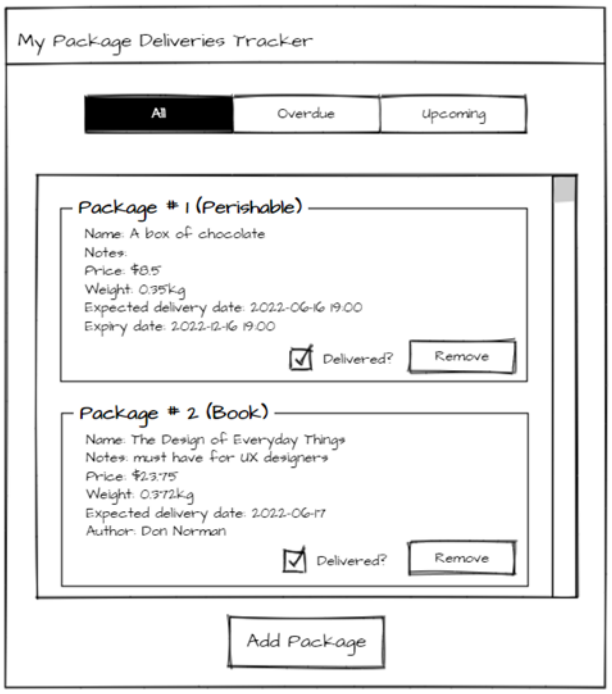
Assignment 4: Package Deliveries Tracker Web Service (25% of course total)

- Due date is **8 August, 11:59pm**. Refer to Canvas for any updates.
- Late penalty is 10% of max per calendar day (0 to 24 hour period past due), max. 2 days late.
- The assignment might be graded automatically. Make sure that your code compiles without warnings/errors (IntelliJ-specific warnings might be ok), and produces the required output. Also use the file names and structures indicated. Deviation from that might result in 0 mark.
- This assignment is to be done individually. Do not show another student your code, do not copy code from another person/online. Post all questions on Canvas (without your solution).
- You may use general ideas you find online and from others, but your solution must be your own.
- Your code MUST compile and run, or marks will be deducted. Unless otherwise specified, your code MUST start from an empty project in IntelliJ and use OpenJDK 18 (or any sub-versions).
- You may use any tool for development. But we will be grading your submission using IntelliJ IDEA Community, so make sure your code compiles and runs there. We support only IntelliJ.

Description

In this assignment you are going to keep the GUI version of the query system of packages you created in Assignment 3 and move the tracker portion into a web service. This system will allow a user to maintain a list of packages of 3 different types recording a small collection of their attributes at a web server.



Package # 1 (Perishable)
Name: A box of chocolate
Notes:
Price: $8.5
Weight: 0.35kg
Expected delivery date: 2022-06-16 19:00
Expiry date: 2022-12-16 19:00

Package # 2 (Book)
Name: The Design of Everyday Things
Notes: must have for UX designers
Price: $23.75
Weight: 0.372kg
Expected delivery date: 2022-06-17
Author: Don Norman

*Figure 1. A mock-up of the Package Deliveries Tracker GUI.**

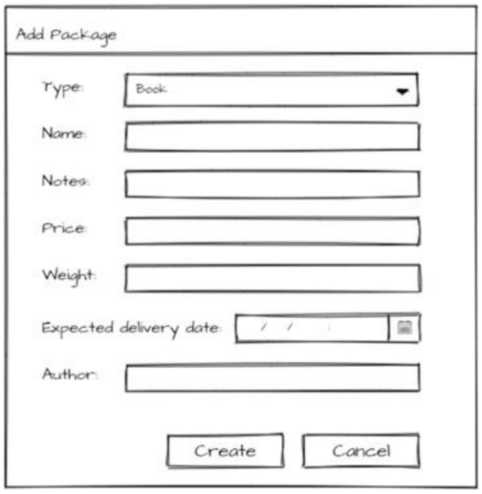


*Figure 2. A mock-up of a dialog for adding a book.**

*these figures are just mock-ups. Yours don't need to match exactly but needs to be of equal quality. You however need to include a few specific UI elements, refer to the rest of this document for details.

Overall Requirements

You are expected to base this assignment on your Assignment 3. However, the part where the package items are managed is now handled by a web server, and the GUI now becomes a web client handling user input. Information between server and client will be transferred using the REST API.

