İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
FEN BİLİMLERİ ENSTİTÜSÜ

Web ve Mobil Uyumlu Şifre – Envanter Yönetim Yazılımı

Yazılım Mühendisliği Ana Bilim Dalı

Dönem Projesi

Avni Burak DALDAL

Öğrenci No Y220240009

Proje Danışmanı: Prof. Dr. Ayşegül ALAYBEYOĞLU SOY

Ocak 2024

Web ve Mobil Uyumlu Şifre – Envanter Yönetim Yazılımı

Öz

Şifre sızıntısı bilgi güvenliği için dikkat edilmesi gereken en önemli konulardan biridir. Dolayısı ile şifre bilgilerinin güvenli şekilde saklanması ve yönetilmesi gerekmektedir. Şifre sızıntısını ortadan kaldırmak için, şifrelerin güvenli şekilde saklanabileceği, yönetilebileceği ve takip edilebileceği mobil ve web uyumlu bir yazılım geliştirilecektir. Geliştirilecek olan bu yazılımda, PHP dilinde MVC mimarisi ile OOP standartlarında, laravel framework ve eloquent model yapısı kullanılacaktır. MySQL üzerinde ise veri tabanı oluşturulacak olup bu verilerin güvenliği bcrypt, argon2 ve sha-1 encrypt metotları ile sağlanacaktır. Bununla birlikte 2FA ile giriş, arama, loglama, çıkış yapmadan ekran kilitleyebilme, iki aşamalı şifre görüntüleme, yetkilendirme, güvenli şifre oluşturabilme ve kopyalayabilme fonksiyonları da yer alacaktır. Şifre ve şifreye bağlı envanter oluşturmak için CRUD işlemleri gerçekleştirecektir. Bu sayede şifre ve envanterlerin yönetimi, yetkilendirme fonksiyonu ile güvenli şekilde sağlanabilecektir. Sonuç olarak oluşan bu verilerin güvenli şekilde saklanabilmesi ve yönetilebilmesi sayesinde şifre sızıntısı gibi durumlarının önüne geçilmesi hedeflenmektedir.

Anahtar Sözcükler: PHP, MySQL, Laravel, OOP, MVC, veri tabanı, encrypt, 2FA, web, mobil, CRUD

Web and Mobile Compatible Password – Inventory Management Software

Abstract

Password leakage is one of the most critical aspects to consider for information security. Therefore, password information needs to be securely stored and managed. To eliminate password leakage, a mobile and web-compatible software will be developed to securely store, manage, and track passwords. In the development of this software, PHP language will be utilized with MVC architecture following OOP standards, employing the Laravel framework, and incorporating the Eloquent model structure. Additionally, a database will be created using MySQL, ensuring the security of the data through bcrypt, argon2, and sha-1 encryption methods. The software will feature functions such as 2FA for login, search capabilities, logging, screen locking without logging out, two-step password viewing, authorization, secure password creation, and copying. CRUD operations will be implemented for creating password and inventory-related records. This will enable the secure management of passwords and inventories through authorization functions. Ultimately, the goal is to prevent situations like password leakage by securely storing and managing the generated data.

Keywords: PHP, MySQL, Laravel, OOP, MVC, database, encrypt, 2FA, web, mobile, CRUD

Teşekkür

Proje çalışmasına katkılarından dolayı müstakbel eşim Gizem ÇETİN'e teşekkür ederim.

İçindekiler

Şekiller Listesi

Tablolar Listesi

Kısaltmalar Listesi

PHP	Hypertext Preprocessor
OOP	Object-Oriented Programming
MVC	Model-View-Controller
SHA-1	Secure Hash Algorithm 1
CRUD	Create Read Update Delete
2FA	Two Factor Authentication
CSS	Cascading Style Sheets
DB	Database
RDBMS	Relational Database Management System
ms	Milisaniye

Bölüm 1

Giriş

Günümüzde, kişisel bilgilerin güvenliği her zamankinden daha önemli hale gelmektedir. İlerleyen teknolojiyle birlikte birçok uygulama kullanılmakta olup, her bir uygulama için güçlü ve birbirinden bağımsız şifreler belirlenmesi gerekmektedir. Ancak, birçok kişi ve kurum, zayıf veya aynı şifreyi birden fazla hesap için kullanmaktadır. Bu durumda, aynı şifreye sahip olan bir hesabın, bilgisayarın veya sunucunun ele geçirilmesi diğer uygulamaların da tehlikeye girmesine neden olabilmekte olup, bu durum, kişisel verilerin sızdırılmasına ve büyük problemlerin ortaya çıkmasına yol açabilmektedir. Ayrıca, kişi ve kurumlar bu şifreleri güvenlik önlemi olmayan dijital dokümanlarda tuttuğu için şifre sızıntısı kolayca gerçekleşebilmektedir. Bu sorunu çözmek amacıyla bir şifre yönetim sistemi geliştirilmesi hedeflenmektedir.

Bu proje, kullanıcılara güvenli ve benzersiz şifreler oluşturmak için yardımcı olacak olup aynı zamanda kullanıcıların tüm şifrelerini tek bir güvenli yerde saklamalarına olanak tanıyacaktır. Bu sayede, bir hesabın ele geçirilmesi durumunda diğer hesapların tehlikeye girmesi önlenmiş olacaktır. Ayrıca, kurumlar için şifrelerin merkezi bir yerde güvenli bir şekilde yönetilmesi, olası şifre sızıntılarını ortadan kaldıracaktır. Temel amaç, kullanıcı güvenliğini artırmak ve şifre oluşturma ve yönetme süreçlerini kolaylaştırmaktır. Bu proje, kullanıcıların güvenlik düzeyini yükseltirken aynı zamanda kurumların şifre yönetim süreçlerini kolaylaştırmayı amaçlamaktadır.

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.
- Şifre ve envanterlerin merkezi bir yerde, güvenilir şekilde yönetilmesi ve saklanabilmesini sağlamak.
- Güvenli ve benzersiz şifre oluşturma sistemi ile zayıf veya aynı şifrelerin birden fazla hesap için kullanımını engellemek.
- Şifrelerin, kullanıcı yetkilendirme ve log sistemi ile izlenebilirliğini sağlamak.

1.2 Literatür Taraması

1.2.1 Veri Tabanı Tasarımı

Veri tabanı tasarımı, bir veri tabanının fiziksel ve mantıksal yapısını oluşturma sürecidir. Bu süreç, verilerin kullanıcı ihtiyaçlarına uygun, doğru, tutarlı, erişilebilir ve verimli olmasını sağlamak amacı ile verilerin bütünlüğünün koruması, sorgu performansı gibi hedeflere odaklanır.

Bu süreç, aşağıdaki adımları içerir:

- Veri gereksinimlerini belirleme: Veri tabanı tasarımının ilk adımı, veri tabanının gereksinimlerini belirlemektir. Bu, veri tabanının kullanılacağı uygulamayı ve kullanıcıların ihtiyaçlarını anlamayı gerektirir. Bu adımda, veri tabanında depolanacak verileri, bu verilerin nasıl kullanılacağını ve bu verilerin nasıl erişileceğini belirlemek gerekir.
- Veri modelini oluşturma: Veri modeli, veri tabanında depolanan verinin yapısını tanımlar. Veri modelleri, nesne yönelimli, ilişkisel veya hiyerarşik gibi çeşitli türlerde olabilir. Veri modeli, veri tabanının mantıksal yapısını tanımlar.

- Veri tabanı şemasını tasarlama: Veri tabanı şeması, veri modelinin fiziksel temsilini tanımlar. Veri tabanı şeması, tablolar, sütunlar, anahtarlar ve ilişkiler gibi unsurları içerir. Veri tabanı şeması, veri tabanının fiziksel yapısını tanımlar.
- Veri tabanını oluşturma: Veri tabanı şeması oluşturulduktan sonra, veri tabanı oluşturulmalıdır. Bu, veri tabanı yönetim sisteminin kullanılmasını gerektirir.

Veri tabanı tasarımı, bir veri tabanının başarısı için kritik öneme sahiptir. İyi bir veri tabanı tasarımı, verilerin doğruluğunu ve tutarlılığını sağlar, verilere hızlı ve verimli bir şekilde erişimi sağlar ve verilerin esnekliğini ve genişletilebilirliğini sağlar. Bu sayede veri tabanının performansını artırır ve kullanıcıların verilere güvenli ve etkili bir şekilde erişmelerini sağlar (Powell, 2006).

1.2.1.1 Veri Tabanı Gereksinimlerini Belirleme

Veri tabanının gereksinimlerini belirleme, bir organizasyonun veya proje için bir veri tabanı sistemi kurmadan önce, kullanıcı ihtiyaçlarını ve iş süreçlerini anlama ve bu ihtiyaçları karşılayacak bir veri tabanı tasarımını oluşturma sürecidir. Bu aşama, başarılı bir veri tabanı oluşturmak için temel bir adımdır (Silberschatz vd., 2001).

Bu adımda, veri tabanının aşağıdaki gereksinimleri belirlenir:

- Veri türleri: Veri tabanında depolanacak verilerin türleri belirlenir. Bu, verilerin sayısal, metinsel, tarihsel veya başka bir türde olup olmadığına karar verilmesini içerir.
- Veri miktarları: Veri tabanında depolanacak verilerin miktarları belirlenir. Bu, verilerin ne kadar olacağı ve ne kadar sık güncelleneceği gibi faktörleri içerir.
- Veri ilişkileri: Veri tabanında depolanan verilerin nasıl ilişkili olacağı belirlenir. Bu, bir veri öğesinin başka bir veri öğesine nasıl bağlı olduğuna karar verilmesini içerir.
- Kullanıcı gereksinimleri: Veri tabanının nasıl kullanılacağı ve kullanıcıların ondan ne beklediği belirlenir. Bu, kullanıcıların veri erişimi ve işleme gereksinimlerini içerir.

