İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
FEN BİLİMLERİ ENSTİTÜSÜ

Web ve Mobil Uyumlu Gelir – Gider Takip Yazılımı

Yazılım Mühendisliği Ana Bilim Dalı

Dönem Projesi

Gizem ÇETİN

Öğrenci No Y220240081

Proje Danışmanı: Dr. Öğr. Üyesi Serpil YILMAZ

Ocak 2024

Web ve Mobil Uyumlu Gelir – Gider Takip Yazılımı

Öz

Gelir ve gider dengesi para yönetimindeki en önemli unsurdur. Bu dengede yer alan kredi kartı gibi alternatif harcama yöntemlerinin avantajları olduğu gibi dezavantajları da bulunmaktadır. Bu dezavantajları ortadan kaldırmak için harcamaların; hatırlatma, öneri ve uyarı fonksiyonları ile çalışan bir bildirim sistemi sayesinde kontrol altına alınması hedeflenmektedir. Bu amaçla gelir ve giderlerin detaylı olarak takibinin yapılabileceği web ve mobil uyumlu bir yazılım geliştirilecek olup; bu yazılım PHP dilinde OOP (Nesne Yönelimli Programlama) ve MVC (Model-View-Controller) mimarisi ile ilişkisel veri tabanı yapısı kullanılarak geliştirilecektir. Yapı olarak Laravel Framework kullanılacak olup ilgili veriler ilişkisel olarak Eloquent model yapısı ile yönetilecektir. Yapıda CRUD işlemleri yer alacaktır ve verilerin kayıt altına alınabilmesi için MySQL veri tabanı kullanılacaktır. Ayrıca oluşan verilere göre; bildirim sistemi de aktif olarak devrede olacaktır. Bunun yanı sıra hedef sistemi de yer alacak olup bu sayede ulaşılması hedeflenen herhangi bir ihtiyacın doğru şekilde gerçekleşebilmesi de sağlanacaktır. Sonuç olarak raporlama ve bildirim sistemi sayesinde harcamaların optimum seviyede tutulması hedeflenmektedir.

Anahtar Sözcükler: OOP, MVC, PHP, MySQL, Laravel, CSS, veri tabanı, web, mobil, yazılım

Web and Mobile Compatible Income - Expense Tracking Software

Abstract

The balance between income and expenses is a crucial element in financial management. Alternative spending methods such as credit cards have both advantages and disadvantages. To eliminate these disadvantages, it is aimed to control expenses through a notification system that operates with reminder, suggestion, and warning functions. For this purpose, a web and mobile-compatible software will be developed for detailed tracking of income and expenses. This software will be developed using the PHP language with Object-Oriented Programming (OOP) and Model-View-Controller (MVC) architecture, utilizing a relational database structure. Laravel Framework will be used as the development framework, and the relevant data will be managed relationally using the Eloquent model structure. CRUD operations will be included in the structure, and a MySQL database will be used to record the data. Additionally, an active notification system will be in place based on the generated data. Furthermore, a target system will be integrated, ensuring the accurate realization of any intended need. In conclusion, the goal is to maintain expenses at an optimum level through reporting and notification systems.

Keywords: OOP, MVC, PHP, MySQL, Laravel, CSS, database, web, mobile, software

Teşekkür

Proje çalışmasına katkılarından dolayı müstakbel eşim A. Burak DALDAL'a teşekkür ederim.

İçindekiler

Şekiller Listesi

Tablolar Listesi

Kısaltmalar Listesi

MVC	Model-View-Controller
OOP	Object-Oriented Programming
CRUD	Create Read Update Delete
PHP	Hypertext Preprocessor
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
2FA	Two Factor Authentication
ms	Milisaniye

Bölüm 1

Giriş

Günümüzde finansal dengeyi sağlamak, gelir ve giderleri kontrol altında tutmak her zamankinden daha önemli hale gelmiştir. Para yönetimi, bireylerin ve işletmelerin sürdürülebilir bir mali sağlık elde etmelerinde kritik bir rol oynamaktadır. Ancak, gelir ve giderler arasındaki dengeyi korumak, özellikle farklı harcama yöntemlerinin etkisiyle karmaşık hale gelmiştir.

Bu zorluğu aşmak ve finansal yönetimi kolaylaştırmak adına, gelir ve giderlerin detaylı bir şekilde takip edilebileceği, hatırlatma, öneri ve uyarı fonksiyonlarıyla desteklenen bir bildirim sistemi geliştirme hedeflenmektedir. Bu amaca yönelik olarak, web ve mobil uyumlu bir yazılım geliştirilecek olup, bu yazılım PHP dilinde OOP ve MVC mimarisi kullanarak, ilişkisel veri tabanı yapısıyla entegre edilecektir.

1.1 Dönem Projesinin Amacı

Bu dönem projesinin amaçları şunlardır;

- Literatür taraması yaparak, proje konusuna uygun veri tabanı tasarımı geliştirmek ve bu tasarımın ilişkisel bir veri tabanı yapısı ile oluşturulmasını sağlamak.
- PHP dilinde Laravel framework ile MVC ve OOP tabanlı, web ve mobil uyumlu yazılım geliştirmek.
- Gelir ve giderlerin başarılı şekilde takip edilebilmesi, raporlanması ve izlenebilirliğini sağlamak.
- Hatırlatma, öneri ve uyarı fonksiyonları içeren bildirim sistemi ile kullanıcıların harcamalarını daha etkin bir şekilde yönetebilmelerini sağlamak.

1.2 Literatür Taraması

1.2.1 Veri Tabanı Tasarımı

Veri tabanı tasarımı, bir bilgi sistemi içindeki verilerin düzenlenmesi, depolanması ve yönetilmesi için bir plan veya yapı oluşturma sürecidir. Bu süreç, veri tabanındaki verilerin kullanıcı ihtiyaçlarına uygun, etkili bir şekilde depolanmasını, erişilmesini ve güncellenmesini sağlamak amacıyla gerçekleştirilir. Veri tabanı tasarımı, veri bütünlüğünü koruma, veri anlamını artırma, sorgu performansını optimize etme gibi hedeflere odaklanır (Kralev ve Kraleva, 2017).

Veri tabanı tasarımının temel unsurları şunlardır:

- Veri Modelleme: Veri modelleme, veri tabanındaki verilerin ve aralarındaki ilişkilerin soyut bir temsilini oluşturmayı içerir. İlişkisel model, en yaygın kullanılan modeldir.
- Normalizasyon: Normalizasyon, veri tabanındaki ilişkisel tabloları daha düzenli ve optimize hale getirmeyi amaçlar. Bu süreç, veri depolama yapısını iyileştirerek gereksiz tekrarları ve veri tutarsızlıklarını önler.
- İlişkisel Şema Tasarımı: İlişkisel şema tasarımı, veri tabanındaki tabloların, sütunların ve bu tablolar arasındaki ilişkilerin belirlenmesini içerir. Bu adım, kullanıcıların verilere etkili bir şekilde erişmelerini sağlar (Sumathi ve Esakkirajan, 2007).
- Veri tabanı Yönetim Sistemi (DBMS) Seçimi: Veri tabanı tasarımında kullanılacak DBMS'nin seçimi önemlidir. DBMS, veri tabanının oluşturulması, güncellenmesi, sorgulanması ve yönetilmesi için gerekli olan araçları sağlar.
- Performans Optimizasyonu: Veri tabanı tasarımında performans optimizasyonu, sorgu hızlarını artırmak ve veri tabanı erişimini iyileştirmek için indeksleme, normalizasyon ve diğer teknikleri içerir.
- Güvenlik ve Yetkilendirme: Veri tabanı tasarımı sırasında güvenlik ve yetkilendirme unsurları dikkate alınmalıdır. Bu, verilere yetkisiz erişimi önlemek ve veri bütünlüğünü korumak için güvenlik politikaları oluşturmayı içerir.

- Yedekleme ve Kurtarma Stratejileri: Veri tabanı tasarımında, veri kaybını önlemek ve felaket durumlarına karşı hazırlıklı olmak için düzenli yedekleme ve kurtarma stratejileri geliştirilmelidir.

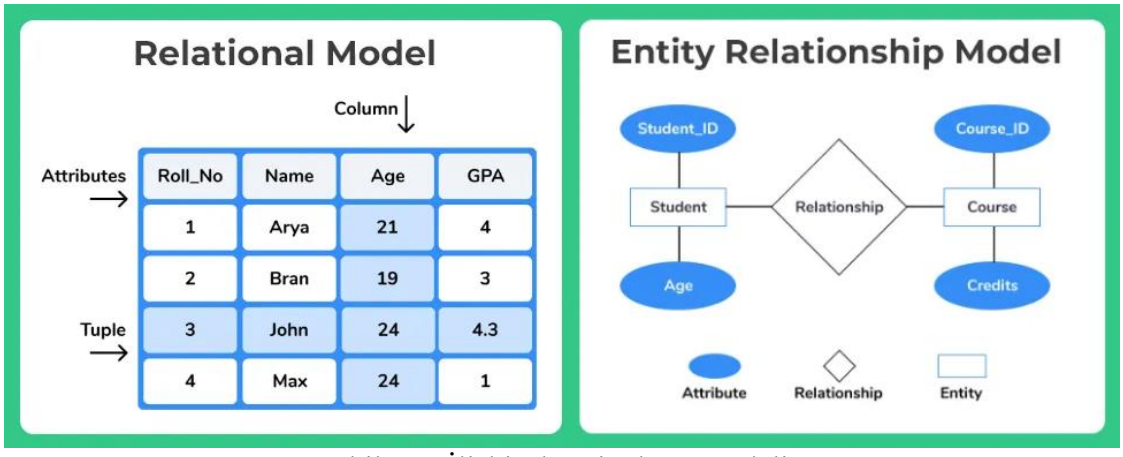
Dolayısıyla; doğru bir veri tabanı tasarımı, veri entegrasyonunu kolaylaştırır, veri tabanı performansını artırır ve kullanıcıların verilere güvenli ve etkili bir şekilde erişmelerini sağlar.