The Web Client Side

From the perspective of the user, it is basically the same interface, except now the client needs to communicate with the server every time the user performs an action that would affect the list of packages, and the server will respond with the update. It is up to you to design what needs to be sent and what can be done within the client (e.g., input error by the user).

You must have (at least) the following set of classes (in separate .java files):

- A set of classes for holding package information. Reuse the Package-Book/Perishable/Electronic class hierarchy (you can also reuse the Factory design pattern to create them if you want)
    - Same as before, name may contain more than one word (e.g., “Peanut butter cookies”), so is notes (e.g., “Contains nuts”). Only the notes and author of Books can be empty.
    - While you are implementing a GUI version here, this class must still correctly implement toString() to return all the field values as a String object, as discussed in lecture.
- A class for managing packages
    - Has a field to store all the packages (via an ArrayList) as received from the web server.
    - Has methods to perform operations on the packages-list as described in the next section.
        - These operations will directly (or indirectly through class(es) you define) incorporate code that communicates with the web server affecting the list.
    - NO LONGER loads and saves items in JSON format as in Assignments 1-3.
- A class for displaying the user interface
    - Uses the Java Swing containers and components to construct the UI
    - Implements event listeners to handle user inputs
- A class for the main application
    - Contains a main method doing only one thing: call SwingUtilities.invokeLater(Runnable) to start the GUI. For details refer to slides on Java GUI.

You need to create a separate IntelliJ project (from empty) for the web client and properly place your files in the following package structure:

- All your classes that are used to generate/represent a package are in cmpt213.assignment4.packagedeliveries.client.model
- All your classes that are used to manage packages are in cmpt213.assignment4.packagedeliveries.client.control
- All your classes that are used to display the user interface are in cmpt213.assignment4.packagedeliveries.client.view
- The class containing the main function is in cmpt213.assignment4.packagedeliveries.client

Again, this class placement is an example of the **Model-View-Controller architecture** where the model classes represent data (packagess) and handle their logic, the view classes determine how information are displayed to the user, and the controller classes control the data flow between the model and view.

Even though in this assignment the web client does not have to save the items-list, you still need to customize the closing behaviour of the application window (top-level JFrame) to trigger a save operation at the web server. Refer to Assignment 3 for details on how to do the customization.

The Web Server Side

The server is responsible for handling all operations sent from the client, for example, adding/removing a package, and returning a list of overdue packages. The packages-list is maintained by the server.

The web server is built using Spring Boot REST API, **and you must create it in IntelliJ as a Maven project**. It must respond to these messages (except for adding a new item which allows some variations) from the client using HTTP requests (you can test them from a command prompt or terminal using curl):

HTTP method & URL	Description
GET /ping	Return a string greeting the user saying: **System is up!**
GET /listAll	Return all packages as a JSON object.

POST /addPackage
 	
*note: this is just one possible idea, you can change the URL and the body to fit your design. One possibility could be using more specific paths, like
POST /addPackage/book
POST /addPackage/perishable
POST /addPackage/electronic
 	
or simply
POST /addBook
POST /addPerishable
POST /addElectronic
	Add a new package. Body of request is a JSON object representing the package, for example,
	{
	"type": "Book",
	"name": "The Design of Everyday Things",
	"notes": "must have for UX designers",
	"price": 23.75,
	“weight”: 0.372,
	"expectedDeliveryDate": "2022-06-17T10:40",
	"author": "Don Norman"
	}
	• The actual content of the request depends on how you implement the book/perishable/electronic classes.
	• Return the updated packages-list as a JSON object.

POST /removePackage	Remove a package from the system • The actual mechanism of determining which package to remove is up to you. In the data you can send an id, or a copy of the book/perishable/electronic object. • Return the updated packages-list as a JSON object.
GET /listOverduePackage	Return all undelivered packages with expected delivery date before the current date (and time), ordered by their expected delivery dates (oldest first).
GET /listUpcomingPackage	Return all undelivered packages with expected delivery dates on or after the current date (and time), ordered by their expected delivery dates (oldest first)
GET /markPackageAsDelivered	Toggle between a package being delivered or not. • Similar to removePackage, actual mechanism is up to you. • Return the updated packages-list as a JSON object.
GET /exit	Triggers the server to save the current packages-list into a JSON file (the client does not save anything).