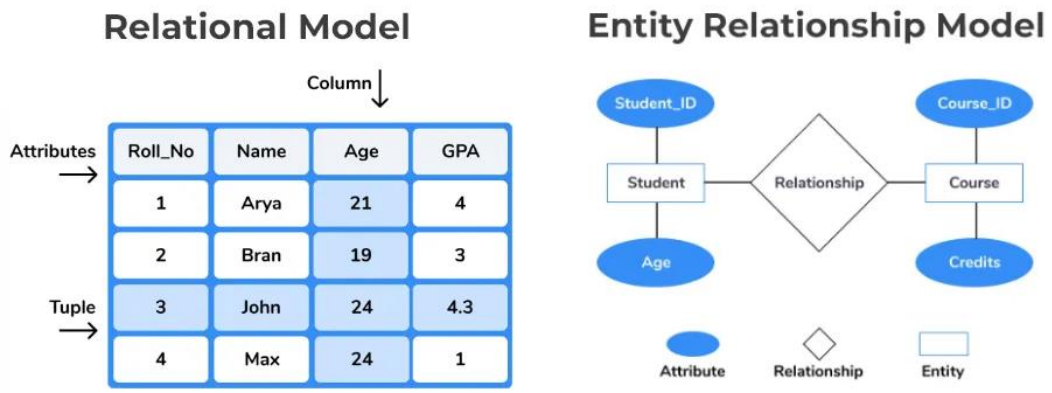
Veri tabanı gereksinimlerinin belirlenmesi, projenin başlangıcında sağlıklı bir temel oluşturarak, veri tabanının kullanıcı ihtiyaçlarına uygun, güvenli, performanslı ve ölçeklenebilir bir şekilde tasarlanmasını sağlar.

1.2.1.2 Veri Modelleme

Veri modelleme, bir veri tabanında depolanacak verilerin yapısını ve ilişkisini tanımlayan bir süreçtir. Bu süreç, veri tabanındaki verilere nasıl erişileceğinin ve nasıl kullanacağının anlaşılmasında yardımcı olur. Genellikle bir diyagram veya şema şeklinde olabilir ve veri tabanının modelini anlamak ve uygulamak için tasarlanır.

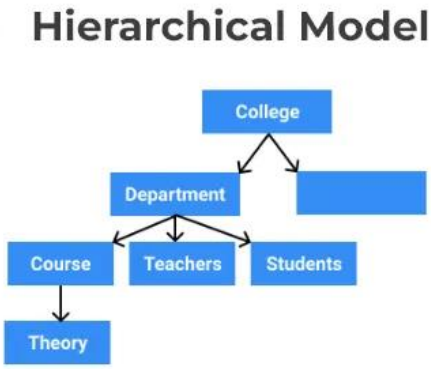
4 adet veri modeli tasarımı bulunmaktadır. Bu model tasarımları ise şunlardır:

- İlişkisel Veri Modeli: İlişkisel veri modeli, bir veri tabanında depolanan verilerin yapısını ve ilişkilerini tanımlayan ayrıca, veri tabanı tasarımında en çok kullanılan veri modelidir. İlişkisel veri modeli, bir tablo kümesi olarak tanımlanır. Her tablo, benzer verileri içeren sütunlar ve satırlardan oluşur. Tablolar, aralarında ilişkileri tanımlayan birincil ve yabancı anahtarlarla birbirine bağlanabilir. Örneğin, bir envanter veri tabanında "Envanterler" ve "Departmanlar" adlı iki tablo arasında bir ilişki olabilir ve bunlar birbirine bağlı olabilir (Kralev ve Kraleva, 2017).



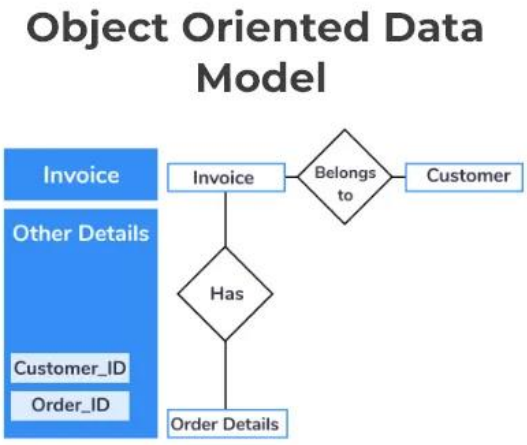
Şekil 1.1: Varlık-ilişki veri modeli

- Hiyerarşik Veri Modeli: Hiyerarşik veri modeli, ağaç benzeri bir yapıya sahiptir. Bu yapıda, her veri öğesi, bir üst veri öğesine bağlıdır. Dolayısıyla her veri öğesinin bir üst ilişkisi ve alt ilişkisi olabilir. Örneğin, bir organizasyon yapısını temsil eden bir hiyerarşik modelde, her birim bir bağlılığı temsil edebilir ve bu birimlere bağlı alt birimler olabilir. Hiyerarşik veri modeli, verilerin hiyerarşik bir yapıda depolanması, verilerin ilişkisinin kolayca anlaşılmasını sağlar. Ayrıca, verilere hızlı ve verimli bir şekilde erişimi sağlar. Ancak verilerin genişletilmesine verimli bir olanak sağlamaz. Bundan dolayı daha karmaşık ilişkiler gerektiren uygulamalarda, ilişkisel veri modeli tercih edilmektedir.



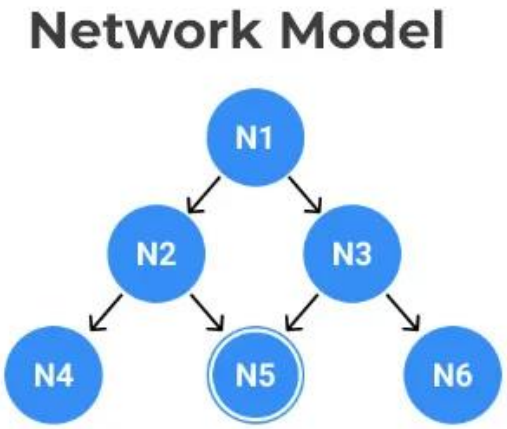
Şekil 1.2: Hiyerarşik veri modeli

- Nesne Tabanlı Veri Modeli: Nesne yönelimli programlama (OOP) kavramlarını kullanır. Her bir veri öğesi bir nesneyi temsil eder ve bu nesneler arasındaki ilişkiler OOP metotları ile gerçekleştirilir. Nesne tabanlı veri modeli, verilerin karmaşık olduğu bir yapıda, verilere daha kolay bir erişim sağlar. Ayrıca verilerin genişletilebilmesini daha verimlidir. Ancak karmaşık bir yapıda performans açısından ilişkisel veri modelinden daha az verimli olabilir. Örnek olarak ürün yönetimi, müşteri yönetimi, öğrenci yönetimi gibi veri tabanlarında kullanılabilir (Dimitrieski vd., 2015).



Şekil 1.3: Nesne tabanlı veri modeli

- Ağ Veri Modeli: Her bir veri öğesi bir köşe olarak kabul edilir. Köşeler kenarlar ile birbirine bağlanır. Verilerin köşeler ve kenarlar aracılığıyla ilişkilendirildiği bir veri modelidir. Örneğin, bir saç metal üretim sürecini temsil eden bir ağ modelde, her bir üretim aşaması bir köşeyi temsil edebilir ve bu köşeler birbirleriyle bağlantılı olabilir. Ağ veri modelinde verileri ilişkilendirmek için birden fazla kenar kullanılır bu da performans açısından ilişkisel veri modeline göre daha az verimlidir.

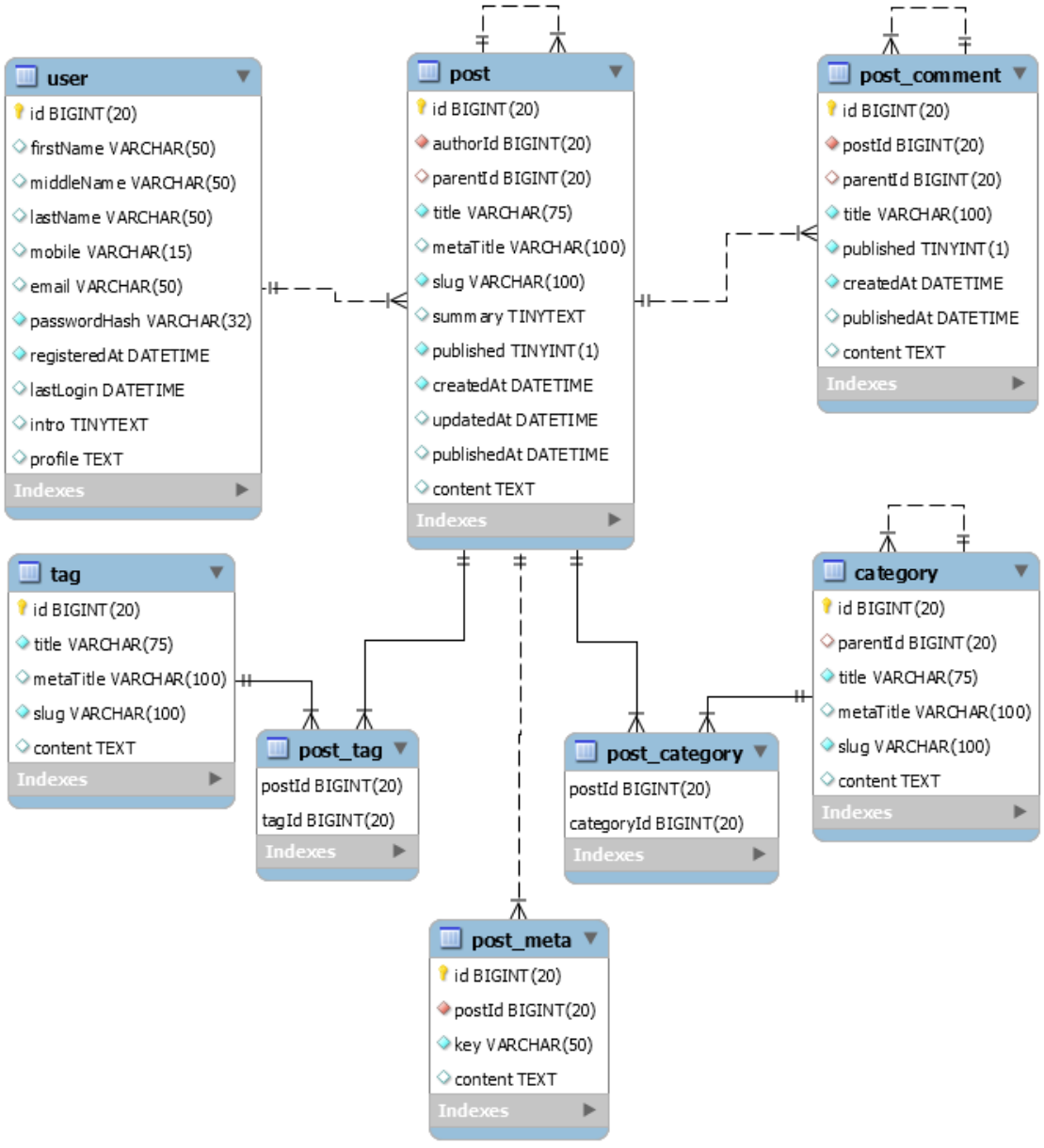


Şekil 1.4: Ağ veri modeli

Her bir veri modeli türü, belirli bir problem alanına veya uygulama gereksinimine daha uygun olabilir. Ancak ilişkisel veri modeli genellikle veri tabanı tasarımında performans ve verim açısından tercih edilen bir veri modelidir.

