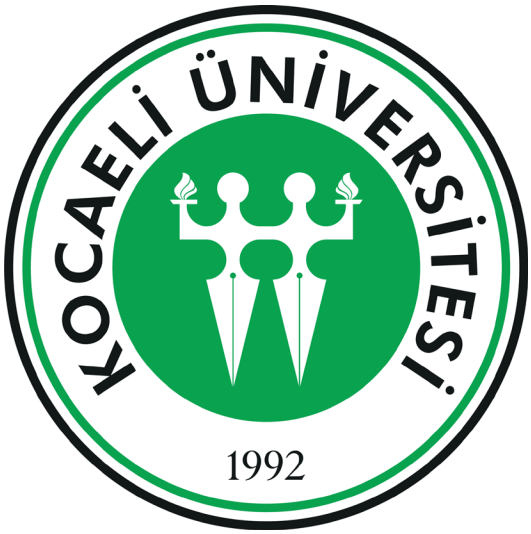


T.C.

Kocaeli Üniversitesi

Teknoloji Fakültesi

Bilişim Sistemleri Mühendisliği

Kare Bulmaca Oyunu

Mehmet Arda YÜKSEL

Güven KAZICI

Levent KIRIMGERİ

2022-2023 Bahar Dönemi

Yazılım Geliştirme Laboratuvarı-II 1.Proje

Mehmet Arda Yüksel
191307001
m.arda.yuksel@gmail.com

Güven Kazıcı
191307058
191307058@kocaeli.edu.tr

Levent Kırımgeri
191307017
191307017@kocaeli.edu.tr

***Özet*—Geliştirilen web tabanlı proje, kullanıcının yüklediği bir görseli öncelikle kullanıcı isteğine göre kırpılmasını ve kırpılmış görselin sonrasında 16 parçaya tekrardan bölünmesini(kırpılmasını) sağlayarak bir oyun alanı oluşturan, 16 görselin rastgele karıştırılması ile devamında görsellerin doğru yere eşleştirilmesine dayanan bir kare bulmaca oyunudur.**

Anahtar Kelimeler- Html, Css, Scss, Bootstrap, Javascript, Jquery, NodeJS, ExpressJS, Npm, Git, GitBash, GitHub

I. TANITIM

Bu web projesinde, backend teknolojisi olarak NodeJS kullanılmıştır. Frontend kısmında ise temel Html, Css, Javascript'in yanı sıra Css ön işlemcisi olan Scss'ten ve Css kütüphanesi olarak Bootstrap'ten, ayrıca Javascript kütüphanesi olarak Jquery'den yararlanılmıştır.

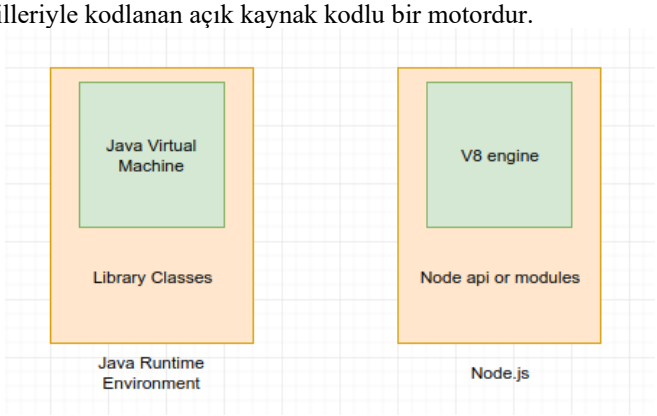
Geliştirilen web sitesinde, ilk aşamada siteye giriş yapan kişiden kullanıcı adı bilgisi istenmektedir. Kullanıcı adı girmeden oyuna başlanılamamaktadır. Kullanıcı adı ile beraber kullanıcı, "beni hatırla" seçeneğini işaretleyerek sonraki girişlerinde tarayıcıya kullanıcı adı bilgisini kaydettirerek farklı bir kullanıcı adı girmek isteyinceye veya tarayıcıdaki verilerini silene kadar siteye her girişinde kullanıcının hatırlanması sağlanmaktadır. Kullanıcı adını girdikten sonra ikinci aşamaya geçilir. Bu aşamada Kullanıcının oyuna başlayabilmesi için bir görsel yüklemesi istenmektedir. Aynı zamanda sayfanın sol kenarında oyunu oynayanlar arasında en yüksek skoru yapan 3 kullanıcının adı, hamle sayısı ve puan bilgileri listelenir. Sayfanın sağ kenarında ise eğer varsa girilen kullanıcı adı bilgisine göre giriş yapan kullanıcının en yüksek skoruna ait bilgiler listelenmektedir. Sayfanın ortasındaki formdan kullanıcı, oyunu oynayacağı görseli ister dosya olarak yükleyebilir, isterse url ile girebilmektedir. İkinci aşamada aynı zamanda kullanıcı adını değiştirebilmek için önceki aşamaya dönülebilmesini sağlayan bir buton bulunmaktadır. Kullanıcı, beni hatırla seçeneğini seçmiş ise siteye her girişinde kullanıcı adını girdiği ilk aşamadan değil, kullanıcı istatistiklerinin listelendiği ve görselin yüklendiği ikinci aşamadan başlatılır. Kullanıcının seçtiği resim yüklendiğinde bir Modal açılmaktadır. Bu modal içerisinde, yüklenen resim kare olsun olmasın oyun alanına yansıyacak görselin kullanıcı tarafından istenilen yerin 1:1 aspect ratio oranı ile kare şeklinde kırpılması sağlanmaktadır. Sonrasında kırpılan kare görsel 16 parçaya javascript ile bölünerek oyun alanı oluşturulmaktadır. Üçüncü aşamada sayfanın sol tarafında karıştırma butonu, sağ tarafında kullanıcının en yüksek skor bilgisi, ortasında ise 16 kırpılmış görselin basıldığı oyun alanı bulunmaktadır. 16 görsel sıra ile oyun alanına basıldıktan sonra karıştırma butonu aktif hale gelmektedir. Kullanıcının karıştır butonuna basmasıyla, 16 parçanın rastgele şekilde yer değişikliği yapması sağlanmaktadır. Parçalardan en az biri doğru yere geldiğinde karıştırma butonu kilitlenip oyun başlatılmaktadır. Kullanıcı 16 parçayı da doğru yere eşleştirinceye kadar oyun devam etmekte ve her hamlede kullanıcıya hamlenin doğru olup olmamasına göre geri bildirim verilmektedir. Aynı zamanda tüm süreç boyunca kullanıcı, hamle sayısı ile puanını oyun alanından görebilmektedir. 16 parçanın tamamı doğru eşleştirildiğinde oyun bitmekte ve kullanıcıya skoru gösterilmektedir. Puan durumuna ve varsa önceki en yüksek skoruna bakılarak en yüksek skorların tutulduğu bir metin belgesine kaydının düşülmesi sağlanmaktadır. Kullanıcı isterse oyun bitiş ekranından bir butona tıklayarak, görsel yüklediği aşamaya geri dönerek oyunu tekrar tekrar oynayabilmektedir.

II. KULLANILAN TEKNOLOJİLER

A. *NodeJS*

NodeJS, Javascript ile sunucu tabanlı uygulamaların geliştirilmesini sağlayan bir **Javascript Runtime** platformudur. Ryan Dahl adında Amerikalı bir yazılım mühendisi tarafından 2009 yılında geliştirilmiştir. Javascript dili normalde client-side çalışan bir dildir. NodeJS, Google'ın V8 motoru üzerinde çalışır ve bu sayede Javascript'in sunucu tarafında (server-side) çalışabilmesi sağlanmaktadır.

V8 Motoru, Google tarafından geliştirilen, C ve Javascript dilleriyle kodlanan açık kaynak kodlu bir motordur.



Şekil.1.1

NodeJS'in çalışma prensibi, Java Runtime Environment'ına oldukça benzerdir. Şekil 1.1'de görüldüğü üzere Java için Java Virtual Machine ne ise Node.js için de V8 Motoru aynısıdır. V8 Motoru, Javascript kodunu makine koduna çevirdiği için Node.js ile geliştirilen uygulamaların performansı çok yüksektir.

Node.js'de en öne çıkan kavramlar; **Asenkronluk**, **Nonblocking**, **Single Thread** ve **Event Loop** kavramlarıdır.

Şekil.1.2

Node.js'de yazılan, daha doğru bir ifadeyle özünde Javascript ile yazılan uygulamalarda işlemler gerçekleştirilirken, sıradaki işleme geçmek için işlemler birbirlerinin bitmesini ve sonuç döndürmesini beklemezler. İşlemler sırayla çalıştırılmakta, hangi işlem önce biterse

onun sonucu gerçekleştirilmektedir. İşlemlerin gerçekleşme sürelerinin farklılığı, birbirlerini engellememektedir. Bu çalışma prensibine Şekil 1.2'de de görüldüğü üzere **Asenkron** çalışma denmektedir. İşlemlerin birbirini bu prensip sayesinde engellememesine ise **non-blocking** denmektedir.

Javascript'te bütün işlemler tek bir thread üzerinde çalıştırılmaktadır. İstenilen işlemler, Javascript Runtime'ına callback veya eventler ile iletilmektedir. Bu istekler, bir kuyruk oluşturmakta ve işlemler bu kuyruktan sırayla gerçekleştirilmektedir. Bu çalışma mantığına ise Javascript'te **Event Loop** denmektedir.

Node.js'i ve modüllerini kullanabilmek için ilgili sitesinden Node.js'in bilgisayara yüklenmesi gerekmektedir. Node.js'in kurulumu yapıldıktan sonra herhangi bir terminal üzerinde "node -v" komutu ile yüklemenin gerçekleşip gerçekleşmediği, versiyon kontrolü ile beraber yapılabilmektedır. Daha sonra ilgili projede yine terminal üzerinden "npm init" komutu ile Node.js projesi oluşturulmaktadır. Bu komut sonrasında **package.json** dosyası ve **node_modules** klasörü oluşturulmaktadır. Proje içerisine npm ile bir modül yüklenmek istendiğinde ise, "npm install <modül ismi>" ya da "npm i <modül ismi>" komutu kullanılarak istenilen paket projeye kolaylıkla dahil edilebilmektedir.

1) NPM

NPM, Node Paket Yönetimi(Node Package Manager) kısaltmasıdır. Node.js uygulamalarında kullanılmak üzere modüller halinde paketler sunan bir paket yönetim sistemidir. Npm, birçok Javascript geliştiricisi tarafından desteklenen, açık kaynak kodlu bir yazılım deposudur. Buradan açık kaynak kodlu paketler geliştiriciler tarafından npmjs.com adresine yüklenmektedir. Buradaki modüller çok kısa ve basit terminal komutlarıyla projelere dahil edilebilmektedir. Buradan indirilen paketlerin kullanılabilmesi için projede **package.json** dosyası bulunmalıdır. Bu dosya, Node.js projesindeki tüm bağımlılıkları(dependencies) ve projenin y azarı, versiyonu, başlangıç noktası, lisansı gibi bilgileri de içinde bulunduran bir dosyadır. Bu dosya, Node projesi ilk initialize edildiğinde otomatik olarak oluşturulmaktadır. Npm ile bir paket, global ve lokal olmak üzere iki farklı şekilde yüklenebilmektedir. Npm, sadece paketlerin yüklenmesini değil, takibi ve güncellenmesi, silinmesi gibi işlemlerin de yapılabilmesini sağlamaktadır. Paketler **node_modules** klasörü içerisinde tutulmaktadır. Projedeki tüm paketlerin bilgileri ise, projenin tüm bilgilerinin tutulduğu **package.json** dosyasındaki **dependencies** kısmında isim ve versiyonlarıyla bulunmaktadır. Her yeni modül kurulduğunda indirilen modül, bu iki temel dosyada yer edinmektedir.

