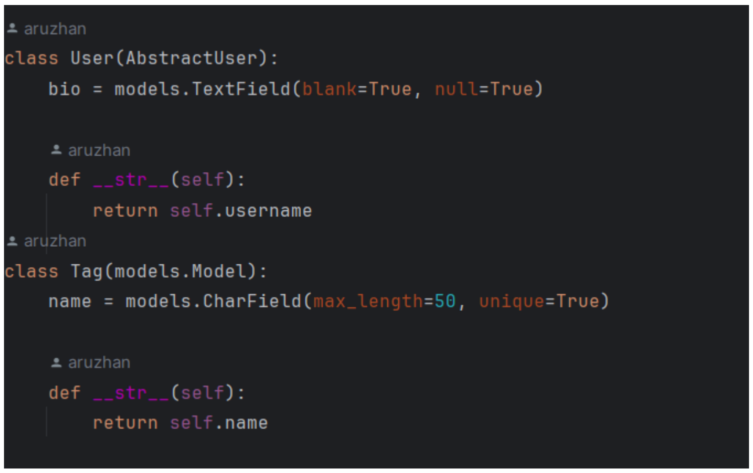
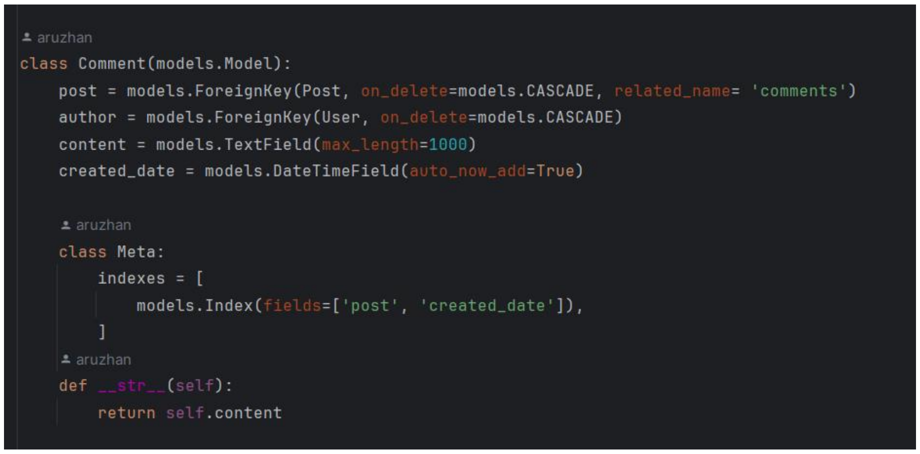
For assignment 2 I have referenced assignment 1st “Blog” project.

Exercise 1:

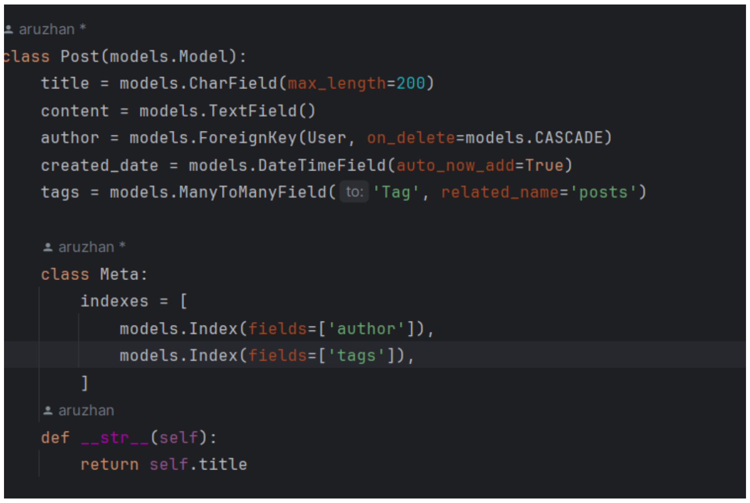
Schema Design and **Indexing**:



```
aruzhan
class User(AbstractUser):
    bio = models.TextField(blank=True, null=True)

    aruzhan
    def __str__(self):
        return self.username
aruzhan
class Tag(models.Model):
    name = models.CharField(max_length=50, unique=True)

    aruzhan
    def __str__(self):
        return self.name
```



```
aruzhan
class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name= 'comments')
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField(max_length=1000)
    created_date = models.DateTimeField(auto_now_add=True)

    aruzhan
    class Meta:
        indexes = [
            models.Index(fields=['post', 'created_date']),
        ]
    aruzhan
    def __str__(self):
        return self.content
```



