Hugbúnaðarverkefni 1 / Software Project 1

6. Persistence Layer

HBV501G – Fall 2018

Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

Miðmisseriskönnun

Evaluate this course in Ugla!

(survey open until today)

In-Class Quiz Prep

- Please prepare a scrap of paper with the following information:

- ID: _____@hi.is   Date: ________

- a) ______________ e) ______________
- b) ______________ f) ______________
- c) ______________ g) ______________
- d) ______________ h) ______________

- During class, I’ll show you questions that you can answer very briefly
  - No elaboration necessary
- Hand in your scrap at the end of class

- All questions in a quiz weigh same
- All quizzes (ca. 10 throughout semester) have the same weight
  - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam

Quiz #4 Solution: Spring Web MVC Framework

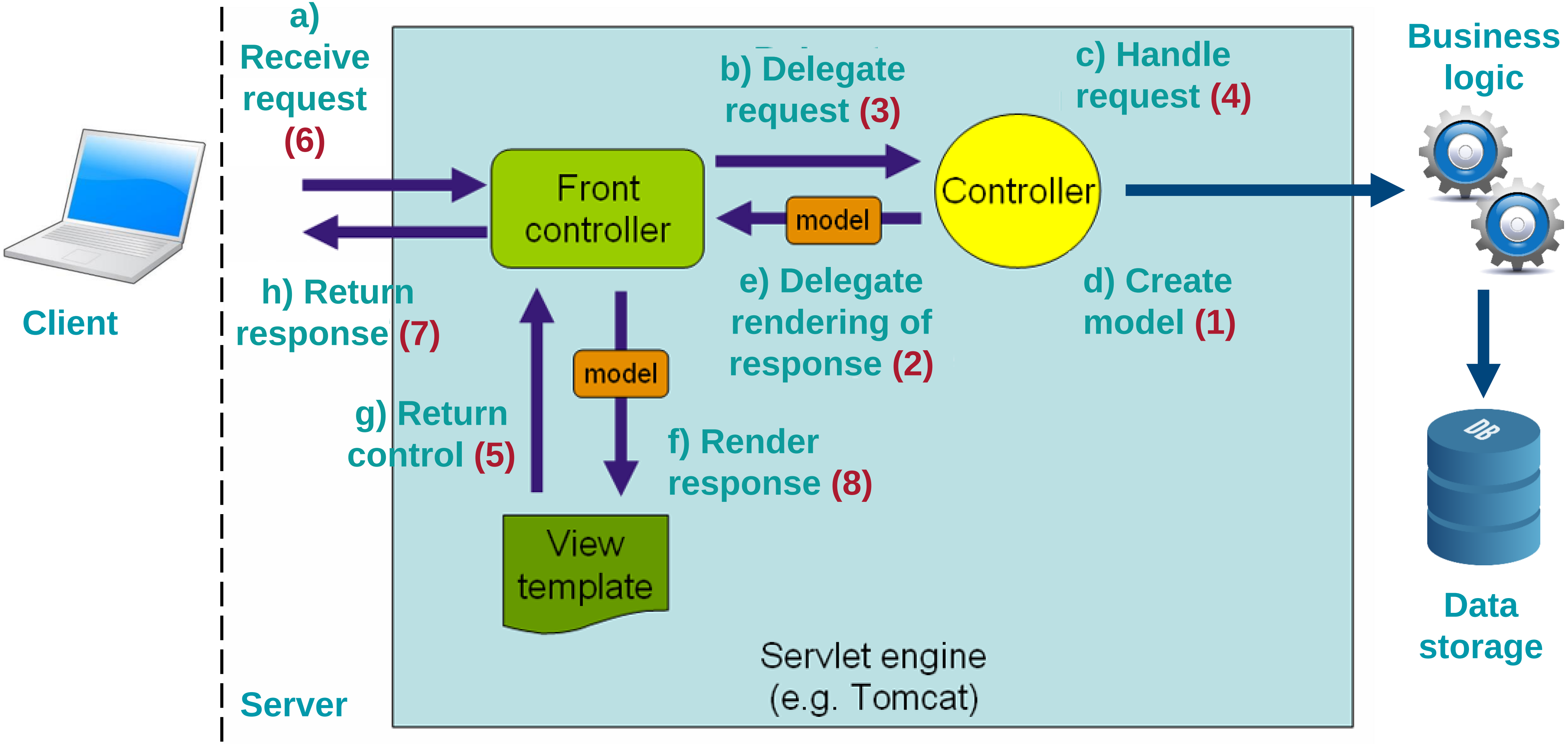
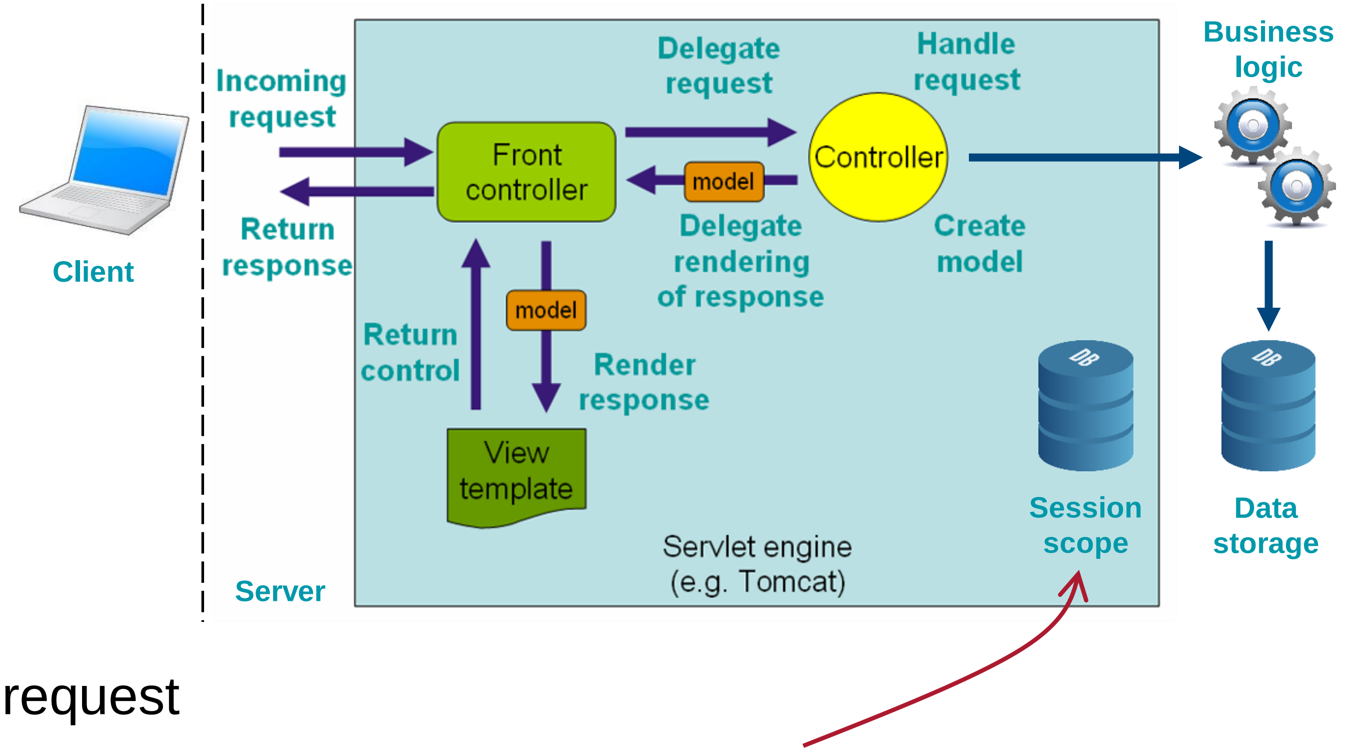