2) Express.js

Express.js, Node.js için bir uygulama çatısı(framework)dır. Node.js uygulamalarının geliştirilme süreçlerini, sağladığı middleware'ler ve yardımcı araçlar sayesinde oldukça kolaylaştırmaktadır. Express.js Framework'ü; Pug, Ejs, Handlebars gibi **template engine**'nin kullanılabilmesini de sağlamaktadır. Statik dosyaların yönetimini kolaylaştırmakta, veritabanı yönetim sistemleri ile bağlantılarının gerçekleştirilmesini ve **Route** yapısı sayesinde siteye gelen isteklerin yönetimini sağlayabilmektedir. Bir Node.js projesine dahil edilmesi normal bir modül'de olduğu gibidir. Terminal veya komut satırı üzerinden "npm i express" komutu ile indirilebilmektedir. Modülü indirdikten sonra projenin çalıştırıldığı temel .js dosyası içerisinde "const express = require('express')" ile proje içerisinde çağrılmakta ve bir constant değişkene aktarılmaktadır. Bu değişken üzerinden de bir nesne türetilmekte ve Express.js işlemleri bu nesne üzerinden yürütülmektedir.

3) EJS Template Engine

Web projelerinde oluşturulan .html uzantılı sayfalar statik yapıda yani dışarıdan veriler ile değiştirilemez haldedir. Statik bir .html uzantılı dosyada en basit haliyle veritabanından çekilmiş bir veri, kullanıcıya dinamik bir şekilde gösterilememektedir. Bunun yapılabilmesi için **Şablon Motoru(Template Engine)** kullanılmaktadır. Template Engine'ler dosyanın uzantısını değiştirerek istenilen dinamik veri alış verişini ve bu verilerin/js kodlarının, html kodlarıyla beraber kullanılabilmesini mümkün kılar. Pug, Handlebars ve Ejs gibi farklı Template Engine'ler bulunmaktadır.

Bu projede, **EJS** Şablon Motoru kullanılmaktadır.
EJS, Embedded Javascript'in kısaltmasıdır. Adından da anlaşılabileceği üzere html kodlarıyla beraber javascript'in kullanılabilmesini sağlamaktadır. Dosyanın uzantısı ise .html'den .ejs'e değiştirilir. Projeye Ejs, "npm i ejs" komutu ile indirilmektedir. Devamında Express.js modülünde olduğu gibi, projenin yürütüldüğü javascript dosyası içerisinde "const ejs = require('ejs')" komutu ile proje içerisine dahil edilip bir değişkene aktarılmaktadır. Sonrasında daha önce indirilen **Template Engine** sınıfından türetilen nesne üzerinden "set" metodu çalıştırılmaktadır. Üretilen template engine nesnesi "app" olmak üzere, "app.set('view engine','ejs')" komutu ile projede kullanılacak view engine'nin ejs olduğu belirtilmektedir. Bu işlemden sonra dosyanun uzantısı .ejs olarak değiştirilmektedir ve artık projede bu modül .ejs uzantılı dosyaları varsayılan olarak "**views**" klasörü altında aramaya başlamaktadır. Dolayısıyla views klasörü yoksa oluşturulup, ilgili uzantısı .ejs yapılmış dosya da bu klasör altına yerleştirilmelidir. Bu işlemlerin yanı sıra oluşturulan **app** nesnesi üzerinden projenin çalışabilmesi için olmazsa olmaz **port** dinleme işlemi de **listen** metodu içerisinde port numarası verilerek gerçekleştirilmektedir. Ayrıca projede kullanılacak statik dosyalar, app nesnesi üzerinden **use** metodu içerisinde Express.js'in **static** middleware'i sayesinde "app.use(express.static('public'))" komutu ile belirtilmektedir. Bu sayede statik dosyalara referans verileceği zaman public klasörüne yerleştirildikleri takdirde fazladan bir dosya yolu girilmeden sadece public klasörü içerisindeki yolu belirtilerek erişilebilmektedir.

4) Routing

Projenin yürütüldüğü temel javascript dosyası içerisinde daha önce Express.js modülünden oluşturulan "**app**" nesnesi üzerinden "**route**" metodu çalıştırılarak sayfaya yapılan isteklere ilgili yanıtların yönlendirilmesi gerçekleşmektedir.

Route metodu içerisine, parametre olarak isteğin gerçekleştiği url'i almaktadır. Devamında zincirleme şekilde **get** metodu ve onun ardından **post** metodu (ikisi ya da yalnızca biri) çağrılmaktadır. Bu metodların içerisinde isteklerin yönetildiği asıl yer olan **callback fonksiyonu** çalıştırılmaktadır. Get/Post metodu içerisindeki callback fonksiyonu iki parametre almaktadır. Bu parametrelerin birincisi gelen isteği (**request**), ikincisi ise yanıtı (**response**) temsil eden değişkenlerdir. Callback fonksiyonu içerisinde ise temel olarak kullanıcıya yanıt olarak bir sayfa gönderilecek ise response nesnesi üzerinden, **render** metodu çalıştırılarak içerisine, gönderilecek sayfanın ismi argüman olarak verilmektedir. Yanıt olarak bir sayfa değil, veri gönderilecekse response nesnesi üzerinden, bu sefer **send** metodu çalıştırılarak argüman olarak bir javascript objesi halinde istenilen veriler gönderilebilmektedir.

5) File System

Bu proje içerisinde, kullanıcıların oyun bilgileri bir metin belgesine yazdırılmaktadır. Metin belgesine yapılan dosya işlemleri javascript'in File System modülü ile gerçekleşmektedir. Bir dosya oluşturmak, silmek, içeriğini okumak, ekleme yapmak, tamamen yeni içerik yazmak gibi işlemleri gerçekleştirebilmektedir. File System modülü Node.js ile birlikte gelen çekirdek modüllerden biridir. Dışarıdan herhangi bir modülün "npm i" komutu ile indirilmesi gerekirken, bu modül halihazırda yüklü gelmektedir. Projenin çalıştırıldığı javascript dosyasında "require('fs')" komutu ile tanımlanıp bir nesneye atanarak kullanılmaktadır. Bu nesne üzerinden çeşitli metodlar çalıştırılarak kullanılmaktadır. Çalıştırılacak metodların senkron ve asenkron olmak üzere iki farklı versiyonu bulunmaktadır. Okuma yapmak için oluşturulan nesne "fs" olmak üzere "fs.readFile()" komutu çalıştırılır. Buradaki **readFile** metodu içerisine üç parametrealmaktadır. Birincisi dosyanın adı, ikincisi karakter kodlaması (örn: utf8), üçüncüsü ise bir callback fonksiyonudur. Callback fonksiyonu içerisine biri hatayı, diğeri okunan dosyanın içeriğini tutan iki parametre ile çalışmaktadır. Yazma işlemi için ise "fs.writeFile()" komutu çalıştırılmaktadır. Buradaki **writeFile** metodu içerisine üç parametre almaktadır. Birincisi dosyanın adı, ikincisi dosyanın içerisine yazılacak veri, üçüncüsü de bir callback fonksiyonudur. Callback fonksiyonu ise içerisinde hatayı alan bir parametre alarak çalışmaktadır.

B. JQuery

JQuery, 2006 yılında John Resig tarafından geliştirilen, ücretsiz, açık kaynak kodlu bir javascript kütüphanesidir.
"Write less do more" prensibine sahip olan jquery kütüphanesi vanilla javascript denilen yalın javascriptin nispeten uzun ve karmaşık kodlarının yaptığı bir işi; çok daha kısa, basit ve pratik kodlar ile gerçekleştirebilmeyi sağlamaktadır. Açık kaynak kodlu büyük bir geliştirici topluluğuna sahiptir, dolayısıyla pek çok yararlı paketi tüm geliştiricilerin kullanımına sunmaktadır. Zengin kütüphanesinin yanısıra, en öne çıkan özelliklerinden birisi **cross-platform** olmasıdır. Tüm tarayıcılarda sorunsuz çalışabilmektedir. Vanilla javascript ile yazıldığı taktirde tarayıcı farklılıklarından dolayı çıkabilecek sıkıntıları göz önünde bulundurarak fazladan gerekecek pek çok ekstra kodu kendi bünyesinde halletmekte ve uyumlu bir uygulama geliştirmeyi kolaylaştırmaktadır. Kendi içerisinde bulunan css dosyasıyla, yerleşik bir şekilde bulunan çeşitli metodları ile kendine özgü animasyonlar vb. css işlemlerini yapmayı da kolaylaştırmaktadır. Javascript'te kullanılan Belge Nesne Modelini (Document Object Model - DOM) kullanarak tüm html manipülasyon işlemlerini de kolaylaştırmaktadır. Kullanmak için iki farklı versiyonu vardır. Biri üretim versiyonu, sıkıştırılmış sadece kullanım amaçlı; bir diğeri ise geliştirici versiyonu, sıkıştırılmamış okunaklı kodlar ile test ve geliştirme amaçlıdır. Yine daha öncesinde bahsedilen, bütün sağladığı kolaylıklar gibi, Ajax gibi tarayıcılarda farklılık gösterip sorun çıkartabilecek bir işlemi de kendi içerisinde bütün uyumluluk sorunlarını çözerek basit ve sade bir şekilde kullanılmasını sağlamaktadır.

C. SCSS

SCSS, css'te bulunmayan programlama özelliklerini kullanabilmeyi sağlayan bir css ön işlemcisi (pre-processor) dir. Syntactically Awesome Style Sheet (Sass)'ın aynısıdır. Sass ile Scss'in aralarındaki tek fark, syntax farklılıklarıdır. Scss'in syntax'i css'e daha çok benzemektedir. Css'te bulunan süslü parantez ve noktalı virgül kullanımı Scss'te kullanılırken, Sass'ta bu karakterlerin yeri boş bırakılmakta yani kullanılmamaktadır. Scss, **Sorgu yapıları**, **Döngüler**, **Değişkenler**, **Import yapısı**, **Fonksiyonlar** iç içe yapılar(**Nesting**), Kalıtım(**Inheritence**), çeşitli **Matematik fonksiyonları** gibi pek çok css'te bulunmayan özellikleri eklemektedir. Bir css ön işlemcisi olduğu için Scss kodunun nihai çıktısı yine bir Css'tir. Scss, Css'e derlenmektedir. Bu derleme işlemi için çeşitli açık kaynak kodlu compiler'lar bulunmaktadır. Bu projede, Scss compiler olarak **Koala Compiler** kullanılmıştır. Scss dosyaların uzantısı ise .scss'tir.

