Give you a clean, production-minded Java implementation for a **Small Computer Shop Management** app that uses **Cloud storage (Google Firestore)** and provides **fully implemented CRUD** (Create, Read, Update, Delete) operations. I'll include:

* Maven `pom.xml` dependencies
* Domain model (`Computer`)
* DAO interface + Firestore implementation (`FirestoreComputerDAO`)
* Service layer (`ComputerService`)
* A simple CLI `Main` to demo CRUD usage
* Setup and deployment notes (how to configure Firebase/Firestore)

This is modular and ready to extend into a web app or desktop GUI later.

What you'll need (quick)

1. Java 11+ (works with Java 8+ but recommended 11+).
2. Maven.
3. A Google Cloud project with **Firestore** enabled and a **service account JSON** (downloaded) — save it as `serviceAccountKey.json` in your project root or a secure path.
4. Firestore in Native mode (recommended for server apps).

`pom.xml` (Maven)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example.computershop</groupId>
  <artifactId>computer-shop</artifactId>
  <version>1.0.0</version>
  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- Firebase Admin SDK -->
    <dependency>
```

```
      <groupId>com.google.firebase</groupId>
      <artifactId>firebase-admin</artifactId>
      <version>9.3.0</version> <!-- verify latest when you implement -->
    </dependency>

    <!-- Google Cloud Firestore (pulled in by firebase-admin usually) -->
    <dependency>
      <groupId>com.google.cloud</groupId>
      <artifactId>google-cloud-firestore</artifactId>
      <version>3.16.4</version>
    </dependency>

    <!-- JSON (Gson) -->
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.10</version>
    </dependency>

    <!-- Logging -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>2.0.7</version>
    </dependency>

    <!-- JUnit 5 for tests (optional) -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.9.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

> *Tip:* Confirm the latest Firebase Admin SDK version when you implement. The code below uses stable APIs.

Domain model — `Computer.java`

```java
package com.example.computershop.model;

import java.util.Objects;

public class Computer {
    private String id;          // Firestore document id
    private String brand;
    private String model;
    private String cpu;
    private int ramGb;
    private int storageGb;
    private double price;
    private int stockQuantity;
    private String notes;

    public Computer() { } // needed for Firestore deserialization

    public Computer(String id, String brand, String model, String cpu, int ramGb, int storageGb, double price, int stockQuantity, String notes) {
        this.id = id;
        this.brand = brand;
        this.model = model;
        this.cpu = cpu;
        this.ramGb = ramGb;
        this.storageGb = storageGb;
        this.price = price;
        this.stockQuantity = stockQuantity;
        this.notes = notes;
    }

    // getters and setters

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getBrand() { return brand; }
    public void setBrand(String brand) { this.brand = brand; }

    public String getModel() { return model; }
    public void setModel(String model) { this.model = model; }
    public String getCpu() { return cpu; }
    public void setCpu(String cpu) { this.cpu = cpu; }

    public int getRamGb() { return ramGb; }
    public void setRamGb(int ramGb) { this.ramGb = ramGb; }

    public int getStorageGb() { return storageGb; }
    public void setStorageGb(int storageGb) { this.storageGb = storageGb; }

    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }

    public int getStockQuantity() { return stockQuantity; }
    public void setStockQuantity(int stockQuantity) { this.stockQuantity = stockQuantity; }

    public String getNotes() { return notes; }
    public void setNotes(String notes) { this.notes = notes; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Computer computer = (Computer) o;
        return Objects.equals(id, computer.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }

    @Override
    public String toString() {
        return "Computer{" +
               "id='" + id + '\'' +
               ", brand='" + brand + '\'' +
               ", model='" + model + '\'' +
               ", cpu='" + cpu + '\'' +
               ", ramGb=" + ramGb +
               ", storageGb=" + storageGb +
               ", price=" + price +
               ", stockQuantity=" + stockQuantity +
               ", notes='" + notes + '\'' +
            '}';
    }
}
```

DAO interface — `ComputerDAO.java`

```java
package com.example.computershop.dao;

import com.example.computershop.model.Computer;

import java.util.List;
import java.util.Optional;

public interface ComputerDAO {
    Computer create(Computer c) throws Exception;
    Optional<Computer> findById(String id) throws Exception;
    List<Computer> findAll() throws Exception;
    Computer update(Computer c) throws Exception;
    boolean delete(String id) throws Exception;
}
```