1.2.1.3 Veri Tabanı Şeması Tasarlama

Veri tabanı şeması tasarlama, bir veri tabanını oluşturan tabloları, sütunları ve ilişkileri tanımlayan bir süreçtir. Veri tabanı şeması, veri tabanının yapısını ve verilerin nasıl depolandığını tanımlar. Veri tabanı şeması tasarımının temel amacı, verileri düzenli bir şekilde depolayarak veri bütünlüğünü korumak ve performanslı bir şekilde sorgulama yapılmasını sağlamaktır (Sumathi ve Esakkirajan, 2007).



Şekil 1.5: Veri tabanı şema tasarımı örneği

Veri tabanı şema tasarımının temel kavramları ise şunlardır:

- Tablo Tanımları: İlk olarak, veri tabanında hangi bilgilerin saklanacağı belirlenir. Her bir tablo, bir kavramı veya varlığı temsil eder. Örneğin, bir okul veri tabanında "Öğrenciler" tablosu, öğrencileri temsil eder.
- Sütun Tanımları: Sütunlar, tablolarda depolanan verilerin özelliklerini tanımlar. Her sütun, bir veri türü ve bir özellik adı ile tanımlanır. Örneğin, bir "Öğrenciler" tablosunda "ogrenciAdi" sütunu, öğrencilerin adını, "ogrenciID" sütunu ise öğrencilerin benzersiz kimliklerini temsil eder.
- Birincil Anahtar (Primary Key) Tanımları: Her tablonun birincil anahtarı, tablodaki her bir kaydı benzersiz bir şekilde tanımlayan bir veya birden fazla sütundan oluşur. Birincil anahtar, tablonun temelini oluşturur ve tablodaki verileri eşsiz bir şekilde tanımlamak için kullanılır. Birincil anahtar genellikle ID alanı olmaktadır.
- Yabancı Anahtar (Foreign Key) Tanımları: Yabancı anahtarlar, iki tablo arasındaki ilişkiyi tanımlar. Bir tablonun birincil anahtarı, diğer tabloda bir yabancı anahtar olarak referans alınabilir. Bu sayede, iki tablodaki kayıtlar ilişkilendirilebilir.
- Normalizasyon: Normalizasyon, veri tabanında tutarlı ve verimli veri depolama için kullanılan bir tekniktir. Normalizasyon, tabloları, verilerin tekrarlanmasını ve tutarsızlıklarını önlemek için basit, özlü parçalara böler. Normalizasyon, 1NF, 2NF, 3NF gibi seviyelere ayrılır.

  Temel normalizasyon seviyeleri:

  - 1NF, veri tablolarında tekrarlanan verilerin önlenmesi için kullanılan bir ilk adımdır. 1NF'ye uygun bir tabloda, her hücrede yalnızca bir değer bulunmalıdır. Yani, bir hücre içinde birden fazla veri veya alt bileşen bulunmamalıdır. Aynı zamanda, her sütun tek bir değer tipini içermelidir.
  - 2NF, 1NF'nin bir uzantısıdır ve tablolardaki kısmi bağımlılıklardan kurtulmayı amaçlar. Kısmi bağımlılık, bir sütunun birincil anahtarın yalnızca bir kısmına bağlı olduğu durumdur. 2NF'ye uygun bir tabloda, tüm sütunlar birincil anahtara tamamen bağımlıdır. Bu, bir sütunun birincil anahtarın tüm bileşenlerine bağlı olduğu anlamına gelir.

- ❖ 3NF, 2NF'nin bir uzantısıdır ve tablolardaki geçişli bağımlılıklardan kurtulmayı amaçlar. Geçişli bağımlılık, bir sütunun doğrudan değil, başka bir sütundan dolaylı olarak birincil anahtara bağlı olduğu durumdur. 3NF'ye uygun bir tabloda, hiçbir sütun geçişli bağımlılıklara sahip değildir. Bu, bir sütunun yalnızca birincil anahtara doğrudan bağlı olduğu anlamına gelir.

- İndeksleme: İndeksleme işlemi, genellikle sık sık sorgulanan sütunların performanslı şekilde yanıt verebilmesi için kullanılır. Bir sütun indekslendiğinde, veri tabanı motoru, sorgu için gerekli verileri daha hızlı bulmak için indeksi kullanır. Bu, sorgunun daha hızlı sonuçlanmasını sağlar.

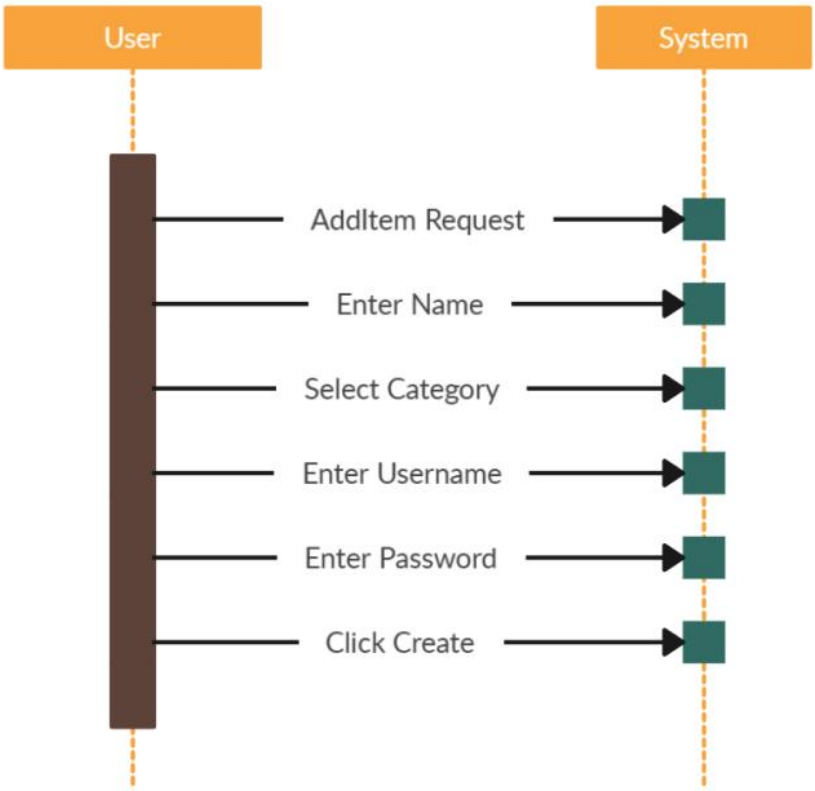
Veri tabanı şema tasarımı, veri tabanının verimli ve sürdürülebilir olmasını sağlayan önemli bir süreçtir. Bu süreç, genellikle kullanıcı ihtiyaçlarına, veri bütünlüğüne ve performansa odaklanarak gerçekleştirilir. İyi bir veri tabanı tasarımı, veri tabanının hangi tipte verileri saklayacağını, tablolar arasındaki ilişkileri ve tablo bağımlılıklarını anlaşılır bir şekilde ortaya koyar. Bu sayede, veri tabanının sürdürülebilirliği daha net bir şekilde belirlenir. Dolayısıyla veri tabanına eklenecek olan yeni bir tablo, veri tabanı şema tasarımına bağlı olarak, veri bütünlüğünü bozmadan veri tabanına dahil edilir (Selfa vd., 2006).

Özetle doğru bir veri tabanı şema tasarımı, kullanıcıların ihtiyaçlarını karşılamak, veri bütünlüğünü korumayı, performansı artırmayı ve ölçeklenebilirliği sağlamayı hedefler. Bu tasarım süreci, veri tabanının güncelliğini ve doğruluğunu sürdürmek için dikkatlice planlanmalı ve uygulanmalıdır. Veri modelleme ve veri tabanı 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 Filament Tallstack, Tailwind CSS ve Blade Template Engine üzerinde çalışmaktadır. Bu sayede kişi veya kurumların, şifre ve envanterlerini kategorilere bağlı olarak girişini sağladığı, şifrelerin görüntülenmesini ve değiştirilmesini yetkilendirilebildiği ve bu işlemleri log takibi ile kayıt altına alabildiği bir web uygulaması sunmaktadır.



Şekil 2.1: Şifre ekleme adımlarının use-case diyagramı

2.1 Uygulama Veri Tabanı Tasarımı

Veri tabanı tasarımı, veri yönetiminin temelini oluşturur. Veri tabanının veri modeli, verilerin performansını ve sürdürülebilirliğini doğrudan etkiler. Bundan dolayı veri tabanının hangi veri modeli ile tasarlanacağının seçimi, veri yönetiminde etken bir rol oynar.