1.2.1.1 Veri Modelleme

Veri modelleme, bir bilgi sistemi içinde kullanılacak verilerin, bu veriler arasındaki ilişkilerin ve veri yapılarının soyut bir temsilini oluşturmayı içeren bir süreçtir. Bu temsil, genellikle bir diyagram veya şema şeklinde olabilir ve bilgi sistemini anlamak, tasarlamak ve uygulamak için kullanılır.

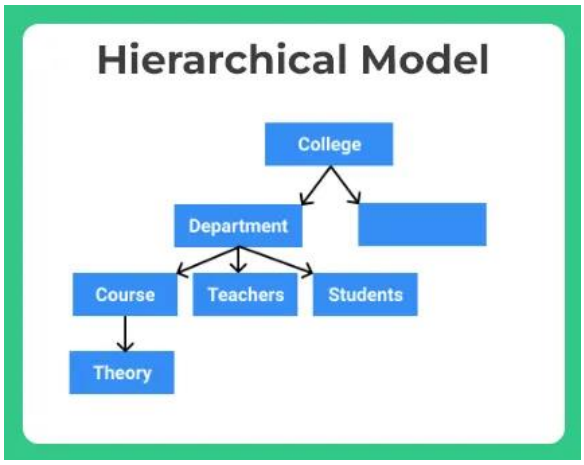
4 adet veri modeli türü bulunmaktadır bu modeller ise şunlardır:

- İlişkisel Model: Veri tabanı tasarımında en yaygın kullanılan veri modelidir. Bu model, tablolar arasında kullanılan yapıyı temsil eder. Her tablo, belirli bir veri türündeki bilgileri içerir ve tablolar arasındaki ilişkiler, genellikle birincil anahtarlar ve dış anahtarlar kullanılarak tanımlanır (Dimitrieski vd., 2015). Örneğin, bir e-ticaret veri tabanında "Ürünler" ve "Siparişler" adlı iki tablo arasında bir ilişki olabilir.



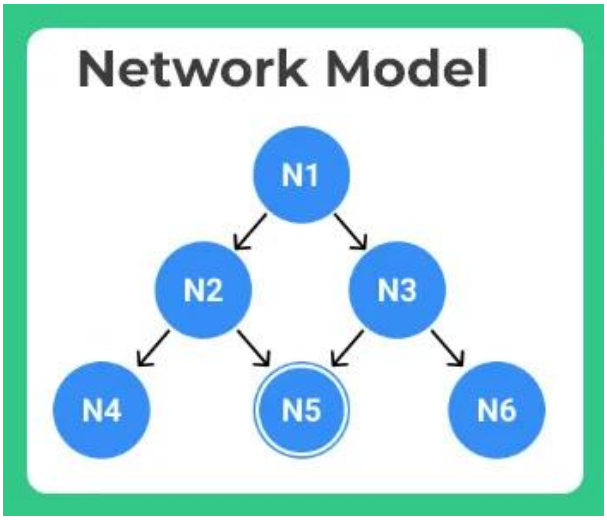
Şekil 1.1: İlişkisel veri tabanı modeli

- Hiyerarşik Model: Verilerin ağaç benzeri bir hiyerarşi içinde düzenlendiği bir veri modelidir. Bu modelde, her düğüm, bir üst düğümle ilişkilidir ve altındaki bir veya daha fazla düğümü olabilir. Örneğin, bir organizasyon yapısını temsil eden bir hiyerarşik modelde, her birim bir düğümü temsil edebilir ve bu birimlere bağlı alt birimler olabilir.



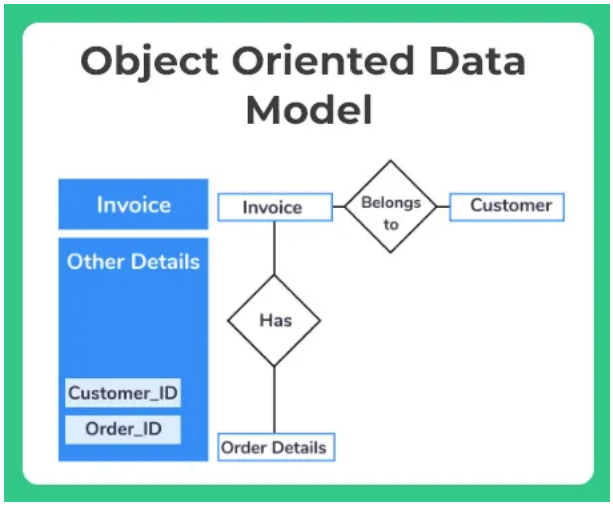
Şekil 1.2: Hiyerarşik veri tabanı modeli

- Ağ Model: Verilerin düğümler ve kenarlar aracılığıyla ilişkilendirildiği bir veri modelidir. Bu modelde, bir düğüm, birden çok diğer düğümle bağlantılı olabilir. Örneğin, bir otomobil üretim sürecini temsil eden bir ağ modelde, her bir üretim aşaması bir düğümü temsil edebilir ve bu düğümler birbirleriyle bağlantılı olabilir.



Şekil 1.3: Ağ veri tabanı modeli

- Nesne Tabanlı Model: Nesne yönelimli programlama (OOP) prensiplerini temel alır. Her bir veri öğesi bir nesneyi temsil eder ve bu nesneler arasındaki ilişkiler, OOP prensiplerine göre modellenir. Bu model, karmaşık sistemlerin tasarımını ve anlaşılmasını kolaylaştırabilir (Silberschatz vd., 2001).



Şekil 1.4: Nesne tabanlı veri tabanı modeli

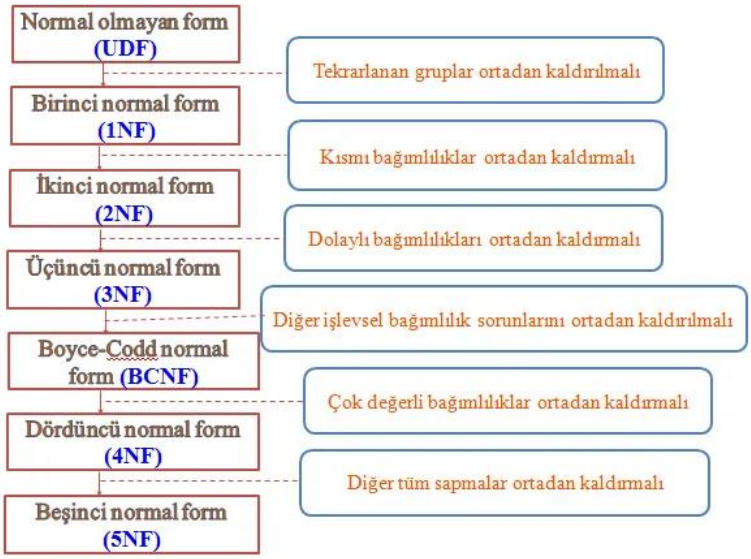
Her bir veri modeli türü, belirli bir problem alanına veya uygulama gereksinimine daha uygun olabilir. Genellikle, kullanılacak olan veri modeli, bilgi sistemini daha etkili ve kullanışlı hale getirmek adına dikkatlice seçilir.

1.2.1.2 Normalizasyon

Normalizasyon, veri tabanı tasarım aşamasında veri tekrarını, veri kaybını veya veri yetersizliğini önlemek için gerçekleştirilen işlemlerdir. Veri tabanındaki tabloların yapısını düzenleyerek, depolanan verilerin bütünlüğünü ve performansını artırmayı hedefler. Normalizasyon uygulanan veri tabanlarının performansı artar, sabit diskteki boyutu azalır ve tablolardaki satır ve sütun sayısı azalacağından veri tekrarı önlenmiş olur. Özellikle silme, güncelleme gibi işlemlerde çıkabilecek sorunlar büyük oranda azaltılmış olur (Powell, 2006).

Normalizasyonun veri tabanına uygulanabilmesi için belli seviyeleri vardır. Normalizasyonun her bir kuralı yani seviyeleri normal form olarak adlandırılır. Bu seviyeler gereksiz veri tekrarlarını ne derecede engellediği ve tutarlılığı ne kadar sağladığına bağlı olarak derecelendirilir. Seviye yükseldikçe veri tutarlılığı artar, veri tekrarı düşer.

Normalizasyon seviyeleri 1NF (Birinci Normal Form), 2NF, 3NF, BCNF (Boyce-Codd Normal Form, 3.5NF'de denir), 4NF şeklinde adlandırılır ve yukarı doğru devam eder.