Figure adapted from Spring Framework Reference Documentation

Preserving State Between Requests

- Often, we want to maintain a certain state on the server between requests
  - i.e. keep data on the server that is specific to a particular user's session
    - but should not be transferred back and forth with each request
- Tedious approach: We could
  - store such data in the database and retrieve it every time, based on some user ID transferred with each request



- The Java Servlet API simplifies this by providing a session scope that
  - is associated with a user's requests automatically
  - lets us store and retrieve objects by name
  - exists only for the duration of user's session (i.e. from first to typ. 30 min. after last request)

Working with Session Attributes

```
@Controller
public class GameStateController {
  @RequestMapping(value="/update", method=RequestMethod.GET)
  public String stateUpdate(HttpSession session, Model model) {
    PlayerState ps = (PlayerState) session.getAttribute("playerstate");
    // [application logic providing Score object]
    session.addAttribute("myscore", score);
    // [application logic]
    session.removeAttribute("tempState");
    model.addAttribute("userinfo", new UserInfo());
    return "Game";
  }
}
```

Make HTTP session available in request handler

Retrieve object stored under given label from the session

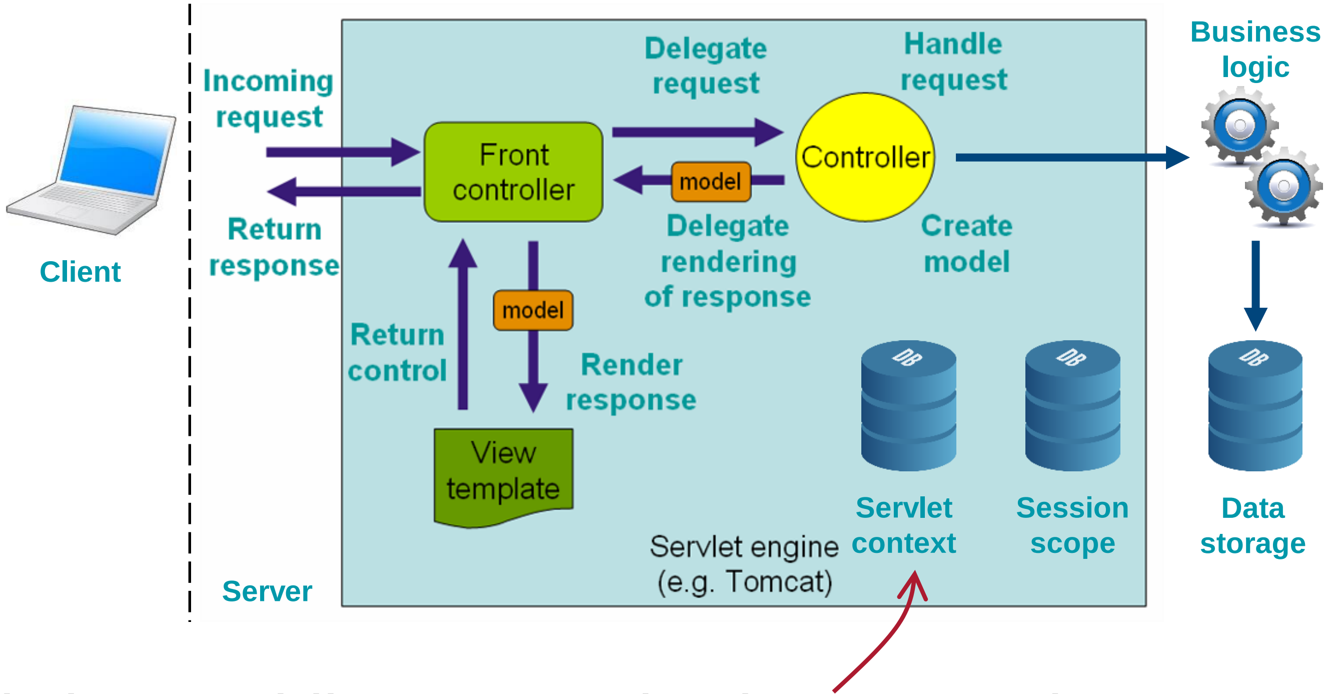
Store given object under given label in the session

Remove object stored under given label from session

Sharing Data Across Sessions

- Sometimes, we want to make certain information available to all sessions of an application

- Tedious approach: We could
  - store such data in the database and retrieve it anytime we need it
  - but it is not object-oriented there



- The Java Servlet API simplifies this by providing an application scope that
  - is available in sessions of all users
  - lets us store and retrieve objects by name
  - exists as long as the application is deployed on the server and the server is running

Working with the Servlet Context

```
@Controller
public class GameStateController {
  @RequestMapping(value="/next", method=RequestMethod.GET)
  public String nextRound(HttpSession session, Model model) {
    ServletContext context = session.getServletContext();
    GameState gs = (GameState) context.getAttribute("gamestate");
    // [application logic providing Score object]
    context.addAttribute("hiscore", score);
    // [application logic]
    context.removeAttribute("someState");
    return "Start";
  }
}
```

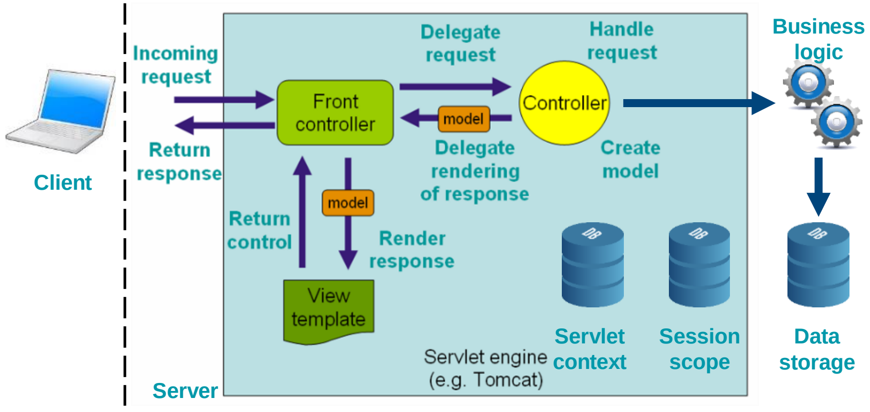
Make application context available in request handler

Retrieve object stored under given label from context

Store given object under given label in context

Remove object stored under given label from context

Summary: Spring Web MVC Basics



- Build `@Controller`s to respond to different requests
- Use `@RequestMapping`s to define which controller method should react to which URI
- Extract input from requests using
  - `@RequestParameter`s for single request parameters
  - `@PathVariable`s for parts of the URI path
  - `@ModelAttribute`s for parameters describing objects
- Store and retrieve information spanning multiple requests in the `HttpSession`
- Store and retrieve information available to all sessions in the `ServletContext`
- Invoke application logic in regular Java classes from controller
- Store information to be provided to the next view in the `Model`
- Specify the next view in the return value of the request handler
- Construct views as JSPs that incorporate information from the `Model`
- Or use a `@RestController` to respond with JSON data instead of a view