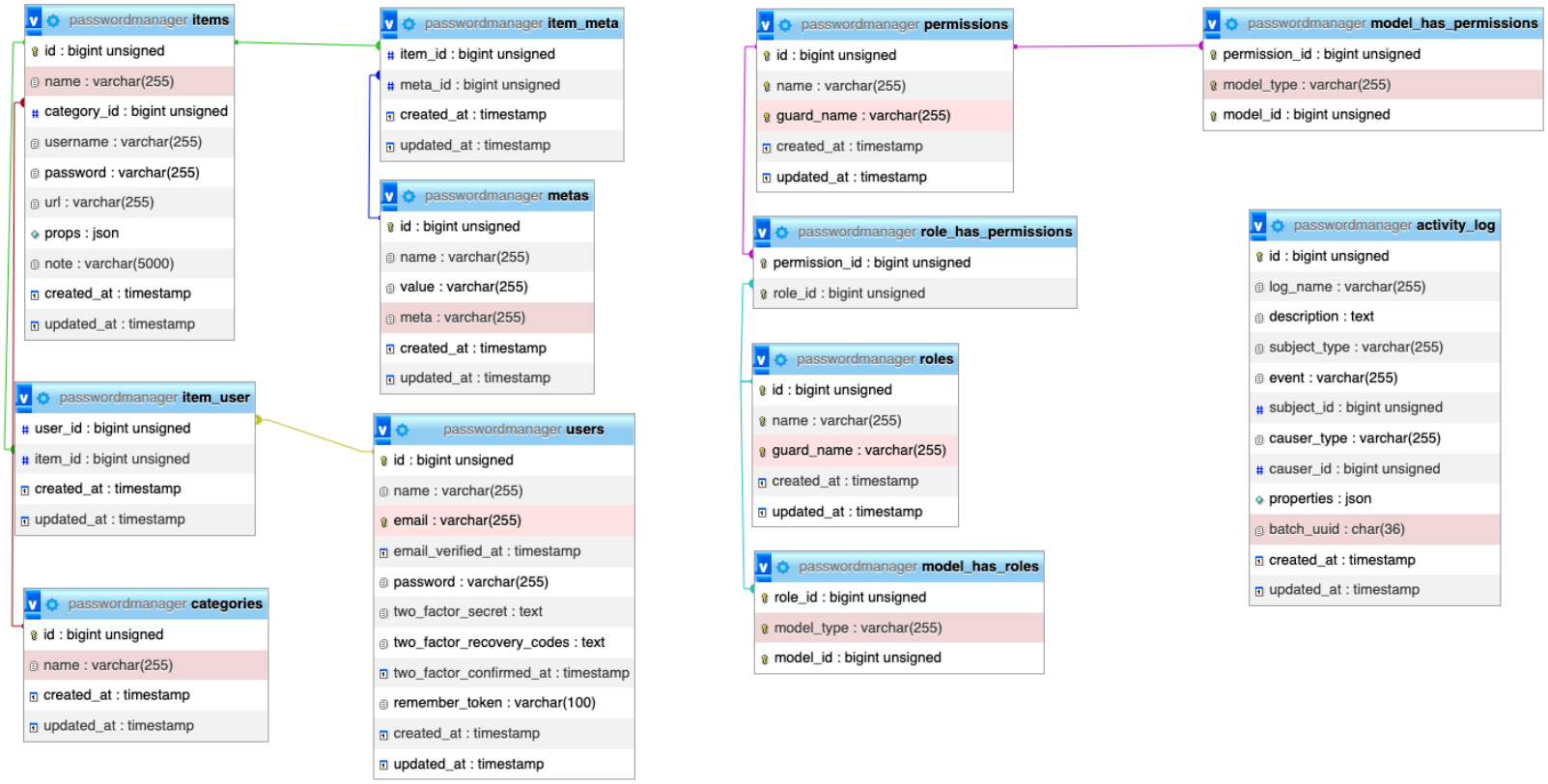
Bu dönem projesinde veri modeli olarak, varlık – ilişki veri modeli kullanılmıştır. Şifre, envanter, kategori, kullanıcı, yetkilendirme işlemlerine ait verilerin kayıt altına alınması yani CRUD işlemleri için tablolar arasındaki varlık ilişkileri normalizasyon kurallarına bağlı olarak, Laravel 10 Framework tarafında Eloquent Model kütüphanesi ile MySQL sisteminde oluşturulmuştur.

Varlık – ilişki modeline bağlı olarak toplam 12 adet tablo oluşturulmuştur. Bu tablolar, activity_log, categories, items, item_meta, item_user, metas, model_has_permissions, model_has_roles, permissions, role, role_has_permissions, users şeklindedir. Tablolar arasında bire çok (one to many) veya çoka çok (many to many) ilişki bulunmaktadır. Örnek olarak, items tablosu ile categories tablosu arasındaki bire çok ilişki şu şekildedir; bir şifrenin (items) birden fazla kategorisi olabilmektedir. Ayrıca, kullanıcı tablosu ile items tablosu arasındaki çoka çok ilişki ise şu şekildedir; birden fazla şifre birden fazla kullanıcıya atanabilmektedir. Tablolar arasındaki ilişkilerin bağlayıcılığı foreign key ile sağlanmıştır. Örnek olarak, items tablosunda bulunan category_id (FK), categories tablosunda bulunan id (PK) alanı ile bağlıdır.

Tablolarda yer alan alanların veri tipleri ise kaydedilecek olan verilerin türlerine göre belirlenmiştir. Örnek olarak, items tablosunda yer alan props alanı key – value şeklinde bir değer içereceği için json veri tipi atanmıştır. Yine bu tabloda yer alan password alanı şifrelenmiş bir metin değeri içereceği için varchar tipinde ve 255 karakter uzunluğunda atanmıştır. Ayrıca tablolar arasında yer alan foreign key (yabancı anahtar) ilişkilerine ait alanların veri tipi ise biginteger unsigned şeklindedir. Tüm tablolardaki ID alanı primary key (birincil anahtar) olarak belirlenmiştir. Kayıt ve güncelleme işlemlerinin zaman bilgisinin kayıt altına alınabilmesi için tüm tablolarda created_at ve updated_at alanları timestamps şeklinde oluşturulmuştur.

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 12 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 veri tabanı şema tasarımında ise tablo ve sütun tanımlamaları, ilişkisel anahtar tanımlamaları (Primary Key, Foreign Key) yer almaktadır. Tasarıma ait görsel Şekil 2.2'de verilmiştir.



Şekil 2.2: Veri tabanı şema tasarımı

2.2 Uygulama Geliştirme

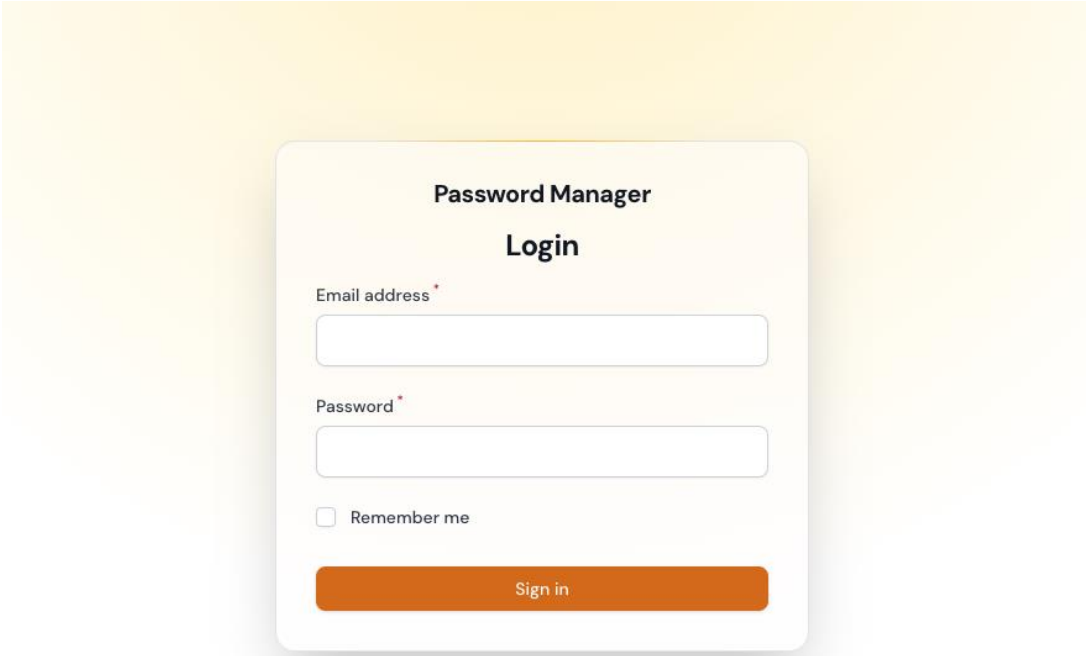
Günümüzde PHP web programlama dili çok popülerdir. PHP içerisinde birçok OOP ve MVC tabanlı framework bulunmaktadır (Dockins, 2017). Bu frameworkler içerisinde en gelişmiş ve en popüler olanı ise Laravel frameworktür. Laravel, login işlemleri, CRUD işlemleri, loglama işlemleri, event işlemleri gibi birçok yazılım ihtiyacını karşılamaktadır. Ayrıca blade template engine sistemi ile esnek bir view sistemi sunmaktadır (Stauffer, 2019).

View 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.

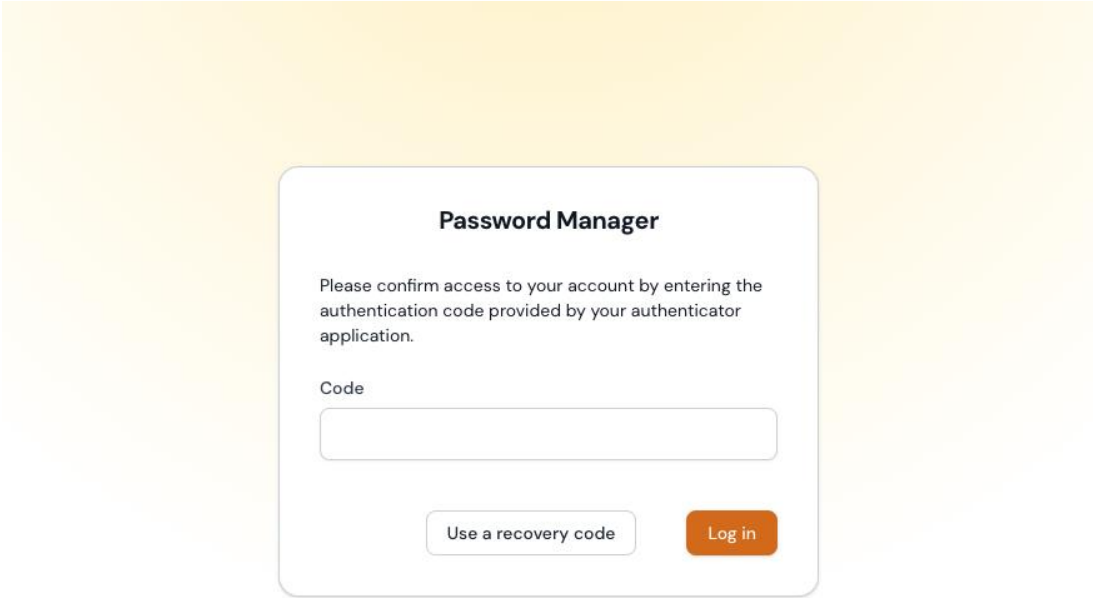
Bu dönem projesinde login ve ikili doğrulama (2FA) işlemlerini Laravel'de yer alan Filament paketi ile, CRUD işlemlerini Laravel'de yer alan controller, model ve resource ile, view sistemi ve mobil uyumlu görünümü, blade template engine ve Tailwind CSS ile, loglama işlemlerini ise yine Laravel'de yer alan Filament activity log paketi ile gerçekleştirdik.