Şekil 1.5: Normalizasyon seviyelerinin çalışma mantığı

Temel normalizasyon seviyeleri:

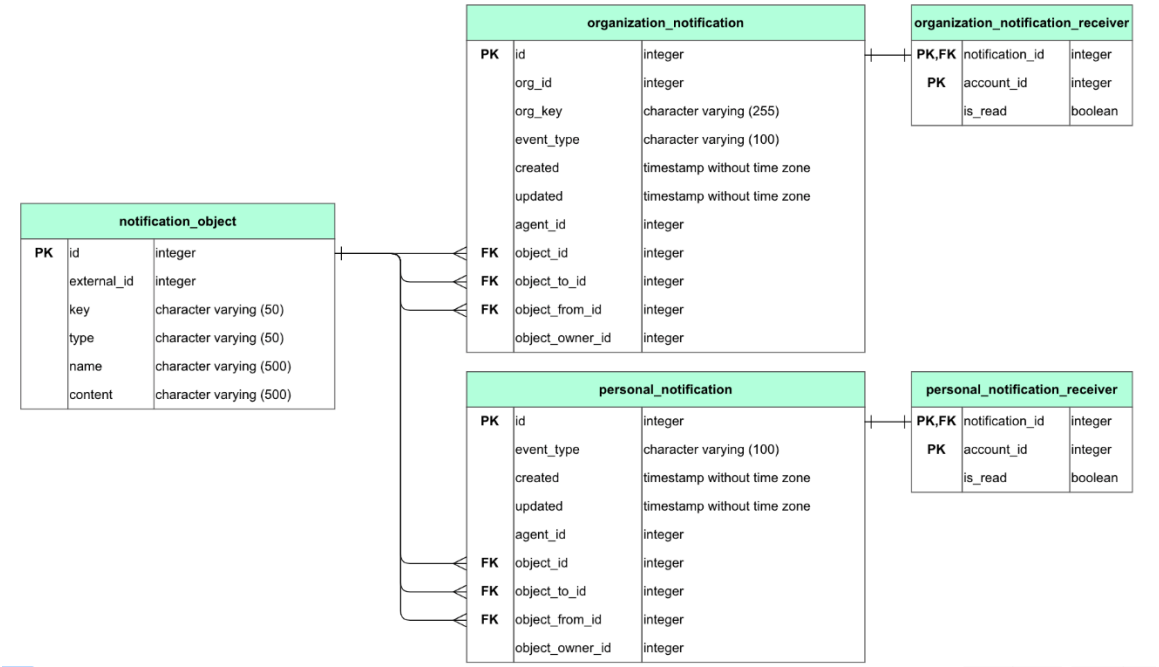
- 1NF (1. Normal Form): Bir tablonun 1. Normal Form'a uygun olması için her hücrede sadece bir değer bulunmalıdır. Yani, bir hücre içinde birden fazla veri veya alt bileşen bulunmamalıdır Aynı zamanda, her sütun tek bir değer tipini içermelidir. Bu aşama, veri tablolarındaki temel düzeni sağlar.
- 2NF (2. Normal Form): Bir tablonun 2. Normal Form'a uygun olması için öncelikle 1. Normal Form'a uyması gerekmektedir. 2NF, tablodaki tüm sütunların, tablo anahtarına tamamen bağımlı olmasını sağlar. Yani, tablo içindeki her bir değer, tablo anahtarına bağlı olarak belirgin bir şekilde tanımlanmalıdır. Bu, birincil anahtarın her sütunu tam olarak tanımlamasını sağlar.
- 3NF (3.Normal Form): Bir tablonun 3. Normal Form'a uygun olması için öncelikle 2. Normal Form'a uyması gerekmektedir. 3NF, tablodaki tüm sütunların anahtar dışında hiçbir anahtar tarafından belirlenmemesini sağlar. Bu, bir sütunun diğer bir sütuna bağımlı olması önler. Yani, her sütun yalnızca tablo anahtarına bağlı olarak belirlenmelidir.

- 4NF (4. Normal Form): Bir tablonun 4. Normal Form'a uygun olması için öncelikle 3. Normal Form'a uyması gerekmektedir. 4NF, tablodaki her çok-değerli bağımlılığın, birincil anahtarın bir alt kümesine bağımlı olmadığı anlamına gelir.

Bu temel normalizasyon seviyeleri, veri tabanı tasarımında veri bütünlüğünü sağlamak ve veri tabanı yapılarını optimize etmek için kullanılır. Ancak, her durumda aşırı normalizasyonun kaçınılması ve belirli uygulama gereksinimlerine göre esneklik gösterilmesi önemlidir. Normalizasyon süreci, veri tabanı tasarımının bir parçasıdır ve belirli bir uygulama veya kullanım senaryosuna bağlı olarak, aşırı normalizasyondan kaçınarak veya belirli durumlarda geçerli olan normalizasyon seviyelerini seçerek uyarlanabilir (Lightstone vd., 2007).

1.2.1.3 İlişkisel Şema Tasarımı

İlişkisel şema tasarımı, bir veri tabanındaki veri tablolarının, sütunların ve bu tablolar arasındaki ilişkilerin tanımlandığı süreci ifade eder. İlişkisel veri tabanı tasarımının temel amacı, verileri düzenli bir şekilde depolayarak veri bütünlüğünü korumak ve etkili bir şekilde sorgulama yapılmasını sağlamaktır.



Şekil 1.6: İlişkisel şema tasarımı örneği

İlişkisel şema tasarımının temel kavramları ise şunlardır:

- Tablo Tanımları: İlk adım, veri tabanında hangi bilgilerin saklanacağını belirlemektir. Her bir konsept veya varlık için bir tablo oluşturulur. Örneğin, bir kütüphane veri tabanında "Kitaplar", "Yazarlar" ve "Kategoriler" gibi tablolar olabilir.
- Sütun Tanımları: Her tablo içinde, bu tabloya ait özellikleri temsil eden sütunlar tanımlanır. Her sütun, belirli bir veri türünü ve özelliği temsil eder. Örneğin, bir "Kitaplar" tablosunda "Kitap Adı", "Yazar ID", "Yayın Yılı" gibi sütunlar olabilir.
- Birincil Anahtar (Primary Key) Tanımları: Her tablonun birincil anahtarı (primary key) belirlenir. Birincil anahtar, tablodaki her bir kaydı eşsiz bir şekilde tanımlayan bir veya birden fazla sütundan oluşur. Bu genellikle bir ID sütunu olabilir.
- Yabancı Anahtar (Foreign Key) Tanımları: İki tablo arasındaki ilişkiyi ifade etmek için yabancı anahtarlar kullanılır. Yabancı anahtar, bir tablonun birincil anahtarını diğer bir tablo içinde referans alır. Bu sayede ilişkilendirilmiş kayıtlar arasında bağlantı kurulur.
- Normalizasyon: İlişkisel şema tasarımında, tabloların normalizasyonu önemlidir. Bu, verilerin depolanma şeklini optimize eder ve veri tabanının bütünlüğünü korur. Normalizasyon, 1. Normal Form (1NF), 2. Normal Form (2NF), 3. Normal Form (3NF) gibi seviyeleri içerir.
- İndeksleme: İndeksleme, sık sık sorgulanan sütunlar üzerinde performansı artırmak için kullanılır. Bir veya daha fazla sütunun indekslenmesi, sorguların daha hızlı gerçekleşmesine olanak tanır.

İlişkisel şema tasarımı, veri tabanının etkili ve sürdürülebilir olmasını sağlar. Tasarım süreci, genellikle kullanıcı ihtiyaçlarına, veri bütünlüğüne ve performansa odaklanarak gerçekleştirilir. İyi bir tasarım, veri tabanının hangi tipte verileri barındıracağını, tablolar arasındaki ilişkileri ve tablo bağımlılıklarını anlaşılır bir şekilde ortaya koyar. Bu sayede, veri tabanının sürdürülebilirliği daha net bir şekilde anlaşılır. Dolayısıyla veri tabanına eklenecek olan yeni bir tablo, şema tasarımına bağlı olarak, veri bütünlüğünü bozmadan sisteme entegre edilebilir.

Özetle doğru bir veri tabanı tasarımı, kullanıcının ihtiyaçlarına uygun, veri bütünlüğünü koruyan, performanslı ve ölçeklenebilir bir yapı oluşturmayı hedefler. Bu tasarım süreci, veri tabanının güncelliğini ve doğruluğunu sürdürebilmesi için dikkatlice planlanmalı ve uygulanmalıdır. Veri modelleme, normalizasyon ve ilişkisel şema tasarımı, bu sürecin başlıca adımlarını oluşturur.

Bölüm 2

Materyal ve Metot

Bu dönem projesi, işletim sistemi olarak Ubuntu 22.04 LTS Server, web sunucusu olarak NGNIX, veri tabanı sunucusu olarak MySQL 8.0, web uygulama katmanı olarak PHP 8.1 – Laravel 10 Framework, sunum katmanı olarak Tailwind CSS ve Blade Template Engine üzerinde çalışmaktadır. Bu sayede kullanıcının gelir ve giderlerini kategorilere bağlı olarak girişini sağladığı, giderlerin kategori bazlı hesaplanması, raporlanması ve bildirim sistemi ile kullanıcının finansal durumunu izlemesi ve yönetmesi için kapsamlı bir web uygulaması sunmaktadır.

2.1 Uygulama Veri Tabanı Tasarımı

Veri tabanı tasarımı veri yönetiminde önemli bir yere sahiptir. Veri tabanının hangi veri modeli ile tasarlanacağının seçimi ise verilerin performans ve sürdürülebilirliğini belirlemede etken bir rol oynar (Akhmad ve Hisyam, 2015).

