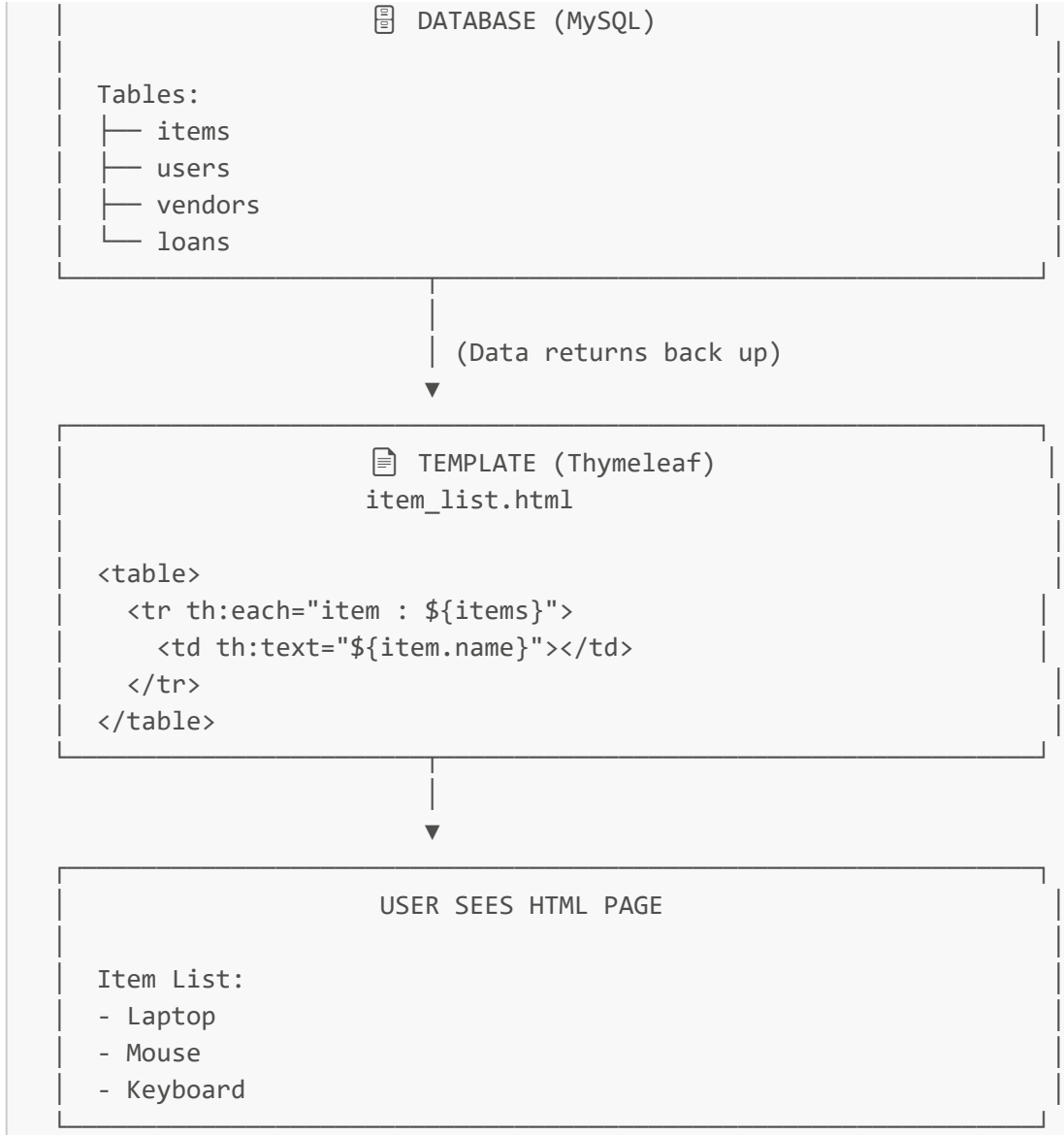
Spring Boot Request Flow - Visual Guide

How a Web Request Works in Your Project



Complete Example: Adding an Item

1. User fills form and clicks "Submit"

```
Browser → POST /items/create
```

2. Controller receives request

```
@PostMapping("/items/create")
public String createItem(@ModelAttribute ItemDto itemDto) {
    itemService.addItem(itemDto);  // ← Calls service
    return "redirect:/items";       // ← Redirects to list
}
```

3. Service processes business logic

```
@Service
public class ItemService {
    public void addItem(ItemDto dto) {
        Item item = convertDtoToEntity(dto);  // Convert
        itemRepository.save(item);            // ← Calls repository
    }
}
```

4. Repository saves to database

```
itemRepository.save(item);  // Spring auto-generates INSERT query
```

5. Database stores data

```
INSERT INTO items (name, quantity, price)
VALUES ('Laptop', 10, 50000);
```

File Structure with Real Examples

```
src/main/java/com/example/IMS/

├── controller/
│   ├── ItemController.java          ← Handles /items/* URLs
│   ├── UserManagementController.java ← Handles /admin/users/*
│   └── VendorController.java         ← Handles /vendors/*
│
├── service/
│   ├── ItemService.java              ← Item business logic
│   ├── UserService.java              ← User management logic
│   └── VendorService.java            ← Vendor operations
│
├── repository/
│   ├── IItemRepository.java          ← Item database access
│   ├── IUserRepository.java          ← User database access
│   └── IVendorRepository.java        ← Vendor database access
│
├── model/
│   ├── Item.java                     ← Item table structure
│   ├── User.java                     ← User table structure
│   └── Vendor.java                   ← Vendor table structure
│
├── dto/
```

```
│   ├── ItemDto.java                ← Item form data
│   └── UserRegistrationDto.java    ← Registration form data
│
└── ImsApApplication.java           ← Main entry point (DON'T MODIFY)
```