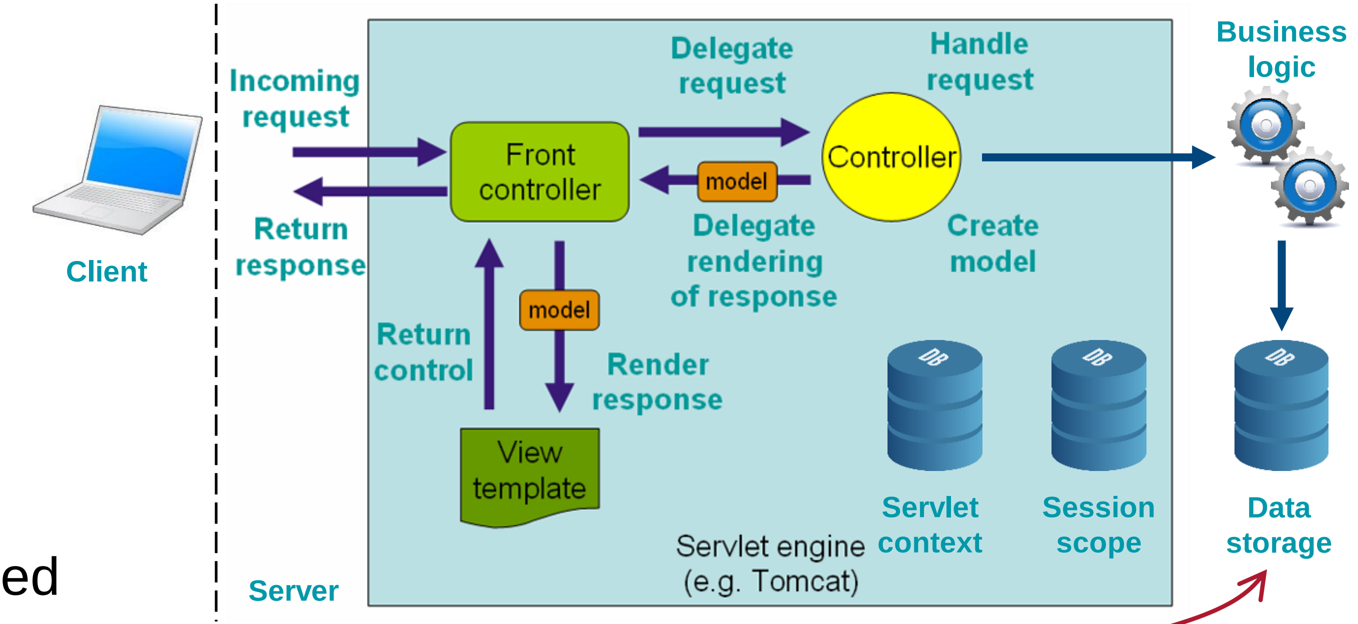
Java Persistence API

see also:

- Williams: Professional Java for Web Applications, Ch. 19-22, 24
- http://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Persistent Data Storage

- Some data is not suitable for storage in the servlet engine’s scopes, e.g.
  - data that shall be stored even when the server is down
  - data that is too large to be kept in memory
  - data that is most efficiently stored and retrieved in a non-object-oriented structure (e.g. relational data)
  - data that is retrieved from external sources



- For these purposes, a database or other data sources can be accessed from the business logic
  - Persistence frameworks can help with the mapping of objects to database structures

Recap from HBV401G: Object-Relational Mapping (ORM)

- All our object-oriented data structures exist in memory at run-time.
- However, we also need data structures outside our program…
  - to preserve information while the system is not running
  - to work with data structures that are larger than available memory
  - to exchange information with other (remote) systems

- There are a number of solutions for this
  - e.g. databases, XML, JSON, binary files…
- Most of them are not (or not fully) object-oriented though

- **Challenge: Object-Relational Mapping**
  - Transforming object-oriented data structures into a non-object-oriented persistent format

Motivation: Database Access without ORM

```
public Product getProduct(long id) throws SQLException {
  try (Connection c = this.getConnection();
       PreparedStatement s = c.prepareStatement(
         "SELECT * FROM dbo.Product WHERE productId = ?")) {
    s.setLong(1, id);
    try (ResultSet r = s.executeQuery()) {
      if (!r.next()) return null;
      Product p = new Product(id);
      p.setName(r.getNString("Name"));
      p.setDatePosted(r.getObject("DatePosted", Instant.class));
      p.setPrice(r.getDouble("Price"));
      // ...mapping a dozen more attributes...
      return p;
    }
}
```

Need to identify objects through their primary keys *and* OO references

Need to deal with connection technicalities

Need to map data types, object and table structures

Need to pick apart query results and piece together objects field by field

Need to deal with linked entities (efficient retrieval, cascading deletions etc.)

Need to maintain similarly complex code for creating and updating entities
- tedious
- ideal breeding ground for bugs

Database Access Implementation Options

- Previously, you may likely have done this manually
  - Connect to database
  - Formulate SQL statements
  - Map data back and forth between objects and relational structures
- In this lecture, we will see how to abstract from most of the technical steps
  - Just let the Java Persistence API (JPA) know which objects you want to be persistent…
  - …and what information you want to retrieve from the database
  - Necessary database operations will be executed automatically

- Pros and cons
  - Easy-to-use standard database operations without a lot of technical overhead
  - More complex non-standard queries still require manual work
  - A manual implementation that does exactly what you need may be more efficient