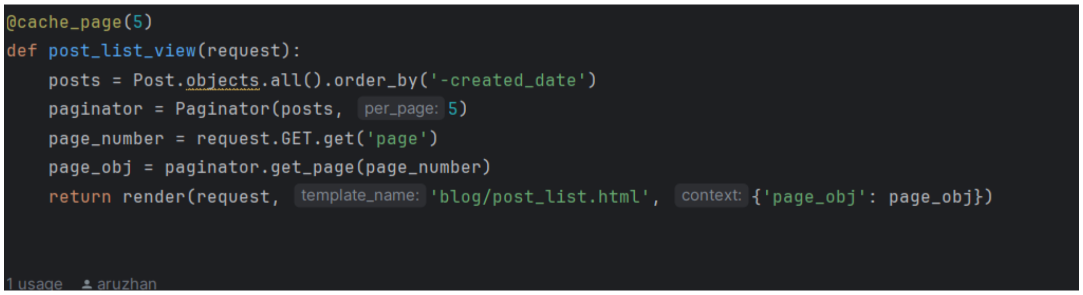
```
aruzhan *
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_date = models.DateTimeField(auto_now_add=True)
    tags = models.ManyToManyField( to: 'Tag', related_name='posts')

    aruzhan *
    class Meta:
        indexes = [
            models.Index(fields=['author']),
            models.Index(fields=['tags']),
        ]
    aruzhan
    def __str__(self):
        return self.title
```

Query Optimization:

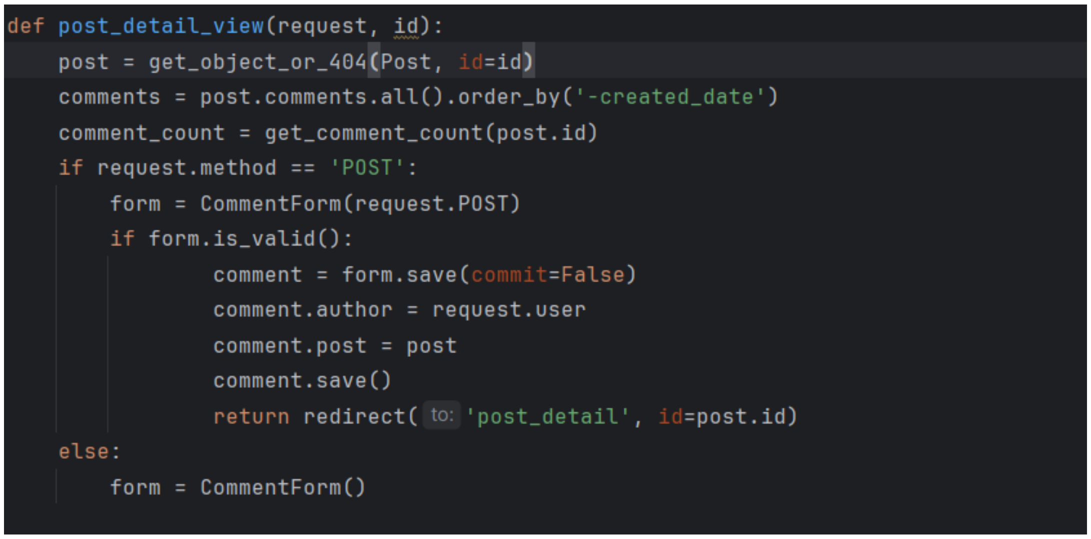
- Post.objects.prefetch_related('comments').all()
  was used to request posts with all their comments in post_list_view
- The use of select_related might not be suitable because it only works with ForeignKey relationships. Prefetch_related is more suitable in this case.