More details:

- All data exchanged between the server and client is in JSON format (except for /ping which is returned as just plain text, and you can decide what /exit returns).
- Unless otherwise specified, HTTP status of successful GET responses are 200 (OK) and POST are 201 (created).

You need to create a separate IntelliJ project for the web server and properly place your files in the following package structure (refer to the Creating A REST API Using Spring Boot in IntelliJ section):

- All your classes that respond to REST messages are in cmpt213.assignment4.packagedeliveries.webappserver.controllers
- All your classes that are used to generate/represent a package are in cmpt213.assignment4.packagedeliveries.webappserver.model
- All your classes that are used to manage items are in cmpt213.assignment4.packagedeliveries.webappserver.control

The server must be able to save the packages-list in JSON format (e.g., a JSON array object), as triggered by the exit action at the client side. When the server runs for the first time, there is no JSON file to load and this list is empty (because nothing has been added yet). When the user exits the system, the server automatically generates a JSON file (list.json) and saves it in your project folder (use a hard-coded relative path starting with ./). When the server runs again (and subsequently), this JSON file will be loaded and used to populate the list. Whenever the client exits, this file will be updated with the new information. Both the loading and saving are done automatically by the application – the user doesn't need to do anything to trigger that (besides the user closes the web client).

To help us understand how to communicate with the server directly via curl commands, create a "curlCommands.txt" text file describing how to perform each of the operations. In your IntelliJ project, right-click the project name in the Project view and select New > Directory, and name it "docs". You should see a new folder called "docs" along with other folders like "src". Put the text file there.

Graphical User Interface Requirements

The mock-ups in the first page are for illustration only. You are free to design how your UI looks like as long as all the operations are supported. However, some specific UI components need to be used.

The operations are the same as Assignment 3. Refer to the assignment description for details.

Again, your UI need not match the sample exactly, but it should be of equal quality and include all the required information, without the need of the console. **To help us understand how to use your UI, same as in Assignment 3, create a "userManual.txt" text file describing how to perform each of the operations**. In your IntelliJ project, right-click the project name in the Project view and select New > Directory, and name it "docs". You should see a new folder called "docs" along with other folders like "src". Put the text file there.

Coding Requirements

- Your code must conform to the programming style guide for this course; see course website.
- Each interface/class must be in their own .java file and have both class-level and method-level JavaDoc comments describing the corresponding purpose. Field-level comments are optional.

Suggestions

- Think about the design before you start coding.
    - List the classes you expect to create.
    - For each class, decide what its responsibilities will be.
    - Think through some of the required features. How will each of your classes work to implement this feature? Can you think of design alternatives?
    - What need to be sent as REST messages and in what format should the data be.
- Write your code in progressing level of details.
    - Your code does not need to be fully functional at the firs time. Start with just printing a message in a method to have the logic correct.
    - Use refractoring to improve your code in later stages.
- You can reuse some of the methods to save time. For example, think about how to populate all the items in a view in various operations that list the items.

Submission Instructions

- Submit a ZIP file of your project to the corresponding folder on Canvas. Name it using this format: **<firstname_lastname>_<studentID>_Assignment4.zip** (e.g., John_Smith_012345678_Assignment4.zip). See course website for directions on creating and testing your ZIP file for submission. **Deviation from that will result in mark deductions**. If you have any difficulties in submitting your file on Canvas, email it to the instructor. **Make sure you include the curlCommands.txt and userManual.txt files!**
- Since you are submitting 2 IntelliJ projects, first create a zip file for each project (<firstname_lastname>_<studentID>_Assignment4_client.zip, <firstname_lastname>_<studentID>_Assignment4_webServer.zip), then zip these 2 into your submission zip file.
- Email a copy to yourself at your SFU account before the deadline for save keeping.
- If you use any libraries they have to be included in the project as well.
- All submissions will automatically be compared for unexplainable similarities.

Curl Commands

The cURL tool is a command letting users to send and receive data between a client and a server. It is available in OSX, Linus, and recent Windows. It supports many options but here we only need to do simple GETs/POSTs. For example, the following sends a GET request to a localhost server.

```
curl -i -H "Content-Type: application/json" -X GET localhost:8080/hello
```

For details read the links and watch the videos in the Useful Resources section. You can choose to use this way for the client and server to communicate, or you can use the built-in Java HttpUrlConnection for the client to perform HTTP requests.