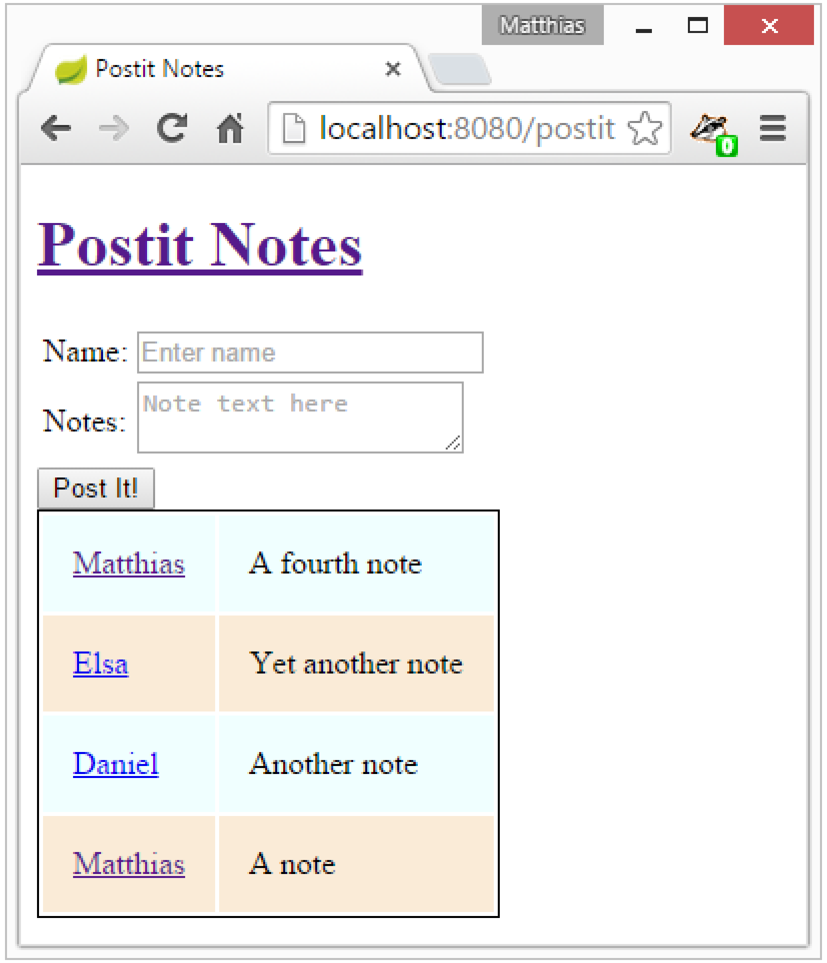
Technology Stack

- Structured Query Language (SQL) *(→ TÖL303G)*
  - Language for formulating queries of relational databases
- Java Database Connectivity (JDBC) *(→ Williams, Ch. 19)*
  - API for creating connections to a variety of databases (through appropriate JDBC drivers) and sending SQL queries to them
- Object-Relational Mapper (ORM) *(→ Williams, Ch. 19)*
  - Framework taking care of the mapping of object structures to relational structures, formulating suitable queries etc. Popular ORMs: Hibernate, MyBatis, EclipseLink…
- Java Persistence API (JPA) *(→ Williams, Ch. 20, 21)*
  - Additional layer abstracting from any particular ORM implementation
- Spring Data JPA *(→ Williams, Ch. 22, 24)*
  - Extension of JPA providing convenient methods for query formulation (among other things)

The Skeleton App’s PostIt Demo

- The skeleton app includes a small “PostIt” demo.
  - See “Spring Boot Intro” slides in Verkefni folder in Ugla for setup instructions

- After starting your database server and running `Application.main` as usual, you can play with a demo of the persistence layer at http://localhost:8080/postit
  - Note that data you enter here remains persistent even after you restart the web application!



Postit Notes

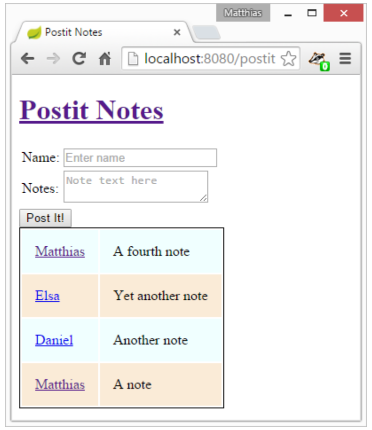
Name: Enter name

Notes: Note text here

Post It!

Matthias	A fourth note
Elsa	Yet another note
Daniel	Another note
Matthias	A note

PostIt View, Part 1: Data Entry Form (PostitNotes.jsp)



Postit Notes

Name: Enter name

Notes: Note text here

Post It!

Matthias	A fourth note
Elsa	Yet another note
Daniel	Another note
Matthias	A note

```
<!DOCTYPE html>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form" %>
<html lang="en">
  <head><!-- ... --></head>
  <body>
    <sf:form method="POST" commandName="postitNote" action="/postit">
      <table><tr>
        <td>Name:</td><td><sf:input path="name" type="text" placeholder="Enter name"/></td>
      </tr><tr>
        <td>Notes:</td><td><sf:textarea path="note" type="text" placeholder="Note text here"/></td>
      </tr></table>
      <input type="submit" value="Post It!"/>
    </sf:form>
    <!-- ... -->
```

Definitions of custom tags to be used in construction of HTML code (“smart” tags interpreted by server)

Name of @ModelAttribute in which we expect the data

Name of entity attribute in which to store the input

Persistent Data Entity
(PostitNote)

```
@Entity
@Table(name = "postitnote")
public class PostitNote {

  @Id
  @GeneratedValue(strategy =
      GenerationType.IDENTITY)
  private Long id;

  private String name;
  private String note;

  public PostitNote() {
  }
  public PostitNote(
      String name, String note) {
    this.name = name;
    this.note = note;
  }
  public Long getId() {
    return id;
  }
  public void setId(Long id) {
    this.id = id;
  }
  // [...other getters & setters...]
}
```

Indicates that O/R mapping shall be performed for instances of this class

Optional: Name of DB table to use for entity (default: class name)

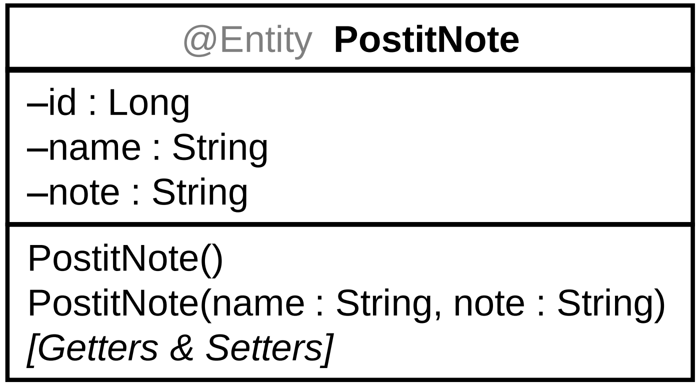
Indicates that the following attribute shall contain the primary key, and that it shall be populated with unique values

Required in order to create the object populated by the web form

Note: Initializing the id attribute is being taken care of automatically due to `@GeneratedValue`

All attributes with getters and setters will turn into DB table columns

Design Model of Skeleton App



@Entity **PostitNote**

–id : Long
–name : String
–note : String

PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]

Data Type Mappings

Java types	**SQL types** (one of listed or equivalent, depending on DB)
short, Short	SMALLINT, INTEGER, BIGINT
int, Integer	INTEGER, BIGINT
long, Long, BigInteger	BIGINT
float, Float, double, Double, BigDecimal	DECIMAL
byte, Byte	BINARY, SMALLINT, INTEGER, BIGINT
char, Character	CHAR, VARCHAR, BINARY, SMALLINT, INTEGER, BIGINT
boolean, Boolean	BOOLEAN, BIT, SMALLINT, INTEGER, BIGINT, CHAR, VARCHAR
byte[], Byte[]	BINARY, VARBINARY
char[], Character[], String	CHAR, VARCHAR, BINARY, VARBINARY
Date, Calendar (with @Temporal annotation)	DATE, TIME, DATETIME
enum	SMALLINT, INTEGER, BIGINT, CHAR, VARCHAR
Serializable	VARBINARY (object stored in serialized form)

Request Handling, Part 1: Submitting a New PostIt

(PostitNoteController)

```
@RequestMapping(value = "/postit", method = RequestMethod.POST)
public String postitNoteViewPost(@ModelAttribute("postitNote") PostitNote postitNote,
                                 Model model) {

  postitNoteService.save(postitNote);

  model.addAttribute("postitNote", new PostitNote());

  model.addAttribute("postitNotes", postitNoteService.findAllReverseOrder());

  return "postitnotes/PostitNotes";

}
```

Doing business logic with the postitNote received in the request (here: saving it)

Preparing a new, empty postitNote to populate the form with in the response

Getting data from the business logic (here: the list of existing PostIts to display)