D. Bootstrap

Bootstrap; açık kaynak kodlu, oldukça popüler, Twitter tarafından geliştirilen bir Css kütüphanesidir.
Javascript için JQuery ne ise, Css için de Bootstrap o'dur. Css kodlarının yaptığı işleri daha kısa ve basit şekilde yapmayı sağlamaktadır. Bootstrap'in temelinde biri css diğeri javascript olmak üzere iki dosya görev almaktadır. Bootstrap'in css dosyası, ilgili class'lara karşılık gelen stillerin tanımlandığı yer iken javascript dosyası ise bu stillere aktiflik ve dinamikliğin kazandırıldığı dosyadır. Bootstrap'in proje içerisinde kullanılabilmesi için bu iki dosyanın projeye dahil edilmesi gerekmektedir. Bunun için ise farklı yöntemler bulunmaktadır. Bu yöntemlerden birincisi npm ile indirmek, ikincisi cdn sunucusu ile dahil etmek, üçüncüsü ise dosyaları doğrudan harici olarak indirip statik bir dosya olarak projeye dahil etmektir. Npm ile harici olarak indirmek arasındaki fark, npm yönteminde dependencies kısmına kütüphanenin eklendiği gibi dosyaların node_modules içerisinde yer almasıdır. Harici yöntemde ise dosyalar, doğru referans vermek kaydıyla istenilen yerde tutulabilmektedir. Kullanım mantığı, esasında kullanılacak html dokümanının içerisinde hem css hem de js referansı verildikten sonra, istenilen html elemanının class attribute'üne Bootstrap dokümanında yer

alan stillerin hangisi uygulanmak isteniyorsa ilgili class'ı tanımlayarak gerçekleşmektedir.

E. Git, GitBash ve GitHub

Git, açık kaynak kodlu bir sürüm/versiyon ve kaynak kod kontrol-yönetim sistemidir. Normalde bir proje geliştirilirken projenin kaynak kodu, yazılımcının isteğine göre harddiskinde veya bulut ortamı gibi yerlerde saklanabilmektedir. Proje geliştirilmeye devam edildikçe, projenin sürümleri oluşmaktadır ve depolandığı yerlerde bir karmaşa söz konusu olmaktadır. Ayrıca, örneğin harddiskte depolanan bir projenin güvenliği de şüphelidir. Bir projenin geriye döndürülebilirliği ve de oldukça önemli bir diğer konu olan ekip ile çalışma esnasındaki kodların paylaşılabilirliği açısından da geleneksel yöntemler çok sıkıntılıdır. İşte bu ve bunun gibi tüm sorunlara çözüm olarak Git kullanılmaktadır.

Git sayesinde projelerin versiyon kontrolü çok daha kolay yapılabilmekte, ekip çalışması ve ekipteki yazılımcıların eş zamanlı proje üzerinde çalışmalarında kodların paylaştırılması ve birleştirilmesi oldukça kolaylaşmakta ve tüm bu işlemlerin hızlı ve pratik bir şekilde gerçekleştirilmesi sağlanmaktadır. Proje geliştirilmesi esnasında herhangi bir aşamada istenilen değişikliklerin geri ve ileri alınabilmesini sağlamaktadır.

Kullanabilmek için öncelikle, git-scm.com sitesinden GitBash ile birlikte indirilmesi gerekmektedir.

GitBash, Git özelliklerinin kullanılabildiği bir komut satırıdır. Pek çok Unix komutlarının da kullanılabilmesini sağlamaktadır. Git'i projede kullanabilmek için GitBash komut satırı harici olarak kullanılabildiği gibi Visual Studio Code içerisinde terminal kısmından dahili olarak editör içerisinden de erişilerek de kullanılabilmektedir. Git kullanımının temelinde 3 farklı ortam yer almaktadır. Bunlardan birincisi "**working directory**" ikincisi "**staging area**" üçüncüsü ise "**local repository**"dir. Git kullanımında kaynak kodlar bu üç farklı bölgede ileri geri hareket ettirilmektedir. Working Directory(çalışma dizini) kaynak kodların yer aldığı dosyaların dizinidir. Staging Area(Geçiş Bölgesi-Index Bölgesi) dosyaların tutulduğu ara bir bölgedir. Local Repository ise proje kodlarının lokal ortamda tutulabileceği son duraktır.

Git kullanımında öncelikle ilgili proje klasörü içerisinde GitBash üzerinden "git -init" komutu ile local repo'nun oluşturulması gerekmektedir. Bu komut ile çalışma dizininde .git uzantılı gizli bir klasör oluşturulmaktadır. Bu klasör projenin Local Repository'si bir diğer deyişle zaman makinesidir. Git kullanımında en sık kullanılan komutlardan biri de "git status" komutudur. Bu komut ile dosyaların güncel durumu sorgulanabilmektedir. Git kullanımında öncelikle tüm dosyalar, "git add" komutu ile working directory'den staging directory'e aktarılarak izlenmeye başlanmaktadır. Staging Directory'den Local Repo'ya ise "git commit" komutu ile aktarılmaktadır. Bu işlemler ilgili git komutlarıyla tersine de çevrilebilmekte, istenilen tek bir işlem geri alınabildiği gibi belli bir işlemin olduğu yere kadarki tüm değişiklikler de geriye ya da ileriye döndürülebilmektedir.

GitHub ise esasında **Remote Repository**'lerden en popüler olanıdır. Yazılımcılar arasında kaynak kodların paylaşılıp depolandığı bir yerdir. Aynı zamanda bir sosyal ağ görevi de görmektedir. Git kullanımıyla beraber GitHub içerisinde bir repo oluşturulmakta ve bunu local repo ile bağlayarak versiyon yönetiminin bulut ortamında gerçekleştirilmesi sağlanmaktadır. GitHub ortamı, ekip çalışmasında da elbette Git ile beraber kullanımıyla projenin eş zamanlı geliştirilmesinde kolaylık sağlamaktadır.

III. GELİŞTİRME AŞAMALARI

A. Yeni Javascript Projesi

1) Visual Studio Code ve Extensions

Proje için geliştirme ortamı olarak Visual Studio Code kullanılmıştır. Visual Studio Code ortamında extensions kısmından Html, Css, Javascript ve NodeJS kullanımında oluşturulan ".ejs" uzantılı dosyalar içerisinde gerekli EJS dili için intellisense sağlamak amacıyla birkaç farklı uzantı indirilmiştir. Bunların yanı sıra dosyaların ikonlarını kendilerine verilen isimlere göre daha anlamlı ikonlarla değiştiren Material Icon Theme uzantısı indirilip, ayarlardan tema olarak seçilmiştir. Projede kullanılan diğer gerekli kütüphaneler extension olarak ya da npm kullanılarak Node Modules içerisine değil statik dosyalar halinde harici şekilde indirilip klasörlere yerleştirilerek ve ilgili yerlerde referans gösterilerek kullanılmıştır.

2) Proje Klasörlerinin Oluşturulması, Harici Paketlerin Yüklenmesi

Projede backend teknolojisi olarak NodeJS kullanılmaktadır. NodeJS'in kullanılabilmesi için öncelikle nodejs.org sitesi üzerinden tüm kullanıcılara önerilen sürüm indirilmiştir. NodeJS'in bilgisayarda yüklü olup olmadığının kontrolü bir işletim sisteminde yer alan veya git bash gibi farklı bir komut istemcisi üzerinden "node -v" komutu ile versiyonunu sorgulayarak öğrenilebilmektedir. NodeJS kurulumu sonrası VS Code üzerinden proje kodlarının yer alacağı "**SquarePuzzleGame**" adında bir ana klasör oluşturulmuştur.

a) Public Klasörü

"SquarePuzzleGame" klasörü içerisinde statik dosyaların yer alacağı "public" klasörü oluşturulmuştur. Public klasöründe "css", "js" ve "img" adında üç klasör oluşturulmuştur. Bu klasörlerin her birinde, adından da anlaşılabileceği üzere ilgili dilden harici indirilmiş statik dosyalar gruplandırılmış şekilde bulunmaktadır. Public klasörü içerisindeki ilk klasör olan "css" klasörü altında; "bootstrap", "cropper" ve "custom" adında üç klasör ve bir "all.css" dosyası oluşturulmuştur. Bootstrap klasöründe sitesinden indirilen "bootstrap.min.css" ve "bootstrap.min.css.map" dosyaları yerleştirilmiştir. Bu sayede bootstrap kütüphanesi'nin stilleri projeye dahil edilmiştir. Cropper klasöründe kullanıcı taraflı kırpma işleminde kullanılacak cropper.js kütüphanesi, ilgili github repository'sinde bulunan "cropper.css" dosyası indirilip yerleştirilerek projeye dahil edilmiştir. Custom klasöründe ise "animation.scss" ve "reset.css" dosyaları oluşturulmuştur. Animasyon scss dosyası, oyun alanı

içerisindeki 16 kırpılmış görselin yer değiştirmesi esnasında bir animasyon uygulanması için gerekli algoritmanın yazıldığı dosyadır. Reset css dosyası ise html elementlerinin stillerinde, tarayıcı farklılıklarından kaynaklanan problemleri engellemek, yeniden düzenlenmiş varsayılan bir görünüm elde etmek ve çok sık kullanılan css kodlarını yine varsayılanmış gibi kullanılmasını sağlamak amacıyla oluşturulmuştur. Public klasörü içindeki css klasöründe yer alan dosyalardan sonuncusu "all.css" dosyası, sitedeki tüm elementlere başta class seçicisi olmak üzere seçiciler kullanılarak stillerinin tanımlandığı ana stil dosyasıdır.

Public klasörü içerisinde oluşturulan ikinci klasör "img" klasörüdür. Bu klasörde sadece sitenin favicon resmi bulunmaktadır.

Public klasörü içerisinde oluşturulan üçüncü klasör "js" klasörüdür. Burada css klasöründe stil dosyaları yerleştirilen bootstrap ve cropper kütüphanelerinin javascript dosyaları kendi isimleriyle birer klasör içerisine yerleştirilmiştir. Jquery kütüphanesi de yine bu klasör altında "min.js" ve "min.map" dosyaları indirilip yerleştirilerek dahil edilmiştir. Son olarak "extra.js" adında bir dosya oluşturulmuştur. all.css nasıl tüm stillerin yer aldığı dosya ise bu dosya da bütün javascript'in yer aldığı ana script dosyasıdır.

b) Views Klasörü

Proje baştan sona tek bir sayfada başlayıp bitmektedir. Farklı işlemler için farklı sayfalar oluşturulmamıştır. Views klasörü içerisinde bundan dolayı tek bir "**index.ejs**" dosyası oluşturulmuştur. Bu dosyada projede gerekli html elemanlarından bazıları ön tanımlı bazıları ise javascript ile runtime anında üretilmektedir. Projede NodeJS kullanıldığı için dosya .html değil .ejs uzantısına sahiptir.

c) Ana Dizin ve Node Modules Klasörü

Ana dizinde başlıca projenin çalıştırıldığı "**app.js**" dosyası oluşturulmuştur. Terminal üzerinden "npm i <paket adı>" komutu ile projede gerekli olan kütüphaneler/modüller indirilmiştir. İndirilen modüller: Express.js, File System, Ejs, Body Parser ve Cors paketleridir. Buradaki tüm dosyalar Node_Modules klasörü içerisine indirilmektedir.

