**Django ile WhiteNoise kullanma**

**Not**

Django olmayan bir uygulama ile WhiteNoise kullanmak için [genel WSGI belgelerine](http://whitenoise.evans.io/en/stable/base.html)bakın.

Bu kılavuz, WhiteNoise ile bir Django projesi kurma boyunca size yol gösterin. Çoğu durumda yapılandırma satırbir çift daha fazla almamalıdır.

Ben WhiteNoise oluşturmak için beni istenir ilk kullanım durumda olduğu gibi birkaç yerde Heroku söz, ama hiçbir şey Heroku-WhiteNoise hakkında özel ve aşağıdaki talimatlar ne olursa olsun barındırma platformu uygulamak gerekir.

**1. *Statik dosyaların* doğru yapılandırıldığından emin olun**

Django'ya aşinaysan ne yapacağını biliyorsun. Yeni bir Django projesiyle yeni başlıyorsanız, dosyanızın altına aşağıdakileri eklemeniz gerekir:settings.py

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

Uygulamanızı dağıtmanın bir parçası olarak, tüm statik dosyalarınızı . (Heroku üzerinde çalışıyorsanız, bu sizin için otomatik olarak yapılır.)./manage.py collectstaticSTATIC\_ROOT

Statik dosyalarınıza başvurmak için [statik](https://docs.djangoproject.com/en/stable/ref/templates/builtins/#std:templatetag-static) şablon etiketini kullandığınızdan emin olun, bunun yerine URL'yi doğrudan yazmak. Örneğin:

{% **load** static %}

<img src="{% **static** "images/hi.jpg" %}" alt="Hi!" />

<!-- DON'T WRITE THIS -->

<img src="/static/images/hi.jpg" alt="Hi!" />

Daha fazla bilgi için Django [staticfiles](https://docs.djangoproject.com/en/stable/howto/static-files/) kılavuzuna bakın.

**2. WhiteNoise etkinleştirin**

Dosyanızı edin ve listeye WhiteNoise ekleyin. WhiteNoise middleware doğrudan Django [SecurityMiddleware](https://docs.djangoproject.com/en/stable/ref/middleware/#module-django.middleware.security) sonra (bunu kullanıyorsanız) ve diğer tüm middleware önce yerleştirilmelidir:settings.pyMIDDLEWARE

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'whitenoise.middleware.WhiteNoiseMiddleware',

*# ...*

]

Bu kadar - WhiteNoise şimdi statik dosyaları [(aşağıdaki adımları](http://whitenoise.evans.io/en/stable/django.html#check-its-working)kullanarak çalıştığını teyit edebilirsiniz) hizmet verecek. Ancak, en iyi performansı elde etmek için aşağıdaki adım 3'e devam etmeli ve sıkıştırma ve önbelleğe alma nızı etkinleştirmelisiniz.

**Not**

Ara yazılım listesinin en üstünde en yüksek önceliğe sahip olması gerektiğini öneren diğer üçüncü taraf ara yazılımları bulabilirsiniz. Tam olarak ne olduğunu anlamak sürece bu tavsiye göz ardı etmeli ve her zaman diğer middleware yukarıda yer. WhiteNoise'dan önce başka ara yazılımlar çalıştırmayı planlıyorsanız, Django'daki [request\_finished hatadan](https://code.djangoproject.com/ticket/29069) haberdar olmalısınız.WhiteNoiseMiddleware

**3. Sıkıştırma ve önbelleğe alma desteği ekleme**

WhiteNoise, dosyalarınızı otomatik olarak sıkıştırmayı ve her sürüm için benzersiz adlar oluşturmayı, böylece sonsuza kadar güvenli bir şekilde önbelleğe alınabilen bir depolama arka ucuyla birlikte gelir. Kullanmak için, sadece bu ekleyin:settings.py

STATICFILES\_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'

Bu django's [ManifestStaticFilesStorage](https://docs.djangoproject.com/en/stable/ref/contrib/staticfiles/#manifeststaticfilesstorage) arka uç tarafından sağlanan önbelleğe alma davranışı ile otomatik sıkıştırma birleştirir. Sıkıştırma uygulamak istiyorsanız ancak önbelleğe alma davranışını istemiyorsanız şunları kullanabilirsiniz:

STATICFILES\_STORAGE = 'whitenoise.storage.CompressedStaticFilesStorage'

**Not**

WhiteNoise depolama arka ucuna geçtikten sonra sorun yaşıyorsanız lütfen [sorun giderme kılavuzuna](http://whitenoise.evans.io/en/stable/django.html#storage-troubleshoot)bakın.