Optimization report:

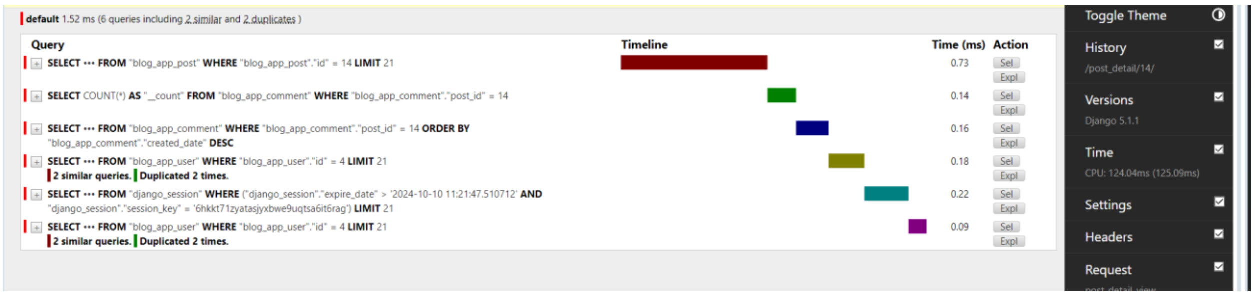
-Here are views without prefetch_related command and the time that sql query took is 0.73ms.



```python
@cache_page(5)
def post_list_view(request):
    posts = Post.objects.all().order_by('-created_date')
    paginator = Paginator(posts, per_page: 5)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
    return render(request, template_name: 'blog/post_list.html', context: {'page_obj': page_obj})


1 usage   aruzhan
```

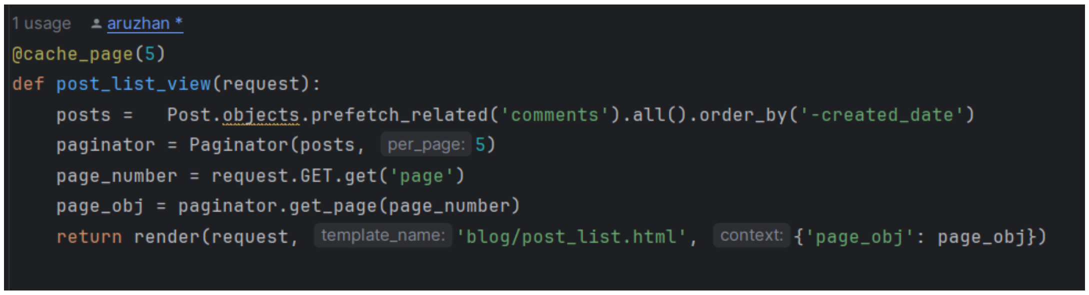


```python
def post_detail_view(request, id):
    post = get_object_or_404(Post, id=id)
    comments = post.comments.all().order_by('-created_date')
    comment_count = get_comment_count(post.id)
    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
                comment = form.save(commit=False)
                comment.author = request.user
                comment.post = post
                comment.save()
                return redirect( to: 'post_detail', id=post.id)
    else:
        form = CommentForm()
```

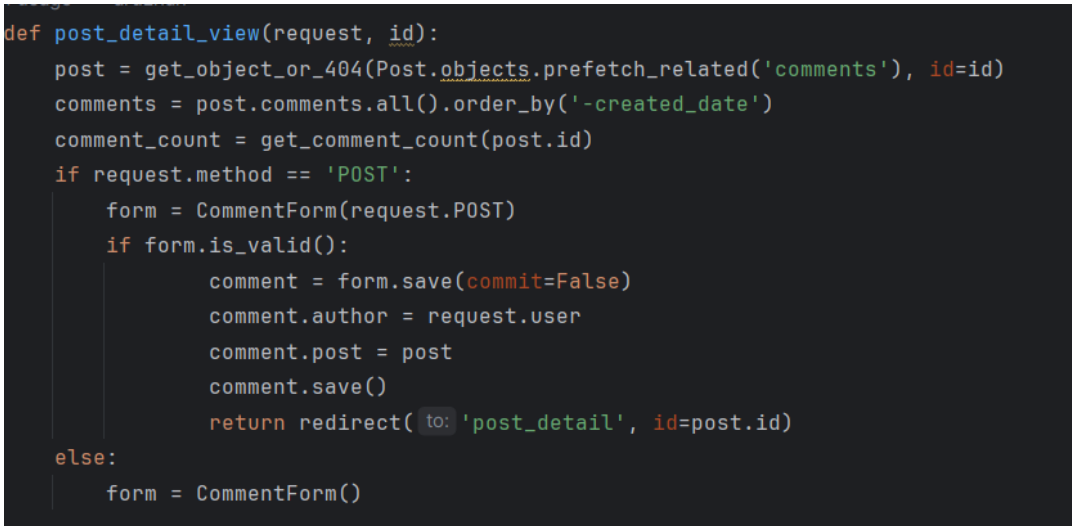


```
default 1.52 ms (6 queries including 2 similar and 2 duplicates )

Query                                                                                    Timeline   Time (ms)  Action
SELECT ••• FROM "blog_app_post" WHERE "blog_app_post"."id" = 14 LIMIT 21                             0.73      Sel Expl
SELECT COUNT(*) AS "__count" FROM "blog_app_comment" WHERE "blog_app_comment"."post_id" = 14         0.14      Sel Expl
SELECT ••• FROM "blog_app_comment" WHERE "blog_app_comment"."post_id" = 14 ORDER BY                  0.16      Sel Expl
"blog_app_comment"."created_date" DESC
SELECT ••• FROM "blog_app_user" WHERE "blog_app_user"."id" = 4 LIMIT 21                              0.18      Sel Expl
2 similar queries. Duplicated 2 times.
SELECT ••• FROM "django_session" WHERE ("django_session"."expire_date" > '2024-10-10 11:21:47.510712' AND   0.22   Sel Expl
"django_session"."session_key" = '6hkkt71zyatasjyxbwe9uqtsa6it6rag') LIMIT 21
SELECT ••• FROM "blog_app_user" WHERE "blog_app_user"."id" = 4 LIMIT 21                              0.09      Sel Expl
2 similar queries. Duplicated 2 times.

Toggle Theme
History
/post_detail/14/
Versions
Django 5.1.1
Time
CPU: 124.04ms (125.09ms)
Settings
Headers
Request
```

