JAX-RS 2.1 AND BEYOND

ANDY MCCRIGHT – IBM

ANDYMC@US.IBM.COM

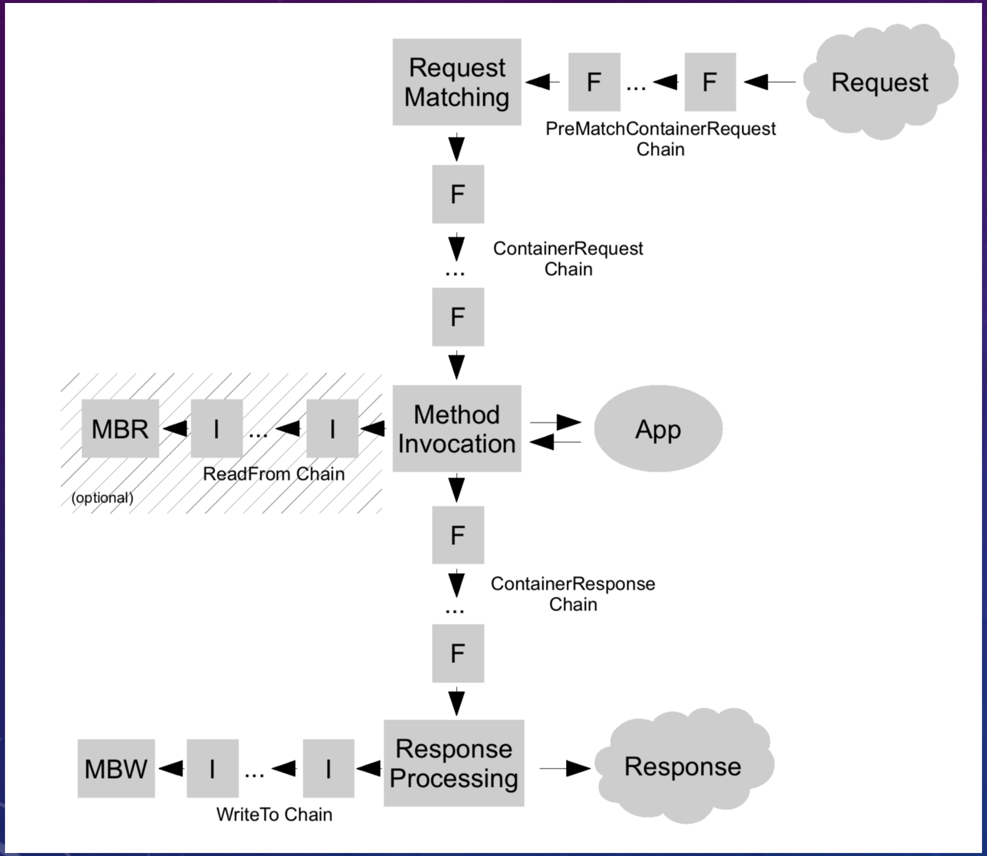
@ANDREWMCCRIGHT

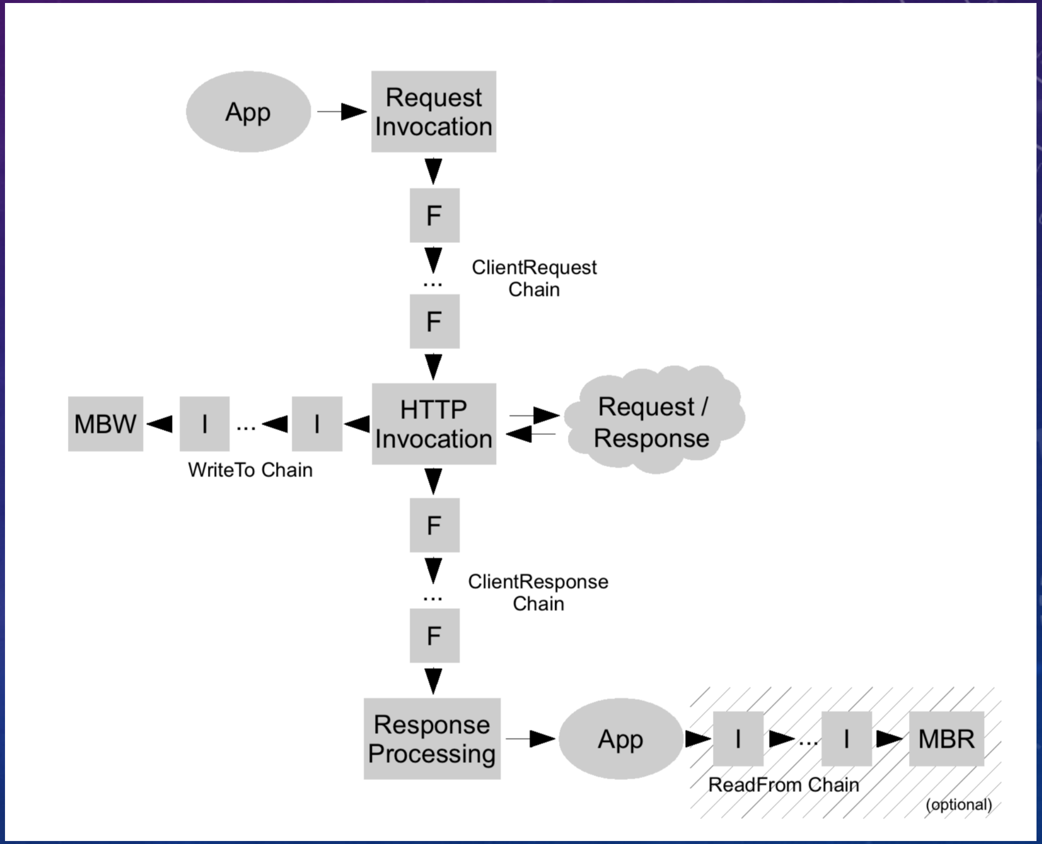
- What’s In JAX-RS 2.1?
  - Reactive Client
  - Server Sent Events
  - Other New APIs
- Beyond JAX-RS 2.1…
  - JAX-RS in Java SE
  - Type-safe Client APIs
  - Better CDI integration

JAX-RS REFRESHER

- JAX-RS provides RESTful services via Resources and Providers.
- Resources implement the business logic of a RESTful request.
- Providers handle infrastructure tasks, like converting HTTP entity data into objects and vice-versa (MessageBodyReaders/Writers), converting exceptions into HTTP responses (ExceptionMappers), filtering requests/responses (Filters), etc.

