vault app

vaultcores -> spring boot application

Vault client (receives the credentials)

VaultcoresApplication.java

```
@SpringBootApplication
public class VaultcoresApplication {
	public static void main(String[] args) {
		SpringApplication.run(VaultcoresApplication.class, args);
	}
}
```

@SpringBootApplication:

- Combo of @Configuration, @EnableAutoConfiguration, and @ComponentScan
- Tells Spring to auto-configure beans and scan the current package for components (controllers, services, etc.)

main(...):

- Standard Java entry point
- SpringApplication.run(...) bootstraps the Spring app, starts an embedded server (like Tomcat)

AppConfig.java

```
@Configuration
public class AppConfig {
	@Bean
	public RestTemplate restTemplate() {
		return new RestTemplate();
	}
}
```

@Configuration marks this as a class that declares Spring beans.
@Bean registers RestTemplate as a bean in the Spring context.
So anywhere you @Autowired RestTemplate, you get this instance.

Credential.java (Model)

```
public class Credential {
	private String key;
	private String value;

	// Constructors, getters, setters
}
```

A simple POJO used to **map JSON data** from the Core Java app into Java objects. Jackson (the default JSON parser in Spring Boot) uses **getters/setters** to serialize/deserialize JSON.

CredentialService.java (WebClient-based cache loader)

```
@Service
public class CredentialService {

	private final WebClient webClient = WebClient.create("http://localhost:8081");
	private final Map<String, String> credentialMap = new ConcurrentHashMap<>();

	@PostConstruct
	public void loadCredentialsFromVault() {
		System.out.println("Fetching credentials from core Java vault Server...");
		Credential[] credentials = webClient.get()
				.uri("/credentials")
				.retrieve()
				.bodyToMono(Credential[].class)
				.block();

		if(credentials != null) {
			for(Credential credential : credentials) {
				credentialMap.put(credential.getKey(), credential.getValue());
			}
		System.out.println("loaded credentials: " + credentialMap);
		} else {
			System.out.println("Failed to load credentials!");
		}
	}

	public String getCredentialValue(String key) {
		return credentialMap.get(key);
	}
}
```

This class is responsible for:

- Fetching credentials **once at app startup**
- Storing them in a thread-safe in-memory map
- Providing fast, in-memory lookups when users hit your API

@Service is a Spring annotation that marks this class as a **service layer component**.It gets detected during component scan (@SpringBootApplication) and registered as a **Spring Bean**.
Services usually contain **business logic** (like caching, computing, transforming data).

```
private final WebClient webClient = WebClient.create("http://localhost:8081");
```

WebClient is a **non-blocking, reactive HTTP client** introduced in Spring 5.
WebClient.create(...) creates a client configured to call the vault server.
We chose WebClient here to show modern, reactive-style calls.
You could also inject this as a bean if you wanted to reuse the instance elsewhere.

** non-blocking refers to operations that allow a program to continue its execution without waiting for the operation to complete

```
private final Map<String, String> credentialMap = new ConcurrentHashMap<>();
```

Stores the key-value credentials in memory (e.g., "db-user" → "admin")
ConcurrentHashMap is used instead of HashMap because:

- Spring Boot runs with multiple threads
- HTTP requests might hit this map concurrently
- So we need **thread-safety without locking the whole map** (like synchronizedMap would)

Method: loadCredentialsFromVault()

@PostConstruct tells Spring to run this method automatically after bean initialization.
It's the perfect place to pre-fetch data that your app depends on before serving users.

```
Credential[] credentials = webClient.get()
        .uri("/credentials")
        .retrieve()
        .bodyToMono(Credential[].class)
        .block();
```

- Makes an HTTP GET call to /credentials endpoint of your Core Java server.
- .retrieve() triggers the call.
- .bodyToMono(Credential[].class) parses the JSON array into a Credential[].
- .block() waits for the response (because WebClient is normally async).

**** Even though WebClient is reactive, calling .block() converts it into a synchronous call — suitable for startup initialization.**

```
for (Credential credential : credentials) {
        credentialMap.put(credential.getKey(), credential.getValue());
}
```

Loops through each credential and puts it in the map using the key as the lookup.

```
getCredentialValue(String key)
```

```
public String getCredentialValue(String key) {
        return credentialMap.get(key);
}
```

Simple map lookup.
When /get-credential?key=... is called in the controller, it pulls value from here.
It's **fast**, since it avoids any network call and just uses in-memory data.

Summary CredentialService

Aspect	What it does
When it's called	Only once — at Spring Boot startup
Where it stores data	In-memory ConcurrentHashMap
Why it's fast	All credentials are preloaded
Why @PostConstruct	Ensures the app is ready before serving users
Why ConcurrentHashMap	Thread-safe access during parallel HTTP calls
Why WebClient	Modern, async-capable HTTP client (can .block() for sync)

LegacyCredentialClient.java — *RestTemplate-based live fetch*

This class handles:

- Making a live HTTP call to the Core Java vault app **every time a credential is requested**
- No caching involved — always real-time

```
@Component
public class LegacyCredentialClient {
```

@Component is another Spring stereotype (like @Service, @Repository)

Spring will detect this and manage its lifecycle

Called *legacy* because RestTemplate is considered an older HTTP client (but still widely used and simpler)

```
private final RestTemplate restTemplate;
```

RestTemplate is a **blocking, synchronous HTTP client**.
Great for simple use cases.
Doesn't require reactive setup or mono/flux wrapping.

```
public LegacyCredentialClient(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
}
```

Constructor-based dependency injection.