How Different Parts Connect

Example: View Item List

```
1. Browser Request:
   GET http://localhost:8080/items

2. ItemController.java:
   @GetMapping("/items")
   public String viewItems(Model model) {
       List<Item> items = itemService.getAllItems();  ← Get data
       model.addAttribute("items", items);            ← Add to model
       return "Item/View";                            ← Return template name
   }

3. ItemService.java:
   public List<Item> getAllItems() {
       return itemRepository.findAll();  ← Get from DB
   }

4. IItemRepository.java:
   // Spring auto-generates:
   SELECT * FROM items;

5. Item/View.html:
   <div th:each="item : ${items}">
       <p th:text="${item.name}"></p>
   </div>
```

Important Annotations Explained

@Controller vs @RestController

```java
@Controller  // Returns HTML pages
public class ItemController {
    @GetMapping("/items")
    public String viewItems() {
        return "item_list";  // Returns item_list.html
    }
}

@RestController  // Returns JSON/XML (for APIs)
```

```
public class ItemApiController {
    @GetMapping("/api/items")
    public List<Item> getItems() {
        return items;  // Returns JSON: [{"id":1,"name":"Laptop"}]
    }
}
```

@Autowired (Dependency Injection)

```
@Controller
public class ItemController {
    @Autowired  // Spring automatically creates and injects this
    private ItemService itemService;

    // No need to write: itemService = new ItemService();
    // Spring does it for you!
}
```

@Entity (Database Table)

```
@Entity
@Table(name = "items")  // Optional: customize table name
public class Item {
    @Id  // Primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)  // Auto-increment
    private Long id;

    @Column(nullable = false)  // Required field
    private String name;

    private Integer quantity;  // Optional field
}

// This creates:
// CREATE TABLE items (
//   id BIGINT AUTO_INCREMENT PRIMARY KEY,
//   name VARCHAR(255) NOT NULL,
//   quantity INT
// );
```

🚀 Quick Commands Reference

Building

```

```

```
# Clean and build
mvn clean install

# Skip tests (faster)
mvn clean install -DskipTests

# Package as JAR
mvn package
```

Running

```
# Development mode (with auto-restart)
mvn spring-boot:run

# Production mode
java -jar target/IMS-AP-0.0.1-SNAPSHOT.jar
```

Testing

```
# Run all tests
mvn test

# Run specific test
mvn test -Dtest=ItemServiceTest
```

Database

```
# Create database
.\setup-mysql-database.bat

# Check database connection
.\check-database.bat
```

Debugging Tips

1. Check Application Logs

Look for these patterns in console:

```
✅ Started ImsApApplication in 3.456 seconds
   → App started successfully
```

```
❌ Error creating bean with name 'itemRepository'
   → Database connection issue

❌ Whitelabel Error Page - Status 404
   → URL mapping not found

❌ Whitelabel Error Page - Status 500
   → Java error in code
```

2. Enable Debug Mode

Add to `application.properties`:

```
# See all SQL queries
spring.jpa.show-sql=true
logging.level.org.hibernate.SQL=DEBUG

# See SQL parameters
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

# Enable debug logging
logging.level.root=DEBUG
```

3. Test in Browser

```
URL to test          What it does
------------------   ---------------------------------
/items               View all items
/items/create        Create new item form
/login               Login page
/api/items           JSON API (if RestController exists)
```

📊 Project Metrics

Your FlowTrack project has:

- ☑ **Controllers:** 11 (handling web requests)
- ☑ **Services:** 13 (business logic)
- ☑ **Repositories:** 10 (database access)
- ☑ **Models:** 11 (database tables)
- ☑ **Templates:** 30+ (HTML pages)
- ☑ **Dependencies:** Spring Boot 2.7.18, Java 11

🎓 Learning Path

Week 1: Understand Existing Code

- Read through `ItemController.java`
- Trace how data flows from browser to database
- Modify a simple HTML template

Week 2: Make Small Changes

- Add a new field to `Item` model
- Create a simple search function
- Customize form validation

Week 3: Create New Feature

- Add "Category" entity
- Create CRUD operations (Create, Read, Update, Delete)
- Link categories to items

Week 4: Advanced Topics

- Learn Spring Security (authentication)
- Implement REST APIs
- Add file upload functionality

Common Pitfalls for Beginners

❌ **Mistake:** Modifying `target/` folder
☑ **Solution:** Always edit files in `src/`, `target/` is auto-generated

❌ **Mistake:** Forgetting to rebuild after changes
☑ **Solution:** Run `mvn clean install` after code changes

❌ **Mistake:** Committing `.idea/` or `target/` to git
☑ **Solution:** Already fixed! Your `.gitignore` is configured

❌ **Mistake:** Hardcoding database passwords in code
☑ **Solution:** Use environment variables or `application-local.properties`

❌ **Mistake:** Not understanding Spring's "magic"
☑ **Solution:** It's not magic! Spring uses annotations + reflection + proxies

---

Resources

Official Documentation:

- Spring Boot Docs
- Spring Data JPA
- Thymeleaf

Video Tutorials:

- YouTube: "Spring Boot Tutorial for Beginners"
- YouTube: "Spring Boot Full Course"

Practice:

- Spring Initializr - Create new projects
- Baeldung - Excellent tutorials

You're all set! Start with the BEGINNER_GUIDE.md and explore the code! 🚀