Request Handling, Part 1: Submitting a New PostIt

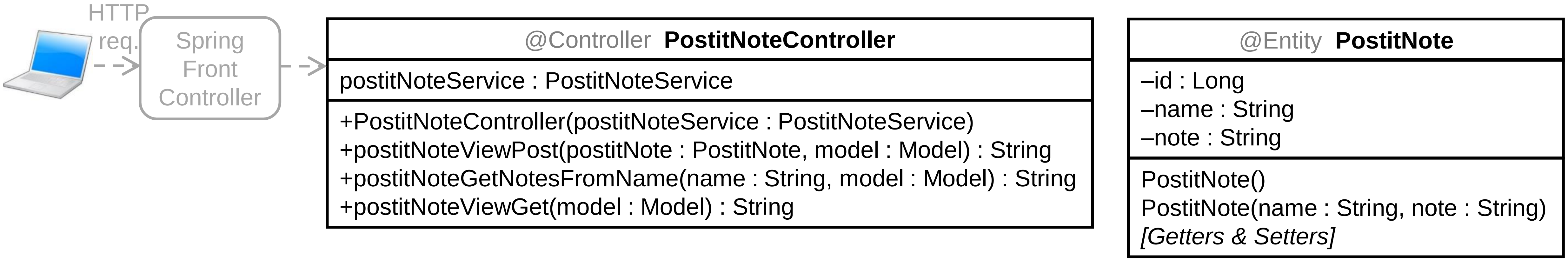
(PostitNoteController)

```
@Controller
public class PostitNoteController {
  PostitNoteService postitNoteService;

  @Autowired
  public PostitNoteController(PostitNoteService postitNoteService) {
    this.postitNoteService = postitNoteService;
  }

  @RequestMapping(value = "/postit", method = RequestMethod.POST)
  public String postitNoteViewPost(@ModelAttribute("postitNote") PostitNote postitNote,
                                   Model model) {
     // ...
```

Obtain an instance of the service providing required business logic. Note:

- Dependency injection: Controller does not instantiate the class it depends on, but expects to receive it from outside
- `@Autowired` indicates that Spring takes care of instantiating and providing (injecting) the service, so we don't have to worry about when and where to do this

Design Model of Skeleton App



@Controller **PostitNoteController**

postitNoteService : PostitNoteService

+PostitNoteController(postitNoteService : PostitNoteService)
+postitNoteViewPost(postitNote : PostitNote, model : Model) : String
+postitNoteGetNotesFromName(name : String, model : Model) : String
+postitNoteViewGet(model : Model) : String

@Entity **PostitNote**

–id : Long
–name : String
–note : String

PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]

Declaration of Business Service (PostitNoteService)

```
import project.persistence.entities.PostitNote;
import java.util.List;

public interface PostitNoteService {

  PostitNote save(PostitNote postitNote);
  void delete(PostitNote postitNote);
  List<PostitNote> findAll();
  List<PostitNote> findAllReverseOrder();
  PostitNote findOne(Long id);
  List<PostitNote> findByName(String name);

}
```

Declaration of various functionalities that our business logic offers

Implementation of Postit Handling Services (PostitNoteServiceImplementation)

```
@Service
public class PostitNoteServiceImplementation
    implements PostitNoteService {
  PostitNoteRepository repository;

  @Autowired
  public PostitNoteServiceImplementation(
      PostitNoteRepository repository) {
    this.repository = repository;
  }

  @Override
  public PostitNote save(PostitNote postitNote) {
    return repository.save(postitNote);
  }
  @Override
  public void delete(PostitNote postitNote) {
    repository.delete(postitNote);
  }
```

```
@Override
public List<PostitNote> findAll() {
  return repository.findAll();
}
@Override
public PostitNote findOne(Long id) {
  return repository.findOne(id);
}
@Override
public List<PostitNote> findByName(String name) {
  return repository.findByName(name);
}
@Override
public List<PostitNote> findAllReverseOrder() {
  List<PostitNote> postitNotes =
    repository.findAll();
  Collections.reverse(postitNotes);
  return postitNotes;
}
```

```
}
```

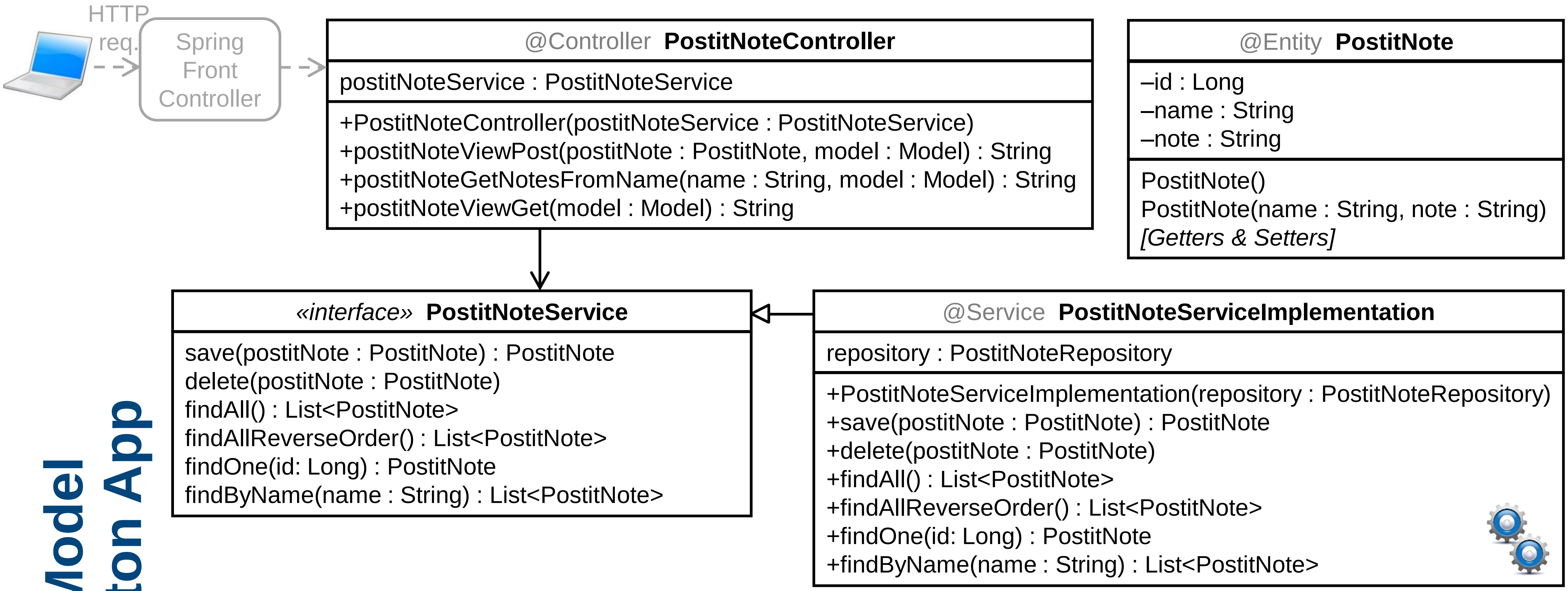
Indicates this is a business logic implementation

Obtain data repository through dependency injection

Business logic implementation, heavily relying on repository in this case

This line does everything we had to code manually on slide 13!