Spring injects the RestTemplate bean from AppConfig.java.

Method: fetchCredentials()

```
public Credential[] fetchCredentials() {
```

```
        String url = "http://localhost:8081/credentials";

        return restTemplate.getForObject(url, Credential[].class);
}
```

Calls the vault API endpoint live.

getForObject(...) does the job of:

1. Sending GET request
2. Receiving the JSON array
3. Mapping it into a Credential[]

Method: getCredentialValueByKey(...)

```
public String getCredentialValueByKey(String key) {
        Credential[] credentials = fetchCredentials();
        for (Credential c : credentials) {
                if (c.getKey().equalsIgnoreCase(key)) {
                        return c.getValue();
                }
        }
        return null;
}
```

- Loops through the returned array
- Matches the key (case-insensitive)
- Returns the matching value if found
- Returns null if not found

Summary — LegacyCredentialClient

Aspect	What it does
When it's called	Every time /get-credential-legacy is hit
Speed	Slower than cache (calls over HTTP every time)
Accuracy	Always fresh, reflects latest state of vault
Why RestTemplate	Simple to use, blocking style, still common in legacy systems
No cache	Doesn't store anything in memory

Vaultcorej -> core java application

Credential server

Class: Credential.java

```
public class Credential {

    private String key;
    private String value;

    public Credential(String key, String value) {
        this.key = key;
        this.value = value;
    }

    public String getKey() {
        return key;
    }

    public String getValue() {
        return value;
    }
}
```

- A simple pojo class defining fields, constructors and getters.

Class: CredentialHandler.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

```
import com.sun.net.httpserver.*;

public class CredentialHandler implements HttpHandler {

    public static final List<Credential> credentials = new ArrayList<>(Arrays.asList(
            new Credential("db-user", "admin"),
            new Credential("db-pass", "secret123"),
            new Credential("api-key", "querty123")));

    @Override
    public void handle(HttpExchange exchange) throws IOException {
        String method = exchange.getRequestMethod();
        if (method.equals("GET")) {
            handleGet(exchange);
        } else if (method.equals("POST")) {
            handlePost(exchange);
        } else {
            exchange.sendResponseHeaders(405, -1); // method not allowed
        }
    }

    private void handleGet(HttpExchange exchange) throws IOException {
        StringBuilder json = new StringBuilder("[");
        for (int i = 0; i < credentials.size(); i++) {
            Credential c = credentials.get(i);
            json.append(String.format("{\"key\": \"%s\", \"value\": \"%s\"}", c.getKey(),
c.getValue()));
            if (i != credentials.size() - 1)
                json.append(",");
        }
        json.append("]");
        exchange.getResponseHeaders().add("Content-Type", "application/json");
        exchange.sendResponseHeaders(200, json.toString().getBytes().length);
        OutputStream os = exchange.getResponseBody();
        os.write(json.toString().getBytes());
        os.close();
    }

    private void handlePost(HttpExchange exchange) throws IOException {
        // read request body
        InputStream is = exchange.getRequestBody();
        BufferedReader reader = new BufferedReader(new InputStreamReader(is,
StandardCharsets.UTF_8));
```

```
        StringBuilder body = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            body.append(line);
        }
        String json = body.toString();
        String key = extractValue(json, "key");
        String value = extractValue(json, "value");

        if (key == null || value == null) {
            exchange.sendResponseHeaders(400, 0);
            exchange.getResponseBody().write("invalid json".getBytes());
            exchange.getResponseBody().close();
            return;
        }

        boolean exists = credentials.stream().anyMatch(c ->
c.getKey().equalsIgnoreCase(key));

        if (exists) {
            exchange.sendResponseHeaders(409, 0); // conflict
            exchange.getResponseBody().write("Key already exists. use a different
key".getBytes());
        } else {
            credentials.add(new Credential(key, value));
            exchange.sendResponseHeaders(201, 0);
            exchange.getResponseBody().write("Credential added".getBytes());
        }

        exchange.getResponseBody().close();
        reader.close();
        is.close();
    }

    private String extractValue(String json, String fieldName) {
        String pattern = "\"" + fieldName + "\":\\s*\"(.*?)\"";
        return json.matches(".*" + pattern + ".*") ? json.replaceAll(".*" + pattern + ".*", "$1") :
null;
    }

}
```

Class: VaultServer.java

```
import java.io.IOException;
import java.net.InetSocketAddress;

import com.sun.net.httpserver.HttpServer;

public class VaultServer {

    public static void main(String[] args) throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(8081), 0);
        server.createContext("/credentials", new CredentialHandler());
        server.setExecutor(null);
        server.start();
        System.out.println("Vault Server running at http://localhost:8081/credentials");
    }
}
```