Statik dosya depolama sisteminin dışında dosyaları sıkıştırmanız gerekiyorsa, verilen [komut satırı yardımcı programını](http://whitenoise.evans.io/en/stable/base.html#cli-utility)kullanabilirsiniz

**Brotli sıkıştırma**

Ortak gzip sıkıştırma biçiminin yanı sıra WhiteNoise, daha yeni ve daha verimli [brotli](https://en.wikipedia.org/wiki/Brotli) biçimini destekler. Bu, bant genişliğini azaltmaya ve yükleme hızını artırmaya yardımcı olur. Brotli sıkıştırmayı etkinleştirmek için [Brotli Python paketinin](https://pypi.org/project/Brotli/) çalıştırılarak yüklenmesi gerekir.pip install whitenoise[brotli]

Brotli [tüm büyük tarayıcılar](https://caniuse.com/#feat=brotli) tarafından desteklenir (IE11 hariç). WhiteNoise brotli desteği sağlayan hiçbir uyumluluk sorunları vardır bu yüzden sadece bunu talep tarayıcılara brotli veri hizmet verecektir.

Ayrıca, tarayıcıların brotli verilerini yalnızca BIR HTTPS bağlantısı üzerinden isteyeceğini de unutmayın.

**4. İçerik Dağıtım Ağı Kullanma**

Yukarıdaki adımlar orta trafik sitelerinde iyi performans alacak, ancak yüksek trafik siteleri için, ya da performans bir CDN kullanarak bakmak gerekir bir endişe olduğu siteler.

WhiteNoise statik içeriğinizle uygun önbellek üstbilgileri gönderdiğinden, CDN dosyalarınızı önbelleğe alıp uygulamanızla yeniden iletişim kurmanıza gerek kalmadan bunları sunabilecektir.

Aşağıda Amazon CloudFront, CDN popüler bir seçim ile WhiteNoise kurmak için talimat vardır. Diğer CDN'ler için işlem olsa çok benzer görünmelidir.

**Amazon CloudFront için talimatlar**

AWS Web Konsolu'nun CloudFront bölümüne gidin ve "Dağıtım Oluştur"u tıklatın. Uygulamanızın etki alanını (http öneki olmadan) "Başlangıç Etki Alanı Adı" alanına koyun ve geri kalan ayarları olduğu gibi bırakın.

Dağıtımınızın etkinleşmesi birkaç dakika sürebilir. Hazır olduğunda, dağıtım alan adını dosyanıza kopyalayın, böylece aşağıdaki gibi görünür:settings.py

STATIC\_HOST = 'https://d4663kmspf1sqa.cloudfront.net' **if** **not** DEBUG **else** ''

STATIC\_URL = STATIC\_HOST + '/static/'

Ya da daha da iyisi, böyle bir şey yaparak CDN'nizi ayarlarınıza kodlamaktan kaçınabilirsiniz:

STATIC\_HOST = os.environ.get('DJANGO\_STATIC\_HOST', '')

STATIC\_URL = STATIC\_HOST + '/static/'

Bu şekilde CDN'nizi yalnızca bir ortam değişkeni ayarlayarak yapılandırabilirsiniz. Heroku'daki uygulamalar için bu komutu çalıştırabilirsiniz

heroku config:set DJANGO\_STATIC\_HOST=https://d4663kmspf1sqa.cloudfront.net

**gzip dışındaki sıkıştırma algoritmalarını kullanma**

Varsayılan olarak CloudFront, üstbilginin değeri gzip olmadığı sürece, isteklere dahil edilen üstbilgi tarayıcılarını atar. Gzip ise, CloudFront sıkıştırılmamış dosyayı kaynaktan getirir, sıkıştırır ve isteyen tarayıcıya geri döndürecektir.Accept-Encoding