Bu dönem projesinde veri modeli olarak, ilişkisel (varlık – ilişki) model kullanılmıştır. Gelir ve gider verilerinin kayıt altına alınması yani CRUD işlemleri için tablolar arasındaki varlık ilişkileri normalizasyon kurallarına bağlı olarak, Laravel 10 Framework tarafında Eloquent Model kütüphanesi ile MySQL sisteminde oluşturulmuştur.

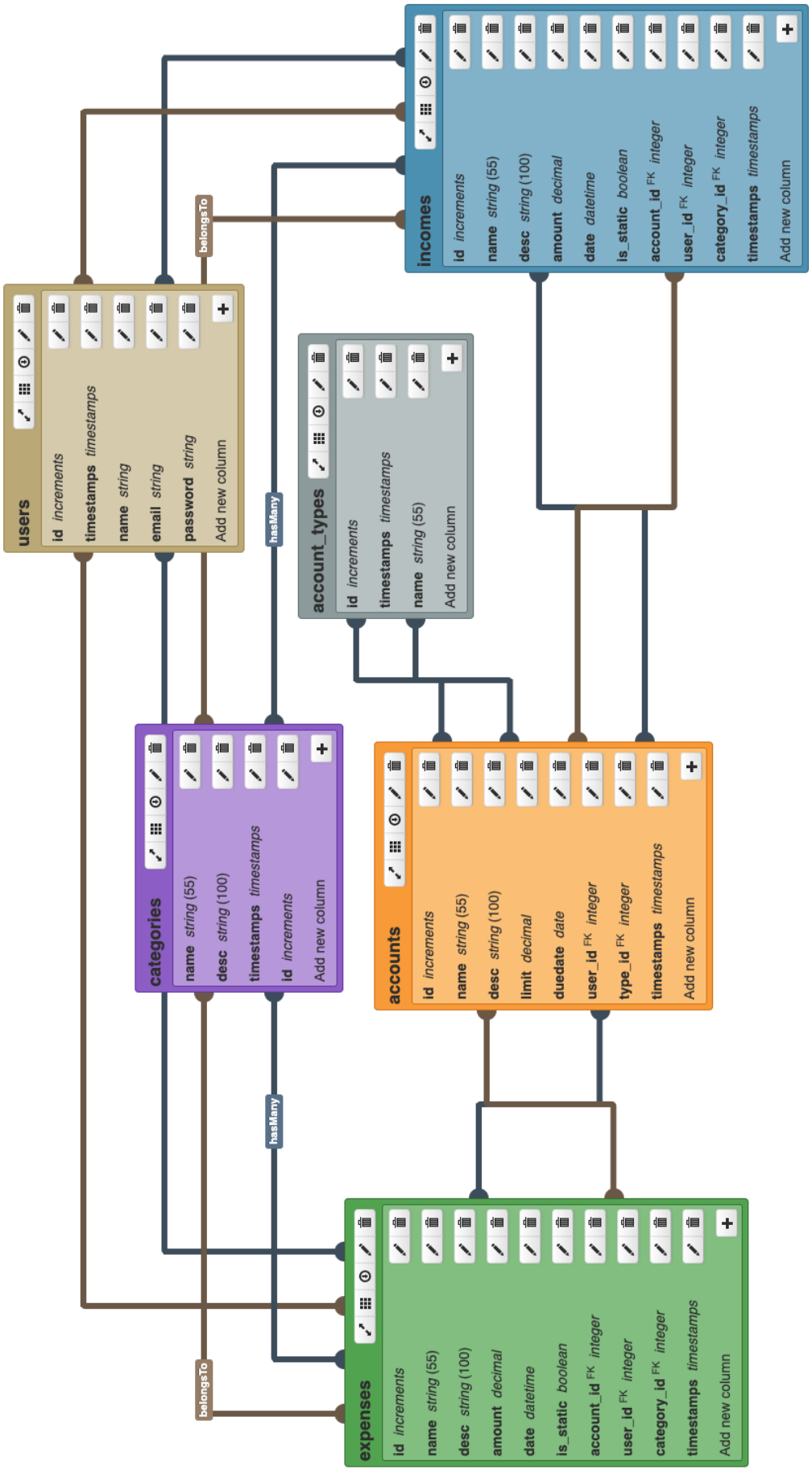
OOP yazım standartları baz alınarak toplam 6 adet tablo oluşturulmuştur. Bu tablolar, accounts, account_types, categories, expenses, incomes, users şeklindedir. Tablolar arasında bire çok (one to many) veya bire bir (one to one) ilişki bulunmaktadır. Örnek olarak, expenses tablosu ile accounts tablosu arasındaki bire çok ilişki şu şekildedir; bir varlık hesabının (banka hesabı, kredi kartı… vb.) birden fazla gider kalemi olabilmektedir. Ayrıca accounts tablosu ile, account_types tablosu arasındaki bire bir ilişki ise şu şekildedir; bir varlık hesabının bir hesap türü olabilmektedir. Tablolar

arasındaki ilişkilerin bağlayıcılığı foreign key ile sağlanmıştır. Örnek olarak, incomes tablosunda bulunan category_id (FK), categories tablosunda bulunan id (PK) alanı ile bağlıdır.

Tablolarda yer alan veri tipleri kaydedilecek olan verilerin türlerine göre belirlenmiştir. Örnek olarak, incomes ve expenses tablolarında yer alan amount alanı sayısal bir değer içereceği için decimal veri tipi atanmıştır. Yine bu tablolarda yer alan name alanı bir metin değeri içereceği için ise varchar veri tipi atanmıştır. Ayrıca bu iki tabloda yer alan categories_id (FK) alanının, yani anahtar ilişkisinin veri tipi ise unsigned integer şeklindedir.

Tüm bu işlemler Laravel'de yer alan Eloquent ORM ile gerçekleştirilmiştir. MySQL üzerinde bir veri tabanı ve veri tabanı kullanıcısı oluşturulmuştur. Oluşturulan veri tabanı ve veri tabanı kullanıcısı, Laravel'deki .env dosyasında veri tabanı bilgisi olarak belirtilmiştir ve Laravel ile MySQL veri tabanı bağlantısı sağlanmıştır. Veri tabanında oluşturulacak olan 6 adet tabloya ait migration ve modeller oluşturulmuştur. Oluşturulan migrationlarda tablo isimleri ve tabloya ait alanlar ile bu alanların veri tipleri de yer almaktadır. Modellerde ise tablolar arasındaki ilişkiler her tablo için nesne tabanlı şekilde oluşturulmuştur. Tablolar, migrate işlemi ile varlık-ilişkisi modeline bağlı olarak MySQL üzerindeki veri tabanında oluşturulmuştur.

Oluşturulan bu tablolara ait ilişkisel şema tasarımında ise tablo ve sütun tanımlamaları, anahtar tanımlamaları (Primary Key, Foreign Key) yer almaktadır. Tasarıma ait diyagram Şekil 2.1'de verilmiştir.



Şekil 2.1: Veri Tabanı Diyagramı

2.2 Uygulama Geliştirme

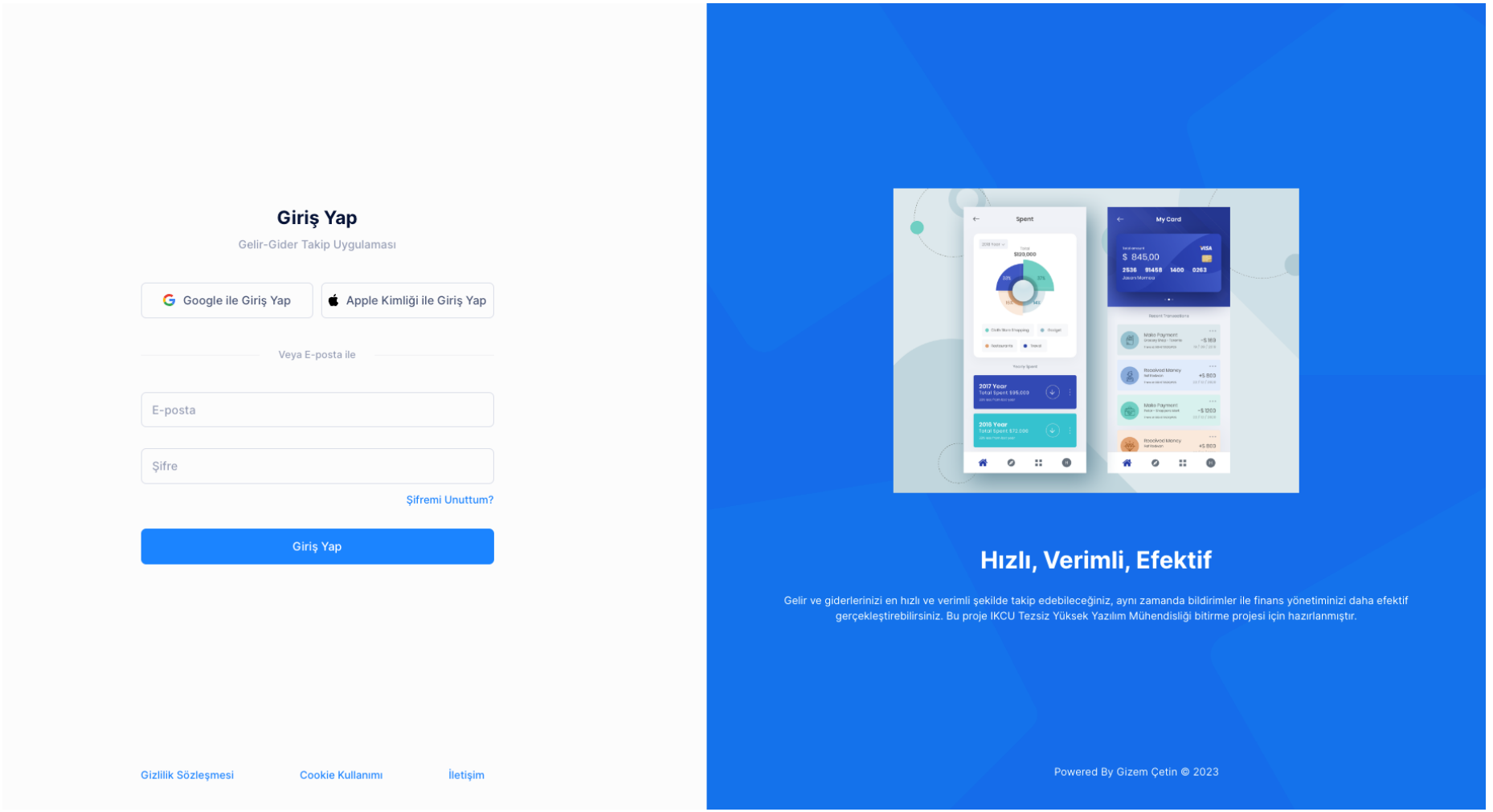
PHP günümüzde çok popüler olan bir web programlama dilidir ve birçok OOP ve MVC tabanlı framework barındırmaktadır. Laravel ise bu frameworkler arasında en gelişmiş ve en popüler olanıdır (Solanki vd., 2017). Laravel, login işlemleri, CRUD işlemleri, anlık bildirim işlemleri, otomatik işlerin yapılması gibi birçok yazılım ihtiyacını karşılamaktadır. Ayrıca blade template engine sistemi ile esnek bir view yani önyüz sistemi sunmaktadır (Parkar ve Shinde, 2015).