-Here is an example with prefetch_related ORM query used, and the time sql request took is 0.48ms



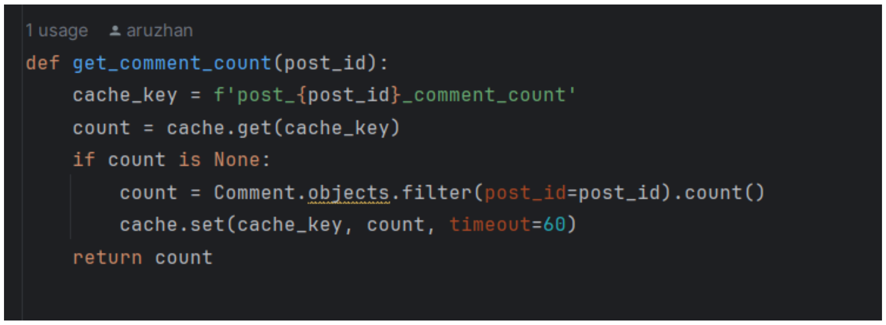
```
1 usage  aruzhan *
@cache_page(5)
def post_list_view(request):
    posts =   Post.objects.prefetch_related('comments').all().order_by('-created_date')
    paginator = Paginator(posts, per_page: 5)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
    return render(request, template_name: 'blog/post_list.html', context: {'page_obj': page_obj})
```



```
def post_detail_view(request, id):
    post = get_object_or_404(Post.objects.prefetch_related('comments'), id=id)
    comments = post.comments.all().order_by('-created_date')
    comment_count = get_comment_count(post.id)
    if request.method == 'POST':
        form = CommentForm(request.POST)
        if form.is_valid():
                comment = form.save(commit=False)
                comment.author = request.user
                comment.post = post
                comment.save()
                return redirect( to: 'post_detail', id=post.id)
    else:
        form = CommentForm()
```

Recommendations for optimization:

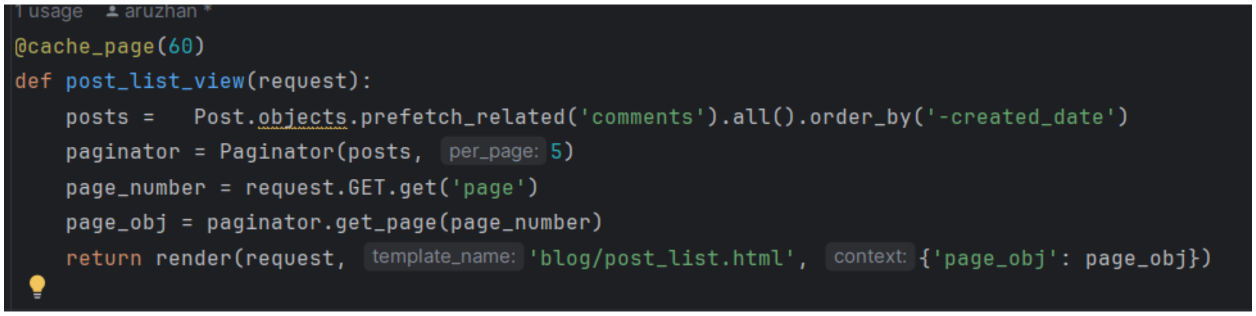
- Using comment counting denormalization method:



```
1 usage  aruzhan
def get_comment_count(post_id):
    cache_key = f'post_{post_id}_comment_count'
    count = cache.get(cache_key)
    if count is None:
        count = Comment.objects.filter(post_id=post_id).count()
        cache.set(cache_key, count, timeout=60)
    return count
```