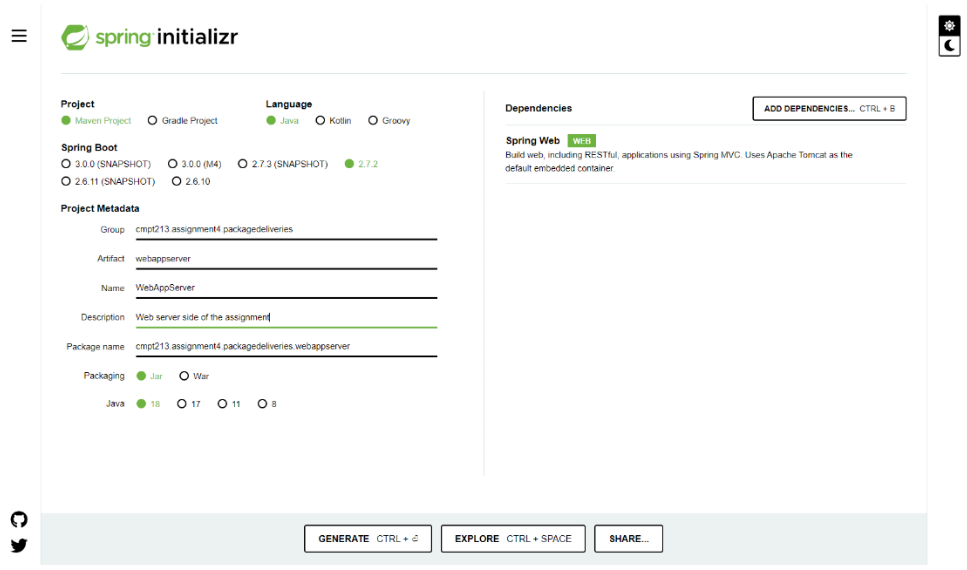
Creating A REST API Using Spring Boot in IntelliJ

Instead of creating from an empty project (which is the case for the Web Client), the Web Server must be created as a Maven project in IntelliJ using the method described here.

Go to https://start.spring.io/. This takes you to the Spring Initializr.

Use the Spring Initializr to download a **Spring web project** in a zip file. Choose Project as “Maven Project”, Language as “Java”, and Spring Boot as the latest release (just with the numbers, without snapshot/RC, …etc.). Make sure you add the “Spring Web” dependency.

For the Project Metadata section, input Group as “cmpt213.assignment4.packagedeliveries”, Artifact as “webappserver” (the Package name should be set automatically and you don’t have to change it), Name as “WebAppServer”, Description as “Web server side of the assignment”, Packaging as “Jar”, and Java as “18” (this means that for your JDK setting in IntelliJ it should also be 18).



Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M4) 2.7.3 (SNAPSHOT) [✓ 2.7.2]
2.6.11 (SNAPSHOT) 2.6.10

Project Metadata

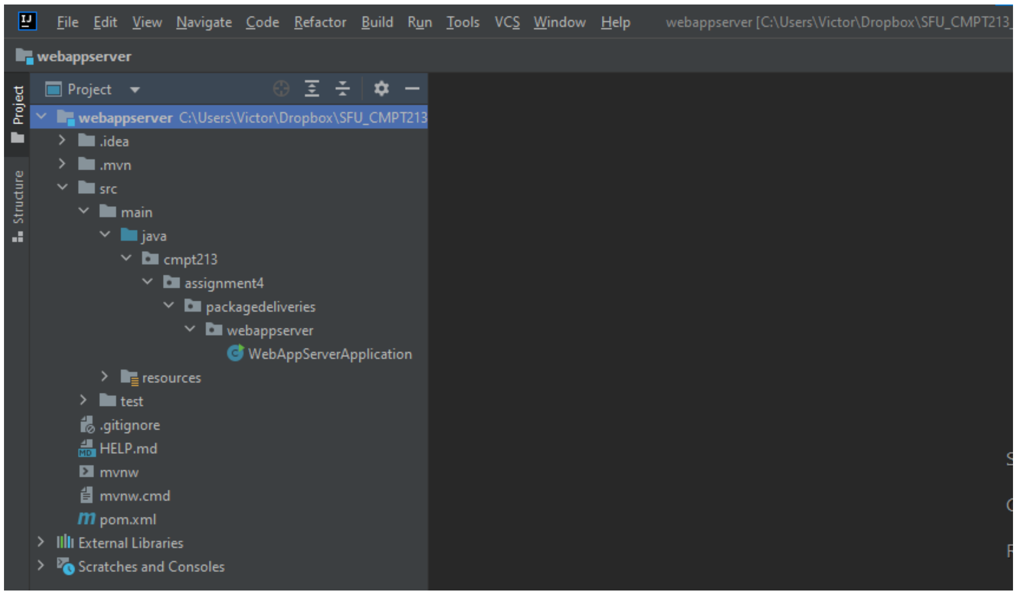
Group	cmpt213.assignment4.packagedeliveries
Artifact	webappserver
Name	WebAppServer
Description	Web server side of the assignment
Package name	cmpt213.assignment4.packagedeliveries.webappserver
Packaging	[✓ Jar] War
Java	[✓ 18] 17 11 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Click the GENERATE button at the bottom of the page and download the zip file.

