıklamalı Teknik Şartname

Versiyon: 1.1 | Tarih: 02.01.2026

Amaç: Öğrencinin, yalnızca Python standart kütüphaneleri kullanarak kalıcı veri saklama(dosyalama istege bağlı SQLite) dahil olmak üzere, çalışma takip ve aralıklı tekrar (spaced repetition) sistemini uçtan uca tasarlayıp geliştirmesi.

Bu doküman; proje konusunu, işlevsel gereksinimleri, veri modelini (dosya tabanlı), modülleri, kabul kriterlerini, test senaryolarını ve değerlendirme rubriğini içerir.



1. Proje Tanımı

StudyBuddy, öğrencinin kendi çalışma materyalini (kartlar/flashcards) oluşturduğu, bu kartları çalıştıkça 0–5 arası bir puanla değerlendirdiği ve sistemin aralıklı tekrar mantığıyla bir sonraki tekrar tarihini hesapladığı komut satırı (CLI) uygulamasıdır.

Uygulama; kullanıcı yönetimi, deste (deck) yönetimi, kart yönetimi, çalışma (review) akışı, raporlama ve yedekleme gibi modüllerden oluşacaktır.

1.1 Öğrenme Çıktıları

- Dosya tabanlı kalıcı veri saklama (json, csv, pathlib, shutil).
- Katmanlı mimari: veri erişimi (repository), iş mantığı (service), arayüz (CLI) ayrımı.
- Güvenlik temelleri: parola hashleme (hashlib + salt), basit oturum yönetimi.
- Algoritmik düşünme: aralıklı tekrar (SM-2 benzeri) güncelleme formülü.
- Test yazımı: unittest ile birim testler ve kabul test senaryoları.
- Hata yönetimi ve loglama (logging).

1.2 Kapsam ve Kısıtlar (Çok Önemli)

1. SQL/Veritabanı YOK: sqlite3 dahil hiçbir SQL tabanlı yaklaşım kullanabilirsiniz.
2. Kalıcı veri ZORUNLU: JSON/CSV gibi dosyalara yazıp okumak zorunlu.
3. Uygulama CLI (komut satırı) olacaktır. GUI/web zorunlu değildir.
4. Kod, tek seferde çalışabilir olmalı: python main.py ile açılmalıdır.
5. README.md zorunludur: kurulum, kullanım örnekleri, veri formatı ve komutlar.

2. İşlevsel Gereksinimler (Fonksiyonlar)

2.1 Kullanıcı Yönetimi (İsteğe Bağlı

- Kayıt ol: e-posta + parola (parola hash'lenerek saklanır).
- Giriş yap: e-posta + parola doğrulanır, oturum başlatılır.
- Çıkış yap: oturum kapatılır.
- Kullanıcılar birbirinin verisini göremez (deck/kart izolasyonu).

2.2 Deck (Deste) Yönetimi

- Deck oluştur / listele / güncelle / sil.

- Her deck, yalnızca ilgili kullanıcıya aittir.
- Deck silinirse, bağlı kartlar da silinir (cascade).

2.3 Card (Kart) Yönetimi

- Kart ekle / listele / güncelle / sil.
- Her kart: front (soru) + back (cevap) alanlarına sahiptir.
- Kartlar deck'e bağlıdır.
- Kart eklenince SRS (tekrar durumu) kaydı otomatik oluşur.

2.4 Review (Çalışma) Akışı

Öğrenci, bir kartı çalışıktan sonra 0–5 arası bir kalite puanı verir:

- 0: Hiç hatırlamadım
- 1: Çok zor hatırladım
- 2: Kısmen hatırladım
- 3: Doğru ama zor
- 4: Doğru ve rahat
- 5: Mükemmel / akıcı

Sistem, puana göre aralıklı tekrar durumunu günceller: repetition, interval_days, easiness_factor, due_date.

2.4.1 Aralıklı Tekrar Kuralı (SM-2 Benzeri)

Kural özeti (öğrenci uygulayacak):

- EF (easiness factor) başlangıçta 2.5 alınır.
- EF güncelleme formülü:

$$EF = EF + (0.1 - (5-q) * (0.08 + (5-q)*0.02))$$

$$EF < 1.3 \text{ ise } 1.3\text{'e sabitlenir.}$$
- $q < 3$ ise repetition=0, interval=1 gün.
- $q \geq 3$ ise repetition artar:

$$\text{repetition}=1 \rightarrow \text{interval}=1 \text{ gün}$$

$$\text{repetition}=2 \rightarrow \text{interval}=6 \text{ gün}$$

$$\text{repetition}\geq 3 \rightarrow \text{interval} = \text{round(interval} * \text{EF})$$
- due_date = bugün + interval

2.5 Raporlama

- Bugün tekrar edilecek kartları listele (due_date <= today).
- Son 7 gün: kaç review yapıldı, ortalama kalite puanı.
- Deck bazlı istatistik: toplam kart, due olan kart, ortalama EF (opsiyonel).