Önyüz sisteminin iskelet yapısını oluşturan ise CSS programlama dilidir. CSS programlama dilleri arasında ise en popüler olan framework, Tailwind CSS'tir. Tailwind CSS, modern web sayfası tasarımları ve mobil uyumlu görünümler oluşturmak için önceden tanımlanmış buton, tablo, sayfa yapıları gibi öğeleri, sınıf şeklinde sağlamaktadır (Yu, 2015).

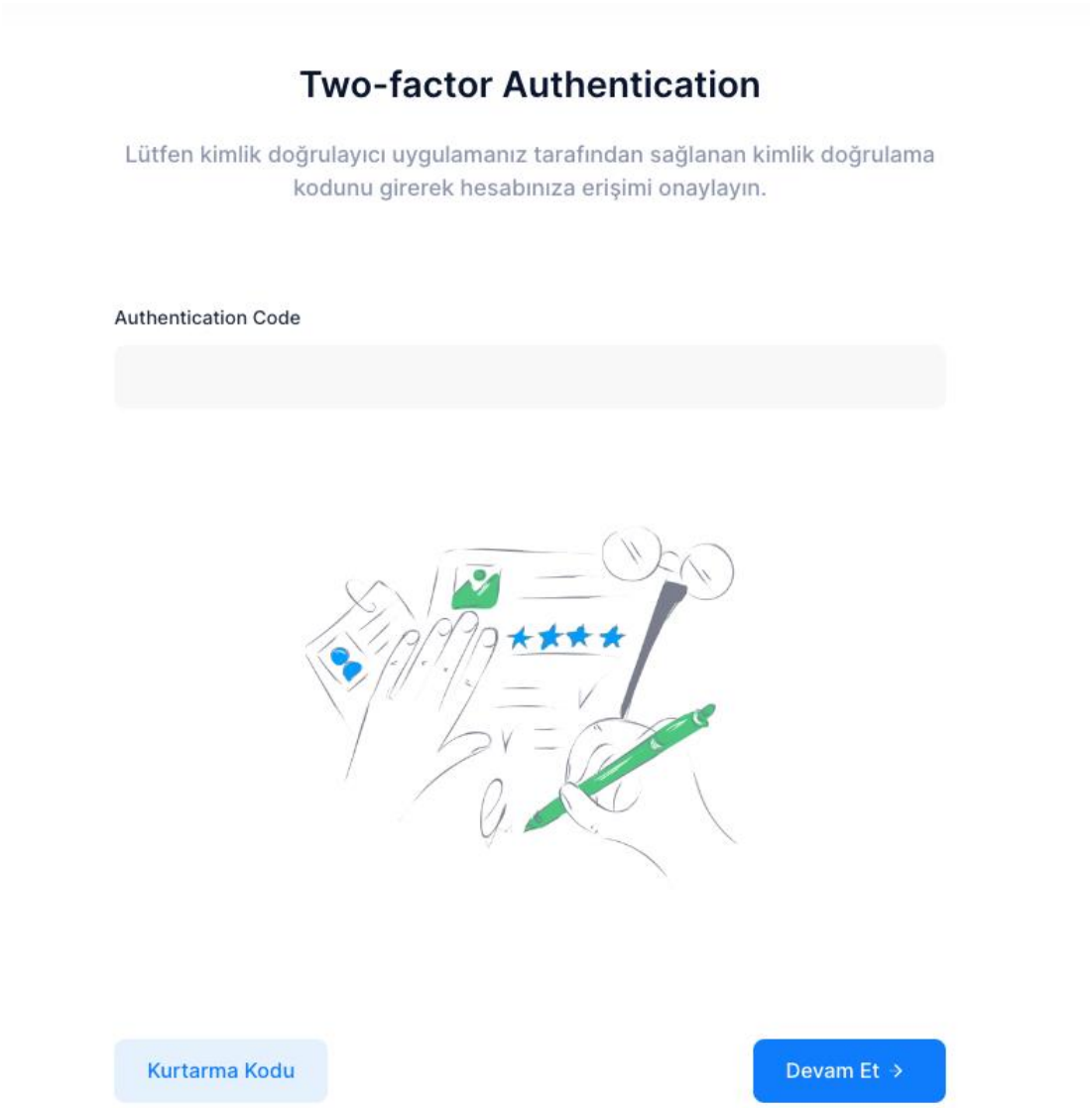
Bu dönem projesinde login ve ikili doğrulama (2FA) işlemlerini Laravel'de yer alan Fortify paketi ile, CRUD işlemlerini Laravel'de yer alan controller ve modeller ile, önyüz sistemi ve mobil uyumlu görünümü, blade template engine ve Tailwind CSS ile, anlık bildirim sistemini ise yine Laravel'de yer alan event ve brodcasting paketleri ile gerçekleştirdik.

2.2.1 Uygulama Giriş Ekranı

Uygulamaya girebilmek için Şekil 2.2'deki ekran kullanıcıya gösterilmektedir. Bu ekranda, kullanıcının Google hesabı, Apple kimliği veya sisteme kayıtlı e-posta adresi ve şifresinin sağlanması istenmektedir. Kullanıcı yanlış bir şifre ile giriş yapmaya çalışır ise hata mesajı ile karşılaşmaktadır. Ayrıca kullanıcı, ikili doğrulama giriş sistemini aktifleştirdi ise; giriş yaptıktan sonra Şekil 2.3'teki ekran gösterilmekte olup bu ekranda kullanıcının telefonundaki authenticator uygulamasındaki tek seferlik şifrenin sağlanması istenmektedir.