B. App.js Yürütme Dosyası ve Routing

Projenin Node.js tarafından çalıştırılan temel yürütme dosyası "**app.js**" tir. Node.js'in hangi dosyayı baz alacağı, package.json dosyasındaki dependencies kısmındaki "**main**" property'si ile belirtilmektedir. App.js içerisinde ilk olarak en başta npm ile indirilen modüller, projeye dahil edilmiştir. **Require** fonksiyonu ile paketler, birer nesnelere aktarılmıştır. Express.js framework'ünden oluşturulan nesne üzerinden "**set**" metodu kullanılarak projede kullanılacak template engine'in ejs olduğu belirtilmiştir. Yine aynı Express.js nesnesi ile "**use**" metodu kullanılarak projedeki statik dosyaların bulunduğu public dizini tanımlanmıştır. Bu sayede html içerisinde statik dosyalar referans olarak verilirken public dizini içerisinden aranmaya başlanacaktır. Use metodu son olarak ileride metin belgesine yazı yazdırmak için ajax isteğinde bulunulduğunda, post edilen verilerin request üzerinden body property'si ile erişebilmeyi sağlayan "bodyParser.json" tanımlamada da kullanılmıştır. Express.js'ten üretilen nesne üzerinden "**route**" metodu çalıştırılarak, metod içerisinde projenin çalışacağı ana url belirtilmiştir. Bu url "/square_puzzle_game"dir. Bu url uzantısı girildiğinde gelen isteklerin nasıl yönetileceği **route** metoduna zincirleme şekilde **get** ve **post** metodları yazılarak sağlanmıştır. Get isteğinde bulunulduğunda, metoda callback fonksiyonu içerisinde parametre olarak girilen response nesnesi üzerinden **render** metodu çalıştırılıp içerisinde belirterek, views klasörü içerisindeki "index.ejs" dosyası kullanıcıya döndürülmesi sağlanmıştır. Route metodunun get isteğinde bu şekilde kullanıcıya index sayfası döndürülmektedir. Post isteği ise yine aynı url'e oyun sonunda ajax'in post metoduyla oyuncunun bilgilerinin gönderilip ilgili metin işlemlerinin yapılmasını sağlamaktadır. Express.js nesnesi, ayrıca yine route metodu ile metin belgesinden oyuncuların verilerinin çekilmesinde kullanılmıştır. Metin belgesi işlemlerine, ilgili başlıklarda daha detaylı değinilecektir. Son olarak projenin çalışacağı port numarası, Express.js nesnesi üzerinden **listen** metodu kullanılarak belirtilmiştir. İçerisinde console.log komutuyla dinlemeye başladığında bir mesaj yazdırılması sağlanmıştır.

C. Index.ejs'in Oluşturulması

Views klasörü içerisinde oluşturulan Index.ejs dosyası, projedeki bütün işlemlerin yapıldığı tek bir dosyadır. Proje içerisinde index.ejs'ten başka bir sayfa/view kullanılmamıştır. Bu sayede yapılan tüm işlemler, javascript tarafından dom(document object model) manipülasyonu sayesinde yönetilip runtime anında gerçekleştirilmesi sağlanmaktadır. Dosyada ilk olarak doctype html bildirimi yapılarak html etiketleri ile içerik oluşturulmaya başlanmıştır. Head etiketi içerisinde karakter ve ekran uyumunu sağlayan meta etiketlerinin yanı sıra projedeki stillerin yer aldığı dosyalar, link etiketleriyle href attributelerinde dizinleri belirtilerek çağrılmıştır. Çağrılan stil dosyaları; projedek, harici tüm css'lerin yazıldığı "all.css", cropper kütüphanesinin css dosyası olan "cropper.css" ve bootstrap kütüphanesinin "bootstrap.min.css" dosyasıdır. Bunların dışında head etiketi içerisinde son olarak sayfanın başlığı title etiketi ile yazılmış, faviconu da yine link etiketi ile tanımlanmıştır.

Head etiketi sonrasında body etiketi oluşturulmuştur. Body etiketi içinde, projenin tasarım tercihi olarak bütün elemanları kaplayacak bir wrapper div'i oluşturulmuştur. Projede bütün olan biten işler bu sabit wrapper içerisinde gerçekleşmektedir. Bu wrapper(flex-box), css'i flexbox property'sinden yararlanılarak içerisine gelen elemanlar(flex-item) merkeze hizalanmıştır. Genişlik ve yüksekliği ise sayfanın 100%'üne eşitlenmiştir. Bu sayede içerisindeki elemanlar için bir container görevi görmektedir. Overflow css property'si hidden olarak ayarlanarak, sayfada herhangi bir taşma olmayacak şekilde bir tasarım esas alınmıştır. Yani herhangi bir scroll/kaydırma işlemi gerekmeyecek şekilde proje tasarımı yapılmıştır. Son olarak projenin/oyunun tamamında yer alacak bir arka plan animasyonu, bu wrapper'a bağlı olmak üzere gerçekleştirilmiştir. Bu animasyona ilgili başlıkta değinilecektir.

Wrapper div'i içerisinde kullanıcının oyuna başlamadan önceki biri kullanıcı adı diğer görsel yükleme yeri olmak üzere iki formu barındıran bir "welcome-section" class'ına sahip bir div daha oluşturulmuştur. Bu div içerisinde sayfasının sol ve sağ taraflarında gösterilmek üzere birer oyuncu istatistik kısımları oluşturulmuştur. Bu welcome section div'inin de içerisine bir form container daha oluşturulup içerisine bahsedilen iki form birbirlerinden ayrı olacak şekilde birer wrapper ile yazılmıştır.
Sayfada, yüklenen görselin kırpılma işleminin gerçekleştiği bir bootstrap elemanı olan modal eklenmiştir. Body etiketi içerisinde, en sona projede kullanılan javascript kodları script etiketleri ile ilgili dosyalar referans verilerek tanımlanmıştır. Bu scriptler: projedeki tüm harici javascript'in yer aldığı "extra.js" ile ; cropper, bootstrap ve jquery'nin scriptleridir.

D. Genel Tasarım Tercihleri

Projenin tamamında kullanılmak üzere, public/css klasörü içerisinde bootstrap gibi paketlerin dışında, tek bir stil dosyası oluşturulmuştur. Bu stil dosyasının en başında projede tercih edilen yazı tipi stilleri, Google fontları cdn sunucusu aracılığıyla import edilmiştir.
İkon kullanımı için fontawesome tercih edilmiştir ve kullanılabilmesi için cdn sunucusundan stil dosyasının en başına import edilmiştir. Site içerisindeki scroll barı da webkit-scrollbar css komutları ile değiştirilmiştir.
Projede bütün sayfalarda tasarım için kullanılan bir yöntem söz konusudur. Oluşturulan view içeriği her zaman bir wrapper container div'i içerisinde geliştirilmiştir. Bu en dışta kalan wrapper container'ı da css tarafından display'i flexbox olarak ayarlanmıştır. Justify-content ve align-items komutları ile içerisinde oluşturulmuş elemanların genel olarak hizalanması daha kolay sağlanmaktadır. Hizalamanın dışında arkaplan renginde ve overflow komutlarında da değişiklik yapılmaktadır. Sonrasında yazılan tüm html elemanları bu wrapper container'ına göre davranmaktadır.

E. Kullanıcı Adı ve Görsel Yükleme Formunun Dış Container'ı ve Animasyonunun Oluşturulması

Body etiketindeki wrapper içerisinde oluşturulan welcome section div'inin içerisine kullanıcı adının ve sonrasında görselin girileceği iki formu barındıran welcome-form-container class'ına sahip bir container oluşturulmuştur. Bu container'ın içerisine ara bir iç container daha oluşturulup bu içteki container'a form container'ları eklenmiştir. Bu şekilde iç içe iki katman oluşturulmasının sebebi form container'ına eklenecek animasyondan dolayıdır. Bu animasyon, form container'ının border'ının birden fazla geçişli renk ile sürekli döndürülmesidir. Welcome-form-container'ına belirli bir genişlik ve yükseklik verilmiştir. Position css property'si relative yapılarak içerisine gelecek absolute elemanların hizalanması sağlanmıştır. Animasyonda asıl görev alan, dereceyi belirtecek özel bir css propery'si oluşturulmuştur. Welcome-form-container'ının "before" ve "after" pseudo elementleri kullanılarak background property'sine conic-gradient özelliği verilmiştir. Bu property içerisine, oluşturulan özel css property'si derece olarak belirtilmiştir. Ayrıca geçiş uygulanacak 4 farklı mavi renk girilmiştir. Pseudo elementlerine animasyon, "animation" css property'si ile animasyonun adı, süresi, zamanlama fonksiyonu ve tekrarlama adedi girilerek belirtilmiştir. Form validasyonuna göre welcome-form-container'ın class'ına "valid" ya da "invalid" class'ları eklenmektedir. Bu class'lar sayesinde before ve after pseudo elementleri üzerinden başlangıçta verilen mavi rengi invalid class'ı varsa kırmızıya, valid class'ı varsa yeşile çevrilecek şekilde yazılmıştır. Tek başına after pseudo elementine filter css propery'sine blur fonksiyonu verilerek form container border'ının ışık saçıyormuşçasına görünmesi sağlanmıştır. Border'ın sürekli dönme animasyonu için ise css'te "keyframes" özelliğinden yararlanılmıştır. Burada animasyonu tanımlarken 0% ile 100% arasında daha önce oluşturulan conic gradient içerisinde dereceyi belirten özel css property'si 0 ile 360 derece arasında değiştirilmektedir. Derece değeri değiştikçe conic-gradient içerisindeki açı değeri sürekli değişmekte ve bu şekilde border'ın dönme animasyonu sağlanmaktadır.

Welcome-form-container'ının içinde bir inner-container eklenmiştir. Kullanıcı adı ile görsel için iki farklı form container'ı bu iç container'ın içerisinde oluşturulmuştur. Welcome-form-container'ına eklenen border animasyonu before ve after pseudo elementlerine ve bunların da arka planına uygulandığı için absolute olacak şekilde üzerine bir katman daha oluşturacak bir container ile formların eklenmesi gerekmektedir. Bu katmanı oluşturan eleman inner-container'dır. Bu iç container'a position css propery'si absolute verilerek istenilen "katman" özelliği sağlanmıştır. Genişlik ve yükseklik değerleri ise dıştaki container'a bağlı olacak şekilde css'te absolute elemanların bağlı olduğu elemana göre hizalamakta kullanılan top-right-bottom-left property'leri kullanılarak verilmiştir. Tüm yönlere 6 piksel değeri verilerek dış container'ının genişlik ve yüksekliğinden içeriye doğru 6 piksel daha küçük bir genişlik-yükseklik verilmiş olmaktadır. Dıç ile iç container'lar arasındaki 6 piksellik bu fark welcome-form-container'ının border genişliğini oluşturmaktadır. Welcome-form-container'ındaki her iki form başlangıçta bootstrap'in "d-none" class'ı ile görünmez ve etkileşime geçilemez halde başlatılmaktadır. Bu özellik, daha sonrasında jquery'nin hide fonksiyonu ile yer değiştirmektedir. Jquery'nin hide fonksiyonunun kullanılmasının sebebi yine jquery'de bulunan bir elemanın yumuşak bir şekilde ekrana gösterilip kaldırılmasını sağlayan fadeIn ve fadeOut animasyon fonksiyonlarıyla görsel uyum içerisinde çalıştırılmasıdır.