2.2.1 Uygulama Giriş Ekranı

Uygulamaya girebilmek için Şekil 2.3'teki ekran kullanıcıya gösterilmektedir. Bu ekranda, kullanıcıdan sisteme kayıtlı e-posta adresi ve şifresini girmesi istenmektedir. Kullanıcı yanlış bir şifre ile giriş yapmaya çalışır ise hata mesajı ile karşılaşmaktadır. Remember me seçeneğini seçerek bir sonraki girişte şifresiz giriş yapabilmektedir. Ayrıca kullanıcı, ikili doğrulama giriş sistemini aktifleştirdi ise; giriş yaptıktan sonra Şekil 2.4'teki ekran gösterilmekte olup bu ekranda kullanıcıdan telefonundaki Google authenticator uygulamasındaki tek seferlik şifreyi girmesi istenmektedir. Kullanıcının telefonuna erişimi yok ise Use a recovery code butonu ile kurtarma işlemini gerçekleştirebilir.



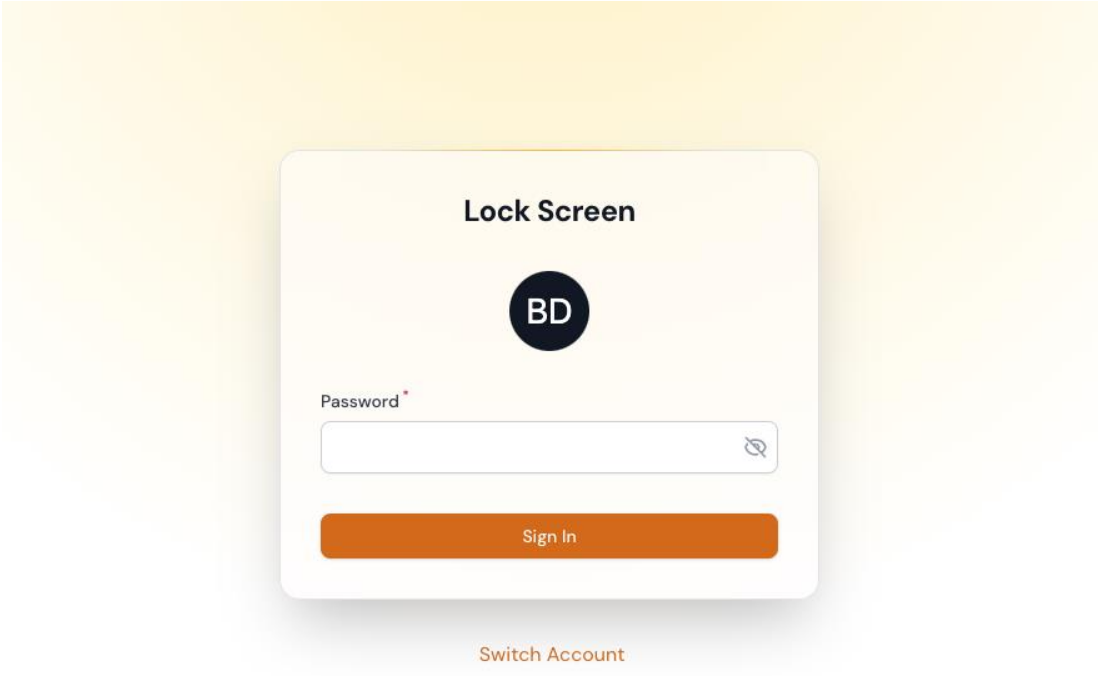
Şekil 2.3: Uygulama giriş ekranı



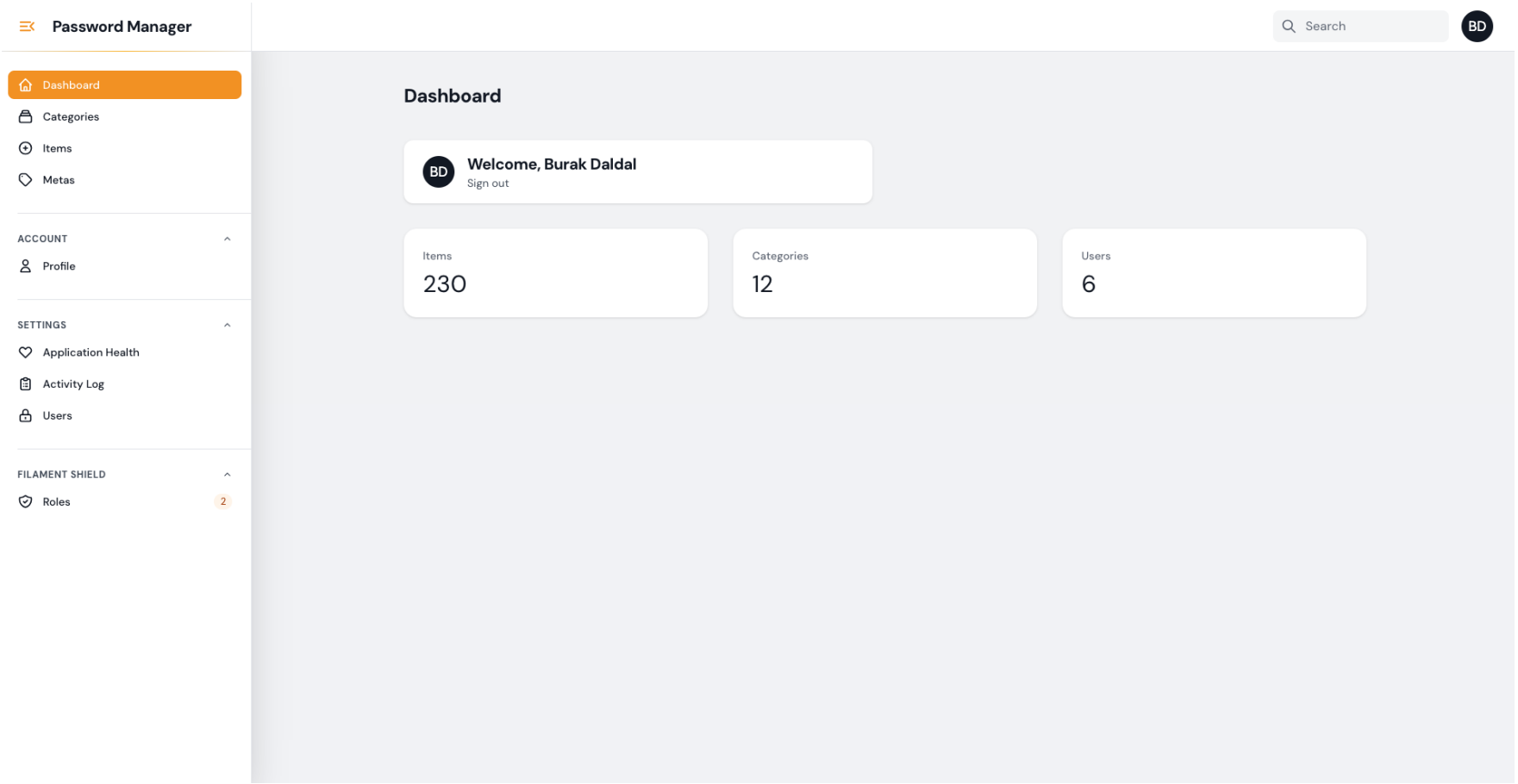
Şekil 2.4: İkili doğrulama ekranı

2.2.2 Uygulama Ana Ekranı

Uygulamada temel işlemleri gerçekleştirebilmek için Şekil 2.6'daki ekran kullanıcıya gösterilmektedir. Bu ekranda, uygulamada kaç adet şifrenin (items), kategorinin ve kullanıcının bulunduğunun sayısal bilgileri gösterilmektedir. Yan menüde yer alan butonlar aracılığı ile ise uygulama yer alan diğer tüm işlemlere erişebilmektedir. Örneğin, items butonundan şifrelerin listelenmesi veya şifre ekleme gibi işlemleri yapabileceği ekrana ulaşmaktadır. Profile butonundan ise, kullanıcı e-posta ve şifre gibi bilgilerini düzenleyebildiği ekrana ulaşmaktadır. Ayrıca üst menüde yer alan search alanından ise bulmak istediği şifreye dair arama yapabilmektedir. Search alanının yanında yer alan, kullanıcının isim ve soyisminin baş harflerinin bulunduğu butondan ise lock screen (ekran kitleme), 2FA ve sign out (çıkış yap) işlemlerini gerçekleştirebilmektedir. Lock screen ekranına ait görüntü ise şekil 2.5'te yer almaktadır.