Design Model of Skeleton App



@Controller **PostitNoteController**

postitNoteService : PostitNoteService

+PostitNoteController(postitNoteService : PostitNoteService)
+postitNoteViewPost(postitNote : PostitNote, model : Model) : String
+postitNoteGetNotesFromName(name : String, model : Model) : String
+postitNoteViewGet(model : Model) : String

@Entity **PostitNote**

–id : Long
–name : String
–note : String

PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]

«interface» **PostitNoteService**

save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllReverseOrder() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>

@Service **PostitNoteServiceImplementation**

repository : PostitNoteRepository

+PostitNoteServiceImplementation(repository : PostitNoteRepository)
+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllReverseOrder() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>

Configuration of Data Repository
(PostitNoteRepository)

```
public interface PostitNoteRepository extends JpaRepository<PostitNote, Long> {
  PostitNote save(PostitNote postitNote);
  void delete(PostitNote postitNote);
  List<PostitNote> findAll();

  @Query(value = "SELECT p FROM PostitNote p where length(p.name) >= 3")
  List<PostitNote> findAllWithNameLongerThan3Chars();

  List<PostitNote> findAllByOrderByIdDesc();
  PostitNote findOne(Long id);
  List<PostitNote> findByName(String name);
}
```

Interface declaration determines what functionality the repository should offer

If you need particular kinds of queries, you can

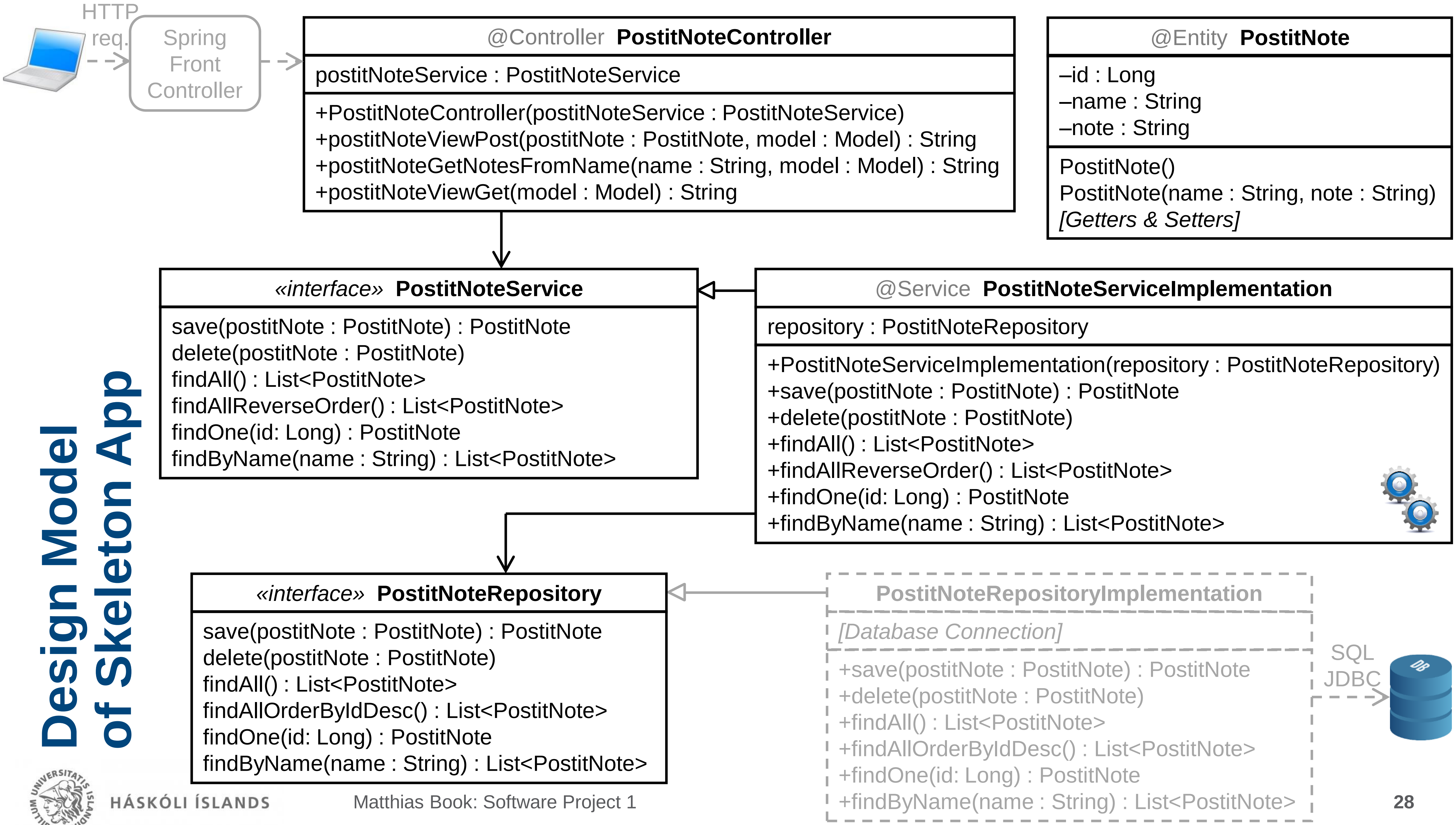
- define many typical ones by using appropriate keywords in your method names (see following two slides)
- or use the `@Query` annotation to create individual queries

Note: Implementation of this interface will automatically be provided by the Java Persistence API (JPA)!

HÁSKÓLI ÍSLANDS

For further query options, see http://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Design Model of Skeleton App



HTTP req.

Spring Front Controller

@Controller **PostitNoteController**

postitNoteService : PostitNoteService

+PostitNoteController(postitNoteService : PostitNoteService)
+postitNoteViewPost(postitNote : PostitNote, model : Model) : String
+postitNoteGetNotesFromName(name : String, model : Model) : String
+postitNoteViewGet(model : Model) : String

@Entity **PostitNote**

–id : Long
–name : String
–note : String

PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]

«interface» **PostitNoteService**

save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllReverseOrder() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>

@Service **PostitNoteServiceImplementation**

repository : PostitNoteRepository

+PostitNoteServiceImplementation(repository : PostitNoteRepository)
+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllReverseOrder() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>

«interface» **PostitNoteRepository**

save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllOrderByIdDesc() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>

PostitNoteRepositoryImplementation

[Database Connection]

+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllOrderByIdDesc() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>

SQL
JDBC

Keywords in Query Method Names

Keyword in method name	Example	Resulting WHERE clause in generated query
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull()	… where x.age is null
IsNotNull, NotNull	findByAge[Is]NotNull()	… where x.age not null

http://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

Keywords in Query Method Names

Keyword	Example	Resulting WHERE clause in generated query
`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended %)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended %)
`Containing`	`findByFirstnameContaining`	`… where x.firstname like ?1` (parameter bound wrapped in %)
`OrderBy`	`findByAgeOrderByLastnameDesc`	`… where x.age = ?1 order by x.lastname desc`
`Not`	`findByLastnameNot`	`… where x.lastname <> ?1`
`In`	`findByAgeIn(Collection<Age> ages)`	`… where x.age in ?1`
`NotIn`	`findByAgeNotIn(Collection<Age> ages)`	`… where x.age not in ?1`
`True`	`findByActiveTrue()`	`… where x.active = true`
`False`	`findByActiveFalse()`	`… where x.active = false`
`IgnoreCase`	`findByFirstnameIgnoreCase`	`… where UPPER(x.firstname) = UPPER(?1)`

http://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

In-Class Quiz #5: JPA Query Methods

Which JPA query method (1-8) will generate which SQL clause (a-h)?

```
SELECT f FROM Foo f WHERE...
a) f.bar = ?1
b) f.bar = ?1 AND f.baz = ?2
c) f.bar BETWEEN ?1 AND ?2
d) f.bar = ?1 ORDER BY f.baz DESC
e) f.bar <= ?1
f) f.bar IN ?1
g) UPPER(f.bar) = UPPER(?1)
h) f.bar = TRUE
```