F. Kullanıcı Adı Formu ve İşlemlerinin Oluşturulması

Welcome-form-container'ı içerisinde oluşturulan iki form container'ının ilki içerisinde kullanıcı adı ve "beni hatırla" bilgisini alan iki input yer almaktadır. Form içerisindeki elemanların hizalanmasında yine css'teki flexbox özelliğinden yararlanılmıştır. Form içerisine sırasıyla kullanıcı adı input container'ı, kullanıcı adı validasyon mesajının yazılacağı bir div'i, beni hatırla checbox container'ı ve son olarak devam etme butonu yerleştirilmiştir. Kullanıcı adı input container'ına bir span etiketiyle, focus olunduğunda, yani kullanıcı input üzerine odaklandığı müddetçe bir kayma efekti gerçekleştirecek placeholder yerleştirilmiştir. Kullanıcı adı input elemanına

"extra.js" javascript dosyasında jquery kullanılarak erişilmektedir. Buradaki girilen değeri kontrol etmek amacıyla bir validasyon fonksiyonu oluşturulmuştur. Globalde, kullanıcı adı regular expression'ının tanımlandığı bir değişken oluşturulmuştur. Validasyon fonksiyonu içerisinde kullanıcı adı input değeri yakalanmaktadır. Oluşturulan regex değişkeni üzerinden "test" metodu içerisine bu değer girilmekte ve buradan true-false değer elde edilmektedir. Elde edilen bu boolean değere göre ilgili container'a, ve input'a duruma göre "valid"-"invalid" class'ları eklenip kaldırılarak renkleri değiştirilmektedir. Kullanıcıya aynı zamanda hatanın mesajı da oluşturulan validasyon mesaj div'i içerisine bastırılarak gösterilmektedir. Bu fonksiyondan, validasyon durumuna göre nihayetinde bir true ya da false değeri return edilmektedir. Elemana, input içerisindeyken her bir tuşa tıklanma anını yakalayacak bir "keyup" eventi eklenmiştir. Bu event içerisinde oluşturulan validasyon fonksiyonu sürekli çağrılarak anlık olarak kullanıcıya geri dönüş sağlanmaktadır. Kullanıcı adı input'undan sonra "beni hatırla" bilgisini içerecek bir checkbox oluşturulmuştur. Bu checkbox'ın tıklanma durumu, jquery'nin "is" metodu üzerinden ":checked" seçicisi sorgulanarak alınmaktadır. Bu metodtan true-false boolean bir değer döndürülmektedir. Kullanıcı adı formu daha önceki başlıkta da belirtildiği üzere, başlangıçta görsel yükleme formuyla beraber kullanıcıya gösterilmemektedir. Eğer kullanıcı önceki girişinde "beni hatırla" seçeneğini seçmiş ise bu form hiç açılmayıp kullanıcı doğrudan ikinci aşamaya yönlendirilip, görsel yükleme formu açılmaktadır. Beni hatırla seçeneği işaretlenmediği sürece ilk başta kullanıcı, bu form ile karşılanmaktadır. Formun son elemanı olarak bir buton oluşturulmuştur. Bu buton, tıklama event'inde kullanıcı adı formunu kapatıp, görsel yükleme formunu açmayı sağlamaktadır. Bu işlemi yapmanın ön şartı olarak kullanıcı adı validasyon fonksiyonu burada tekrar çalıştırılmaktadır. Dönen değer false olduğu müddetçe, yani kullanıcı doğru bir değer girene kadar devam etmeye izin verilmemektedir. Kullanıcı validasyondan geçen doğru bir kullanıcı adı girdiğinde, ilgili validasyon fonksiyonu true değeri döndürmekte ve işlemlere bu durumda devam edilmektedir. Öncelikle kullanıcı adı bilgisiyle "beni hatırla" seçeneğinin işaretlenme durumu, bir javascript objesi oluşturulup, ilgili property'ler üzerinden tanımlanmıştır. Oluşturulan javascript objesi, "JSON.stringify()" metodunun içerisine yazılarak bir json stringine dönüştürülmüştür. Sonrasında bu json string'i javascript'in local storage'ına "localStorage.setItem()" komutuyla key-value şeklinde, key'i "userinfo", value'su oluşturulan json string'i olacak şekilde kaydedilmiştir. Kullanıcının local storage'ında kaydının olduğu tarayıcıdan sonraki her girişinde, kullanıcı başka bir kullanıcı adı girmek istemediği müddetçe "hatırlanmaya" devam edilmektedir. Hatırlama işlemi "localStorage.getItem()" komutu üzerinden kaydedilen key-value ile bilginin geri çağrılıp kontrol edilmesiyle gerçekleştirilmiştir. Geri çağrılan json stringini kullanabilmek için "JSON.parse()" komutu ile tekrardan javascript objesine dönüştürülmüştür. Local storage'taki tutulan kullanıcı adı ve beni hatırla bilgisi, uygulamanın devamında da ihtiyaç duyulan yerlerde tekrar çağrılmaktadır. Local storage'a bilginin kaydedilmesinin ardından kullanıcı adı formu, jquery'nin fadeOut metodu ile yumuşak bir şekilde kaldırılıp yerine, görsel yükleme formunu fadeIn metodu ile yumuşak bir geçiş ile getirilmektedir.

G. Oyun Görseli Seçme Formu ve Yükleme İşlemlerinin Oluşturulması

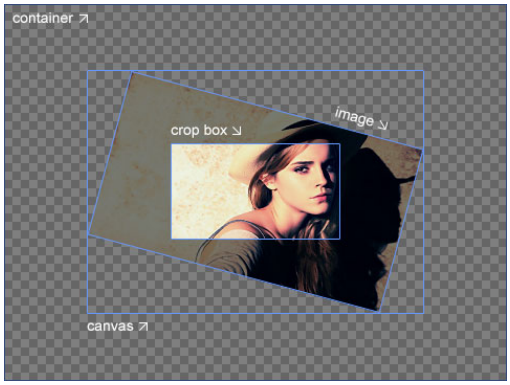
Jquery'nin "fadeIn" metodu ile welcome-form-container içerisine herhangi bir sayfa yenilenmesi olmaksızın ekrana getirilen formun içerisinde sırasıyla: kullanıcı adı formuna dönme butonu, görsel girme file input butonu, görselin url'ini girme text inputu ve url inputuna bağlı gönderme butonu oluşturulmuştur. Geriye dönme butonuna tıklama event'i, görsel yükleme formunu kapatıp, kullanıcı adı girme formunu geri getirmeyi sağlamaktadır. Görsel yükleme butonu tasarımı değiştirilmiş bir file input'udur. Tıklama event'inde, bir dosya seçme penceresi açılmaktadır. Buradan kullanıcı bir dosya/görsel seçtiği zaman, inputun değerinin değişme event'i tetiklenmektedir. Bu event içerisinde, girilen dosyanın validasyonu kontrol edilmektedir. Bu validasyon kontrolü için kullanıcı adı kontrolünde olduğu gibi ayrı bir fonksiyon oluşturulmuştur. Dosya validasyon fonksiyonuna parametre olarak girilen dosya "event.target" ile gönderilmiştir. Fonksiyon içerisinde, seçilen dosyanın boyutu, adedi, uzantısı kontrol edilip boolean değer döndürülmektedir. Uzantı kontrolü sadece **jpeg** dosyalarına izin verilecek şekilde bir **regex** yazılarak yapılmıştır. Yine kullanıcı adı formunda olduğu gibi, formun bulunduğu dış container olan welcome-form-container'a "valid" ve "invalid" class'ları verilmektedir. İlgili hata mesajları, yine aynı fonksiyon içerisinde mesaj olarak, validasyon metni için oluşturulmuş bir div içerisine hatanın türüne göre bastırılmaktadır. Yüklenen dosya validasyon işleminden geçerse, bir "**FileReader**" nesnesi oluşturulmaktadır. Bu nesne üzerinden "**readAsDataUrl**()" komutu çalıştırılıp, içerisine yüklenecek görselin tutulduğu değişken, parametre olarak girilmektedir. Nesne üzerinden "**onload**" eventi çalıştırılarak okuma işlemi bittikten sonra yapılacak işlemler tanımlanmaktadır. Reader nesnesinin "**result**" property'si ile yüklenen görselin url'i, daha önce tanımlanmış hazırda bekleyen bootstrap'in modal elemanının içerisindeki, cropper.js'in kaynağı olacak "**img**" etiketinin "src" attribute'üne atanmaktadır. Buradaki img etiketini temsil eden değişken ile, src attribute'ünün yüklendiğini yakalamak için yine onload eventi kullanılmıştır. Event içerisinde kullanıcının görseldeki kırpılacak alanı seçtiği modal, show metodu ile açılmaktadır. Son olarak kırpma işlemlerinin gerçekleştirildiği fonksiyon çağrılmaktadır.

Kullanıcı, dosyayı file input'u kullanarak seçebildiği gibi, herhangi bir url'i text input'una girerek'te yükleyebilmektedir. Bu işlem için input'a girilen url, ayrı bir fonksiyon içerisinde, kendine ait bir regex sorgusundan geçerek kontrol edilmektedir. Validasyon kontrolü çalışma prensibi olarak daha önce anlatılanların aynısıdır. Validasyondan geçmeyi başaran url için süreç dosya seçme işlemine çok benzer şekilde devam etmektedir. Kullanıcının girdiği url, modal içerisindeki cropper.js'in kaynağı olacak img etiketinin src attribute'üne, bu sefer aracı bir FileReader nesnesi kullanmadan direkt atanmaktadır. Devamında ise file input'unda olduğu gibi, onload eventi içerisinde

modal'ın show metoduyla açılması sağlanmakta ve son olarak kırpılma işlemlerini gerçekleştirecek fonksiyon çalıştırılmaktadır.

H. Cropper.js ve Modal Kullanımı ile Kullanıcının Yüklediği Görselin Kırpılması

Cropper.js, temelde üç yapı ile çalışmaktadır. Bunlar: container, canvas ve kırpma kutusudur. Bir kırpma container'ı içerisinde canvas oluşturulmakta ve yüklenen görsel buraya çizilmektedir. Container'ın boyutu ve varlığı cropper.js'in çalışması için son derece önemlidir. Çünkü çizilen canvasın boyutları bu container'a göre oranlanıp belirlenmektedir. Özellikle bu projede de olduğu gibi bir modal içerisinde kullanılacaksa muhakkak modal tamamen açıldıktan sonra çalıştırılmalıdır. Modal açıldıktan sonra kullanıcının istediği yer ve istenilen boyutta görseli kırpabilmesi için "crop box" elemanı kullanılmaktadır.