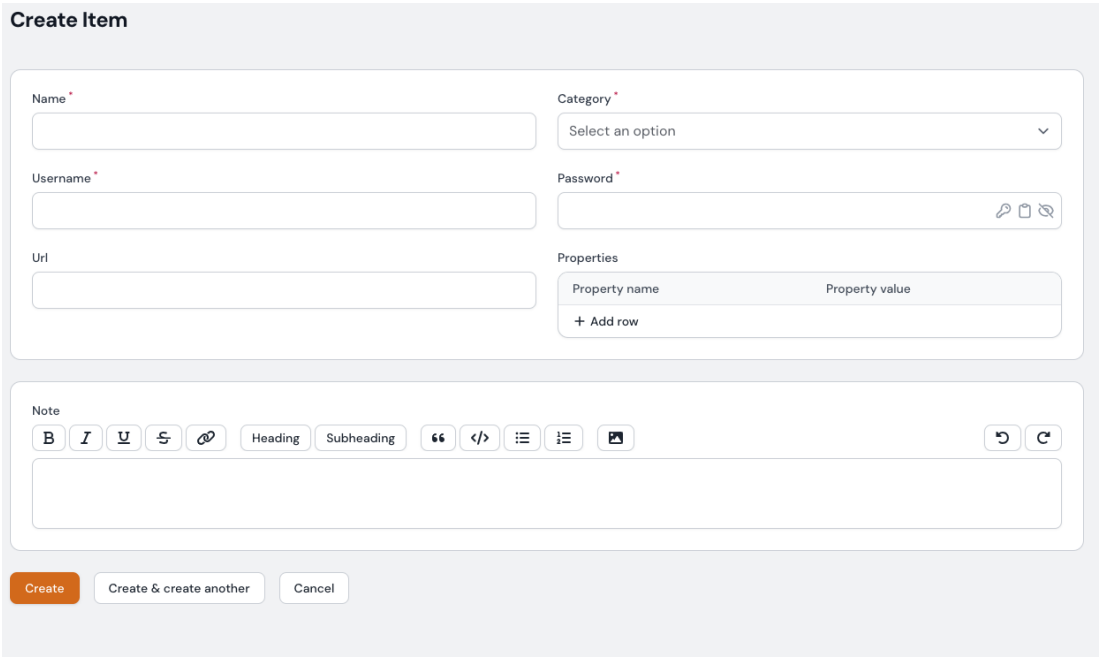
Şekil 2.5: Ekran kilitleme ekranı



Şekil 2.6: Uygulama Ana Ekranı

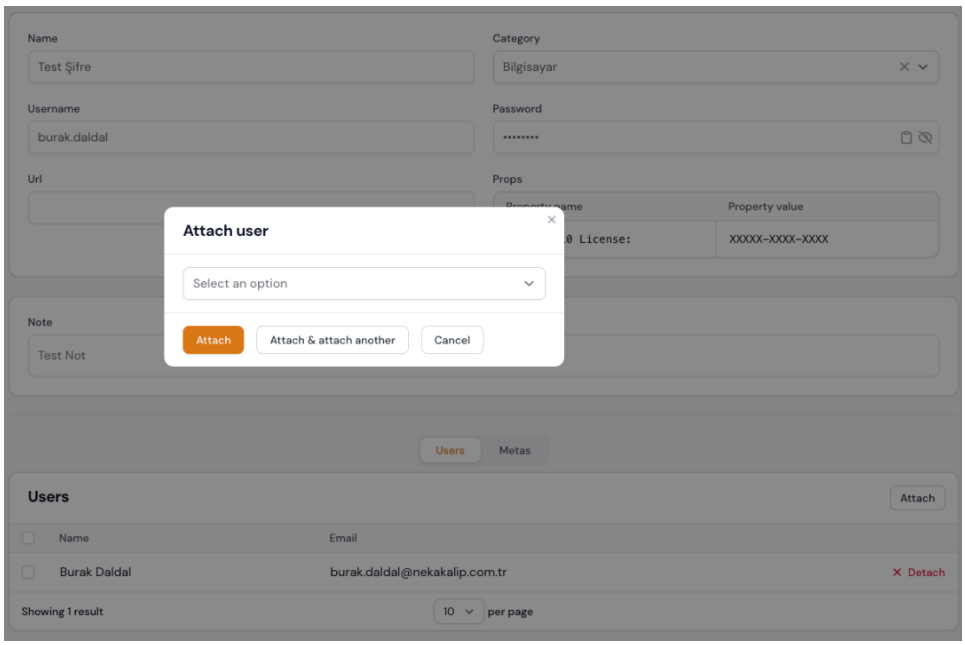
2.2.3 Veri Giriş Ekranı

Uygulamada şifre, kategorileri ve envanter verilerinin kaydedilmesi için 3 ayrı veri giriş sayfası bulunmaktadır. Bu ekranlardan şifre giriş ekranı, Şekil 2.7'de gösterilmektedir. Bu ekran kullanıcının, şifre verilerine ait alanları doldurarak veri girişi işleminin yapılmasını gerçekleştirmektedir. Şifre verilerinin kayıt altına alınabilmesi için zorunlu alanların doldurulması gerekmektedir. Kullanıcı bu alanlardan password alanında, anahtar şekilli buton ile rasgele şifre oluşturma, pano şekilli buton ile oluşturulan bu şifreyi kopyalama işlemlerini gerçekleştirebilmektedir, ayrıca göz şekilli buton ile şifre görünürlüğünü de değiştirebilmektedir. Properties alanında, add row butonuna basarak şifreye key – value şeklinde birden fazla özellik eklenebilmektedir. Note alanından ise metin editörü içerisinde, WYSIWYG Editor formatında notlar yazılabilmekte olup; dokümantasyon, resim, ek dosya gibi bilgiler de eklenebilmektedir. Kullanıcı bu alanları doldurduktan sonra create butonuna basarak veri girişi işlemini gerçekleştirmektedir. Eğer zorunlu alanları doldurmadıysa bununla ilgili uyarı mesajı ile karşılaşmaktadır.

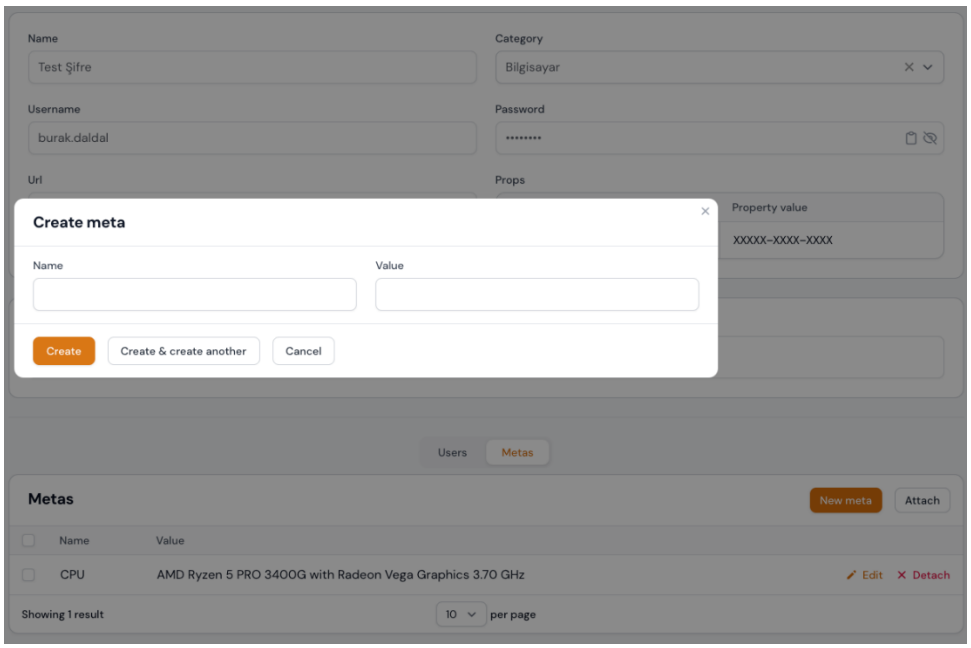


Şekil 2.7: Şifre giriş ekranı

Şifre ekleme işlemi gerçekleştirildikten sonra Şekil 2.8'de yer alan users ve metas alanlarından şifreyi görüntüleyebilecek diğer kullanıcıların ataması, attach butonu ile yapılmaktadır. Yanlış bir kullanıcı atandı ise detach butonu ile bu işlem geri alınabilir. Ayrıca metas alanından envanter attach ve detach işlemi yapılabilmektedir. New meta butonu ile yeni envanter oluşturulup, oluşturulan bu envanterin de otomatik olarak attach işlemi gerçekleştirilmektedir. New meta (meta ekleme) işlemine ait ekran ise Şekil 2.9'da gösterilmektedir.