CloudFront'un sıkıştırmanın kendisini yapmaması ve Brotli gibi diğer algoritmalar kullanılarak sıkıştırılmış dosyalara hizmet etmesini sağlamak için, dağıtımınızı [Kabul-Kodlama üstbilgisini temel alan önbelleğe](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ServingCompressedFiles.html#compressed-content-custom-origin)göre yapılandırmanız gerekir. Bunu dağıtımınızın sekmesinde yapabilirsiniz.Behaviours

**Not**

Varsayılan olarak tüm sitenize CloudFront URL üzerinden erişileceksiniz. Bu URL'ler arama sonuçlarında görünmeye başlarsa, bu SEO sorunlarına neden olabilir. CloudFront'u yalnızca bu yönergeleri izleyerek statik dosyalarınızı proxy [etmesiyle](http://whitenoise.evans.io/en/stable/django.html#restricting-cloudfront)kısıtlayabilirsiniz.

**5. Geliştirme de WhiteNoise kullanma**

Geliştirme django otomatik olarak statik dosya işleme devralır. Çoğu durumda bu durum iyidir, ancak bu, WhiteNoise'un statik dosya işlemeye yaptığı bazı geliştirmelerin geliştirmede kullanılamayacağı ve geliştirme ve üretim ortamları arasındaki davranış farklılıkları olasılığını ortaya çıkaramayacağı anlamına gelir. Bu nedenle whitenoise geliştirme de kullanmak iyi bir fikirdir.runserver

Django'nun statik dosya işlemesini devre dışı bırakabilirsiniz ve WhiteNoise'un yalnızca komuta geçerek devralmasına izin verebilirsiniz, ancak her aradiğinizde bu seçeneği eklemeyi unutmamalısınız. Daha kolay bir yolu dosyanızı ve listenizin üst eklemek tir:--nostaticrunserverrunserversettings.pywhitenoise.runserver\_nostaticINSTALLED\_APPS

INSTALLED\_APPS = [

'whitenoise.runserver\_nostatic',

'django.contrib.staticfiles',

*# ...*

]

**Not**

WhiteNoise'un eski sürümlerinde (v4.0'ın altında) Kanallar runserver'ın kendi uygulamasını sağladığından [Kanallar'da](https://channels.readthedocs.io/) kullanmak mümkün değildi. WhiteNoise'un yeni sürümlerinde bu sorun yoktur ve kanallar veya runserver'ın kendi uygulamasını sağlayan diğer üçüncü taraf uygulamalarla birlikte çalışır.runserver\_nostatic

**6. Dizin Dosyaları**

[**WHITENOISE\_INDEX\_FILE**](http://whitenoise.evans.io/en/stable/django.html#WHITENOISE_INDEX_FILE)seçeneği etkinleştirildiğinde:

* Ziyaret, dosyaya /example//example/index.html
* Ziyaret (302) yönlendirmek olacaktır /example/example/
* Ziyaret (302) yönlendirmek olacaktır /example/index.html/example/

Dizin dosyası dışında bir şey istiyorsanız, bu seçeneği alternatif bir dosya adına da ayarlayabilirsiniz.index.html

**Kullanılabilir Ayarlar**

WhiteNoiseMiddleware sınıfı, WhiteNoise taban sınıfıyla aynı yapılandırma seçeneklerini alır, ancak anahtar kelime bağımsız değişkenlerini oluşturucuya kabul etmek yerine Django ayarlarını kullanır. Ayar adları, 'WHITENOISE\_' önekiyle birlikte büyük harfle ifade edilen anahtar kelime bağımsız değişkenleridir.

**WHITENOISE\_ROOT**

**Varsayılan**

**None**

Uygulamanızın kökünde sunulacak dosyaların dizinine mutlak yol (ayarlanmazsa göz ardı edilir).

Önbellek sürümünden yararlanamayacağınız için bunu statik dosyalarınızın büyük bir kısmı için kullanmayın, ancak belirli bir URL'de hizmet vermek istediğiniz dosyalar gibi veya sizin için uygun olabilir.**robots.txtfavicon.ico**

**WHITENOISE\_AUTOREFRESH**

**Varsayılan**

**settings.DEBUG**

Yanıt vermeden önce dosyaların değişip değişmediğini görmek için dosya sistemini yeniden denetleyin. Bu, sunucuyu yeniden başlatmadan statik dosyalardaki değişiklikleri almak için uygun olabilecek geliştirme de kullanılmak üzere tasarlanmıştır. Hem performans hem de güvenlik nedenleriyle bu ayar üretimde kullanılmamalıdır.