Şekil 2.1

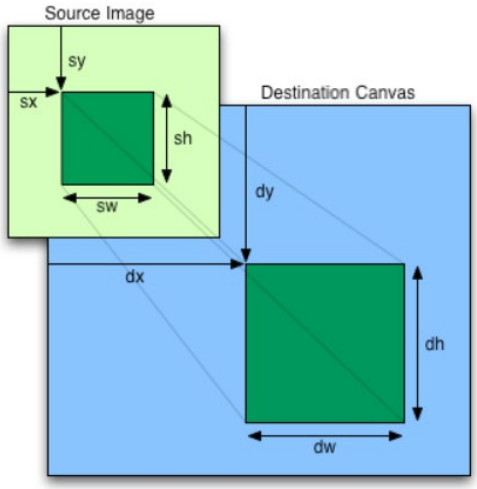
Daha önce görsel yükleme kısmında da bahsedildiği üzere, tüm kırpma işlemleri bir fonksiyon içerisinde gerçekleştirilmektedir. Görsel, hangi yöntemle yüklenirse yüklensin en son bu fonksiyon çağrılmaktadır. Bu fonksiyon içerisinde öncelikle Cropper nesnesinin tutulacağı bir değişken oluşturulmuştur. Her ne kadar bu fonksiyon modal açıldıktan sonra çalıştırılsa da, fonksiyon içerisinde modal değişkeni üzerinden on("shown-bs-modal") eventi çalıştırılıp, modal tamamen açıldıktan sonraki an yakalanmaktadır. Event içerisinde ilk olarak kullanıcının modal'ı açıp kapatması ihtimaline karşılık oluşturulan cropper değişkeni "destroy" metodu ile sıfırlanmaktadır. Sonrasında Cropper sınıfından bir nesne türetilmektedir. Cropper sınıfı içine parametre olarak bir javascript objesi halinde başlangıç değerleri girilmektedir. Buradaki javascript objesi olarak girilen değerlerde: en önemli ve zorunlu olarak kırpılacak görselin kaynağı olan ve daha öncesinde kullanıcının seçtiği görselin src attribute'üne atanan img etiketini temsil eden değişken belirtilmektedir. Sonrasında isteğe bağlı olan; kırpma kutusunun aspect ratio'su, canvasa çizilen görselin nasıl oranlanacağı("view mode"), kırpılacak alanın ön izlemesinin gösterileceği eleman, kırpma kutusunun minimum genişlik ve yüksekliği gibi pek çok özellik belirtilmektedir. Oyun alanı 4x4 parça ile kare şeklinde olacağı için kırpma kutusu da 1:1 oranında ayarlanarak kullanıcının kırpacağı görselin her türlü kare şeklinde olması sağlanmaktadır. Bu aşamada kırpma işlemi yapılmadan modal kapatılırsa yine modal nesnesi üzerinden bu sefer on("hidden-bs-modal") eventi çalıştırılıp, o aşamaya kadarki cropper ve modal nesnesi başlangıçtaki değerlerine sıfırlanmaktadır. Bu sayede birden fazla kırpma kutusunun açılması veya modal'ın hiç açılmaması gibi hataların önüne geçilmektedir. Kullanıcı kırpmak istediği alanı belirledikten sonra modal içerisindeki "kırp butonuna" tıklamasıyla bir event'in içerisine daha girilmektedir. Kırp butonuna tıklandığında, oluşturulan cropper nesnesi üzerinden "**getCropperCanvas**" metodu çalıştırılarak kırpılan görsel elde edilmekte ve bu metoda zincirleme "**toDataUrl**" metodu çalıştırılarak, daha önceden globalde oluşturulan bir img etiketinin atandığı "croppedImage" adındaki bir değişken ile img etiketinin src attribute'üne, kırpılan görselin url'i verilmektedir.

İ. Yüklenen Görselin 16 Parçaya Bölünmesi

CroppedImage değişkeninin src'sine, kırpılan görselin yüklenmesinin tamamlanma anını yakalamak için bir "onload" event'i oluşturulmuştur. Event içerisinde ilk olarak ilerde kullanılmak üzere "sx(source x)" ve "sy(source y)" adında kırpma işlemi yapılırken, kırpılacak görselin x-y koordinatlarında nereden itibaren kırpılmaya başlanacağını tutan, başlangıç değerleri sıfır olan iki değişken tanımlanmıştır. Kırpılan görselin çözünürlüğü/boyutu, kaliteli olması için herhangi bir şekilde düşürülmemiştir. Bu yüzden kullanıcının yüklediği ve kırptığı görselin çözünürlüğünün ne olduğunu ayrıyeten bulmak gerekmektedir. Bunu bulabilmek için kırpılan değişkenin tutulduğu croppedImage değişkeni üzerinden "naturalWidth" property'si kullanılmıştır. Kırpılan görsel 4x4 parçaya ayrılacağı için, kırpılacak 16 parçanın her biri kare şeklinde olmaktadır. Dolayısıyla tek bir parçanın, bir kenarının kaç piksel olduğunu bulmak yeterli olmaktadır. CroppedImage'ın naturalWidth property'si 4'e bölünerek, "**increaseDensity**" adında bir değişkene atanmıştır. 16 parçaya bölme işlemi, bir promise içerisinde gerçekleşmektedir. Javascript, asenkron çalışma prensibine sahiptir. Bir iş bir başka işin bitmesini beklemeden başlatılmaktadır. Bu özellik, normalde bir avantaj olsa da bu projede kırpma aşamasında da olduğu gibi senkron çalışması gereken durumlarda düzgün çalışılmasını engellemektedir. Senkron çalışma yapabilmek için yani belirtilen herhangi bir iş bitmeden diğer bir işe geçilmemesi istenildiğinde javascript'te birkaç farklı çözüm bulunmaktadır. Çözümler: Callback fonksiyonu, promise yapısı, async-wait yapısı, fetch kullanımıdır. Özünde çoğu yöntem, promise yapısına dayanmaktadır. Kırpma işleminde ve projenin diğer kısımlarında, **promise** yapısı ile senkron çalışma sağlanmıştır. Öncelikle bir Promise nesnesi oluşturulmuştur. Promise içerisine bir callback fonksiyonu yazılmıştır. Callback fonksiyonuna birincisi başarılı, ikincisi başarısız sonuç döndürüleceği zaman çalıştırılan resolve ve reject değişkenleri parametre olarak girilmiştir. Callback fonksiyonu içerisinde 16 parçaya kırpılma işlemi gerçekleşmektedir. İçeride kırpma işlemi tamamen bittiği anda resolve metodu çalıştırılmaktadır. Bu sayede oluşturulan promise nesnesi üzerinden "**then**" metodu çalıştırılarak resolve yakalanmakta ve senkron çalışma sağlanmaktadır. Buradaki senkronluğa, kırpma işlemleri bitmeden oyunun başlamaması için ihtiyaç bulunmaktadır. Çünkü ileride değinileceği üzere promise nesnesinin then

metodu içerisinde oyunun başlatılma işlemleri gerçekleşmektedir.
Promise sınıfının içerisinde çalıştırılan callback fonksiyonunda, 16 parça kırpılma işlemi için 16 tur dönecek bir for döngüsü oluşturulmuştur.
For döngüsü içinde 16 da bir parçanın atanacağı bir img etiketi oluşturulmaktadır. Sonrasında asıl kırpma işlemini gerçekleştirecek canvas elementi oluşturulmaktadır. Canvas elementi üzerinden "getContext('2d')" komutu ile iki boyutlu bir context oluşturulmaktadır. Bu context üzerinden ise "**drawImage**" metodu kullanılarak içine verilen parametre değerleri ile kırpma işlemi gerçekleşmektedir. DrawImage metodu içerisinde birinci parametre kırpılacak görselin kendisidir. Sonraki parametreler Şekil 2.2'deki gibidir.

```
drawImage(image,
   sx, sy, sw, sh,
   dx, dy, dw, dh);
```



Şekil 2.2

Buradaki sx ve sy değerleri daha önce bahsedilen değişkenlerin aynısıdır. Çizime, kaynak görselin x ve y koordinatlarında nereden başlanacağını belirtmekte kullanılmaktadır. Sw ve sh, kaynak görselin genişlik ve yüksekliğinin ne kadar alınacağını belirtmektedir. Dx-dy çizim yapılacak canvasın neresinden başlanacağını belirtirken dw ve dh ise kaynak görseldekinde olduğu gibi canvasta yapılacak çizimin genişlik ve yüksekliğini belirtmektedir. S harfi source, d harfi destination, w ve h ise width ve height'i ifade etmektedir. Context nesnesi üzerinden çalıştırılan drawImage metodunda sx ve sy değerlerinin, daha önce elde edilen 16'da bir parçanın genişliği kadar arttırılma ve azaltılma mantığıyla "kırpma" işlemi gerçekleştirilmektedir. Çizim yapıldıktan sonra canvas nesnesi üzerinden "toDataUrl" metodu çalıştırılıp, kırpılmış görsel for döngüsünün başında kırpılan görseli tutmak amacıyla oluşturulmuş img elemanının atandığı değişkenin src'sine tanımlanmaktadır. Sonrasında bu değiken tüm kırpılmış görsellerin tutulacağı runtime anında üretilmiş bir container elemanının içerisine "append" metoduyla eklenmektedir.

For döngüsünün sonunda daha önce bahsedilen sx ve sy değişkenlerinin değerleri bir sonraki döngü için değiştirilmektedir. Sx değişkeni her seferinde, en başta elde edilen kırpılan görselin orijinal çözünürlüğünün 16'da biri olan değerin tutulduğu "increaseDensity"nin değeri kadar arttırılmaktadır. Bu sayede kaynak görselde kırpılmaya en sol üstten başlanıp sağa doğru 4 görseli sırasıyla kırpılması sağlanmaktadır. Satırın sonuna ulaşıldığında yani sx değeri orijinal çözünürlüğün tamamına ulaştığında (4 döngü boyunca increaseDensity kadar arttırıldıktan sonra) sy değeri kaynak görselde bir alt satıra geçilmesi için increaseDensity kadar arttırılmaktadır. Sonrasında tekrardan sx değeri 4 tur daha artarak ikinci satırın tamamı kırpılmaktadır. Bu şekilde 4 satır 4 sütunun kırpılması, for döngüsünün 16 turu ile tamamlanmaktadır. Kırpılan görselin container'a atılması esnasında döngünün son adımında olduğu kontrol edilmektedir. Bu kontrol ile kırpma işleminin tamamen bitmesi yakalanarak promise yapısındaki "**resolve**" metodu çalıştırılmaktadır.
Bu aşamadan sonra kullanıcının kırptığı görsel 16 parçaya bölünmüş olup promise nesnesinin "**then**" metodu tetiklenmektedir. Kırpılan görseller oyun alanına aktarılmaya hazır hale getirilmiştir.