JAX-RS REFRESHER





Appendix C from the JAX-RS 2.1 Specification

REACTIVE CLIENT

- Introduces reactive programming model.
- Built on Java 8’s java.util.concurrent.CompletionStage
    - Allows multiple stages to be invoked in a particular order.
    - Allows handling of exceptions in-line.
    - Async-ready.
- The spec allows for optional support of other reactive platforms such as RxJava and Flow – not portable…

REACTIVE CLIENT DEMO

- Scenario: User runs a consultant shop and needs to schedule consultants based on their skill levels and availability.
- Two microservices apps:
  - Skills – organizes experts by their skill (rated 1=lowest to 5=highest).
  - Scheduling – organizes the schedules of the experts.
- Requirement: We need to leverage these two microservices to provide a scheduling mechanism that finds the most qualified consultant available.

SERVER SENT EVENTS

- Part of the HTML 5 Spec.
- JAX-RS 2.1 provides APIs for SSE server and client.
- Can broadcast events to all registered clients or can send specific events to specific clients.

SSE DEMO

- Chat application – Users register with the chat room and can POST messages that are then broadcast to all registered clients in the room.

OTHER NEW APIS

- New connectTimeout(…) and readTimeout(…) methods in Client API allows more portable timeout code.
- New @PATCH annotation.
- Ordering of providers is now possible using @Priority annotations – note that providers will be invoked in ascending order (i.e. @Priority(5) will be invoked before @Priority(10)).
- Sub-resource locators can now return Class objects instead of only instance objects.
- JSON-B 1.0 Providers out of the box * - if the vendor provides JSON-B.

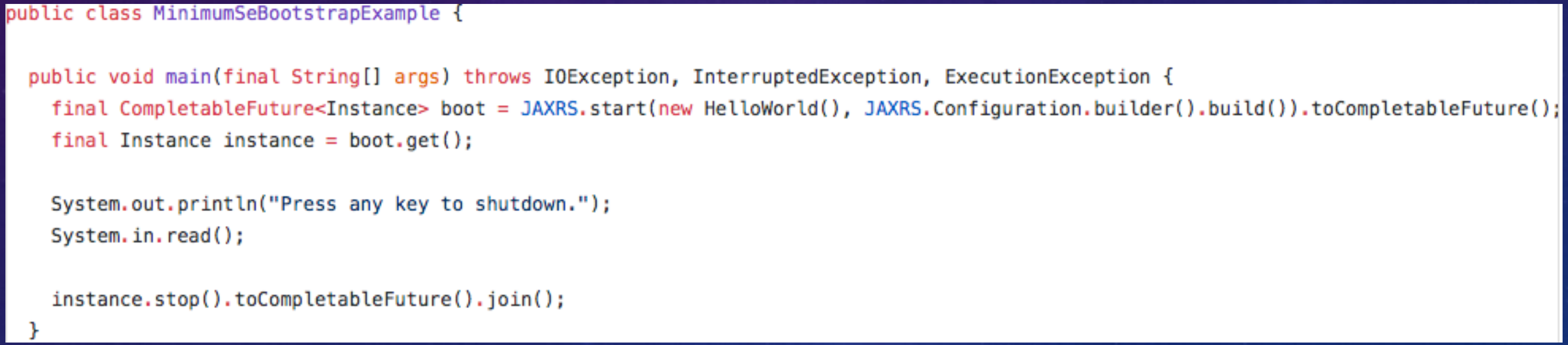
BEYOND JAX-RS 2.1…

- Disclaimer: No promises! Anything in the following slides are subject to change prior to the final release of JAX-RS v.Next.

- Java EE (Oracle) is now Jakarta EE (Eclipse/EE4J)
- Many members of the JAX-RS 2.1 Expert Group are now committers in the Jakarta JAX-RS project. New members have also joined the community.
- Current release planning:
  - The first release of Jakarta EE will be 100% compatible with Java EE 8 – this means JAX-RS 2.1.
  - The next release of JAX-RS is tentatively called JAX-RS 2.2 – some new function, but no breaking changes.
  - The next *major* release of JAX-RS is tentatively called 3.0 – expect some breaking changes.

JAX-RS IN JAVA SE

- The popularity of “micro” (MicroProfile, Spring Boot, etc.) is not lost on the JAX-RS community.
- For many use cases, a JAX-RS environment without all of the “extras” in a full profile application server can have a lot of benefits.
- This proposal intends to allow developers to start and stop a JAX-RS application with a few key bootstrap APIs.
- Consider Markus Karg’s minimal example (https://gist.github.com/mkarg/a38a68f6025f1ef6ddb4916022bd150d):



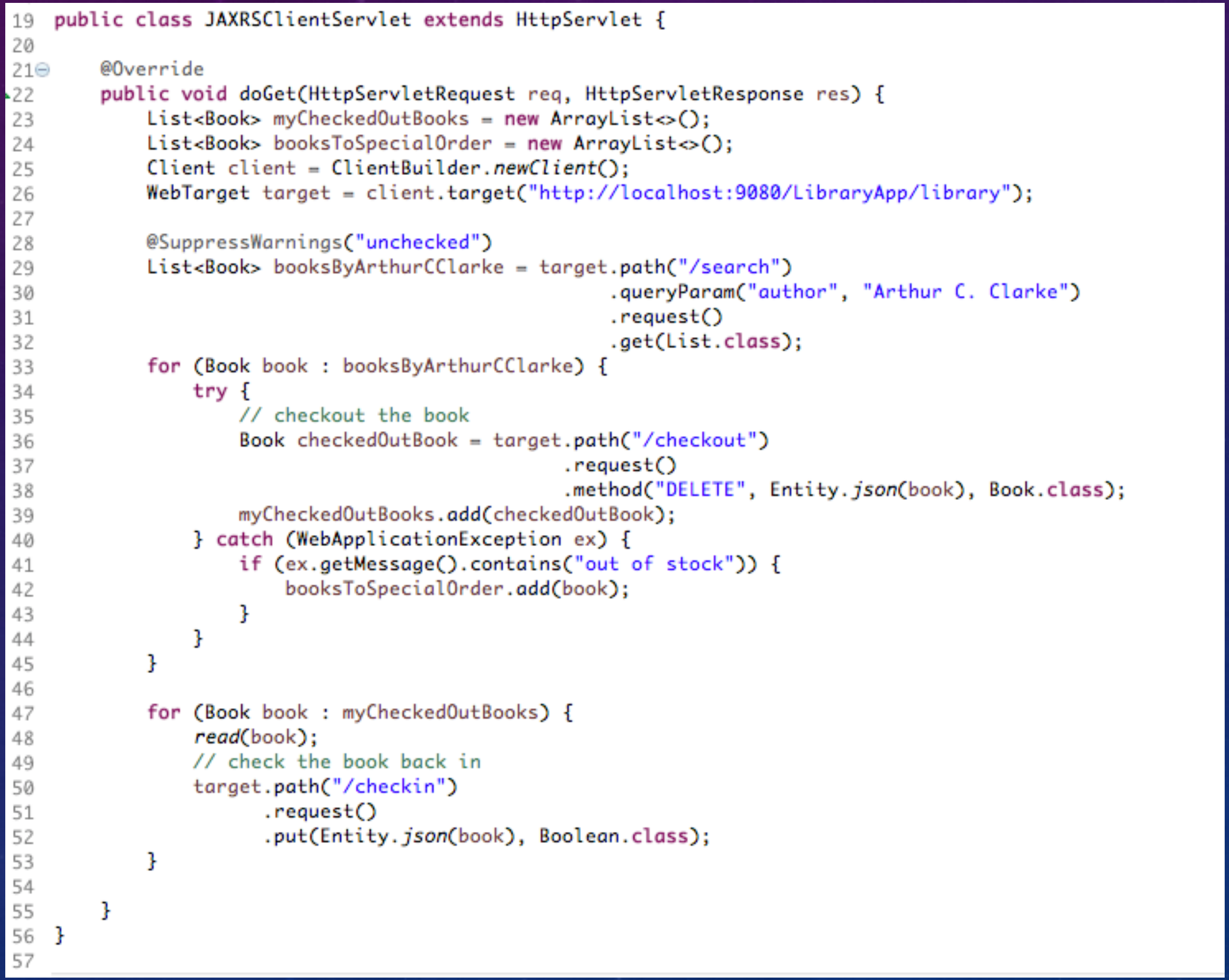
```java
public class MinimumSeBootstrapExample {

  public void main(final String[] args) throws IOException, InterruptedException, ExecutionException {
    final CompletableFuture<Instance> boot = JAXRS.start(new HelloWorld(), JAXRS.Configuration.builder().build()).toCompletableFuture();
    final Instance instance = boot.get();

    System.out.println("Press any key to shutdown.");
    System.in.read();

    instance.stop().toCompletableFuture().join();
  }
```