(Consider the questions as independent – the type of bar varies between questions.)

```
List<Foo> ...
1. findByBarTrue()
2. findByBar(Bar bar)
3. findByBarIgnoreCase(Bar bar)
4. findByBarLessThanEqual(Bar bar)
5. findByBarOrderByBazDesc(Bar bar)
6. findByBarAndBaz(Bar bar, Baz baz)
7. findByBarIn(Collection<Bar> bars)
8. findByBarBetween(Bar bar1,
                   Bar bar2)
```

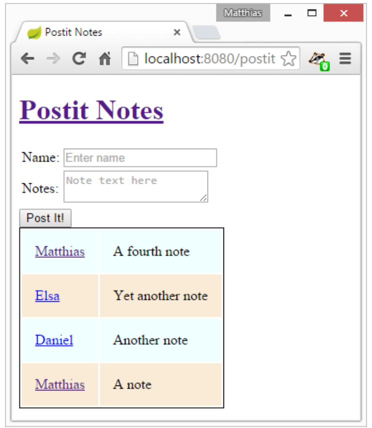
Recap: Request Handling, Part 1: Submitting New PostIt (PostitNoteController)

```
@RequestMapping(value = "/postit", method = RequestMethod.POST)
public String postitNoteViewPost(@ModelAttribute("postitNote") PostitNote postitNote,
                                 Model model) {

  postitNoteService.save(postitNote);

  model.addAttribute("postitNote", new PostitNote());

  model.addAttribute("postitNotes", postitNoteService.findAllReverseOrder());

  return "postitnotes/PostitNotes";

}
```

Next, let's see how these are displayed to the user…

Postit View, Part 2: Postit List

(PostitNotes.jsp)



Postit Notes

Name: Enter name

Notes: Note text here

Post It!

Matthias	A fourth note
Elsa	Yet another note
Daniel	Another note
Matthias	A note

Control tags evaluated on **server** at time of HTML **construction**

```
<c:choose>
  <c:when test="${not empty postitNotes}">
    <table class="notes">
      <c:forEach var="postit" items="${postitNotes}">
        <tr>
          <td><a href="/postit/${postit.name}">${postit.name}</a></td>
          <td>${postit.note}</td>
        </tr>
      </c:forEach>
    </table>
  </c:when>
  <c:otherwise>
    <h3>No notes!</h3>
  </c:otherwise>
</c:choose>
```

Loop through `postitNotes` provided in Model, make each available as `postit` and display its name (as link) and note attributes

Display message instead of table if `postitNotes` is an empty `List`

Request Handling, Part 2: Fetching Lists of PostIts

(PostitNoteController)

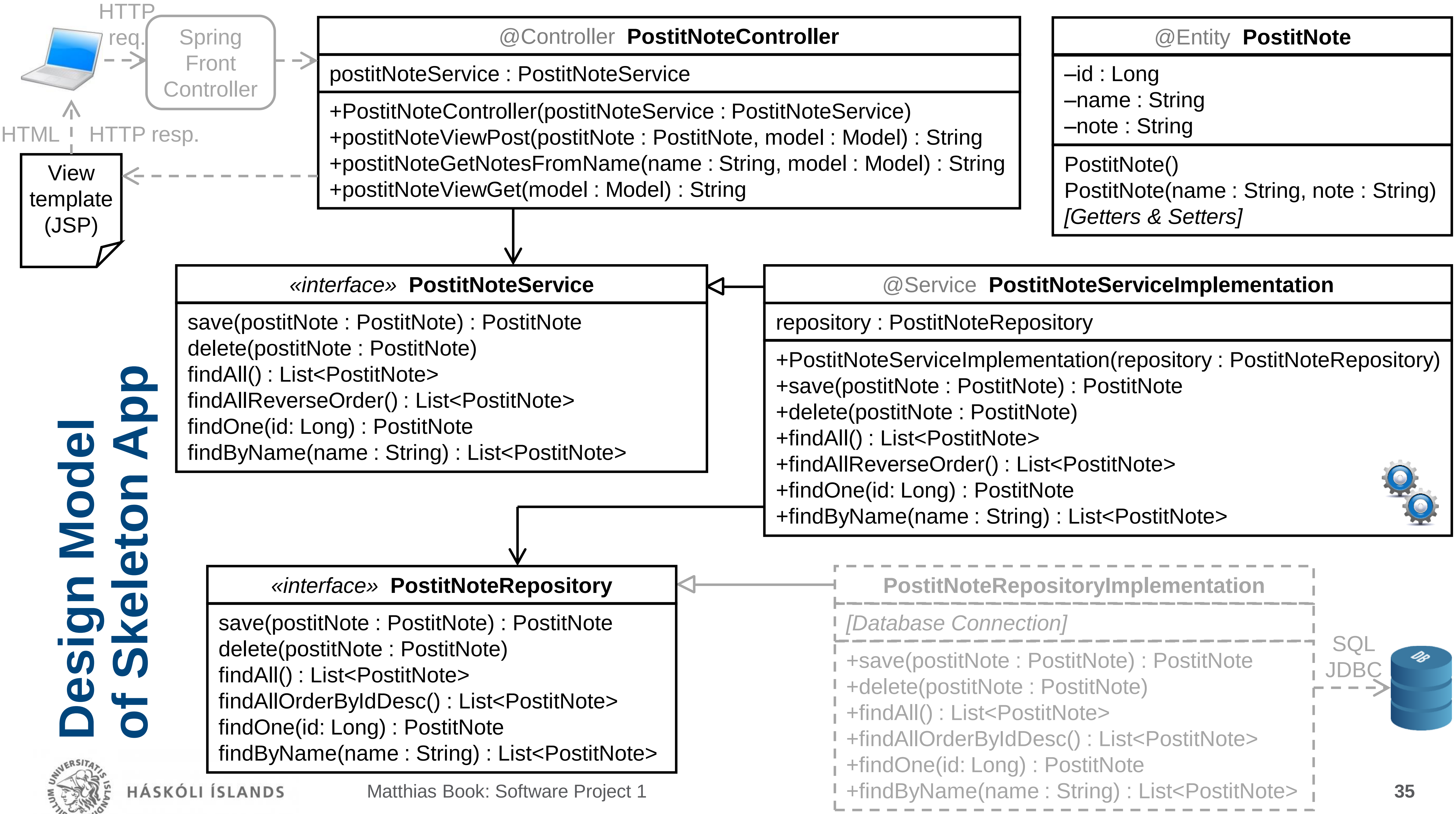
```
@RequestMapping(value = "/postit/{name}", method = RequestMethod.GET)
public String postitNoteGetNotesFromName(@PathVariable String name,
                                         Model model) {
  model.addAttribute("postitNotes", postitNoteService.findByName(name));
  model.addAttribute("postitNote", new PostitNote());
  return "postitnotes/PostitNotes";
}
```

If the URI contains a path variable, retrieve all PostIts with that name

```
@RequestMapping(value = "/postit", method = RequestMethod.GET)
public String postitNoteViewGet(Model model) {
  model.addAttribute("postitNote", new PostitNote());
  model.addAttribute("postitNotes", postitNoteService.findAllReverseOrder());
  return "postitnotes/PostitNotes";
}
```

Otherwise, retrieve all Postits in reverse order

Design Model of Skeleton App



@Controller **PostitNoteController**

postitNoteService : PostitNoteService

+PostitNoteController(postitNoteService : PostitNoteService)
+postitNoteViewPost(postitNote : PostitNote, model : Model) : String
+postitNoteGetNotesFromName(name : String, model : Model) : String
+postitNoteViewGet(model : Model) : String