J. Oyun Alanının Oluşturulması

Projede, kullanıcının bir görsel yükleyip kırpma işleminin gerçekleştirilme aşamasından sonraki tüm elemanlar **runtime** anında javascript tarafından oluşturulmakta, class isimleri verilmekte, içerikleri doldurulup dokümana eklenerek yazdırılmaktadır. Oyun alanında da durum aynı şekildedir. Elemanların oluşturulup ekrana yazdırılması promise nesnesinin then metodu içerisinde gerçekleşmektedir. Hali hazırda bütün içeriğin içine yazıldığı wrapper container içerisine bir **game-section** div'i oluşturulup eklenmiştir. Bu section'a soldan sağa olmak üzere karıştırma butonu ile hamle-puan bilgisinin tutulacağı left side bar, ortada oyun alanı ve sağ tarafta right side bar olarak oyuncunun önceki istatistiğinin yer aldığı birer div eklenmiştir. Oyun alanına, diğer pek çok yerdeki hizalamalarda da kullanıldığı üzere **flexbox** özelliği verilmiştir. Bu sayede 16 kırpılmış görselin, en sol üstten başlayarak sağa doğru sıralı bir şekilde satır satır dizilmeleri sağlanmıştır. Then metodu içerisinde oyun alanı ve elemanlarının oluşturulup eklenmesi sonrasında son olarak kırpılan görsellere **tıklanıldığında** gerçekleşecek işlemlerin tanımlandığı fonksiyon ile oyun alanındaki karıştırma butonuna tıklanma anında gerçekleşecek **karıştırma** işleminin yapıldığı fonksiyon çağrılmaktadır. Bu fonksiyonların içeriğine ileriki başlıklarda değinilmektedir. Oyun alanını barındıran container'a, karıştırma işlemi yapılmadan herhangi bir görsele tıklanıp işlem yapılamaması için "game-mode-zero" isimli bir class verilmiştir. Bu class sayesinde css'in "pointer-events" property'si "none" yapılarak, görseller hiçbir şekilde tıklamalardan veya benzer event tetikleyici işlemlerden etkilenmeyecek duruma getirilmiştir. Oyun başlatıldığında bu class kaldırılmaktadır.

K. *Linked List ile Görseller Arasındaki Bağlantının ve Linked List Algoritmalarının Oluşturulması*

Oyun alanındaki görsellerin pozisyon kontrolleri, tek yönlü bir bağlı liste yapısı kullanılarak gerçekleştirilmektedir. Oyun alanındaki kırpılmış 16 görselin yerleri sadece css animasyonu kullanılarak "değiştirilmiş" gibi gösterilmektedir. 16 img etiketinin konumu hiçbir şekilde değişmemektedir. Dolayısıyla görsellerin herhangi birine tıklanıldığında görsel nerede, hangi pozisyonda görünürse görünsün tıklanan etiketin 16 etiket arasındaki sırası değişmediği için javascript ile aynı **index** değeri elde edilebilmektedir. Elde edilen index değeri, bağlı listedeki düğümlerin **data**'larına karşılık gelmektedir. Bu mantık sayesinde görseller ile bağlı liste arasında soyut bir bağ oluşturulmaktadır.

Javascript'te bağlı liste yapısı için "**LinkedList**" ve "**Node**" adında iki class oluşturulmuştur. Node class'ı içerisinde düğümün data'sını parametre olarak alan bir constructor metodu yazılmıştır. Constructor metodu içinde, parametre olarak girilen data, "**this**" keywordu ile sınıfa ait data değişkenine aktarılmaktadır. Ayrıca sonraki düğümü işaret edecek başlangıç değeri null olan "**next**" değişkeni tanımlanmıştır. LinkedList class'ında da bir constructor yazılmıştır. Parametre olarak herhangi bir şey almamaktadır. İçerisinde this keyword'leri ile sınıfa ait olacak şekilde başlangıç değerleri null olan **header** ve **tail** değişkenleri oluşturulmuştur. Header değişkeni bağlı listenin ilk düğümünü, tail ise son düğümünü tutmak için oluşturulmuştur. Ayrıca constructor içerisinde listenin boyutunu tutacak (her ne kadar sabit boyuta sahip olsa da) bir **size** değişkeni tanımlanmıştır.

LinkedList class'ı içerisinde constructor metodu yazıldıktan sonra listeyi yazdırma, başa, sona, araya düğüm ekleme-çıkarma metodları yazılmıştır. Veri alarak çalışan ekleme-çıkarma metodları, duruma göre veri alıp yeni düğüm üreterek işlemler gerçekleştirmektedir. Bu metodlardaki işlemlerin, yeni düğüm değil, daha önce var olan bir düğümün kendisiyle gerçekleştirildiği versiyonları da yazılmıştır. Klasik ekle-çıkar metodların dışında, ayrıca yer değiştirme, data'dan pozisyon bulma ve pozisyon doğruluğunu bulma işlemlerini yapan metodlar yazılmıştır.

Oyun boyunca pozisyon kontrolleri gibi işlemlerin yürütüleceği bağlı liste, javascript kodları arasında global kısımda oluşturulmaktadır. İçerisine, for döngüsü ile 0-15 datalara sahip 16 düğüm eklenmektedir.

L. *Kırpılan Görsellerin Yer Değiştirme Animasyonunun Oluşturulması*

Oyun alanındaki 16 görsel, birbirleri ile ikili şekilde yer değiştirme yapabilmektedir. Esasında yer değiştirilmiş gibi gösterilmektedir. Html etiketlerinin kendisinde konum/sıra olarak herhangi bir değişiklik olmamaktadır. 16 parçanın her birinin, diğer herhangi bir pozisyona gitmesi için yapılması gereken hareket, css'te 16x16 ihtimal için class'lar yazılarak gerçekleştirilmiştir. Bu işlemde, css'te bulunan "**transform"** ve "**transition"** property'leri öncü rol almaktadır. Transform property'sine "**translate**" fonksiyonu ile farklı değerler verilmektedir. Translate, içerisine x ve y ekseninde kaç piksellik hareket yapılacağını belirten iki parametre almaktadır. Oyun alanındaki görsellerin genişlikleri 150x150 px olmak üzere, örneğin bir görsel bulunduğu pozisyonun bir sağına kaydırılmak istendiğinde translate fonksiyonuna x ekseninde 150px, y ekseninde 0px değerleri girilerek, konumu değiştirilmektedir. Transition property'si ise buradaki hareketin bir anda gerçekleşmeyip belli bir süre ve zamanlama fonksiyonu içerisinde yumuşak bir şekilde gerçekleşmesini sağlamaktadır.

Bütün hareket ihtimalleri için, 16x16 adet class yazmak gerekmektedir. İşte tam da burada Scss'ten faydalanılmıştır. Scss'teki değişkenler, fonksiyon yapıları ve for döngüleri sayesinde bir algoritma yazılmıştır. İç içe iki for döngüsü kullanılarak 16x16 ihtimal için gerekli tüm class'lar yazdırılmıştır. İlgili hareketi yapacak class için, örneğin üçüncü pozisyondan yedinci pozisyona gitme hareketi için "three-to-seven" olacak şekilde bir isimlendirme yapılmıştır. Bu şekilde class'lar, kırpılmış görsellere verildiğinde istenilen konum değişikliği yapılmaktadır.

M. *Kırpılan Görsellerin Karıştırılma Algoritmasının Oluşturulması*

Karıştırma algoritması ve işlemleri için ayrı bir fonksiyon yazılmıştır. Bu fonksiyon, oyun alanındaki karıştırma butonuna tıklanıldığında çalıştırılmaktadır. Algoritmanın mantığı, defalarca rastgele ikili görselin seçilip yer değiştirme fonksiyonunda çalıştırılmalarına dayanmaktadır. Karıştırma fonksiyonu içerisinde öncelikle, 0'dan 15'e kadar, 16 elemanlı bir liste oluşturulmuştur. Liste, bir dizi karıştırma algoritması kullanılarak rastgele karıştırılmıştır. Karıştırılan liste üzerinden bir foreach döngüsünde sıra ile elemanları kullanılmaktadır. Karıştırılan dizinin elemanları, ikili yer değiştirme yapılacak görsellerden birincisinin datasını oluşturmaktadır. Rastgele yer değiştirme yapılacak ikinci düğümün datası ise javascript'in math sınıfının random metodunun 0-16 arası rastgele değer döndürmesi ile elde edilmektedir. Elde edilen iki rastgele düğüm datası, kırpılmış görsellerin yer değiştirme işlemlerinin yapıldığı fonksiyona parametre olarak girilmektedir. Bu fonksiyona ilgili başlıkta değinilecektir.

N. *Oyunun Başlatılması, Puanın ve Hamle Sayısının Eklenmesi*

Oyun, daha önceki başlıkta değinilen karıştırma işlemi sonrasında en az bir parçanın doğru yere eşleşmesini kontrol ederek başlatılmaktadır. Karıştırma işlemi, bir promise yapısının içerisinde gerçekleştirilmektedir. Buradaki promise'in amacı, karıştırma algoritmasının tamamen bitmesini yakalamaktır. Karıştırma işleminde, her yer değişikliğinde işlemleri yapan fonksiyon içerisinde, görsellerin doğru pozisyonda olup olmadığının kontrolü yapılmakta ve buna bağlı olarak eğer bir parça doğru yere eşleşmiş ise "position-correct" class'ı ilgili görselin tutulduğu img etiketine verilmektedir. Karıştırma fonksiyonunda yer alan promise nesnesinin then metodu içerisinde, kaç adet "position-correct" class'ına sahip img etiketinin olduğu jquery ile sorgulanmaktadır. Eğer buradan elde edilen sayı adedi bir veya daha büyük ise oyunun başlatılma işlemleri gerçekleştirilmektedir. Oyunun başlaması için, karıştırma butonu ilgili class verilerek kilitlenmekte, oyun alanına daha önce verilmiş kilitleme class'ı kaldırılarak görsellere tıklama işlemleri açık hale

getirilmektedir. Beraberinde yine runtime anında üretilen oyun boyunca kullanıcının hamlesini ve puanını gösterecek elemanlar oyun alanındaki left side bar içerisine, başlangıç değerleri 0 olmak üzere yazdırılmaktadır.

O. Kırpılan Görsellere Tıklama Event'i Eklenmesi ve Yer Değiştirme İşleminin Gerçekleştirilmesi

Kırpılan görsellere tıklanılma eventinde, gerçekleşecek işlemlerin yapıldığı fonksiyon, oyun alanı ilk oluşturulduğunda çağrılmakta ve tanımlanmaktadır. Bu fonksiyonun içerisinde tıklanan görsel için jquery'nin "**index**" metodu çalıştırılmakta ve bağlı listedeki düğümlere karşılık gelen data bilgisi elde edilmektedir. Kullanıcının her tıklamasındaki kaçıncı kez tıkladığı bilgisi, başlangıç değeri 0 olan global bir "clickCount" adındaki değişken ile tutulmaktadır. Fonksiyon her kırpılmış görsele tıklama anında tetiklenerek değişkenin değerini bir arttırmaktadır. Bu değişken bir **switch-case** yapısı içerisinde sorgulanmaktadır. Tıklama adedi 1 ise ilgili case çalışarak, kullanıcının ilk tıkladığı görselin datası kaydedilmektedir. Kaydedilen bilgi, yer değiştirme yapılacak görsellerden ilkini ifade eden düğümün datasıdır. Benzer şekilde kullanıcının ikinci tıklaması da yakalanıp kaydedilmektedir. İkinci tıklama anında kullanıcının iki kez aynı görsele tıklayıp tıklamadığı kontrolü de yapılmaktadır. Eğer tıklanmış ise, dataların tutulduğu değişkenler ve tıklama adedini tutan değişken, başlangıç değerlerine sıfırlanmaktadır. İkinci tıklama case'inde tıklanan iki görsel aynı değilse, yer değiştirme işlemlerini yapan fonksiyona, iki düğümün datası parametre olarak girilerek çalıştırılmaktadır.