Firestore DAO implementation — `FirestoreComputerDAO.java`

```java
package com.example.computershop.dao;

import com.example.computershop.model.Computer;
import com.google.api.core.ApiFuture;
import com.google.cloud.firestore.*;
import com.google.firebase.cloud.FirestoreClient;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.ExecutionException;

public class FirestoreComputerDAO implements ComputerDAO {
    private static final String COLLECTION = "computers";
    private final Firestore db;

    public FirestoreComputerDAO() {
        this.db = FirestoreClient.getFirestore();
    }

    @Override
    public Computer create(Computer c) throws ExecutionException, InterruptedException {
        CollectionReference col = db.collection(COLLECTION);
        // let Firestore auto-generate ID
        ApiFuture<DocumentReference> added = col.add(c);
        DocumentReference docRef = added.get();
        c.setId(docRef.getId());
        // also set the document with id to be able to read back full object (optional)
        ApiFuture<WriteResult> write = docRef.set(c);
        write.get();
        return c;
    }

    @Override
    public Optional<Computer> findById(String id) throws ExecutionException,
InterruptedException {
        DocumentReference docRef = db.collection(COLLECTION).document(id);
        ApiFuture<DocumentSnapshot> future = docRef.get();
        DocumentSnapshot doc = future.get();
        if (doc.exists()) {
            Computer c = doc.toObject(Computer.class);
            if (c != null) c.setId(doc.getId());
            return Optional.ofNullable(c);
        }
        return Optional.empty();
    }

    @Override
    public List<Computer> findAll() throws ExecutionException, InterruptedException {
        ApiFuture<QuerySnapshot> future = db.collection(COLLECTION).get();
        List<QueryDocumentSnapshot> documents = future.get().getDocuments();
        List<Computer> list = new ArrayList<>();
        for (QueryDocumentSnapshot doc : documents) {
            Computer c = doc.toObject(Computer.class);
            c.setId(doc.getId());
            list.add(c);
        }
        return list;
    }

    @Override
    public Computer update(Computer c) throws ExecutionException, InterruptedException {
        if (c.getId() == null) throw new IllegalArgumentException("Computer id required for update");
        DocumentReference docRef = db.collection(COLLECTION).document(c.getId());
        ApiFuture<WriteResult> writeResult = docRef.set(c);
        writeResult.get(); // wait
        return c;
    }

    @Override
    public boolean delete(String id) throws ExecutionException, InterruptedException {
        DocumentReference docRef = db.collection(COLLECTION).document(id);
        ApiFuture<WriteResult> writeResult = docRef.delete();
        writeResult.get();
        return true;
    }
}
```

Service layer — `ComputerService.java`

```java
package com.example.computershop.service;

import com.example.computershop.dao.ComputerDAO;
import com.example.computershop.model.Computer;

import java.util.List;
import java.util.Optional;
import java.util.concurrent.ExecutionException;

public class ComputerService {
    private final ComputerDAO dao;

    public ComputerService(ComputerDAO dao) {
        this.dao = dao;
    }
    // Create new computer with simple validations
    public Computer addComputer(Computer c) throws Exception {
        validateComputerForCreate(c);
        return dao.create(c);
    }

    public Optional<Computer> getComputerById(String id) throws ExecutionException,
InterruptedException {
        return dao.findById(id);
    }

    public List<Computer> listComputers() throws ExecutionException, InterruptedException {
        return dao.findAll();
    }

    public Computer updateComputer(Computer c) throws Exception {
        if (c.getId() == null) throw new IllegalArgumentException("ID is required to update a
record");
        validateComputerForUpdate(c);
        return dao.update(c);
    }

    public boolean removeComputer(String id) throws Exception {
        if (id == null || id.isBlank()) throw new IllegalArgumentException("ID required");
        return dao.delete(id);
    }

    private void validateComputerForCreate(Computer c) {
        if (c == null) throw new IllegalArgumentException("Computer cannot be null");
        if (c.getBrand() == null || c.getBrand().isBlank()) throw new
IllegalArgumentException("Brand required");
        if (c.getModel() == null || c.getModel().isBlank()) throw new
IllegalArgumentException("Model required");
        if (c.getPrice() < 0) throw new IllegalArgumentException("Price cannot be negative");
        if (c.getStockQuantity() < 0) throw new IllegalArgumentException("Stock cannot be
negative");
    }

    private void validateComputerForUpdate(Computer c) {
        // reuse create validation but allow partial updates if you like
        validateComputerForCreate(c);
    }
}
```