2.6 Yedekleme / Dışa Aktarma (Bonus)

- Veri klasörünü timestamp ile yedekle (shutil.copytree / shutil.make_archive).

- JSON dışa aktar (zaten JSON ise tek dosyada birleştirme) veya CSV rapor çıktısı üret (csv modülü).

3. Veri Modeli (Dosya Tabanlı / SQL İSTEĞE BAĞLI)

Kalıcı veri saklama, dosya tabanlı yapılacaktır. Önerilen yaklaşım JSON dosyalarıdır. Öğrenci farklı bir dosya formatı kullanabilir (CSV/pickle), ancak kabul kriterlerini sağlamalıdır.

3.1 Önerilen Klasör Yapısı

- data/
- data/users.json
- data/decks.json
- data/cards.json
- data/srs_state.json
- data/reviews.json

3.2 JSON Veri Sözleşmesi (Contract)

Aşağıdaki örnek yapılar referanstır. Alan adları değişebilir; fakat aynı anlamları karşılamalıdır.

users.json örnek:

```
[  
  {"id": 1, "email": "a@b.com", "password_hash": "...", "salt": "...", "created_at": "2026-01-02T10:00:00"}  
]
```

decks.json örnek:

```
[  
  {"id": 10, "user_id": 1, "name": "Python", "description": "Temel kavramlar"}  
]
```

cards.json örnek:

```
[  
  {"id": 100, "deck_id": 10, "front": "List nedir?", "back": "Sıralı koleksiyon...", "created_at": "2026-01-02T10:05:00"}  
]
```

srs_state.json örnek:

```
[  
  {"id": 1000, "user_id": 1, "card_id": 100, "repetition": 2, "interval_days": 6, "ef": 2.36,  
  "due_date": "2026-01-08", "last_quality": 4}  
]
```

reviews.json örnek:

```
[  
  {"id": 5000, "user_id": 1, "card_id": 100, "quality": 4, "reviewed_at": "2026-01-02T10:10:00"}  
]
```

3.3 Veri Bütünlüğü Kuralları (Dosya Tabanlı)

- ID alanları benzersiz olmalıdır (ör: artan sayıç veya uuid).
- email UNIQUE olmalıdır.
- decks.user_id mevcut bir kullanıcıyı göstermelidir.
- cards.deck_id mevcut bir deck'i göstermelidir.
- srs_state (user_id, card_id) ikilisi için tekil olmalıdır.
- Deck silinince bağlı card ve srs_state kayıtları da silinmelidir (cascade).

4. Mimari (Önerilen)

Öğrencinin kodu okunur ve sürdürülebilir tutması için katmanlı yapı önerilir:

- main.py -> CLI menüsü ve uygulama akışı
- storage.py -> JSON okuma/yazma, atomic write, id üretimi
- auth.py -> kayıt/giriş/şifre hashleme (hashlib + salt)
- deck_service.py -> deck iş mantığı
- card_service.py -> kart iş mantığı
- review_service.py -> SM-2 hesaplama ve review kaydı
- report_service.py -> rapor hesapları
- utils.py -> input doğrulama, tarih yardımcıları, loglama

4.1 Atomic Write (Zorunlu Tavsiye)

Dosya bozumlalarını önlemek için öneri: JSON kaydederken önce geçici dosyaya yazıp (tmp) sonra replace ile asıl dosyanın üstüne geç. Bu yaklaşım, uygulama kapanırsa yarılmış dosya riskini azaltır. (pathlib + os.replace)

4.2 Kod Kalitesi Beklentileri

- Fonksiyonlar kısa ve tek sorumluluklu olmalı.
- Her modülde en az 1 docstring bulunmalı.
- Hata mesajları anlaşılabilir olmalı (ValueError / custom Exception kullanılabilir).

- logging ile en az: login denemeleri, veri ekleme/silme, review kaydı loglanmalı.

5. CLI (Komut Satırı) Akışları

Uygulama menü tabanlı olabilir. Örnek akış:

6. 1) Kayıt / Giriş
7. 2) Deck işlemleri (oluştur, liste, güncelle, sil)
8. 3) Kart işlemleri (ekle, liste, güncelle, sil)
9. 4) Bugün çalış (due kartları sırayla getir, kalite puanı al, güncelle)
10. 5) Raporlar (bugün due, son 7 gün istatistik)
11. 6) Yedekle (bonus)
12. 7) Çıkış

5.1 Örnek Komut Diyalogları (Kısa)

Örnek: Bugün çalış

Kullanıcı: 4 (Bugün Çalış)

Sistem: 3 kart due. Kart #100: 'HTTP nedir?'