Unzip the file and open this project in IntelliJ using File > Open… and let IntelliJ load the dependencies. When done, you’ll see something like this:


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
webappserver [C:\Users\Victor\Dropbox\SFU_CMPT213
webappserver
Project
Project
Structure

```
webappserver C:\Users\Victor\Dropbox\SFU_CMPT213
  > .idea
  > .mvn
  v src
    v main
      v java
        v cmpt213
          v assignment4
            v packagedeliveries
              v webappserver
                  WebAppServerApplication
      > resources
    > test
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
> External Libraries
> Scratches and Consoles
```

You can now work on this project just like any other IntelliJ projects. Start with the RestController covered in class. Do not modify the WebAppServerApplication.java file that comes with the download.

*Note: You might have to update your IntelliJ to run this REST API project as older versions of IntelliJ do not support JDK18 completely (e.g., you might see an error that says: Cannot determine path to 'tools.jar' library for 18). Version 2022.1.X should work.

Useful Resources

- If you choose to use the Jackson library that comes with Spring Boot to implement your JSON file I/O, take a look at this for how to read and write JSON objects/arrays: https://attacomsian.com/blog/jackson-read-write-json
- Making a REST API using Spring Boot in IntelliJ by Dr. Brian Fraser: https://www.youtube.com/watch?v=rXBsnNCH59o
- Building a RESTful Web Service: https://spring.io/guides/gs/rest-service/
- Using curl in Java (Method #3): https://www.baeldung.com/java-curl
- Java API of the LocalDateTime class: https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/time/LocalDateTime.html
  Tutorial on formatting numeric print output (look at how DecimalFormat is used to pad zero's): https://docs.oracle.com/javase/tutorial/java/data/numberformat.html
- Information on how to customize the closing behaviour of the application window https://docs.oracle.com/javase/tutorial/uiswing/events/windowlistener.html
- Information on how to use different kinds of dialogs in Java Swing https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html
- API of the LGoodDateTimePicker class: https://javadoc.io/doc/com.github.lgooddatepicker/LGoodDatePicker/latest/com/github/lgooddatepicker/components/DateTimePicker.html

Academic Honesty

It is expected that within this course, the highest standards of academic integrity will be maintained, in keeping with SFU's Policy S10.01, "Code of Academic Integrity and Good Conduct." In this class, collaboration is encouraged for in-class exercises and the team components of the assignments, as well as task preparation for group discussions. However, individual work should be completed by the person who submits it. Any work that is independent work of the submitter should be clearly cited to make its source clear. All referenced work in reports and presentations must be appropriately cited, to include websites, as well as figures and graphs in presentations. If there are any questions whatsoever, feel free to contact the course instructor about any possible grey areas.

Some examples of unacceptable behavior:

- Handing in assignments/exercises that are not 100% your own work (in design, implementation, wording, etc.), without a clear/visible citation of the source.
  - Modifying work you copied from elsewhere is not your own work.
- Using another student's work as a template or reference for completing your own work.
- Using any unpermitted resources during an exam.
- Looking at, or attempting to look at, another student's answer during an exam.

- Submitting work that has been submitted before, for any course at any institution.

All instances of academic dishonesty will be dealt with severely and according to SFU policy. This means that Student Services will be notified, and they will record the dishonesty in the student's file. Students are strongly encouraged to review SFU's Code of Academic Integrity and Good Conduct (S10.01) available online at: http://www.sfu.ca/policies/gazette/student/s10-01.html.