Firebase initialization — `FirebaseInitializer.java`

Place the `serviceAccountKey.json` in the project (DO NOT commit to source control). This initializer should be called once at application startup.

```java
package com.example.computershop.config;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.cloud.FirestoreClient;

import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.file.Paths;

public class FirebaseInitializer {
    public static void init(String serviceAccountPath) throws Exception {
        // If already initialized, skip
        if (!FirebaseApp.getApps().isEmpty()) return;

        try (InputStream serviceAccount = new FileInputStream(serviceAccountPath)) {
            GoogleCredentials credentials = GoogleCredentials.fromStream(serviceAccount);
            FirebaseOptions options = FirebaseOptions.builder()
                    .setCredentials(credentials)
                    .build();
            FirebaseApp.initializeApp(options);
            // FirestoreClient.getFirestore() will be usable after this
            System.out.println("Firebase initialized successfully.");
        } catch (Exception e) {
            System.err.println("Failed to initialize Firebase: " + e.getMessage());
            throw e;
        }
    }

    public static String defaultServiceAccountPath() {
        // example default - change as needed
        return Paths.get("serviceAccountKey.json").toAbsolutePath().toString();
    }
}
```

Simple CLI `Main.java` (demo)

```java
package com.example.computershop;

import com.example.computershop.config.FirebaseInitializer;
import com.example.computershop.dao.FirestoreComputerDAO;
import com.example.computershop.model.Computer;
import com.example.computershop.service.ComputerService;

import java.util.List;

public class Main {
    public static void main(String[] args) {
        try {
            // Initialize Firebase with local path - change path if needed
            FirebaseInitializer.init(FirebaseInitializer.defaultServiceAccountPath());

            var dao = new FirestoreComputerDAO();
            var service = new ComputerService(dao);

            // Create
            Computer c1 = new Computer(null, "Dell", "Inspiron 15", "Intel i5-1135G7", 8, 512,
49999.00, 5, "14-inch display");
            Computer created = service.addComputer(c1);
            System.out.println("Created: " + created);

            // Read
            var fetched = service.getComputerById(created.getId());
            fetched.ifPresent(c -> System.out.println("Fetched: " + c));

            // Update
            created.setPrice(47999.0);
            created.setStockQuantity(4);
            Computer updated = service.updateComputer(created);
            System.out.println("Updated: " + updated);

            // List
            List<Computer> all = service.listComputers();
            System.out.println("All computers: ");
            all.forEach(System.out::println);
            // Delete
            boolean deleted = service.removeComputer(created.getId());
            System.out.println("Deleted: " + deleted);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Deployment / Running locally

1. Put your `serviceAccountKey.json` in the project root or change `FirebaseInitializer.defaultServiceAccountPath()` to your path.
2. `mvn clean package`
3. Run `java -jar target/computer-shop-1.0.0.jar` or run from IDE.
4. Check Firestore console — a `computers` collection will be created automatically.

Security & good practices

* **Never commit** `serviceAccountKey.json` to Git. Use environment variables or secret managers for production.
* Use **role-limited service accounts** for production.
* Consider using **Cloud IAM** to restrict Firestore access.
* Implement proper **input sanitization** and permission checks if you add user accounts.
* Use **batch writes** for bulk operations to improve performance.
* Implement **pagination** for `findAll()` if data grows large.
* Consider using **Firestore rules** when you later expose data to client apps.

Enhancements you can add

* Pagination, sorting and filtering in DAO.
* Soft deletes (add `deleted` flag instead of permanently removing).
* Add `Order` entity and link with `Computer` for sales tracking.
* Expose as REST API (Spring Boot) or add a simple web UI.
* Add unit/integration tests using the Firestore emulator for CI.