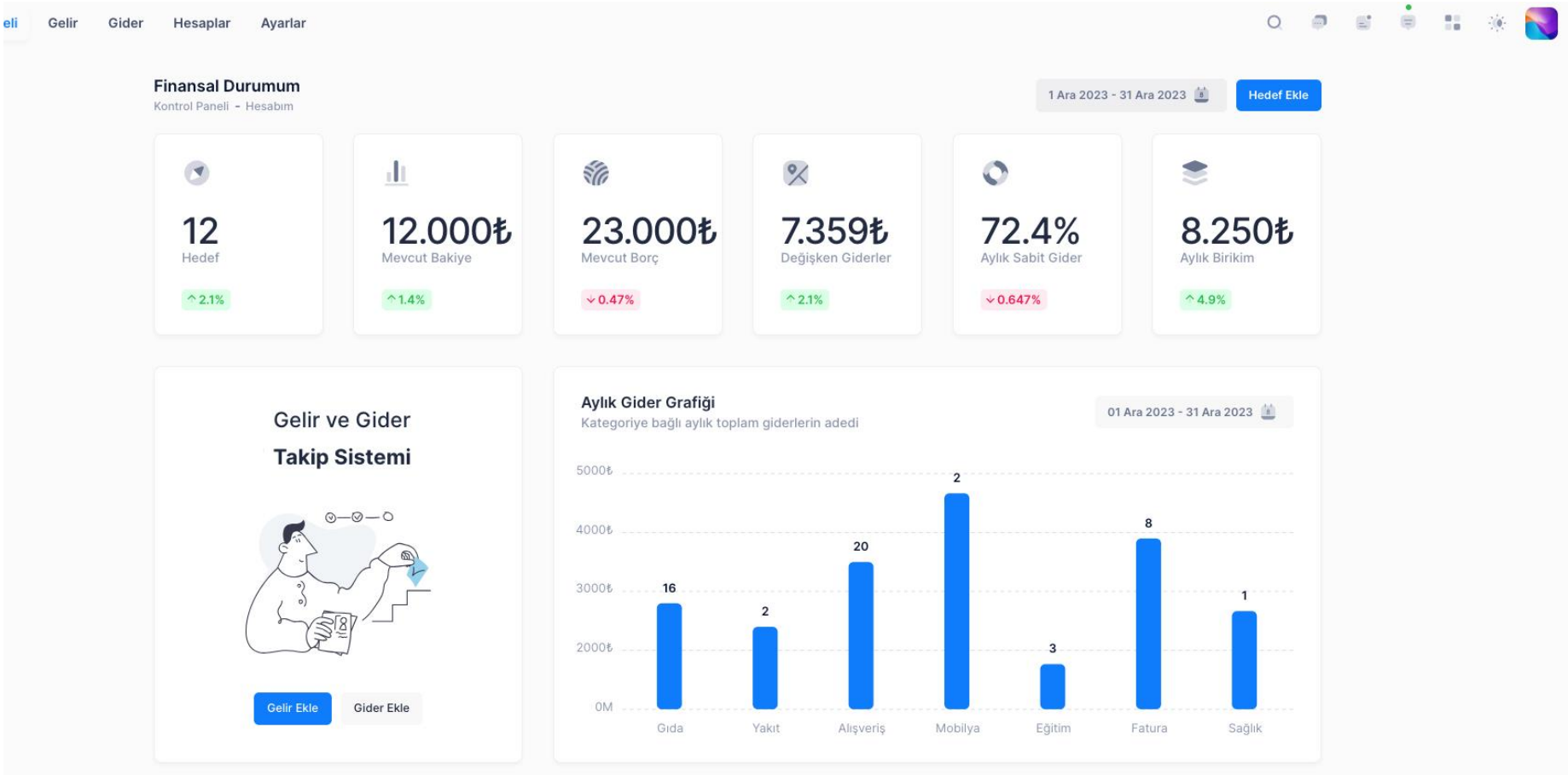
Şekil 2.2: Uygulama Giriş Ekranı



Şekil 2.3: İkili Doğrulama Ekranı

2.2.2 Uygulama Ana Ekranı

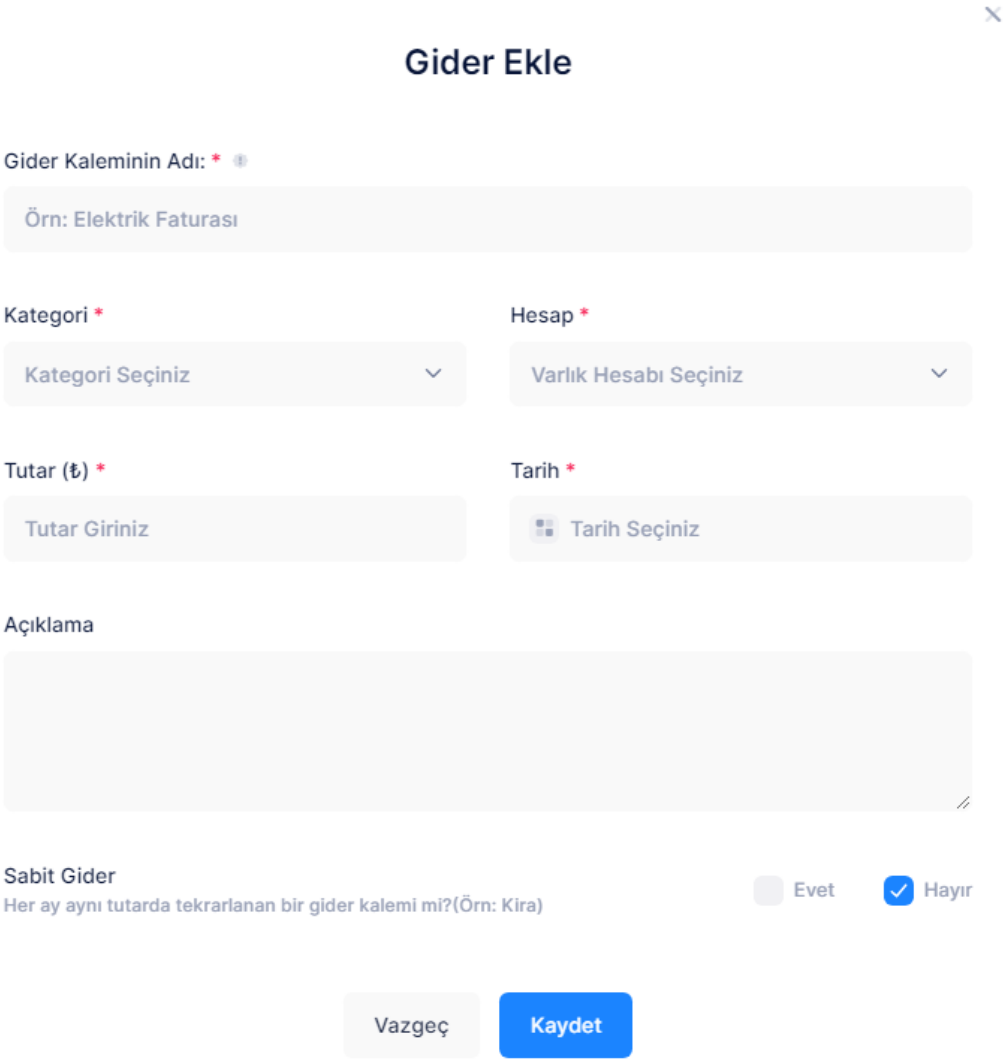
Uygulamada temel işlemleri gerçekleştirebilmek için Şekil 2.4'teki ekran kullanıcıya gösterilmektedir. Bu ekranda, grafiksel olarak kullanıcının finansal durumunun özeti gösterilmektedir. Kullanıcı aylık gider grafiğinin özetine, tarih aralığını değiştirerek ulaşabilmektedir. Kullanıcı gelir, gider ve hedef ekleme işlemlerini bu ekrandaki ilgili butonlardan gerçekleştirebilmektedir. Ayrıca üst menüde yer alan diğer butonlardan ise gelir ve giderlerin listelenmesi, hesap bilgilerinin listelenmesi ve eklenebilmesi gibi işlemleri de gerçekleştirebilmektedir. Sağ üstte yer alan butonlardan ise bildirimlerini kontrol edebilmekte ve uygulamada arama yapabilmektedir.



Şekil 2.4: Uygulama Ana Ekranı

2.2.3 Veri Giriş Ekranı

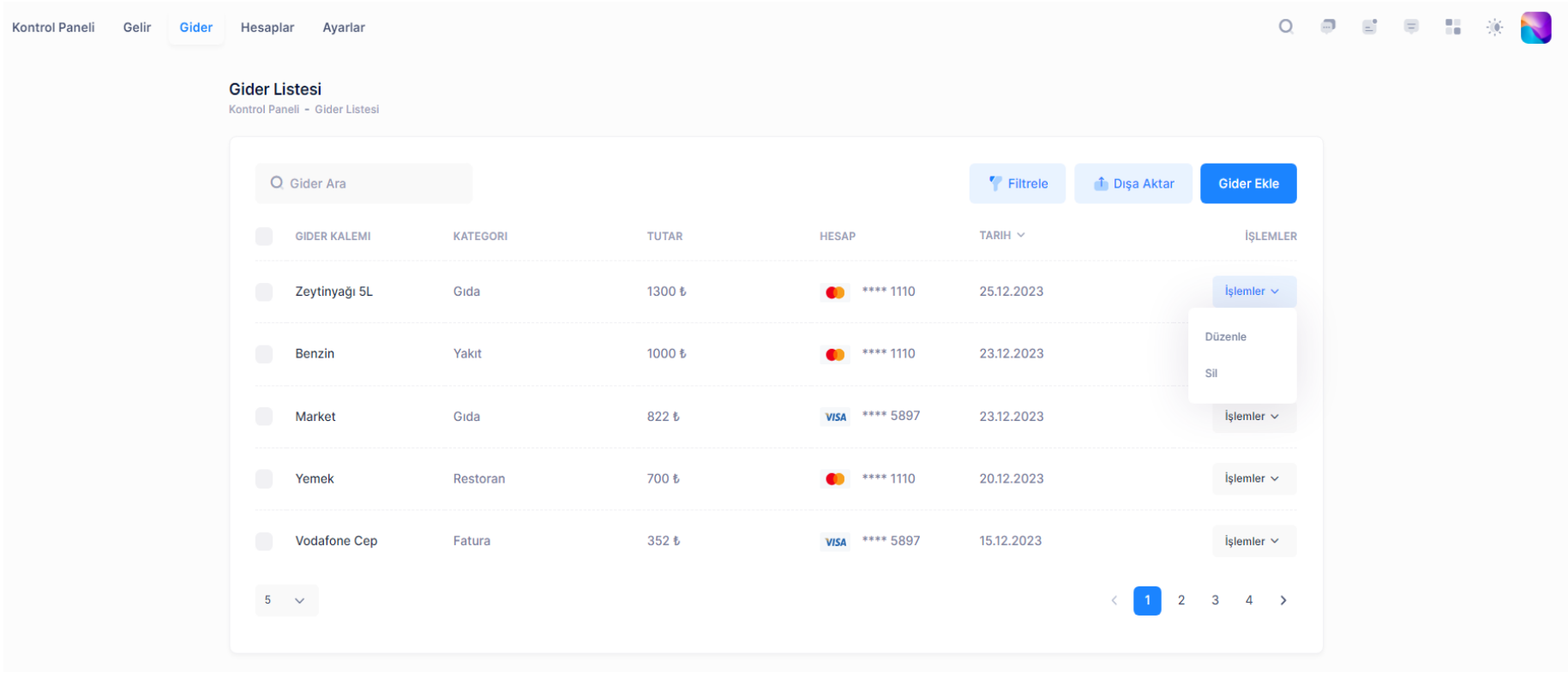
Uygulamada gelir ve gider verilerinin kaydedilmesi için Şekil 2.5'teki ekran kullanıcıya gösterilmektedir. Bu ekran kullanıcının, gelir ve gider verilerine ait alanları doldurarak veri girişi işleminin yapılmasını gerçekleştirmektedir. Gelir ve gider verilerinin kayıt altına alınabilmesi için zorunlu alanların doldurulması gerekmektedir. Kullanıcı bu alanları doldurduktan sonra kaydet butonuna basarak veri girişi işlemini gerçekleştirmektedir. Eğer zorunlu alanları doldurmadıysa bununla ilgili uyarı mesajı ile karşılaşmaktadır.