- https://github.com/eclipse-ee4j/jaxrs-api/issues/509

TYPE-SAFE CLIENT APIS

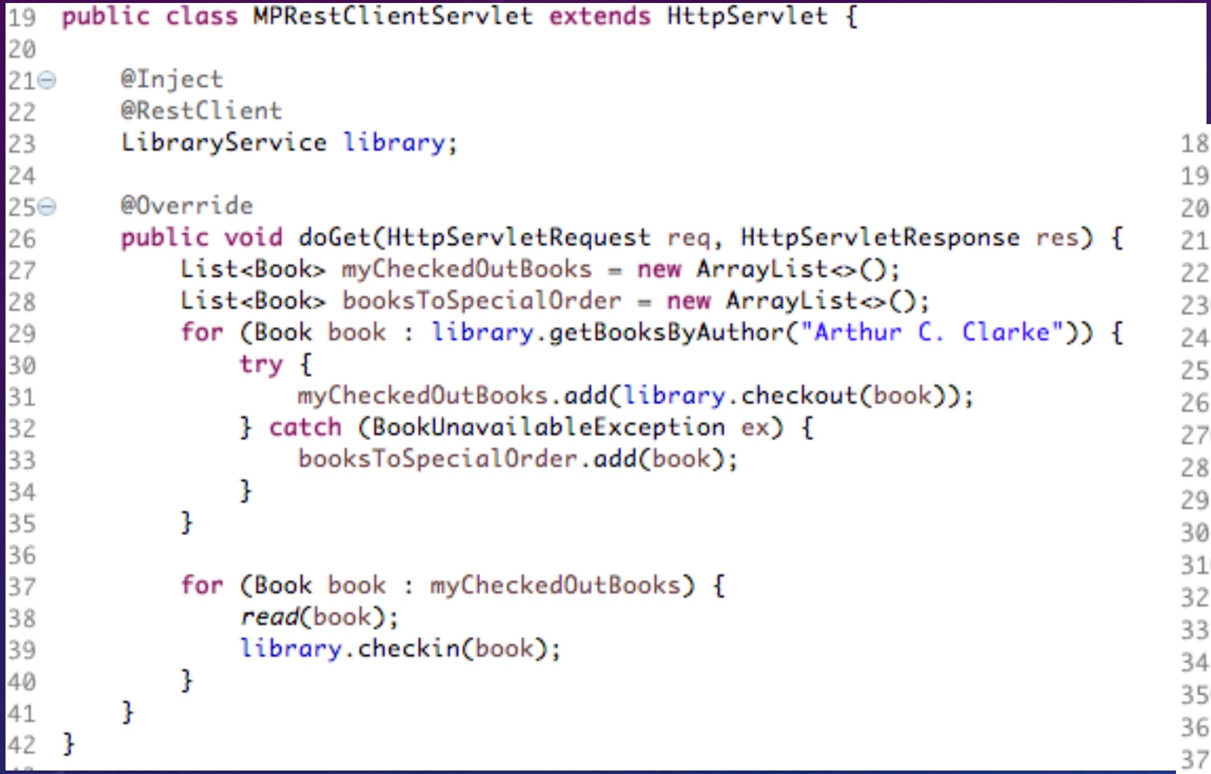
- Current JAX-RS Client APIs are very similar to low-level frameworks – like Apache Commons HTTP Client.
- Many JAX-RS implementations (CXF, Jersey, RESTEasy, etc.) have a type-safe, proxy-based client API that allows developers to code clients that better integrate with their business logic.
- https://github.com/eclipse-ee4j/jaxrs-api/issues/598
- Possibly moving Eclipse MicroProfile Rest Client to Jakarta EE…