Exercise 2:

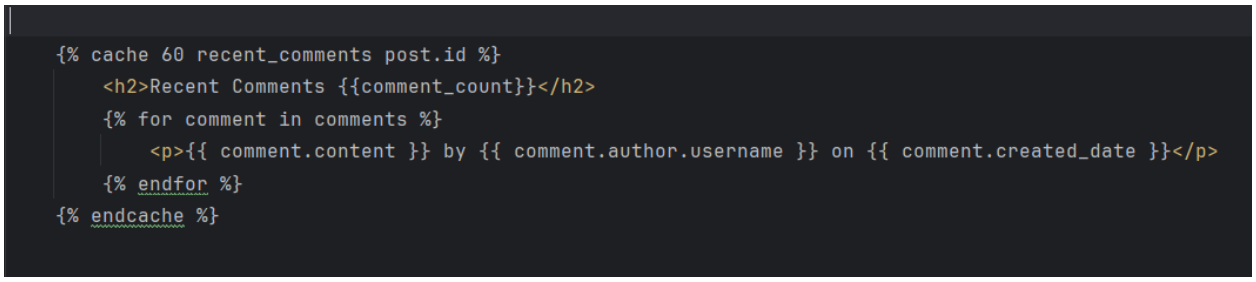
Basic Caching:



```
1 usage   aruzhan *
@cache_page(60)
def post_list_view(request):
    posts =   Post.objects.prefetch_related('comments').all().order_by('-created_date')
    paginator = Paginator(posts,  per_page: 5)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
    return render(request,  template_name: 'blog/post_list.html',  context: {'page_obj': page_obj})
```

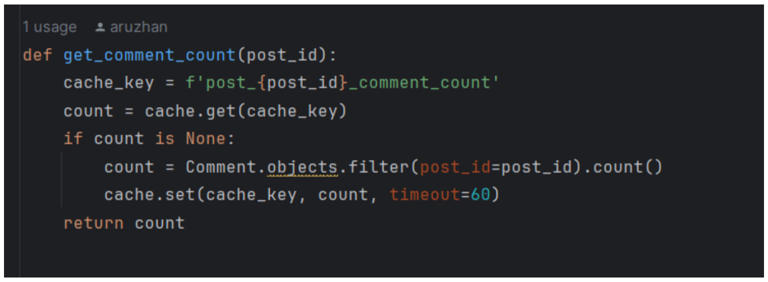
Template Fragment Caching:

In post_detail.html implemented caching of recent comments with timeout in 60 sec.



```
    {% cache 60 recent_comments post.id %}
        <h2>Recent Comments {{comment_count}}</h2>
        {% for comment in comments %}
            <p>{{ comment.content }} by {{ comment.author.username }} on {{ comment.created_date }}</p>
        {% endfor %}
    {% endcache %}
```

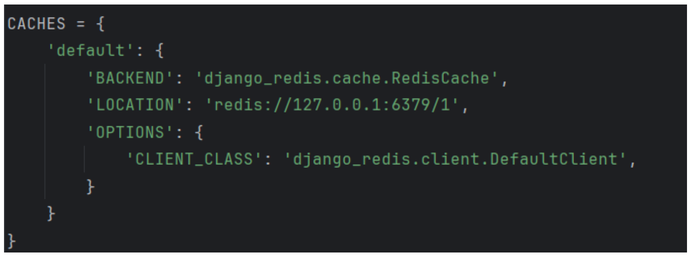
Low-level caching:



```
1 usage   aruzhan
def get_comment_count(post_id):
    cache_key = f'post_{post_id}_comment_count'
    count = cache.get(cache_key)
    if count is None:
        count = Comment.objects.filter(post_id=post_id).count()
        cache.set(cache_key, count, timeout=60)
    return count
```