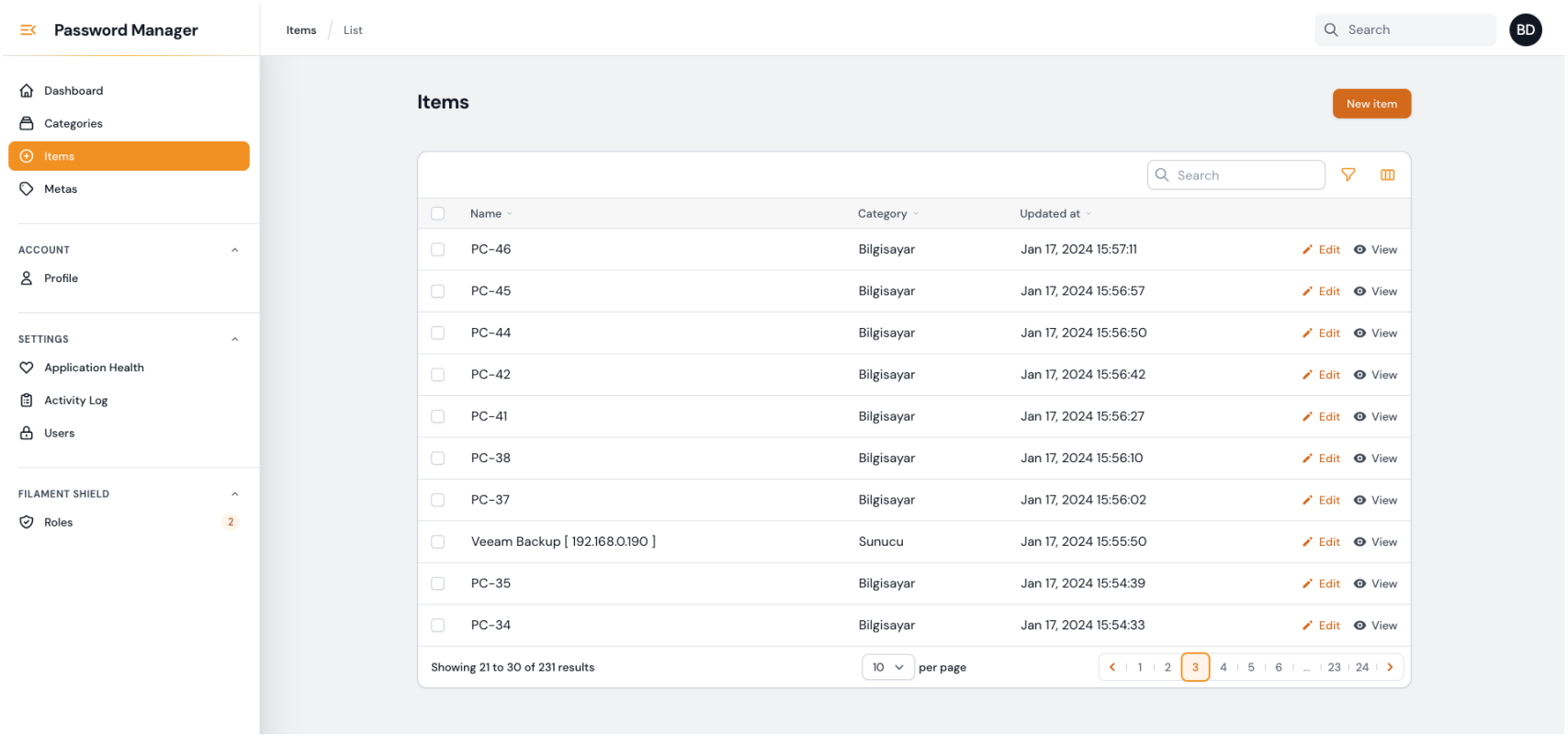
Şekil 2.8: Kullanıcı atama ekranı



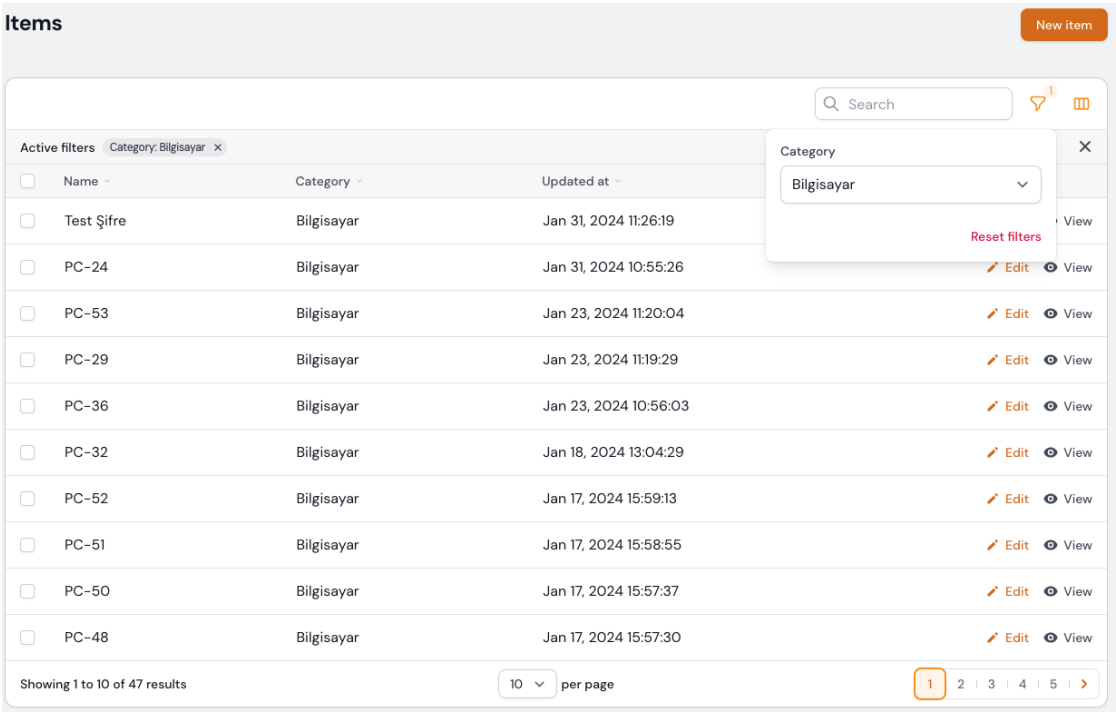
Şekil 2.9: Meta ekleme ekranı

2.2.4 Veri Listeleme Ekranı

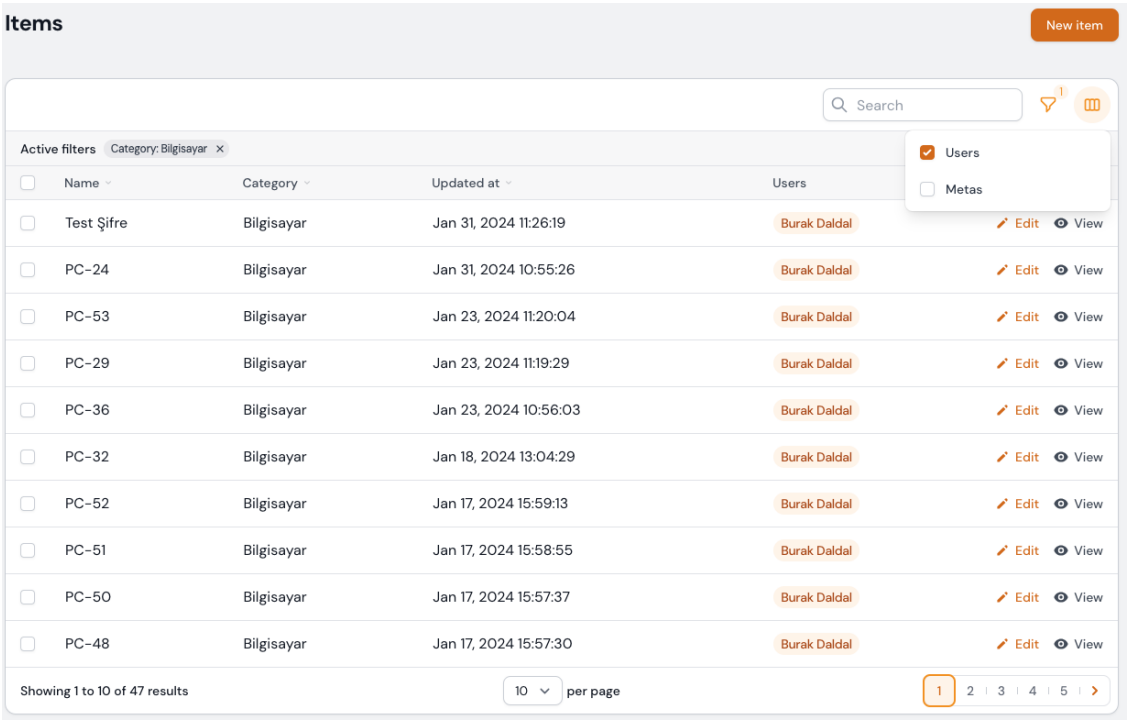
Uygulamada şifre, kategorileri ve envanter verilerinin listelenmesi için 3 ayrı veri listeleme sayfası bulunmaktadır. Bu ekranlardan şifre listeleme ekranı, Şekil 2.10'da gösterilmektedir. Bu ekran kullanıcıya ait ve atanmış olan şifreleri göstermektedir. Kullanıcı bu ekranda şifre ismi, kategori ve kullanıcı adına göre arama yapabilmektedir. Kullanıcı arama kutusuna bu alanlara ait bir kelime yazdığında listede sadece o bilgiye ait aramalar gösterilmektedir. Kullanıcı şeklindeki buton ile kategori bazlı filtreleme yapabilmektedir. Yine aynı butona bastıktan sonra açılan pencereden kategori seçip, seçmiş olduğu kategoriye göre filtreleme yapabilmektedir. Bu filtrelemeyi reset filters butonu ile sıfırlayabilmektedir. Filtreleme işleme ait ekran Şekil 2.11'de gösterilmektedir. Arama işlemindeki gibi aynı şekilde kategori alanlarına ait seçim yaptığında listede sadece o bilgiye ait filtrelenmiş veriler gösterilmektedir. Kullanıcı tablo şekilli butondan ise users veya metas alanlarını listeleme işlemine dahil edebilmektedir. Bu işleme ait ekran Şekil 2.12'de gösterilmektedir. Ayrıca kullanıcı düzenleme butonundan listedeki yer alan şifre bilgisine ait düzenleme işlemi yapabilmektedir. Düzenleme butonuna bastıktan sonra açılan sayfa, Şekil 2.7'deki ekran ile aynı yapıya sahip olup alanların içi ilgili şifre bilgisine ait veriler ile dolu şekilde gösterilmektedir. Kullanıcı bu ekranda delete butonu ile ilgili şifreye ait verinin silme işlemeni gerçekleştirmektedir. Silme butonuna bastıktan sonra kullanıcının ilgili şifre bilgisini silmek isteyip istemediğini onaylayan bir modal sayfası açılmaktadır ve bu sayfada cancel ve confirm şeklindeki butonlardan bir tanesine basarak işlemi gerçekleştirmesi gerekmektedir.



