Assignment 8 - Performance Testing

Zinetov Alikhan

Site: https://aituwka2-0.onrender.com

Scenarios	Virtual Users	Steps	Different browsers
1. Authentification	50	POST /api/auth/login с валидными credentials	Firefox Windows/ Chrome Windows
2. Reading and searching posts	50	GET /api/posts; GET /api/posts/search?username=loadtest	Firefox Windows/ Chrome Windows
3. CRUD methods	50	POST /api/posts PUT /api/posts/:id DELETE /api/posts/:id	Firefox Windows/ Chrome Windows

Code in Python using locust for performance testing:

```
from locust import HttpUser, task, between, constant_pacing
import random
import string


USER_AGENTS = [
    # Chrome Windows
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 "
    "(KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36",
    # Firefox Windows
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:116.0) Gecko/20100101 Firefox/116.0",
    # Safari Mac
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 13_4) AppleWebKit/605.1.15 "
    "(KHTML, like Gecko) Version/16.5 Safari/605.1.15",
    # Edge Windows
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 "
    "(KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36 Edg/114.0.1823.43"
]


def random_text(length):
    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))
```

```python
class BaseUser(HttpUser):

    abstract = True

    host = "https://aituwka2-0.onrender.com"

    token = None

    headers = {}



    def on_start(self):

        self.headers = {

            "User-Agent": random.choice(USER_AGENTS),

            "Accept": "application/json",

            "Content-Type": "application/json",

        }

        creds = {"email": "sasha@gmail.com", "password":
"Alikhan2301!"}

        with self.client.post(

            "/api/auth/login",

            json=creds,

            headers=self.headers,

            name=f"POST /api/auth/login [{self._ua_label()}]",

            catch_response=True

        ) as resp:

            if resp.status_code == 200 and "token" in resp.json():

                self.token = resp.json()["token"]

            else:

                resp.failure(f"Login failed: {resp.status_code}")
```

```
    def _ua_label(self):
        return self.headers["User-Agent"].split()[0]


class ReadUser(BaseUser):
    wait_time = between(2, 5)


    def on_start(self):
        super().on_start()
        res = self.client.get(
            "/api/posts/search?username=loadtest",
            headers=self.headers,
            name=f"GET /api/posts/search [{self._ua_label()}]"
        )
        self.own_ids = [p["_id"] for p in res.json()]


    @task(3)
    def list_all_posts(self):
        self.client.get(
            "/api/posts",
            headers=self.headers,
            name=f"GET /api/posts [{self._ua_label()}]"
        )


    @task(1)
```

```
    def search_posts(self):
        self.client.get(
            "/api/posts/search?username=loadtest",
            headers=self.headers,
            name=f"GET /api/posts/search [{self._ua_label()}]"
        )


class WriteUser(BaseUser):
    wait_time = between(3, 6)


    def on_start(self):
        super().on_start()
        res = self.client.get(
            "/api/posts/search?username=loadtest",
            headers=self.headers,
            name=f"GET /api/posts/search [{self._ua_label()}]"
        )
        self.own_ids = [p["_id"] for p in res.json()]


    @task(3)
    def create_post(self):
        if not self.token:
            return
        hdr = {**self.headers, "Authorization": f"Bearer {self.token}"}
```

```
        payload = {"title": random_text(10), "content":
random_text(50)}

        with self.client.post(

            "/api/posts",

            json=payload,

            headers=hdr,

            name=f"POST /api/posts [{self._ua_label()}]",

            catch_response=True

        ) as resp:

            if resp.status_code == 201 and "_id" in resp.json():

                self.own_ids.append(resp.json()["_id"])

            else:

                resp.failure(f"Create failed: {resp.status_code}")



    @task(1)

    def update_post(self):

        if not self.token or not self.own_ids:

            return

        pid = random.choice(self.own_ids)

        hdr = {**self.headers, "Authorization": f"Bearer {self.token}"}

        payload = {"title": random_text(8)}

        self.client.put(

            f"/api/posts/{pid}",

            json=payload,

            headers=hdr,
```

```
            name=f"PUT /api/posts/[id] [{self._ua_label()}]"
        )

    @task(1)
    def delete_post(self):
        if not self.token or not self.own_ids:
            return
        pid = self.own_ids.pop()
        hdr = {**self.headers, "Authorization": f"Bearer {self.token}"}
        self.client.delete(
            f"/api/posts/{pid}",
            headers=hdr,
            name=f"DELETE /api/posts/[id] [{self._ua_label()}]"
        )

class MixedUser(BaseUser):
    wait_time = constant_pacing(5)

    def on_start(self):
        super().on_start()
        res = self.client.get(
            "/api/posts/search?username=loadtest",
            headers=self.headers,
            name=f"GET /api/posts/search [{self._ua_label()}]"
        )
```

```
        self.own_ids = [p["_id"] for p in res.json()]


    @task(2)
    def browse_and_search(self):
        self.client.get(
            "/api/posts",
            headers=self.headers,
            name=f"GET /api/posts [{self._ua_label()}]"
        )
        self.client.get(
            "/api/posts/search?username=loadtest",
            headers=self.headers,
            name=f"GET /api/posts/search [{self._ua_label()}]"
        )


    @task(1)
    def create_and_update(self):
        if not self.token:
            return
        hdr = {**self.headers, "Authorization": f"Bearer {self.token}"}
        with self.client.post(
            "/api/posts",
            json={"title": random_text(6), "content": random_text(20)},
            headers=hdr,
            name=f"POST /api/posts [{self._ua_label()}]",
```

```
            catch_response=True
        ) as resp:
            if resp.status_code == 201:
                new_id = resp.json()["_id"]
                self.own_ids.append(new_id)
                self.client.put(
                    f"/api/posts/{new_id}",
                    json={"content": random_text(10)},
                    headers=hdr,
                    name=f"PUT /api/posts/[new_id]
[{self._ua_label()}]"
                )
            else:
                resp.failure(f"Create failed: {resp.status_code}")
```