Caching backend:

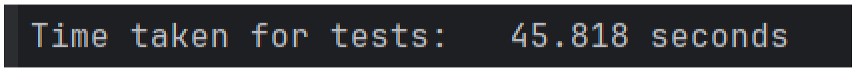
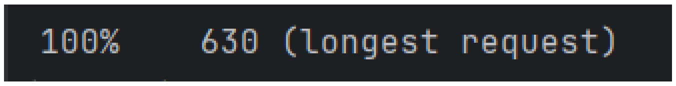
Redis is implemented and used.



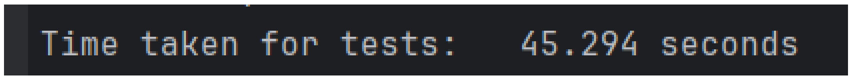
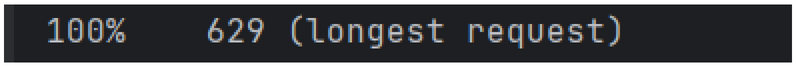
```
CACHES = {
    'default': {
        'BACKEND': 'django_redis.cache.RedisCache',
        'LOCATION': 'redis://127.0.0.1:6379/1',
        'OPTIONS': {
            'CLIENT_CLASS': 'django_redis.client.DefaultClient',
        }
    }
}
```

- For **measuring performance dependence on caching** I used Apache Benchmarking.

Without caching implementation:

- 
```
Time taken for tests:   45.818 seconds
```
- 
```
100%    630 (longest request)
```

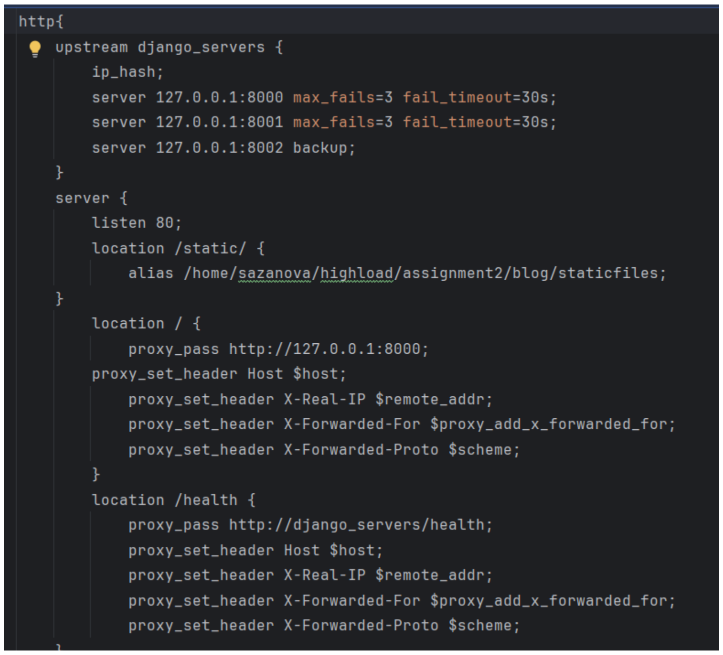
With caching:

- 
```
Time taken for tests:   45.294 seconds
```
- 
```
100%    629 (longest request)
```

Conclusion: There is insignificant change in time because the database is not large enough.

Exercise 3:

Nginx configuration:

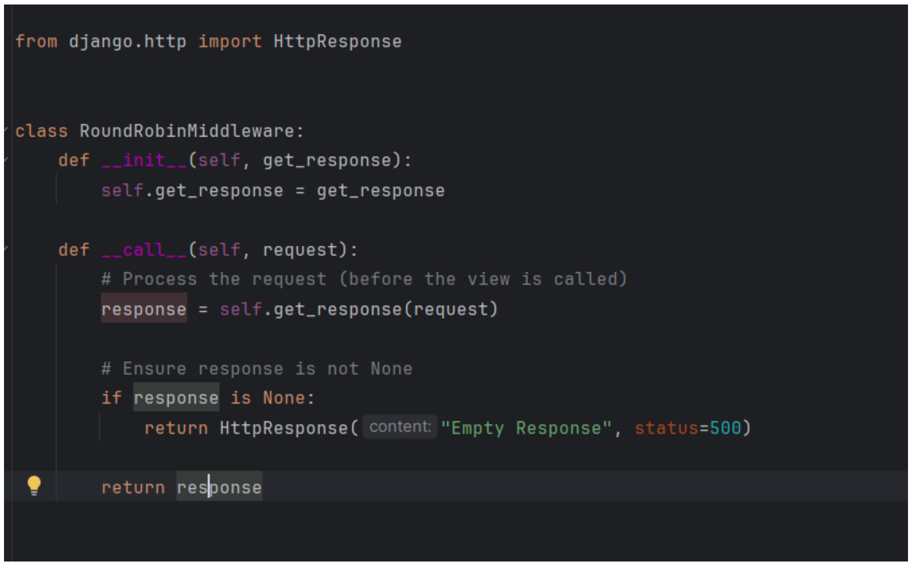


```
http{
    upstream django_servers {
        ip_hash;
        server 127.0.0.1:8000 max_fails=3 fail_timeout=30s;
        server 127.0.0.1:8001 max_fails=3 fail_timeout=30s;
        server 127.0.0.1:8002 backup;
    }
    server {
        listen 80;
        location /static/ {
            alias /home/sazanova/highload/assignment2/blog/staticfiles;
    }
        location / {
            proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
        location /health {
            proxy_pass http://django_servers/health;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
```

I have created also servers for 8001 and 8002 ports.

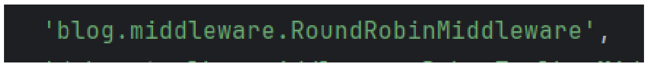
Round-robin load balancing:

I have implemented it in a new middleware.py file.



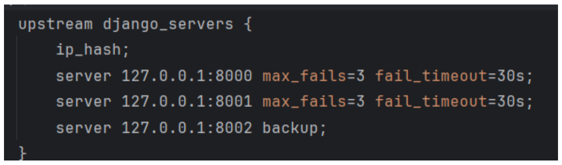
```python
from django.http import HttpResponse


class RoundRobinMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        # Process the request (before the view is called)
        response = self.get_response(request)

        # Ensure response is not None
        if response is None:
            return HttpResponse( content: "Empty Response", status=500)

        return response
```

Then added it into the MIDDLEWARE in settings.py:



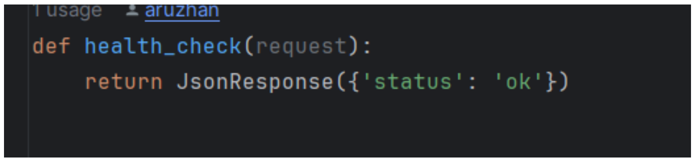
```python
'blog.middleware.RoundRobinMiddleware',
```

To enable **sticky sessions**, I used ip_hash in nginx configuration file.

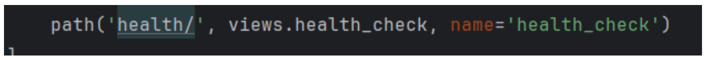


```nginx
upstream django_servers {
    ip_hash;
    server 127.0.0.1:8000 max_fails=3 fail_timeout=30s;
    server 127.0.0.1:8001 max_fails=3 fail_timeout=30s;
    server 127.0.0.1:8002 backup;
}
```

Health check was implemented through writing limits for fails and time into “upstream” segment and writing a response function in view.py then adding it to the urls.py.



```
1 usage  aruzhan
def health_check(request):
    return JsonResponse({'status': 'ok'})
```



```
    path('health/', views.health_check, name='health_check')
```

Analysis of effectiveness of load balancer:

- It works properly and if any of the ports are killed or gone down it distributes tasks between ports.
- One user’s actions are rightly distributed between ports and is transferred properly.

Challenges faced:

1) There were some challenges in creating one static file directory for all css js files needed. I tried to implement django-debug-toolbar, but its css didn’t work when running the project through gunicorn, but worked properly when running it through runserver. I have used the command python manage.py collectstatic and added all needed css, js files manually to STATICFILES_DIRS in settings.py
2) Problems dynamical elements in caching the page. When I cache the post_list page which is cached for 60 seconds, it doesn’t instantly change login logout icons. I may solve it through implementing template fragment caching, but I needed to show how to use view level caching through @cache_page(60).