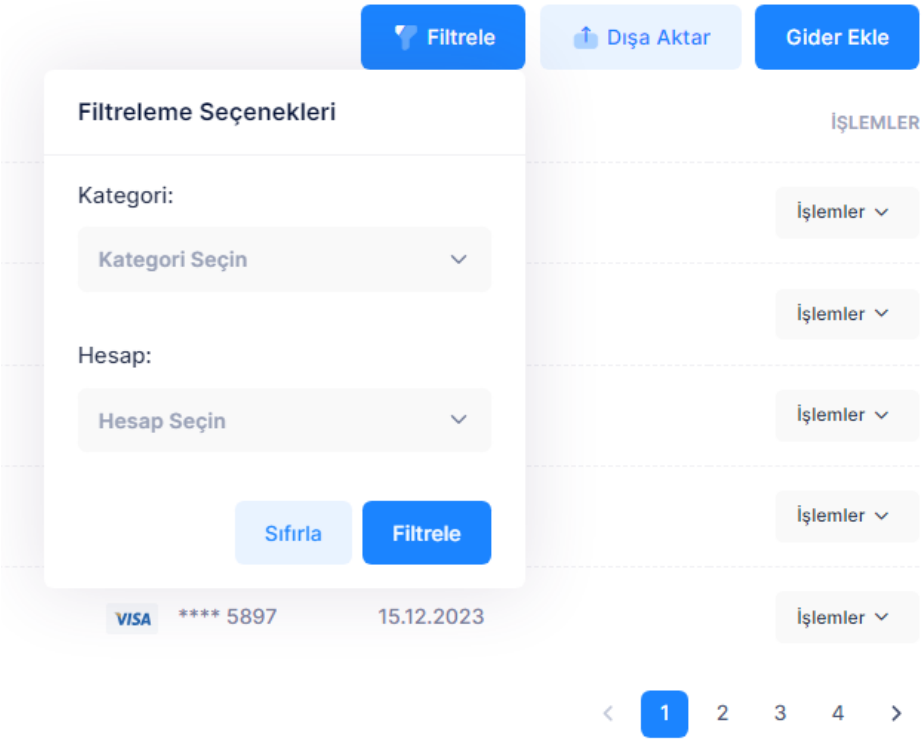
Şekil 2.5: Veri Ekleme Ekranı

2.2.4 Veri Listeleme Ekranı

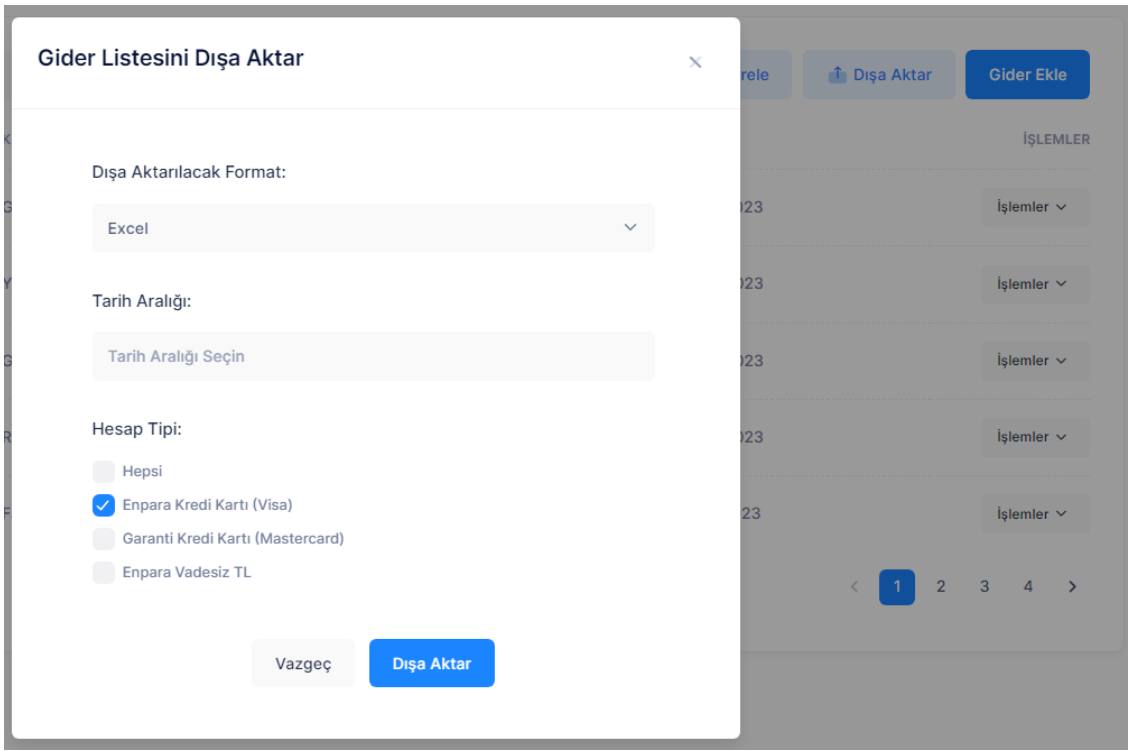
Uygulamada gelir ve gider verilerinin listelenebilmesi için Şekil 2.6'teki ekran kullanıcıya gösterilmektedir. Bu ekran kullanıcının, gelir ve gider verilerini kalem bazlı listelemektedir. Kullanıcı bu ekranda gelir ve gider kalemlerine listede yer alan kalem adı, kategori, hesap, tarih alanlarına göre arama yapabilmektedir. Kullanıcı arama kutusuna bu alanlara ait bir kelime yazdığında listede sadece o bilgiye ait aramalar gösterilmektedir. Kullanıcı filtreleme butonu ile kategori ve hesap bazlı filtreleme yapabilmektedir. Filtrele butonuna bastıktan sonra açılan pencereden kategori veya hesap seçip o kategori ya da hesaba göre filtreleme yapabilmektedir. Bu işleme ait ekran Şekil 2.7'de gösterilmiştir. Arama işlemindeki gibi aynı şekilde hesap ya da kategori alanlarına ait seçim yaptığında listede sadece o bilgiye ait filtrelenmiş veriler gösterilmektedir. Kullanıcı dışa aktar butonu ile verilerini excel'e aktarabilmektedir. Kullanıcı dışa aktar butonuna bastıktan sonra açılan pencerede dışa aktarılacak olan dosya tipini seçmesi ve bununla birlikte tarih aralığı ve hesap bilgisi seçmesi gerekmektedir. Desteklenen dosya tipleri excel ve pdf şeklindedir. Bu ekran Şekil 2.8'de gösterilmiştir. Ayrıca kullanıcı işlemler butonundan listedeki yer alan kalem bilgisine ait düzenleme ve silme işlemi yapabilmektedir. Düzenleme butonuna bastıktan sonra açılan modal sayfası, Şekil 2.5'teki ekran ile aynı yapıya sahip olup alanların içi ilgili kalem bilgisine ait veriler ile dolu şekilde gösterilmektedir. Kullanıcı silme butonu ile ilgili kaleme ait verinin silme işlemeni gerçekleştirmektedir. Silme butonuna bastıktan sonra kullanıcının ilgili kalem bilgisini silmek isteyip istemediğini onaylayan bir modal sayfası açılmaktadır ve bu sayfada vazgeç ve sil şeklindeki butonlardan bir tanesine basarak işlemi gerçekleştirmesi gerekmektedir.