**WHITENOISE\_USE\_FINDERS**

**Varsayılan**

**settings.DEBUG**

Django'nun "bulucuları" API'sini kullanarak yalnızca toplanan dosyaları toplamak yerine, orijinal dizinlerindeki dosyaları bulun ve servis edin. Bu, eski komutun davranışlarıyla eşleştiği geliştirmede yararlıdır. Depolama arka uçları tarafından sağlanan önbelleğe alma ve sıkıştırma özelliklerinden herhangi birini kullanmak istemediğiniz sürece, yapı sırasında komutu çalıştırma gereksinimini önleyerek bu ayarı üretimde de kullanmak mümkündür.**STATIC\_ROOTrunservercollectstatic**

**WHITENOISE\_MAX\_AGE**

**Varsayılan**

**60 if not settings.DEBUG else 0**

Tarayıcıların ve yakınlıkların **sürümdışı** dosyaları önbelleğe alması gereken süre (saniye cinsinden).

Sürümlenmiş dosyalar (yani *base.a4ef2389gibi* benzersiz bir ad verilmiş dosyalar.css kendi içeriğinin bir karma adı dahil edilerek) otomatik olarak algılanır ve sonsuza kadar önbelleğe alınacak şekilde ayarlanır.

The default is chosen to be short enough not to cause problems with stale versions but long enough that, if you’re running WhiteNoise behind a CDN, the CDN will still take the majority of the strain during times of heavy load.

**WHITENOISE\_INDEX\_FILE**

**Default**

**False**

If enable [index file serving](http://whitenoise.evans.io/en/stable/django.html#index-files-django). If set to a non-empty string, enable index files and use that string as the index file name.**True**

**WHITENOISE\_MIMETYPES**

**Default**

**None**

A dictionary mapping file extensions (lowercase) to the mimetype for that extension. For example:

{'.foo': 'application/x-foo'}

Note that WhiteNoise ships with its own default set of mimetypes and does not use the system-supplied ones (e.g. ). This ensures that it behaves consistently regardless of the environment in which it’s run. View the defaults in the [media\_types.py](https://github.com/evansd/whitenoise/blob/v5.2.0/whitenoise/media_types.py) file.**/etc/mime.types**

In addition to file extensions, mimetypes can be specified by supplying the entire filename, for example:

{'some-special-file': 'application/x-custom-type'}

**WHITENOISE\_CHARSET**

**Default**

**'utf-8'**

Charset to add as part of the header for all files whose mimetype allows a charset.**Content-Type**

**WHITENOISE\_ALLOW\_ALL\_ORIGINS**

**Default**

**True**

Toggles whether to send an header for all static files.**Access-Control-Allow-Origin: \***

This allows cross-origin requests for static files which means your static files will continue to work as expected even if they are served via a CDN and therefore on a different domain. Without this your static files will *mostly* work, but you may have problems with fonts loading in Firefox, or accessing images in canvas elements, or other mysterious things.

The W3C [explicitly state](https://www.w3.org/TR/cors/#security) that this behaviour is safe for publicly accessible files.

**WHITENOISE\_SKIP\_COMPRESS\_EXTENSIONS**

**Default**

**('jpg', 'jpeg', 'png', 'gif', 'webp','zip', 'gz', 'tgz', 'bz2', 'tbz', 'xz', 'br', 'swf', 'flv', 'woff', 'woff2')**

File extensions to skip when compressing.

Because the compression process will only create compressed files where this results in an actual size saving, it would be safe to leave this list empty and attempt to compress all files. However, for files which we’re confident won’t benefit from compression, it speeds up the process if we just skip over them.

**WHITENOISE\_ADD\_HEADERS\_FUNCTION**

**Default**

**None**

Reference to a function which is passed the headers object for each static file, allowing it to modify them.