Yer değiştirme işlemleri için, tıklama event'inde olduğu gibi ayrı bir fonksiyon oluşturulmuştur. Bu fonksiyon içerisinde ilk olarak daha önce oluşturulmuş bağlı liste ve metodları kullanılarak, liste nesnesi üzerinden "replace" metodu içerisine kullanıcının seçtiği görsellere göre gelen data0 ve data1 parametreleri verilerek çalıştırılmaktadır. Liste içerisindeki düğümlerin yerleri bu metod ile değiştirilmektedir. Yer değiştirme fonksiyonu içerisinde, listede yer değiştirme yapıldıktan sonra ilgili görsellerin yer değiştirme animasyonunu yapmalarını sağlayacak class'ların isimleri oluşturulmakta ve görsellere eklenmektedir. Liste üzerinden bir metod ile yer değiştirme sonrasında, görsellerin doğru yerde olup olmadığı bilgisi sorgulanıp, boolean değişkenlere kaydedilmektedir. Oyunun başlayıp başlamadığını ifade eden oyun alanındaki "game-mode-zero" class'ı sorgulanarak oyun eğer başlamış ise yeni pozisyonlarının doğruluğuna bakılmaktadır. Bu sorgulama içerisinde eğer iki görsel de yanlış yerdeyse oyuncunun puan bilgisini tutan değişkenin değeri 10 azaltılıp ekrana yazdırılmaktadır. Yanlış konumdaki görsellere aynı zamanda bir css animasyonu uygulamak için "position-false" class'ı eklenmektedir. Aynı zamanda görsellerden biri bile doğru ya da her ikisi de yanlış yerdeyse oyun alanının border'ında bir animasyon oluşturmak üzere yine ayrıca class'lar eklenmektedir. Tüm bu bahsi geçen işlemler yer değiştirme fonksiyonu içerisinde oyun başladığında yapılmaktadır. Her yer değiştirme esnasında bütün işlemler tekrarlanmaktadır. Fonksiyonun devamında son olarak, pozisyonları doğru olan görsellere oyun başlamadan önce ve sonrası, iki durum için de ilgili class'ların verilebilmesi gibi işlemler için ayrıca bir fonksiyon daha çalıştırılmaktadır.

Pozisyonları doğru olan görseller için çalıştırılan fonksiyon içerisinde, öncelikle görsele "position-correct" class'ı verilmektedir. Position-false class'ı sadece animasyon içinken, position-correct class'ı animasyonun dışında görseli kilitlemeyi de sağlamaktadır. Çünkü bu class'ı alan görsel artık doğru yerdedir ve tıklanılamamalıdır.

P. Oyunun Bitirilmesi

Oyunun bitişi, önceki başlıkta bahsedilen, yeni pozisyonun doğruluğu durumunda çalışan fonksiyonun devamında yakalanmaktadır. Bu işlem kırpılmış görsellere verilen "position-correct" class'larının adedi sorgulanarak yapılmaktadır. Class'ların adedi 16 ise bütün görseller doğru yerdedir ve oyun bitmiş demektir. Devamında, ekrana yeni bir absolute eleman ile oyun alanının üzeri kapatılmakta ve oyun bitiş ekranı gelmektedir. Burada da, projenin çoğunda olduğu gibi yine bütün html elemanları, class'ları, içerikleri, runtime anında oluşturulup basılmaktadır. Oyun bitiş ekranında kullanıcının yüklediği ve oyunu oynadığı görselin tamamı, kullanıcı adı, hamle sayısı, puanı gösterilmektedir. Ayrıca oyuna tekrar başlayabilmesi için sayfayı, location nesnesinin "reload" metoduyla yenilemeyi sağlayan bir buton eklenmiştir.

Q. Oyun Sonunda Oyuncu Bilgilerinin Metin Belgesine Yazdırılması, Oyuncuların İstatistiklerinin Ekrana Yazdırılması

Oyunun bitişinin yakalanması ile gerçekleşen işlemlerin sonuncusu olarak, metin belgesi işlemlerini tetikleyecek, ajax isteği yazılmıştır. Kullanıcının ad, hamle sayısı ve puan bilgilerini tutan bir javascript objesi oluşturulmuştur. Jquery'nin ajax fonksiyonu ile projenin hali hazırda çalıştığı url olan "square_puzzle_game" url'ine kullanıcı bilgileri, post metodu ile gönderilmiştir. App.js içerisinde, Express.js nesnesi üzerinden çalıştırılan route metoduna zincirleme bağlı get metoduyla index.ejs render edilmekteydi. Get metodunun ardından yine zincirleme şekilde post metodu yazılarak, gelen istek callback fonksiyonu içerisinde yönetilmiştir. Callback fonksiyonu içerisinde request ve response değişkenleri parametre olarak girilmektedir. Ajax ile gönderilen kullanıcı bilgileri, request değişkeni üzerinden body property'si üzerinden dot notation ile erişilmektedir. Elde edilen kullanıcı bilgileriyle daha önce nasıl çalıştığı anlatılan FileSystem modülü kullanılmaktadır. Öncelikle oyuncuların en yüksek skorlarının tutulduğu "highestscore.txt" metin belgesi okunmaktadır. Okuma işlemi ardından çalıştırılan callback fonksiyonu içerisinde metin belgesinden okunan değer bir json dizisi halinde olduğu için parse edilerek javascript obje dizisine dönüştürülmektedir. Bütün değişiklikler bu javascript obje dizisi üzerinden yapılıp metin belgesine json dizisi formatına dönüştürülüp sıfırdan yazdırılmaktadır. İlk olarak elde edilen dizinin undefined olup olmadığı kontrol edilmektedir. Eğer undefined ise metin belgesinde henüz hiçbir kayıt bulunmamaktadır. Dolayısıyla oyunu bitiren kullanıcının bilgileri doğrudan filesystem nesnesinin

metoduyla yazdırılmaktadır. Metin belgesine kullanıcı bilgileri json formatına çevrilerek yazılmaktadır. Okunacağı zaman da tekrardan javascript objesine parse edilmektedir. Okunan değer eğer undefined değilse, bu sefer oyunu bitiren kullanıcının daha önce bir kaydının olup olmadığı sorgulanmaktadır. Kaydı yoksa direkt yeni kayıt eklenmektedir. Kaydı varsa, en son oynadığı oyunun puanı, kayıtlı puanından yüksek olup olmadığı kontrol edilmektedir. Yüksek değilse rekor kırılamamıştır ve metin belgesine herhangi bir kayıt eklenmemektedir. En son oynadığı oyunun puanı daha yüksek ise metin belgesinden elde edilip dönüştürülen object dizi içerisinde kullanıcının bilgileri güncellenmektedir. Kullanıcı bilgilerinin bulunduğu dizi oyun puanına göre büyükten küçüğe sıralandıktan sonra tekrardan json dizisi haline dönüştürülüp metin belgesine sıfırdan yazdırılmaktadır.

Oyuncuların bilgileri çok benzer şekilde, sitede ekranın kenarlarına yazdırılmak üzere metin belgesinden çekilmektedir. Metin belgesinde oyuncular puanlarına göre büyükten küçüğe göre sıralanmaktadır. Böylelikle en yüksek puan alan 3 kullanıcıyı listelemek için, metin belgesinden çekilen dizinin ilk üç değerini kırpmak yetmektedir. Ayrıyeten siteye giren oyuncunun kullanıcı adına göre, eğer varsa onun da bilgileri çekilip ilgili html elementlerine bastırılmaktadır. Bu işlem için de, oyun sonunda oyuncu bilgilerini yazdırmakta kullanılan ajax yöntemi kullanılmaktadır.

R. Arka Plan Animasyonunun Oluşturulması

Projenin tamamı tek bir “index.ejs” dosyası içerisinde gerçekleştirilmektedir. Burada daha önce de bahsedilen wrapper div’i de site boyunca sabit kalmaktadır. Animasyona geçmeden önce sitenin arkaplanı rengi, linear gradient ile geçişli olacak şekilde ayarlanmıştır. Sitede, kullanıcı adı girme kısmından oyunun sonuna kadar sürekli arka planda bir animasyon çalıştırılmaktadır. Bu animasyon için wrapper div’i içerisinde bağımsız absolute “squares” class’ına sahip bir unordered list (ul) elemanı oluşturulmuştur. İçerisine ise 12 adet list item (li) eklenmiştir. Squares class’ına sahip eleman absolute yapılarak, genişlik ve yükseklik değerleri sayfanın tamamına eşit verilmiştir. İçerideki list item’lara da absolute özelliği verildikten sonra bir animasyon oluşturulmuştur. Bu animasyon, list item’ların aşağıdan yukarıya doğru çıkmalarını sağlayıp aynı anda saydamlık değerlerini azaltmaktadır. List item’ların her birine “nth-child” seçicisiyle erişip genişik ve yükseklik değerleri verilip kare kutular oluşturulmuştur. Animasyon gerçekleşirken rastgele konumlarda olabilmeleri için sola olan uzaklıkları verildiği gibi ayrıca yine her birine animasyon gecikmesi de verilmiştir. Bu sayede bazı elemanlar birbirlerinden farklı zamanlarda ilgili animasyonu gerçekleştirmeye başlayacakları için, kare kutuların farklı konumlarda olmaları sağlanmaktadır.

IV. KAYNAKLAR

[1]https://www.youtube.com/watch?v=CPcpcqaQRvw&list=PLY20HpFruiK3ZCfdEtpUHNhNWn61VGEns

[2]https://www.youtube.com/watch?v=CPcpcqaQRvw&list=PLY20HpFruiK12kqke7T5OQVu1BK2ELQL8

[3]https://www.youtube.com/watch?v=gWBG3ZL3-1g&list=PLfAfrKyDRWrEPrXqp-RA4xG5QWojNip9a

[4] https://www.youtube.com/watch?v=0hZNdTogNo0

[5] https://www.youtube.com/watch?v=SX8bHqXt8ws

[6] https://www.youtube.com/watch?v=ihoPT325jyk

[7]https://stackoverflow.com/questions/26015497/how-to-resize-then-crop-an-image-with-canvas

[8] https://github.com/fengyuanchen/cropperjs

[9] https://code.visualstudio.com/

[10] https://nodejs.org/en

[11] https://git-scm.com/

[12] https://fonts.google.com/

[13] https://stackoverflow.com/

[14] https://www.w3schools.com/

[15] https://tailwindcolor.com/

[16] https://getcssscan.com/css-box-shadow-examples

[17] https://getbootstrap.com/docs/5.2/

[18] https://stackedit.io/

[19] https://medium.com/

[20] https://fontawesome.com/

[21] https://www.cyclic.sh/