TYPE-SAFE CLIENT APIS – THE JAX-RS 2.0/2.1 WAY

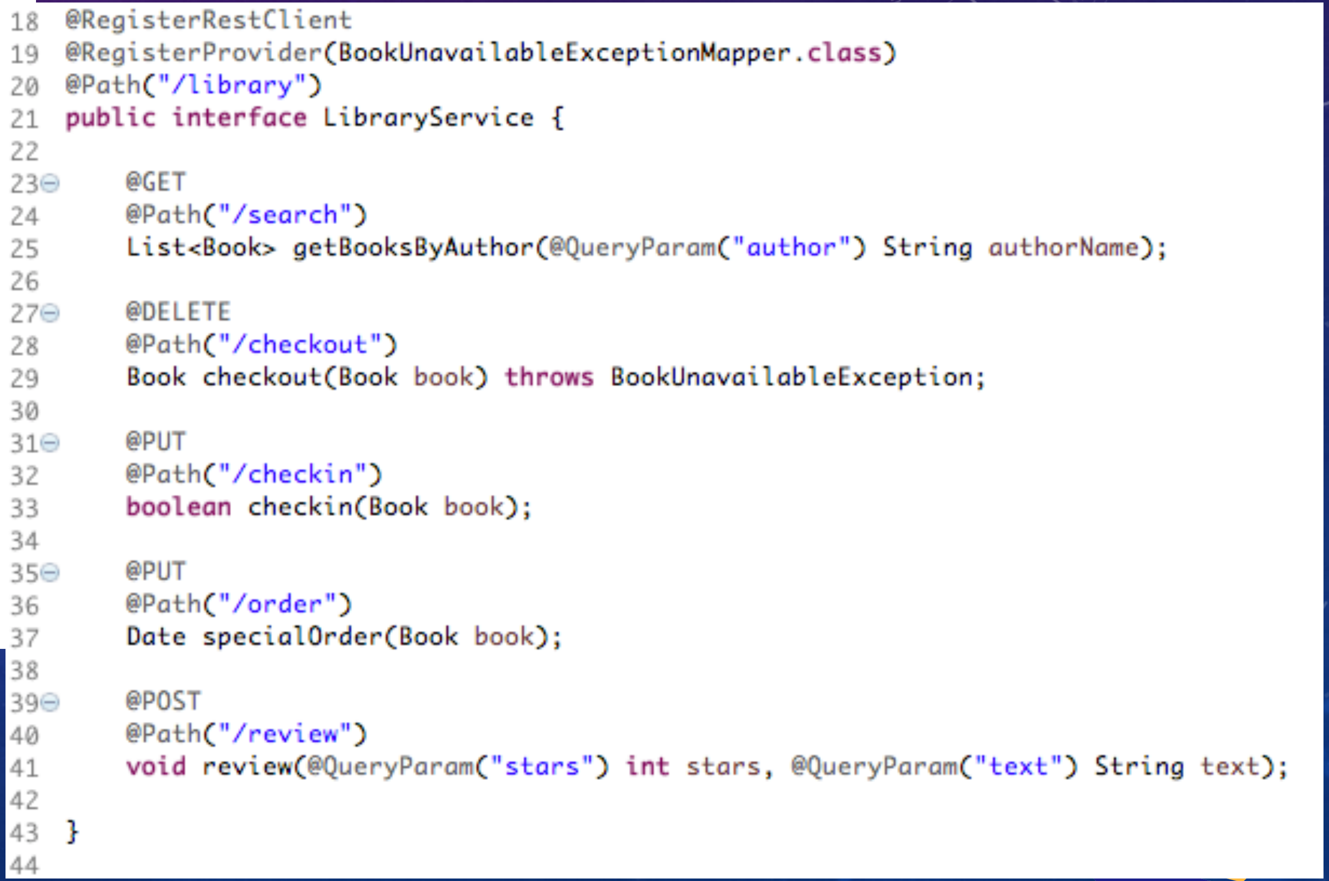


```java
public class JAXRSClientServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        List<Book> myCheckedOutBooks = new ArrayList<>();
        List<Book> booksToSpecialOrder = new ArrayList<>();
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target("http://localhost:9080/LibraryApp/library");

        @SuppressWarnings("unchecked")
        List<Book> booksByArthurCClarke = target.path("/search")
                                                .queryParam("author", "Arthur C. Clarke")
                                                .request()
                                                .get(List.class);
        for (Book book : booksByArthurCClarke) {
            try {
                // checkout the book
                Book checkedOutBook = target.path("/checkout")
                                            .request()
                                            .method("DELETE", Entity.json(book), Book.class);
                myCheckedOutBooks.add(checkedOutBook);
            } catch (WebApplicationException ex) {
                if (ex.getMessage().contains("out of stock")) {
                    booksToSpecialOrder.add(book);
                }
            }
        }

        for (Book book : myCheckedOutBooks) {
            read(book);
            // check the book back in
            target.path("/checkin")
                  .request()
                  .put(Entity.json(book), Boolean.class);
        }

    }
}
```

TYPE-SAFE CLIENT APIS – THE MP REST CLIENT WAY



```java
public class MPRestClientServlet extends HttpServlet {

    @Inject
    @RestClient
    LibraryService library;

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        List<Book> myCheckedOutBooks = new ArrayList<>();
        List<Book> booksToSpecialOrder = new ArrayList<>();
        for (Book book : library.getBooksByAuthor("Arthur C. Clarke")) {
            try {
                myCheckedOutBooks.add(library.checkout(book));
            } catch (BookUnavailableException ex) {
                booksToSpecialOrder.add(book);
            }
        }

        for (Book book : myCheckedOutBooks) {
            read(book);
            library.checkin(book);
        }
    }
}
```



```java
@RegisterRestClient
@RegisterProvider(BookUnavailableExceptionMapper.class)
@Path("/library")
public interface LibraryService {

    @GET
    @Path("/search")
    List<Book> getBooksByAuthor(@QueryParam("author") String authorName);

    @DELETE
    @Path("/checkout")
    Book checkout(Book book) throws BookUnavailableException;

    @PUT
    @Path("/checkin")
    boolean checkin(Book book);

    @PUT
    @Path("/order")
    Date specialOrder(Book book);

    @POST
    @Path("/review")
    void review(@QueryParam("stars") int stars, @QueryParam("text") String text);

}
```

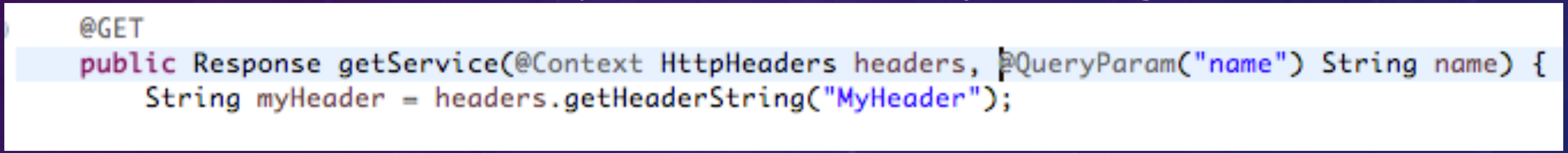
BETTER CDI INTEGRATION

- Currently CDI integration in JAX-RS 2.1 is somewhat limited – see section 11.2.3.
- Developers can use injection of other CDI-managed beans into resource, provider and application instances.
- Most vendors provide integration above and beyond the spec – for example, allowing non-standard lifecycles (i.e. @ApplicationScoped resource classes, or per-request scoped providers, etc.), but this is not standardized – and thus not portable.
- JAX-RS supports multiple injection mechanisms - @Context vs @Inject