For example:

**def** force\_download\_pdfs(headers, path, url):

**if** path.endswith('.pdf'):

headers['Content-Disposition'] = 'attachment'

WHITENOISE\_ADD\_HEADERS\_FUNCTION = force\_download\_pdfs

The function is passed:

**headers**

A [wsgiref.headers](https://docs.python.org/3/library/wsgiref.html#module-wsgiref.headers) instance (which you can treat just as a dict) containing the headers for the current file

**path**

The absolute path to the local file

**url**

The host-relative URL of the file e.g. **/static/styles/app.css**

The function should not return anything; changes should be made by modifying the headers dictionary directly.

**WHITENOISE\_IMMUTABLE\_FILE\_TEST**

**Default**

See [immutable\_file\_test](https://github.com/evansd/whitenoise/blob/v5.2.0/whitenoise/middleware.py#L121) in source

Reference to function, or string.

If a reference to a function, this is passed the path and URL for each static file and should return whether that file is immutable, i.e. guaranteed not to change, and so can be safely cached forever. The default is designed to work with Django’s ManifestStaticFilesStorage backend, and any derivatives of that, so you should only need to change this if you are using a different system for versioning your static files.

If a string, this is treated as a regular expression and each file’s URL is matched against it.

Example:

**def** immutable\_file\_test(path, url):

*# Match filename with 12 hex digits before the extension*

*# e.g. app.db8f2edc0c8a.js*

**return** re.match(r'^.+\.[0-9a-f]*{12}*\..+$', url)

WHITENOISE\_IMMUTABLE\_FILE\_TEST = immutable\_file\_test

The function is passed:

**path**

The absolute path to the local file

**url**

The host-relative URL of the file e.g. **/static/styles/app.css**

**WHITENOISE\_STATIC\_PREFIX**

**Default**

Path component of (with removed if set)**settings.STATIC\_URLsettings.FORCE\_SCRIPT\_NAME**

The URL prefix under which static files will be served.

Usually this can be determined automatically by using the path component of . So if is then will be .**STATIC\_URLSTATIC\_URLhttps://example.com/static/WHITENOISE\_STATIC\_PREFIX/static/**

If your application is not running at the root of the domain and is set then this value will be removed from the path first to give the correct prefix.**FORCE\_SCRIPT\_NAMESTATIC\_URL**

If your deployment is more complicated than this (for instance, if you are using a CDN which is doing path rewriting) then you may need to configure this value directly.

**WHITENOISE\_KEEP\_ONLY\_HASHED\_FILES**

**Default**

**False**

Stores only files with hashed names in .**STATIC\_ROOT**

By default, Django’s hashed static files system creates two copies of each file in : one using the original name, e.g. , and one using the hashed name, e.g. . If WhiteNoise’s compression backend is being used this will create another two copies of each of these files (using Gzip and Brotli compression) resulting in six output files for each input file.**STATIC\_ROOTapp.jsapp.db8f2edc0c8a.js**

In some deployment scenarios it can be important to reduce the size of the build artifact as much as possible. This setting removes the “un-hashed” version of the file (which should be not be referenced in any case) which should reduce the space required for static files by half.

Note, this setting is only effective if the WhiteNoise storage backend is being used.

**WHITENOISE\_MANIFEST\_STRICT**

**Default**

**True**

Set to to prevent Django throwing an error if you reference a static file which doesn’t exist.**False**

This works by setting the [manifest\_strict](https://docs.djangoproject.com/en/stable/ref/contrib/staticfiles/#django.contrib.staticfiles.storage.ManifestStaticFilesStorage.manifest_strict) option on the underlying Django storage instance, as described in the Django documentation:

If a file isn’t found in the manifest at runtime, a is raised. This behavior can be disabled by subclassing and setting the attribute to – nonexistent paths will remain unchanged.**staticfiles.jsonValueErrorManifestStaticFilesStoragemanifest\_strictFalse**

Note, this setting is only effective if the WhiteNoise storage backend is being used.

**Additional Notes**

**Django Compressor**

For performance and security reasons WhiteNoise does not check for new files after startup (unless using Django *DEBUG* mode). As such, all static files must be generated in advance. If you’re using Django Compressor, this can be performed using its [offline compression](https://django-compressor.readthedocs.io/en/latest/usage/#offline-compression) feature.

**Serving Media Files**

WhiteNoise is not suitable for serving user-uploaded “media” files. For one thing, as described above, it only checks for static files at startup and so files added after the app starts won’t be seen. More importantly though, serving user-uploaded files from the same domain as your main application is a security risk (this [blog post](https://security.googleblog.com/2012/08/content-hosting-for-modern-web.html) from Google security describes the problem well). And in addition to that, using local disk to store and serve your user media makes it harder to scale your application across multiple machines.