Chrome:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
DELETE	DELETE /api/posts/[id] [Mozilla/5.0]	11	0	3400	6900	6900	3615.86	1523	6869	40	0.3	0
GET	GET /api/posts [Mozilla/5.0]	118	0	5600	9100	19000	6192.56	2312	24026	134988.34	3.1	0
GET	GET /api/posts/search [Mozilla/5.0]	125	0	1900	19000	23000	4814.28	550	23420	2	2	0
POST	POST /api/auth/login [Mozilla/5.0]	50	0	17000	26000	26000	15492.43	3863	26497	247	0	0
POST	POST /api/posts [Mozilla/5.0]	101	0	1900	11000	18000	3098	581	22795	186.2	1.7	0
PUT	PUT /api/posts/[id] [Mozilla/5.0]	15	0	4400	7700	7700	4376.33	1125	7684	40	0.3	0
PUT	PUT /api/posts/[new_id] [Mozilla/5.0]	41	0	3400	7700	17000	3989.1	1399	17002	40	0.4	0
	Aggregated	461	0	4000	20000	26000	5832.97	550	26497	34626.27	7.8	0

Firefox:

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
DELETE	DELETE /api/posts/[id] [Mozilla/5.0]	10	0	2800	7900	7900	3752.74	1346	7918	40	0.3	0
GET	GET /api/posts [Mozilla/5.0]	123	0	5600	12000	19000	6618.06	2513	28506	156261.99	2.5	0
GET	GET /api/posts/search [Mozilla/5.0]	133	0	1800	25000	27000	5101.45	533	26922	2	2	0
POST	POST /api/auth/login [Mozilla/5.0]	50	0	17000	28000	29000	15678.08	2360	28578	247	0	0
POST	POST /api/posts [Mozilla/5.0]	105	0	2000	12000	18000	3448.98	570	18389	189.64	2.2	0
PUT	PUT /api/posts/[id] [Mozilla/5.0]	13	0	2000	9500	9500	2870.47	1005	9459	40	0.4	0
PUT	PUT /api/posts/[new_id] [Mozilla/5.0]	32	0	3700	6400	14000	3868.15	1005	14205	40	0.7	0
	Aggregated	466	0	4400	22000	28000	6088.38	533	28578	41319.64	8.1	0

Analysis of Chrome table:

- **Authentication: High latency due to bcrypt password hashing and JWT generation (P95=26 s).**
- **List Posts: No pagination or field projection, returning large JSON payload (P99=19 s).**
- **Search Posts: using .populate() and sorting without indexes result in long response times (P99=23 s).**
- **CRUD Operations: Create and update average ~1 s; delete peaks at 12 s under load.**

Difference between Chrome and Firefox:

- **Firefox exhibits 200–1 500 ms higher latency on most endpoints, notably PUT /api/posts/:id.**
- **Chrome performs slightly better for some operations (e.g., POST /api/posts).**

Recommendations:

1. **Implement Pagination.**
2. **Use Field Projection: Return only necessary fields.**
3. **Add Indexes.**
4. **Enable Caching.**
5. **Optimize Authentication: Tune bcrypt cost factor or implement session caching.**

Conclusion:
The performance tests revealed critical bottlenecks in authentication and data retrieval without pagination. Applying the recommended optimizations should reduce average response times to 500–1 000 ms and significantly lower P95/P99 latencies.