Şekil 2.6: Veri Listeleme Ekranı



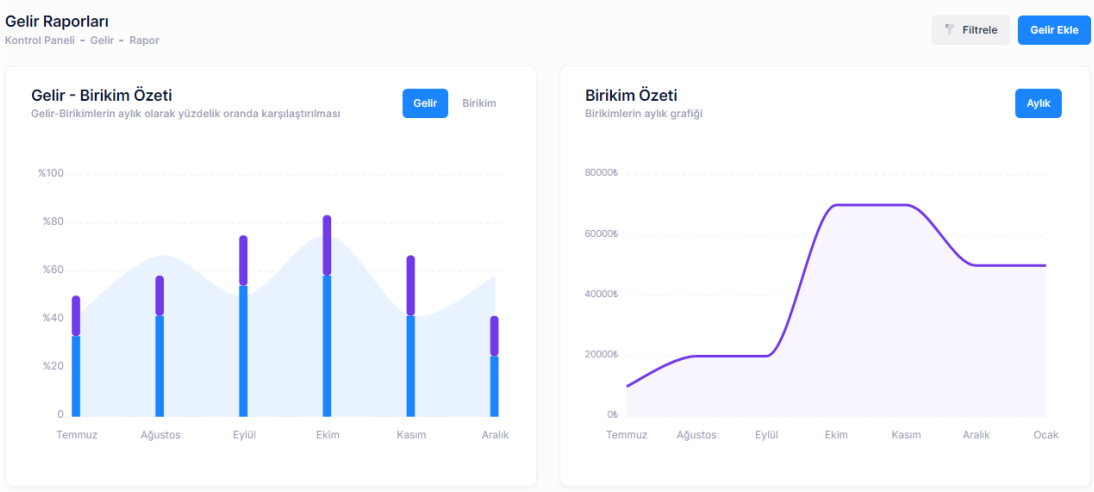
Şekil 2.7: Veri Filtreleme Ekranı



Şekil 2.8: Veriyi Dışa Aktarma Ekranı

2.2.5 Raporlama Ekranı

Uygulamada gelir ve gider verilerinin raporlanabilmesi için Şekil 2.9'daki ekran kullanıcıya gösterilmektedir. Bu ekran kullanıcının, belirli zaman aralıklarına göre gelir ve gider verilerini grafiksel olarak görüntüleyebilmesini sağlamaktadır. Kullanıcı bu ekranda gelir, gider, birikim, borç gibi verilerin oransal karşılaştırmalarını, zamansal olarak değişimlerini, günlük, haftalık, aylık, yıllık zaman dilimleri ile filtreleme işlemi yaparak oluşturabilmektedir. Ayrıca birikim özeti, aylık gider grafiği gibi zamansal olarak aylık veri içeren bu grafiklerin aynı görünümde kullanıcıya e-posta ile de gönderimi sağlanabilmektedir. E-posta içeriğinde bu grafikler pdf formatında yer almaktadır. E-posta gönderim işlemi Laravel'deki Job modülü ile oluşturulup, görev zamanlayıcıya (cronjob) aylık olarak çalışması için eklenmiştir. Bundan dolayı aylık olarak otomatik e-posta gönderimi gerçekleştirilmektedir.



Şekil 2.9: Raporlama Ekranı

Bölüm 3

Sonuç

Veri tabanı tasarımı olarak kullandığımız ilişkisel veri tabanı modeli ve normalizasyon çalışmaları, verilerin yönetilmesinde yani CRUD işlemlerinde büyük kolaylık sağlamıştır. Laravel Eloquent modeli ile ilişkisel veri tabanı modeli, tablolar arasındaki ilişkilerin ve tablolardaki alanların OOP olarak erişilmesini ve kullanılmasını sağladığından dolayı model, controller, view katmanında okunurluğu yüksek ve anlaşılır bir kod bloğu sunmuştur.

Verilerin MySQL veri tabanına veri oluşturulması, veri tabanından veri sorgulanması, güncellenmesi ve silinmesi işlemlerinde, standart SQL kayıt komutlarının kullanılmasına göre; Laravel Eloquent komutlarının (OOP) kullanılması daha hızlı bir CRUD işlemi gerçekleştirilmesini sağlamıştır.

Tablo 3.1: Standart SQL ve Laravel Eloquent CRUD işlemlerinin karşılaştırılması

SQL İŞLEMİ	STANDART SQL	ELOQUENT
INSERT INTO	45ms	20ms
SELECT FROM	32ms	13ms
UPDATE FROM	43ms	18ms
DELETE FROM	46ms	22ms

Ayrıca bildirim ve e-posta gibi otomatik yapılması gereken işlemlerde Laravel Job modülünün zamanlanan tarih ve saatte işlemi yerine getirebilmesi, kullanıcının gelir ve gider takibini verimli bir şekilde gerçekleştirebilmesine olanak sağlamıştır. Raporlama işlemlerinde ise oluşturulan grafiklerin, gösterim şekli ve okunabilirlikleri modern bir şekilde kullanıcıya sunulduğu için; kullanıcının borç ve birikim takiplerini de daha kolay ve bilinçli bir şekilde gerçekleştirebilmesine olanak sağlamıştır.

Kaynaklar

Kralev, V., Kraleva, R. (2017). Approaches to Designing Relational Databases. 7th International Conference. Modern Trends in Science. FMNS-2017, Blageovgrad, Bulgaria.

Dimitrieski, V., Celikovic, M., Aleksic, S., Ristic, S., Alargt, A., Lukovic, I. (2015). Concepts and evaluation of the extended entity-relationship approach to database design in a multi-paradigm information system modeling tool. Computer Languages, Systems & Structures, 44(C): 299-318. https://doi.org/10.1016/j.cl.2015.08.011

Silberschatz-Korth-Sudarshan., (2001). Database System Concepts. Fourth Edition, The McGraw-Hill Inc.

Powell, Gavin., (2006). Beginning Database Design. Wiley Publishing, Inc. Crosspoint Boulevard.

Sumathi, S. and Esakkirajan, S., (2007). Fundamentals of Relational Database Management Systems. Springer-Verlag Berlin Heidelberg.

Lightstone, Sam., Teorey, Toby., And Nadeau, Tom., (2007). Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More. Morgan Kaufmann.

Jauari Akhmad NH, Masfu Hisyam. Implementing Singleton method in Design of MVC-Based PHP Framework Sentinel Web. Informatics Department Electronic Engineering Polytechnic Institute of Surabaya Surabaya, Indonesia 2015.

Solanki N, Shah D and Shah, 2017 A Survey on Different Framework of PHP International Journal of Latest Technology in Engineering, Management & Applied Science (IJTEMAS) 6 (Issue VI)

Parkar VV, Shinde P, 2015 Utilization of Laravel Framework for Development of Web-Based Recruitment Tool J. IOSR Journal of Computer Engineering (IOSR-JCE)

Yu H R 2015 Design and Implementation of Web Based on Laravel Framework. Atl.

Ekler

Ek A

Uygulamada Gelir Tablosunun Migration ile Oluşturulma Kodu

```
<?php

/* Author: Gizem Çetin */

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;



return new class extends Migration

{

    /**

     * Run the migrations.

     */

    public function up(): void

    {

        Schema::create('incomes', function (Blueprint $table) {

            $table->id();

            $table->string('name');
```

```
            $table->string('desc');
            $table->decimal('amount');
            $table->boolean('is_static')->default(false);
            $table->foreign('account_id')->references('id')->on('accounts');
            $table->foreign('user_id')->references('id')->on('users');
            $table->foreign('category_id')->references('id')->on('categories');
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('incomes');
    }
};
```

Ek B

Uygulamada Gelir İşlemlerinin Model Kodu

```
<?php

/* Author: Gizem Çetin */

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;



class Income extends Model

{

    use HasFactory;

    protected $fillable = ['name', 'desc', 'amount', 'is_static', 'account_id', 'user_id', 'category_id'];

    public function account()

    {

        return $this->belongsTo(Account::class);

    }
```

```
    public function user()
    {
        return $this->belongsTo(User::class);
    }
    public function category()
    {
        return $this->belongsTo(Category::class);
    }
}
```

Ek C

Uygulamada Gelir İşlemlerinin Controller Kodu

```
<?php

/* Author: Gizem Çetin */

namespace App\Http\Controllers;



use App\Models\Income;

use Illuminate\Http\Request;

use Illuminate\Http\RedirectResponse;

use Illuminate\Http\View;



class IncomeController extends Controller

{

    /**

     * Display a listing of the resource.

     */

    public function index(): View

    {
```

```
    $incomes = Income::all()->sortByDesc("created_at");
    return view("incomes.index", compact("incomes"));
}
/**
 * Show the form for creating a new resource.
 */
public function create(): View
{
    return view("incomes.create");
}
/**
 * Store a newly created resource in storage.
 */
public function store(Request $request): RedirectResponse
{
    $request->validate([
        'name' => 'required',
        'desc' => 'required',
        'amount' => 'required',
        'date' => 'required',
    ]);
```

```
        Income::create($request->post());

        return redirect()->route('incomes.index')->with('success','Gelir kalemi başarılı şekilde eklendi.');
    }

    /**
     * Display the specified resource.
     */
    public function show(Income $income): View
    {
        return view("incomes.show", compact("income"));
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(Income $income): View
    {
        return view("incomes.edit", compact("income"));
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, Income $income): RedirectResponse
```

```
    {
        $request->validate([
            'name' => 'required',
            'desc' => 'required',
            'amount' => 'required',
            'date' => 'required',
        ]);
        $income->fill($request->post())->save();
        return redirect()->route('incomes.index')->with('success','Gelir kalemi başarılı şekilde güncellendi.');
    }
    /**
     * Remove the specified resource from storage.
     */
    public function destroy(Income $income): RedirectResponse
    {
        $income->delete();
          return redirect()->route('incomes.index')->with('success','Gelir kalemi başarılı şekilde silindi.');
    }
}
```