- https://github.com/eclipse-ee4j/jaxrs-api/issues/569

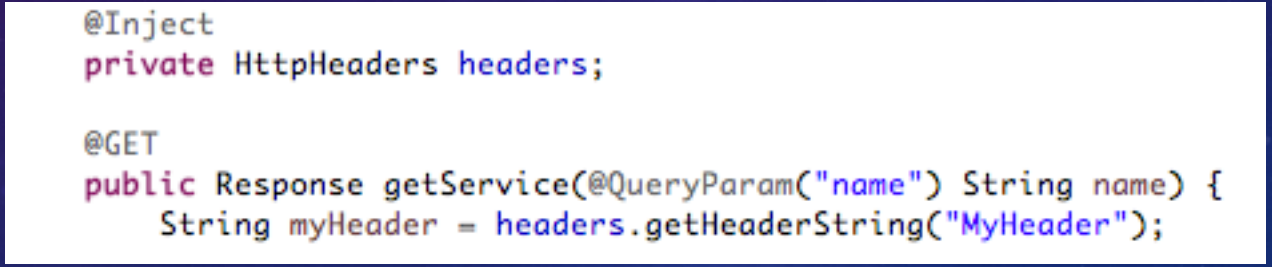
BETTER CDI INTEGRATION - PROPOSALS

- In JAX-RS 2.2, deprecate @Context in favor of @Inject – @Inject does not work on method parameters, so the recommendation would be to put @Context-based injection targets into fields.



```
@GET
public Response getService(@Context HttpHeaders headers, @QueryParam("name") String name) {
    String myHeader = headers.getHeaderString("MyHeader");
```

- Becomes:



```
@Inject
private HttpHeaders headers;

@GET
public Response getService(@QueryParam("name") String name) {
    String myHeader = headers.getHeaderString("MyHeader");
```

- Other things like RuntimeDelegate could also become CDI-managed beans.

BETTER CDI INTEGRATION - PROPOSALS

- The intent is to remove @Context as an injection mechanism in JAX-RS. If we deprecate @Context in JAX-RS 2.2, expect that it will be removed in JAX-RS 3.0.
- Other possibilities would be to standardize non-default lifecycles – allowing resources to be singletons and providers to be per-request, etc.

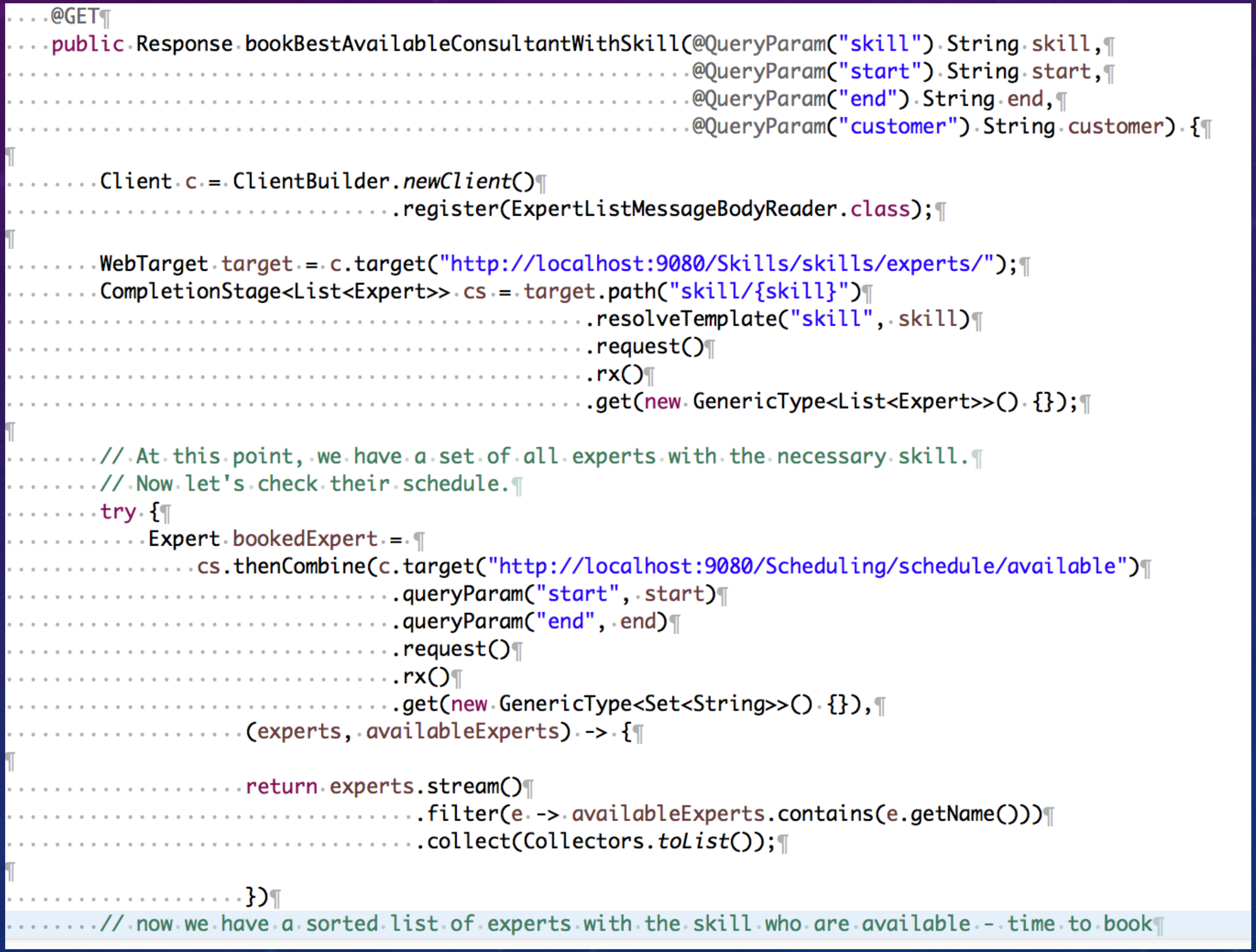
SUMMARY

- JAX-RS 2.1
  - Reactive Client – uses CompletionStage to enable reactive programming on the client side.
  - Server Sent Events – pub/sub for RESTful services.
  - Check the javadoc for additional changes: https://jax-rs.github.io/apidocs/2.1/
- JAX-RS Future
  - More community involvement – get involved at: https://projects.eclipse.org/proposals/eclipse-project-jax-rs
  - Lots of new features coming – stay tuned!

QUESTIONS?