@Entity **PostitNote**

–id : Long
–name : String
–note : String

PostitNote()
PostitNote(name : String, note : String)
[Getters & Setters]

«interface» **PostitNoteService**

save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllReverseOrder() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>

@Service **PostitNoteServiceImplementation**

repository : PostitNoteRepository

+PostitNoteServiceImplementation(repository : PostitNoteRepository)
+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllReverseOrder() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>

«interface» **PostitNoteRepository**

save(postitNote : PostitNote) : PostitNote
delete(postitNote : PostitNote)
findAll() : List<PostitNote>
findAllOrderByIdDesc() : List<PostitNote>
findOne(id: Long) : PostitNote
findByName(name : String) : List<PostitNote>

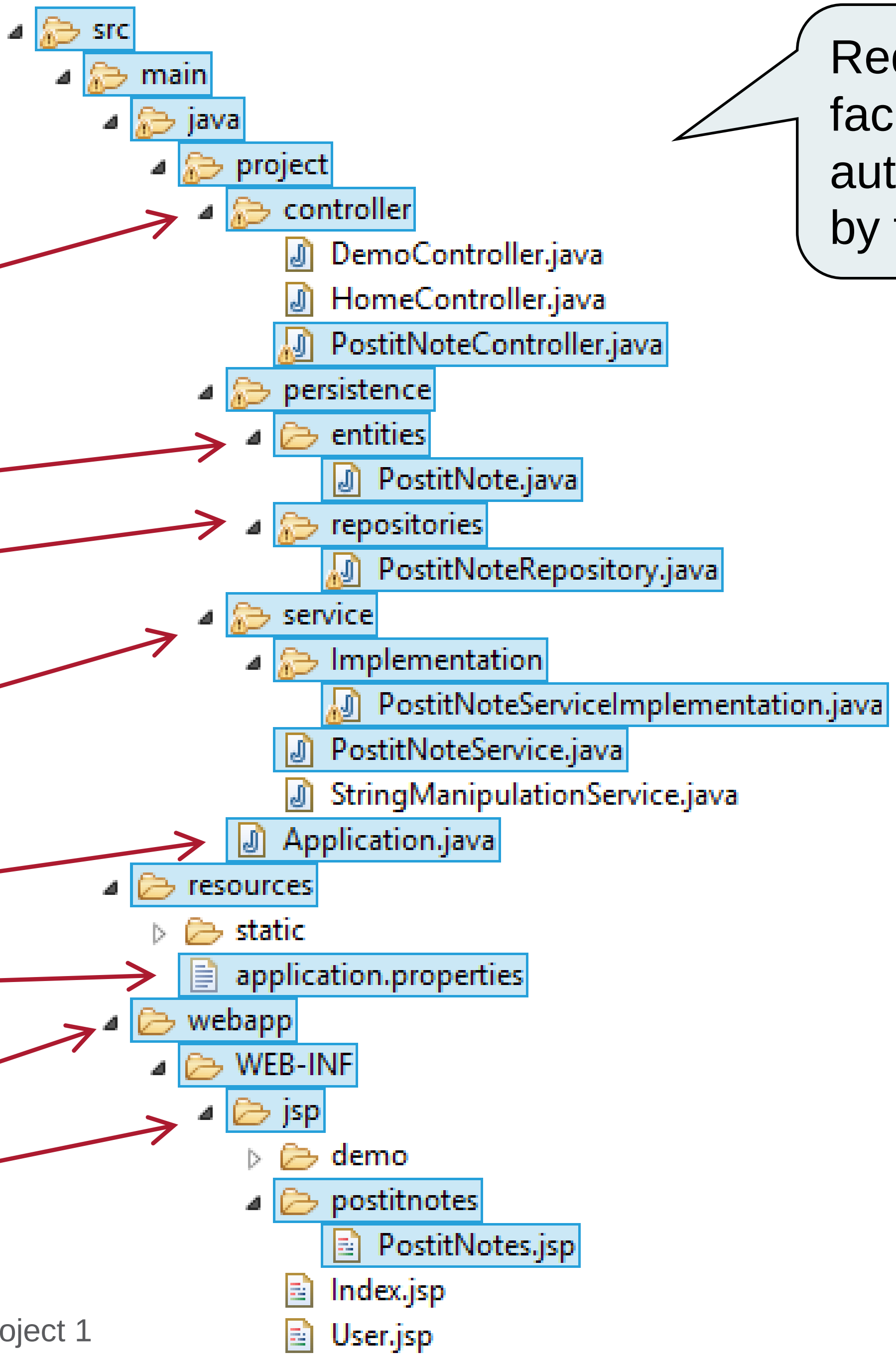
PostitNoteRepositoryImplementation

[Database Connection]

+save(postitNote : PostitNote) : PostitNote
+delete(postitNote : PostitNote)
+findAll() : List<PostitNote>
+findAllOrderByIdDesc() : List<PostitNote>
+findOne(id: Long) : PostitNote
+findByName(name : String) : List<PostitNote>

Project Structure

- **Controller Layer**
  - Request handlers
- **Persistence Layer**
  - Data entities
  - Data repositories
- **Business Layer**
  - Business logic classes
- Configuration
  - Spring Boot main class
  - Configuration file
- **View Layer**
  - Static web content
  - JavaServer Pages



```
src
  main
    java
      project
        controller
          DemoController.java
          HomeController.java
          PostitNoteController.java
        persistence
          entities
            PostitNote.java
          repositories
            PostitNoteRepository.java
        service
          Implementation
            PostitNoteServiceImplementation.java
          PostitNoteService.java
          StringManipulationService.java
        Application.java
    resources
      static
      application.properties
    webapp
      WEB-INF
        jsp
          demo
          postitnotes
            PostitNotes.jsp
          Index.jsp
          User.jsp
```

Required structure to facilitate discovery and autowiring of the classes by the Spring framework

Main Class
(Application)

Let JPA create implementations of repository interfaces automatically

```
@SpringBootApplication
@EnableJpaRepositories
public class Application extends SpringBootServletInitializer {

  @Override
  protected SpringApplicationBuilder configure(SpringApplicationBuilder applicationBuilder) {
    return applicationBuilder.sources(Application.class);
  }

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

}
```

Configuring Database Access
(application.properties)

Example shown for PostgreSQL; adapt accordingly for other database systems

```
# [...]

spring.datasource.url=jdbc:postgresql://localhost:5432/HBV

spring.datasource.username=[your DB username]
spring.datasource.password=[your DB password]

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
```

How to find the database:
[protocol]:[driver]://[host]:[port]/[dbname]

Authentication information

Driver for database connection

Let JPA take care of creating the database tables and updating the schema when we change the structure of our data entity classes. Note:

- Structural changes may corrupt existing data
- If it seems JPA can't keep up with your changes, try running the application *once* with this parameter set to `create` instead of update
- For prototyping only – in a production environment, you'll want to do this manually

Next Week’s Class Schedule

- I’ll be abroad for a seminar next week

- Next Thursday: Consultations will take place as planned
  - Andri and Daníel will answer questions of my teams as well

- Next Friday: No class, but will upload recorded lecture on database access and user interfaces based on JavaServer Pages

- Preview: Assignment #3 will be a UML class diagram of your project, due 28 Oct
- Meanwhile, start implementing your project based on the skeleton project available in Ugla already!