For all these reasons, it’s much better to store files on a separate dedicated storage service and serve them to users from there. The [django-storages](https://django-storages.readthedocs.io/) library provides many options e.g. Amazon S3, Azure Storage, and Rackspace CloudFiles.

**How do I know it’s working?**

You can confirm that WhiteNoise is installed and configured correctly by running you application locally with disabled and checking that your static files still load.DEBUG

First you need to run to get your files in the right place:collectstatic

python manage.py collectstatic

Then make sure is set to in your and start the server:DEBUGFalsesettings.py

python manage.py runserver

You should find that your static files are served, just as they would be in production.

**Troubleshooting the WhiteNoise Storage backend**

If you’re having problems with the WhiteNoise storage backend, the chances are they’re due to the underlying Django storage engine. This is because WhiteNoise only adds a thin wrapper around Django’s storage to add compression support, and because the compression code is very simple it generally doesn’t cause problems.

En yaygın sorun, belirtilen yolda bulunmayan diğer dosyalara (genellikle resimler veya yazı tipleri) başvuran CSS dosyaları nın olmasıdır. Django bu başvuruları yeniden yazmaya çalıştığında ilgili dosyayı arar ve bulamazsa hata atar.

Sorunların WhiteNoise'dan kaynaklanıp kaynaklanmadığını test etmek için, WhiteNoise depolama arka ucunu Django'nun arka ucunu değiştirmeyi deneyin:

STATICFILES\_STORAGE = 'django.contrib.staticfiles.storage.ManifestStaticFilesStorage'

Sorunlar devam ederse, sorununuz Django'nun kendisiyle ilgilidir [(dokümanları](https://docs.djangoproject.com/en/stable/ref/contrib/staticfiles/) veya [posta listesini](https://groups.google.com/d/forum/django-users)deneyin). Sorun yalnızca WhiteNoise ile oluşursa, [sorun izleyicisi](https://github.com/evansd/whitenoise/issues)üzerinde bir bilet yükseltmek.

**CloudFront'u statik dosyalarla sınırlama**

Yukarıda verilen CloudFront'u ayarlama yönergeleri, sitenin tamamına CloudFront URL üzerinden erişilebilmektedir. Bu URL'ler arama sonuçlarında görünmeye başlarsa, bu SEO sorunlarına neden olabilir. CloudFront'u yalnızca statik dosyalarınızı şu yönergeleri izleyerek proxy olarak kısıtlayabilirsiniz:

1. Yeni oluşturduğunuz dağıtıma gidin ve "*Dağıtım Ayarları*", ardından "*Davranışlar*" sekmesini, ardından "*Davranış Oluştur*" sekmesini tıklatın. Yol desenine koyun ve kaydetmek için "*Oluştur*" düğmesine tıklayın.static/\*
2. Şimdi davranışı seçin ve "*Edit*" i tıklayın. "*Viewer Erişimini Kısıtla*" seçeneğini*"*evet " olarak ayarlayın ve kaydetmek için "*Evet, Edit*" düğmesine tıklayın.Default (\*)
3. Desenin listede ilk sırada, varsayılanın ikinci olup olmadığını denetleyin. Bu, statik dosyalar için isteklerin geçirilmesini, ancak diğerlerinin engellenmesini sağlar.static/\*

**Diğer depolama arka uçlarını kullanma**

WhiteNoise yalnızca dosyalarını yerel dosya sisteminde depolayan depolama arka uçlarıyla çalışır. Amazon S3 örneğin, uzaktan dosyaları depolamak backends ile çalışmaz.STATIC\_ROOT

**WhiteNoise testlerimi yavaşlatıyor!**

WhiteNoise, uygulama başladığında mümkün olduğunca çok iş yapmak üzere tasarlanmıştır, böylece uygulama çalışırken dosyaları olabildiğince verimli bir şekilde sunabilir. Bu, uzun süren üretim işlemleri için anlamlıdır, ancak uygulama örnekleri sık sık oluşturulurken ve yok edildiğinde, eklenen başlangıç zamanının test çalıştırmaları sırasında bir sorun olduğunu görebilirsiniz.