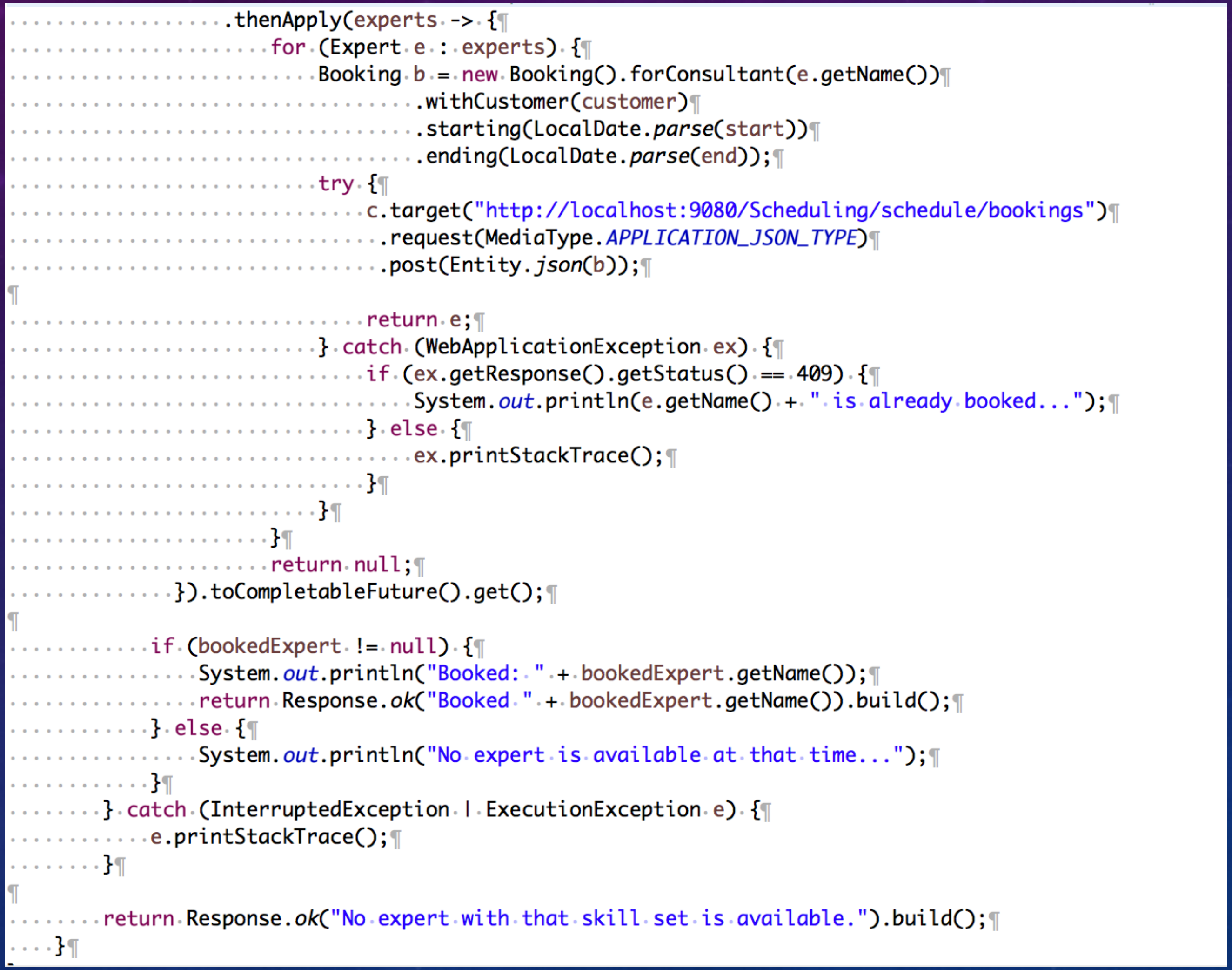
BACKUP SLIDES

RX CLIENT – EXAMPLERESOURCE.JAVA



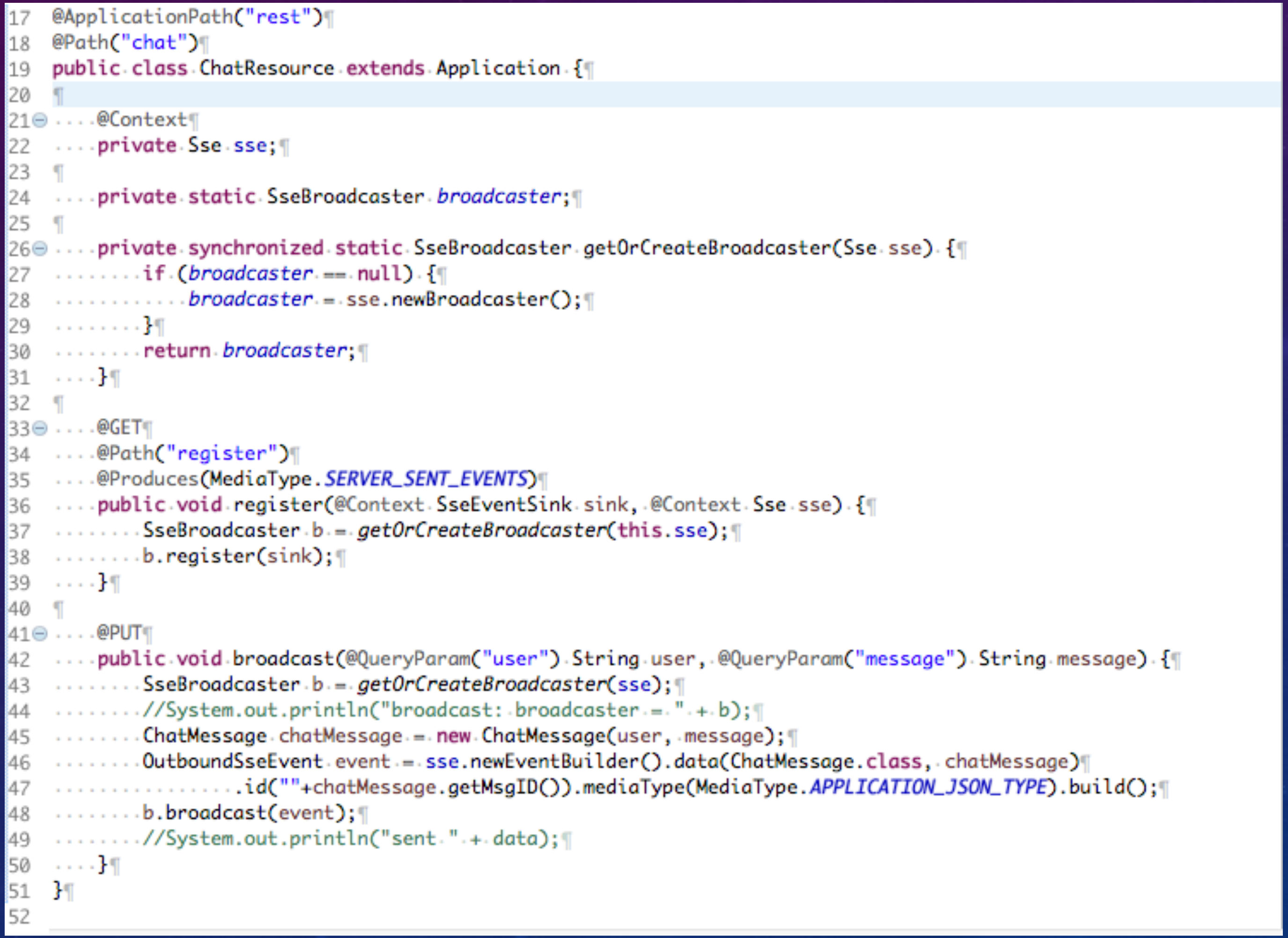
```java
   @GET
   public Response bookBestAvailableConsultantWithSkill(@QueryParam("skill") String skill,
                                                        @QueryParam("start") String start,
                                                        @QueryParam("end") String end,
                                                        @QueryParam("customer") String customer) {

       Client c = ClientBuilder.newClient()
                               .register(ExpertListMessageBodyReader.class);

       WebTarget target = c.target("http://localhost:9080/Skills/skills/experts/");
       CompletionStage<List<Expert>> cs = target.path("skill/{skill}")
                                         .resolveTemplate("skill", skill)
                                         .request()
                                         .rx()
                                         .get(new GenericType<List<Expert>>() {});

       // At this point, we have a set of all experts with the necessary skill.
       // Now let's check their schedule.
       try {
           Expert bookedExpert =
               cs.thenCombine(c.target("http://localhost:9080/Scheduling/schedule/available")
                               .queryParam("start", start)
                               .queryParam("end", end)
                               .request()
                               .rx()
                               .get(new GenericType<Set<String>>() {}),
                   (experts, availableExperts) -> {

                   return experts.stream()
                                 .filter(e -> availableExperts.contains(e.getName()))
                                 .collect(Collectors.toList());

                   })
       // now we have a sorted list of experts with the skill who are available - time to book
```