Şekil 2.10: Veri listeleme ekranı



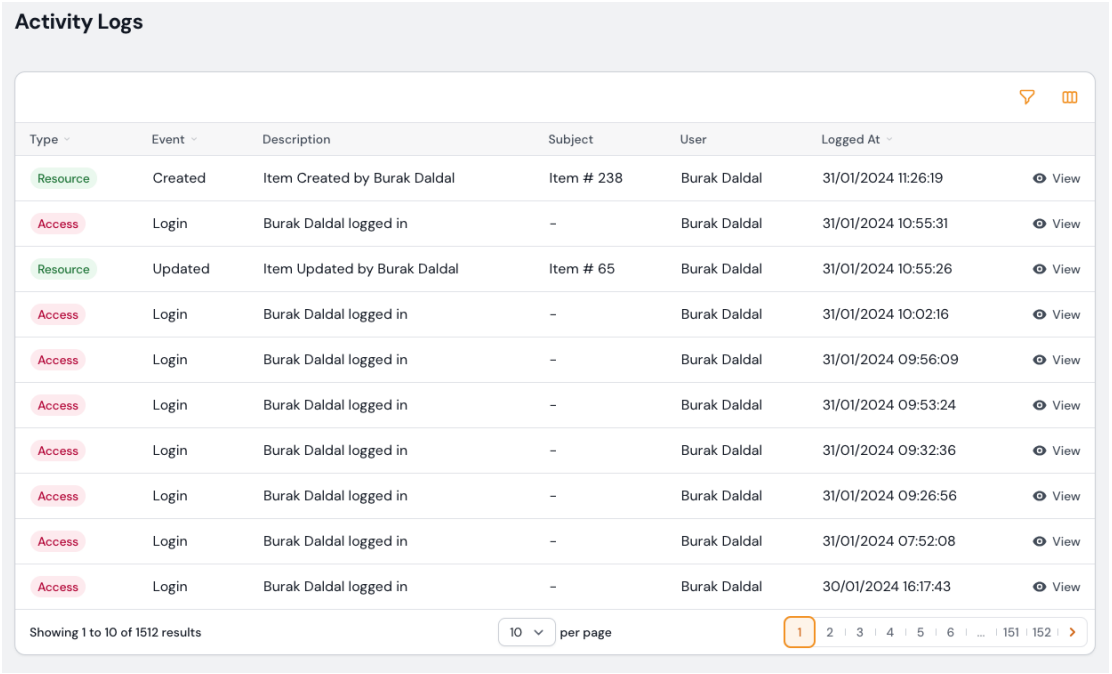
Şekil 2.11: Veri filtreleme ekranı



Şekil 2.12: Liste özelleştirme ekranı

2.2.5 Loglama Ekranı

Uygulamada tüm CRUD ve login işlemlerine ait logların takip edilebilmesi ve raporlanabilmesi için Şekil 2.13'teki ekran kullanıcıya gösterilmektedir. Bu ekran tüm işlemlere ait logların, liste şeklinde görüntülenebilmesini sağlamaktadır. Örnek olarak sisteme giriş yapmış olan bir kullanıcının log kaydı, access tipindeki login event işlemi ile kullanıcı adı ve giriş yaptığı tarih bilgisini içermektedir. View butonu ile bu kayda ait detay bilgisine de ulaşılabilmektedir. Detay bilgisinde ise hangi ip ve tarayıcıdan giriş yapmış olduğunun bilgileri yer almaktadır. Ayrıca CRUD işlemlerine ait log kaydı ise, resource tipinde olup, event alanında yapılan CRUD işleminin bilgisi verilmektedir. Bununla birlikte açıklama, konu, kullanıcı adı ve tarih bilgisi de yer almaktadır. Yine bu kaydın view butonu ile detay bilgisine ulaşılabilmekte olup, view sayfasında ise ilgili kayıt bir güncelleme işlemi ise güncellenmiş alanın; güncellenmeden önceki verisi ile güncellendikten sonraki veri bilgisini de içermektedir.



Şekil 2.13: Loglama ekranı

Bölüm 3

Sonuç

Veri tabanı şema tasarımı olarak kullandığımız ilişkisel veri modeli ve normalizasyon işlemleri, verilerin yönetilmesinde yani CRUD işlemlerinde etkili bir kullanım sağlamıştır. Laravel Eloquent modeli ile ilişkisel veri modeli, tablolar arasındaki ilişkilerin ve tablolarda yer alan alanların OOP olarak erişilmesi ve kullanılmasında performans sağlamıştır. Ayrıca model, controller, view katmanında okunurluğu yüksek ve anlaşılır bir kod bloğu sunmuştur.

Verilerin MySQL veri tabanına kaydedilmesi, veri tabanından bu kayıtların 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: Raw SQL ve Laravel Eloquent CRUD işlemlerinin karşılaştırılması

SQL-CRUD	RAW SQL	ELOQUENT
INSERT INTO	53ms	34ms
SELECT FROM	36ms	19ms
UPDATE FROM	56ms	25ms
DELETE FROM	70ms	36ms

Ayrıca kullanıcıların yetkilendirilmesi gibi veri güvenliğinin sağlanması gereken işlemlerde Laravel Filament modülünün Shield paketi; veri oluşturma, görüntüleme, güncelleme ve silme işlemlerinin erişebilirliğini ve güvenli bir şekilde yönetilebilmesini sağlamıştır. Bu sayede her kullanıcı sadece kendisine ait ve atanmış olan verileri görüntüleyebilmekte ve buna göre takibini yapabilmektedir.

Kaynaklar

Dockins, K. (2017). “Design Patterns in PHP and Laravel.” Apress. Retrieved from http://samples.leanpub.com/larasign-sample.pdf

Stauffer, M. (2019). “Laravel: Up & Running: A Framework for Building Modern PHP Apps.” United States of America: O'ReillyMedia.

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

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

Powell, G., (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.

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

Selfa, D. M., Carrillo, M., and B. M. D. R., "A database and web application based on mvc architecture," In 16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06), pp. 48-48, February 2006.

Ekler

Ek A

Uygulamada Items Tablosunun Migration ile Oluşturulma Kodu

```
<?php


use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;


return new class extends Migration

{

    /**

     * Run the migrations.

     *

     * @return void

     */

    public function up()

    {

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

```
            $table->id();
            $table->string('name');
            $table->foreignId('category_id')->unsigned()->nullable();
            $table->string('username');
            $table->string('password');
            $table->string('url')->nullable();
            $table->json('props')->nullable();
            $table->string('note',5000);
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('items');
    }
};
```

Ek B

Uygulamada Item İşlemlerinin Model Kodu

```
<?php


namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Spatie\Activitylog\Traits\LogsActivity;


class Item extends Model

{

    use HasFactory;

    protected $fillable = [

        'name',

        'category_id',

        'username',

        'password',

        'url',
```

```
    'props',
    'note'
];
protected $casts = [
    'props' => 'array'
];
public function Users()
{
    return $this->belongsToMany(User::class);
}
public function Category()
{
    return $this->belongsTo(Category::class);
}
public function Metas()
{
    return $this->belongsToMany(Meta::class);
}
protected static function booted()
{
    static::deleting(function ($item) {
```

```
        $item->Users()->detach();
        $item->Metas()->detach();
    });
}
```

Ek C

Uygulamada Item İşlemlerinin Resource Kodu

```
<?php


namespace App\Filament\Resources;


use App\Models\Item;

use Filament\Tables;

use Filament\Resources\Form;

use Filament\Resources\Table;

use Filament\Resources\Resource;

use Filament\Forms\Components\Card;

use Filament\Forms\Components\Select;

use Filament\Forms\Components\KeyValue;

use Filament\Tables\Columns\TagsColumn;

use Filament\Tables\Columns\TextColumn;

use Filament\Forms\Components\TextInput;

use Filament\Tables\Filters\SelectFilter;
```

```
use Illuminate\Database\Eloquent\Builder;
use Phpsa\FilamentPasswordReveal\Password;
use App\Filament\Resources\ItemResource\Pages;
use
App\Filament\Resources\ItemResource\RelationManagers\MetasRelationManager;
use
App\Filament\Resources\ItemResource\RelationManagers\UsersRelationManager;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Auth;
use Filament\Forms\Components\RichEditor;


class ItemResource extends Resource
{
    protected static ?string $model = Item::class;
    protected static ?string $navigationIcon = 'heroicon-o-plus-circle';
    public static function form(Form $form): Form
    {
        return $form
            ->schema([
                Card::make()
                    ->schema([
                        TextInput::make('name')->required(),
```

```
            Select::make('category_id')
                ->relationship('category', 'name')
                ->searchable()
                ->required()
                ->preload(),
            TextInput::make('username')
                ->required(),
            Password::make('password')
                ->copyable(true)
                ->generatable(true)
                ->required(),
            TextInput::make('url')
                ->url(),
            KeyValue::make('props')
                ->keyLabel('Property name')
                ->valueLabel('Property value')
        ])
        ->columns(2),
    Card::make()
        ->schema([
            RichEditor::make('note')
```

```
                ])
                ->columns(1)
        ]);
    }
    public static function table(Table $table): Table
    {
        return $table
            ->columns([
                TextColumn::make('name')->sortable()->searchable(),
                TextColumn::make('category.name')->sortable()->searchable(),
                TextColumn::make('updated_at')->dateTime()->sortable(),
                TagsColumn::make('users.name')->separator(',')->searchable()->toggleable()->toggledHiddenByDefault(),
                TextColumn::make('metas.meta')->searchable()->toggleable()->toggledHiddenByDefault(),
            ])
            ->defaultSort('updated_at', 'DESC')
            ->filters([
                SelectFilter::make('category')->relationship('category', 'name'),
            ])
            ->actions([
                Tables\Actions\EditAction::make(),
```

```
            Tables\Actions\ViewAction::make(),
        ])
        ->bulkActions([
            Tables\Actions\DeleteBulkAction::make(),
        ]);
}
public static function getRelations(): array
{
    return [
        UsersRelationManager::class,
        MetasRelationManager::class
    ];
}
public static function getPages(): array
{
    return [
        'index' => Pages\ListItems::route('/'),
        'create' => Pages\CreateItem::route('/create'),
        'edit' => Pages\EditItem::route('/{record}/edit'),
        'view' => Pages\ViewItem::route('/{record}')
    ];
```

```
    }

    protected static ?string $recordTitleAttribute = 'name';

    public static function getGloballySearchableAttributes(): array

    {

        return ['name', 'users.name', 'category.name', 'username', 'url', 'props'];

    }

    public static function getGlobalSearchResultUrl(Model $record): string

    {

        return route('filament.resources.items.view', ['record' => $record]);

    }

    protected static function getGlobalSearchEloquentQuery(): Builder

    {

        if (Auth()->user()->hasRole('super_admin')) {

            return parent::getGlobalSearchEloquentQuery();

        } else {

            return parent::getGlobalSearchEloquentQuery()->whereRelation('users',
'user_id', '=', Auth::id());

        }

    }

}
```