Sistem: Cevap gösterilsin mi? (E/H) -> E

Sistem: Cevap: 'Hypertext Transfer Protocol...'

Sistem: Kalite puanı (0-5): 4

Sistem: Güncellendi -> due_date: 2026-01-08

6. Test Zorunluluğu (unittest)

Öğrenci aşağıdaki minimum testleri yazmalıdır (en az 10 test):

- Kayıt: aynı email ile ikinci kayıt engellenir.
- Giriş: yanlış parola reddedilir.
- Deck CRUD: oluştur-listele-güncelle-sil akışı.
- Card CRUD: oluştur-listele-güncelle-sil akışı.
- Review: kalite<3 durumunda repetition resetlenir ve due_date = today+1 olur.
- Review: kalite>=3 durumunda interval büyür.
- Due list: due_date <= today olan kartlar listelenir.
- İzolasyon: farklı kullanıcı deck/kart verilerini göremez.
- Dosya yazma: atomic write ile kaydedilen JSON bozulmaz (basit senaryo).
- Cascade silme: deck silinince bağlı card ve srs_state kayıtları da silinir.

7. Geliştirme Aşamaları (Milestone)

13. M1 (Gün 1-2): Dosya tabanlı storage katmanı + id üretimi + ilk JSON formatı
14. M2 (Gün 3-4): Auth (register/login) + parola hashleme + oturum yönetimi
15. M3 (Gün 5-6): Deck & Card yönetimi + CLI menü
16. M4 (Gün 7-8): Review akışı + SM-2 hesaplama + due listesi
17. M5 (Gün 9): Raporlama + loglama + yedekleme (bonus)
18. M6 (Gün 10): Testler + README + son düzenlemeler

8. Değerlendirme Rubriği (100 Puan)

Kriter	Açıklama	Puan
Dosya Tabanlı Veri Modeli	JSON/CSV yapısı doğru, veri kalıcı, bütünlük korunuyor.	20
Auth & Güvenlik	Hash+salt, kullanıcı izolasyonu, güvenli doğrulama.	15
Deck/Kart CRUD	Tüm CRUD çalışıyor, hatalar yönetiliyor.	20
Review + SRS	SM-2 mantığı doğru, due_date hesapları doğru.	20
Raporlama	Due listesi + 7 gün istatistikleri.	10
Kod Kalitesi	Modüler yapı, okunabilirlik, docstring, loglama.	10
Testler	unittest ile min 10 test, edge-case kapsama.	5

Bonus (en fazla +10): yedekleme, CSV rapor çıktısı, gelişmiş arama/filtreleme, import özelliği.

9. Teslim Şartları

- Kaynak kod (klasör yapısı düzenli).
- data/ klasörü ve örnek JSON dosyaları (ilk örnek veriler opsiyonel).
- README.md (kurulum, kullanım, menü akışları, veri formatı).
- Test klasörü (unittest).
- Kısa demo videosu (opsiyonel ama güçlü artı).

9.1 Kabul Kriterleri (Minimum)

19. python main.py ile uygulama açılıyor ve menü çalışıyor.
20. Kayıt + giriş + çıkış çalışıyor.
21. Deck ve kart CRUD tamamen çalışıyor.
22. Review akışı kalite puanı alıp due_date güncelliyor.
23. Bugün due olan kartlar listeleniyor.
24. Veriler dosyaya kaydediliyor ve uygulama kapanıp açılınca kaybolmuyor.

25. Testler çalışıyor: `python -m unittest`
26. SQL kullanılmadığı açıkça ispatlanabilir (kodda sqlite3 yok).

Ek A: İpuçları (Pure Python)

- Parola hashleme için `os.urandom` ile salt üret ve `hashlib.pbkdf2_hmac` kullan.
- Tarih alanlarını ISO formatında sakla: `YYYY-MM-DD` ve ISO `datetime`.
- Dosya okuma/yazma için `pathlib.Path` kullan; klasörü yoksa oluştur.
- JSON yazarken atomic write uygula: tmp dosyaya yaz -> `os.replace` ile değiştir.
- ID üretimi için artan sayıç (`counters.json`) veya `uuid.uuid4` kullanabilirsin.
- Hataları kullanıcıya anlaşılır göster, loglara teknik detay yaz.