RXCLIENT – EXAMPLERESOURCE.JAVA (CONTINUED)



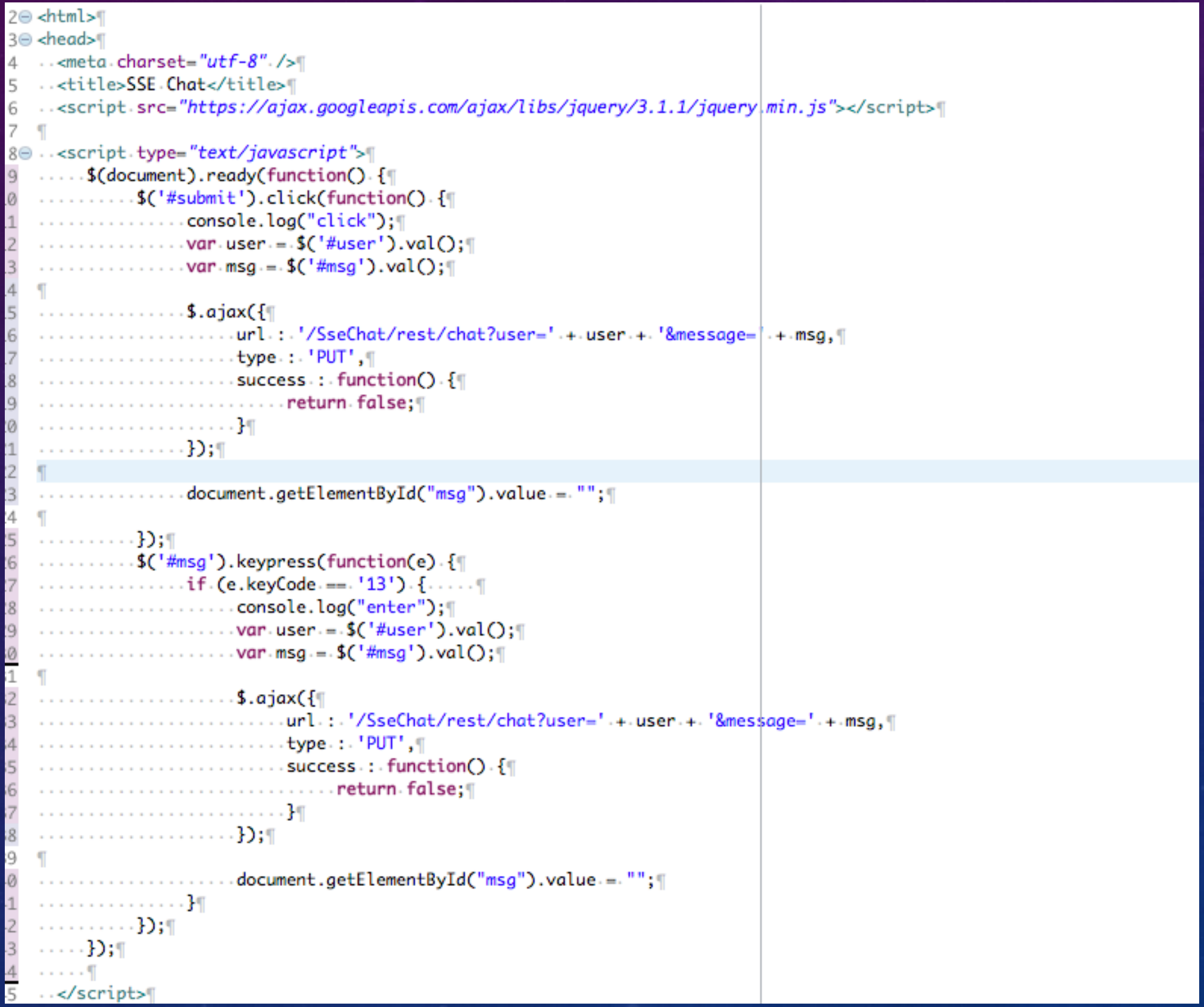
```
                  .thenApply(experts -> {
                      for (Expert e : experts) {
                          Booking b = new Booking().forConsultant(e.getName())
                                  .withCustomer(customer)
                                  .starting(LocalDate.parse(start))
                                  .ending(LocalDate.parse(end));
                          try {
                              c.target("http://localhost:9080/Scheduling/schedule/bookings")
                               .request(MediaType.APPLICATION_JSON_TYPE)
                               .post(Entity.json(b));

                              return e;
                          } catch (WebApplicationException ex) {
                              if (ex.getResponse().getStatus() == 409) {
                                  System.out.println(e.getName() + " is already booked...");
                              } else {
                                  ex.printStackTrace();
                              }
                          }
                      }
                      return null;
              }).toCompletableFuture().get();

            if (bookedExpert != null) {
                System.out.println("Booked: " + bookedExpert.getName());
                return Response.ok("Booked " + bookedExpert.getName()).build();
            } else {
                System.out.println("No expert is available at that time...");
            }
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }

        return Response.ok("No expert with that skill set is available.").build();
    }
```