Bunu düzeltmenin en basit yolu, sınama sırasında ayarın .'a ayarlandıklarına emin olmaktır. (Varsayılan olarak etkinleştirildiğinde ve başka bir şekilde.) Bu, WhiteNoise'un başlangıçta statik dosyalarınızı taramasını durdurur, ancak bunun dışında davranışları tamamen aynı olmalıdır.WHITENOISE\_AUTOREFRESHTrueTrueDEBUGFalse

Ayrıca, dizininizde gereksiz dosyalar olmadığından da emin olmak gerekir. Özellikle, çok sayıda dosya içerebilen ve uygulama başlatmanızı önemli ölçüde yavaşlatabilen bir dizini eklememeye dikkat edin. Belirli dosyaları eklemeniz gerekiyorsa, statik dizininizin içinden yalnızca ihtiyacınız olan dosyalara symlinks oluşturabilirsiniz.STATIC\_ROOTnode\_modulesnode\_modules

**Neden "ValueError: Eksik staticfiles için tezahür girişi alıyorum ..."?**

Bu hatayı görüyorsanız, şablonlarınızda statik bir dosyaya atıfta bulunduğunuz anlamına gelir (böyle bir şeyi kullanmak var olmayan bir şey kullanmak veya en azından Django'nun beklediği yer değildir. Django'nun kullanabileceğiniz dosyayı neden bulamadığını anlamıyorsanız{% static "foo" %}

python manage.py findstatic --verbosity 2 foo

Hangi django dosya "foo" için arar tüm yolları gösterecektir.

Nedense, Django'nun bu tür hataları sessizce yok saymasını istiyorsanız.WHITENOISE\_MANIFEST\_STRICTFalse

**Webpack / Browserify / $LATEST\_JS\_THING ile WhiteNoise kullanma**

Django ile herhangi bir ön uç yapı sistemi entegre etmek için basit bir teknik aşağıdaki gibi bir dizin düzeni kullanmaktır:

./static\_src

↓

$ ./node\_modules/.bin/webpack

↓

./static\_build

↓

$ ./manage.py collectstatic

↓

./static\_root

Burada projeniz için tüm kaynak dosyaları (JS, CSS, vb) içerir. Yapı aracınız (Webpack, Browserify veya seçtiğiniz her neyse olabilir) sonra bu dosyaları işler ve çıktıyı .static\_srcstatic\_build

Dizine giden yol eklenir:static\_buildsettings.py

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static\_build')

]

Bu, Django'nun işlenen dosyaları bulabileceği, ancak bunları üreten araç hakkında hiçbir şey bilmesi gerekmediği anlamına gelir.

Son adım "karma sürüm" ve sıkıştırılmış statik dosyaların kopyalarını üretime hazır olarak yazar.manage.py collectstaticstatic\_root

Not, hem dizinler hem de dizinler sürüm denetiminin dışında tutulmalıdır (örn. aracılığıyla) ve yalnızca dizin iade edilmelidir.static\_buildstatic\_root.gitignorestatic\_src

**Etki alanının kökünde olmayan bir uygulama dağıtma**

Bazen Django uygulamaları, yalnızca [http://example.com](http://example.com/)yerine <http://example.com/my-app/> gibi bir etki alanında belirli bir önekte (veya "alt dizin") dağıtılır.

Bu durumda normalde uygulamanın nerede olduğunu söylemek için Django'nun [FORCE\_SCRIPT\_NAME](https://docs.djangoproject.com/en/1.11/ref/settings/#force-script-name) ayarını kullanırsınız. Ayrıca doğru önek de kullandığından emin olmanız gerekir. Örneğin:STATIC\_URL

FORCE\_SCRIPT\_NAME = '/my-app'

STATIC\_URL = FORCE\_SCRIPT\_NAME + '/static/'

Bu iki değeri ayarladıysanız, WhiteNoise kendisini otomatik olarak doğru şekilde yapılandıracaktır. Daha karmaşık bir şey yapıyorsanız, [**WHITENOISE\_STATIC\_PREFIX**](http://whitenoise.evans.io/en/stable/django.html#WHITENOISE_STATIC_PREFIX)açıkça kendiniz ayarlamanız gerekebilir.