SSE CHATRESOURCE.JAVA



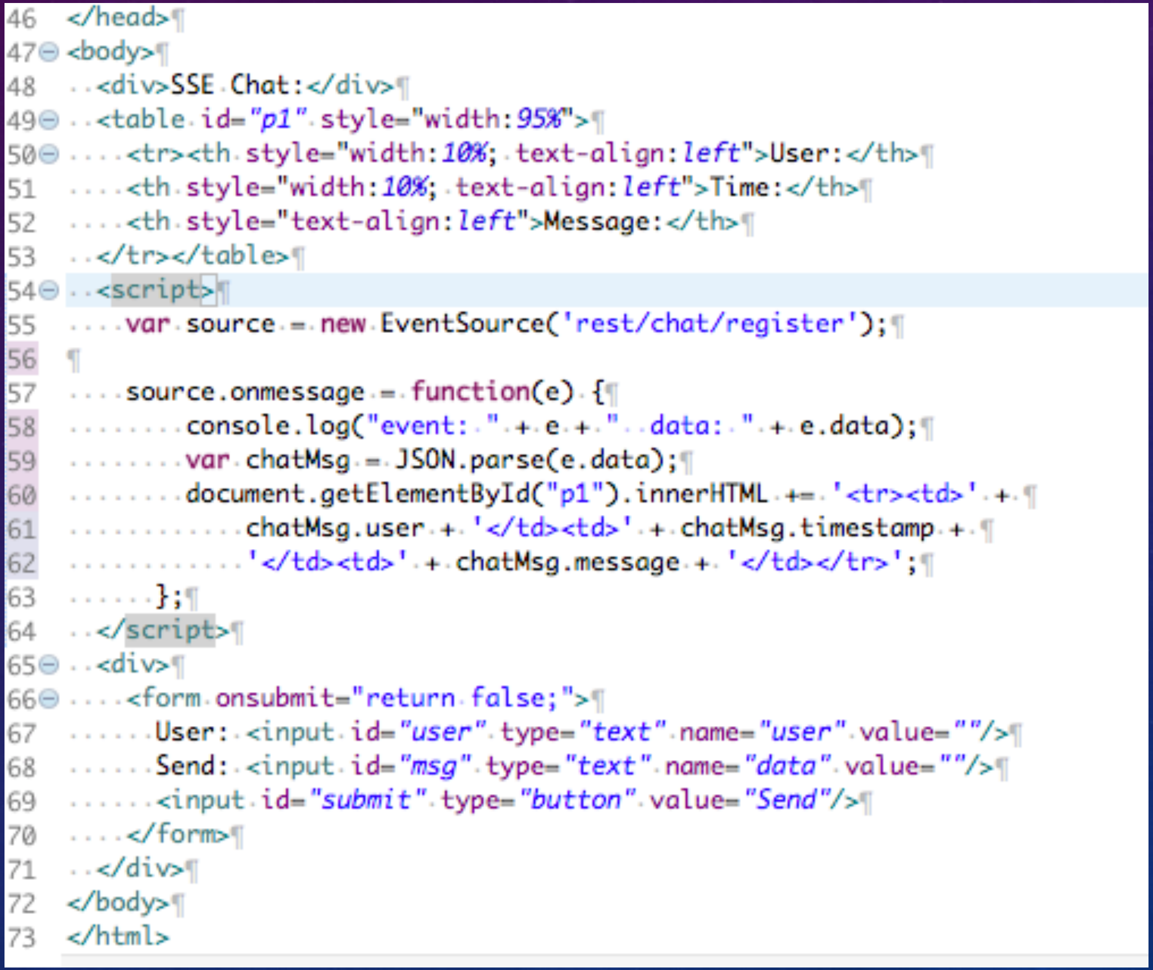
```java
@ApplicationPath("rest")
@Path("chat")
public class ChatResource extends Application {

    @Context
    private Sse sse;

    private static SseBroadcaster broadcaster;

    private synchronized static SseBroadcaster getOrCreateBroadcaster(Sse sse) {
        if (broadcaster == null) {
            broadcaster = sse.newBroadcaster();
        }
        return broadcaster;
    }

    @GET
    @Path("register")
    @Produces(MediaType.SERVER_SENT_EVENTS)
    public void register(@Context SseEventSink sink, @Context Sse sse) {
        SseBroadcaster b = getOrCreateBroadcaster(this.sse);
        b.register(sink);
    }

    @PUT
    public void broadcast(@QueryParam("user") String user, @QueryParam("message") String message) {
        SseBroadcaster b = getOrCreateBroadcaster(sse);
        //System.out.println("broadcast: broadcaster = " + b);
        ChatMessage chatMessage = new ChatMessage(user, message);
        OutboundSseEvent event = sse.newEventBuilder().data(ChatMessage.class, chatMessage)
                .id(""+chatMessage.getMsgID()).mediaType(MediaType.APPLICATION_JSON_TYPE).build();
        b.broadcast(event);
        //System.out.println("sent " + data);
    }
}

```

SSE INDEX.HTML



```
<html>
<head>
  <meta charset="utf-8" />
  <title>SSE Chat</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>

  <script type="text/javascript">
     $(document).ready(function() {
          $('#submit').click(function() {
               console.log("click");
               var user = $('#user').val();
               var msg = $('#msg').val();

               $.ajax({
                    url : '/SseChat/rest/chat?user=' + user + '&message=' + msg,
                    type : 'PUT',
                    success : function() {
                         return false;
                    }
               });

               document.getElementById("msg").value = "";

          });
          $('#msg').keypress(function(e) {
               if (e.keyCode == '13') {
                    console.log("enter");
                    var user = $('#user').val();
                    var msg = $('#msg').val();

                    $.ajax({
                         url : '/SseChat/rest/chat?user=' + user + '&message=' + msg,
                         type : 'PUT',
                         success : function() {
                              return false;
                         }
                    });

                    document.getElementById("msg").value = "";
               }
          });
     });

  </script>
```

SSE INDEX.HTML (CONTINUED)



```
</head>
<body>
  <div>SSE Chat:</div>
  <table id="p1" style="width:95%">
    <tr><th style="width:10%; text-align:left">User:</th>
    <th style="width:10%; text-align:left">Time:</th>
    <th style="text-align:left">Message:</th>
  </tr></table>
  <script>
    var source = new EventSource('rest/chat/register');

    source.onmessage = function(e) {
        console.log("event: " + e + "  data: " + e.data);
        var chatMsg = JSON.parse(e.data);
        document.getElementById("p1").innerHTML += '<tr><td>' +
            chatMsg.user + '</td><td>' + chatMsg.timestamp +
            '</td><td>' + chatMsg.message + '</td></tr>';
      };
  </script>
  <div>
    <form onsubmit="return false;">
      User: <input id="user" type="text" name="user" value=""/>
      Send: <input id="msg" type="text" name="data" value=""/>
      <input id="submit" type="button" value="Send"/>
    </form>
  </div>
</body>
</html>
```