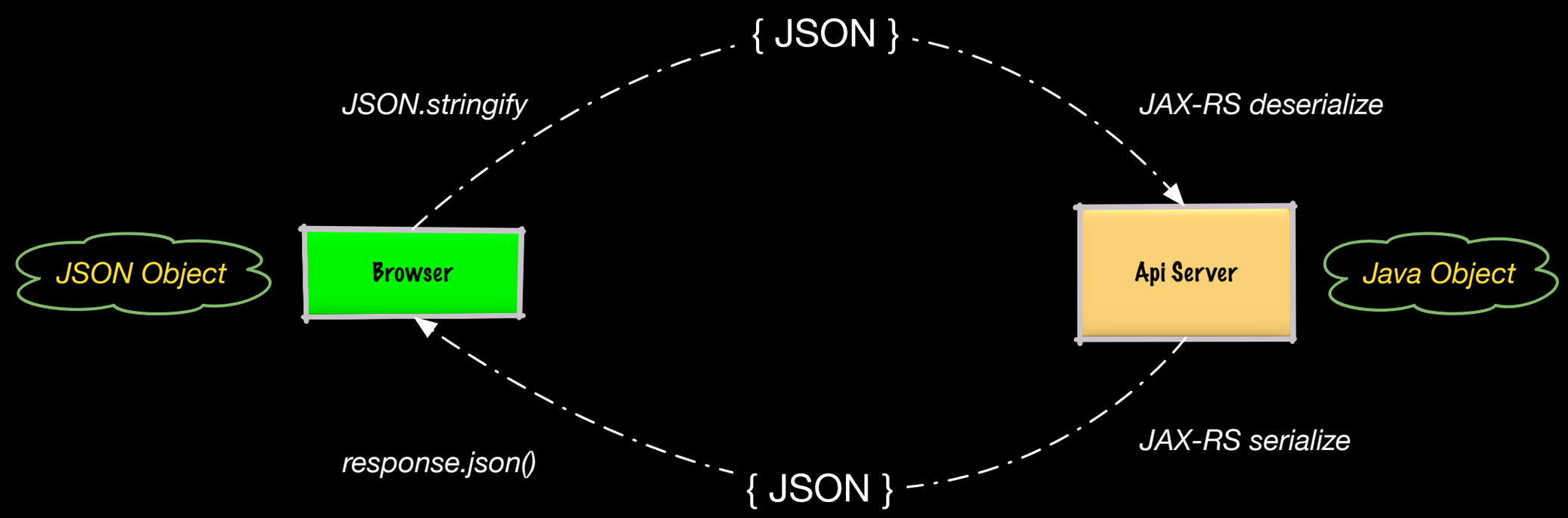
Login

Sending Data From Browser to Server
and Back Again

Overview

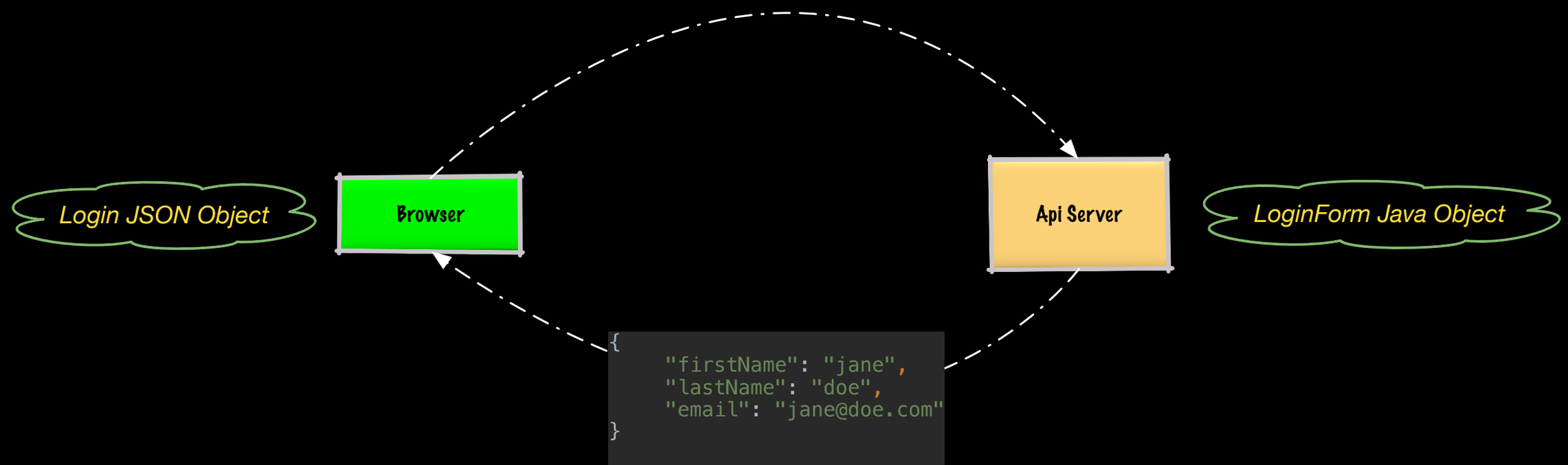


- Browsers send JSON objects to become server Java objects, and back again -- all via JSON strings over the wire.
- Using a login example, let us see how we can control the conversion between JSON and Java objects between the client and server.

An Example: Login

How does this work for Login?

```
POST http://localhost:8080/GreaterGoodsGrocers/api/account/login
Content-Type: application/json

{
  "email": "jane@doe.com",
  "passwordHash": "f8ab56b3058d01189c1421394b4a8fb85b40057f8972bbc5f6ed0a9b6466986d"
}
```



- Jane logs in with an email and password hash. She receives her login details in return (including her full name).

The Login Endpoint on the Server

JAX-RS specifies all annotations' meanings

The login endpoint consumes a JSON LoginForm and produces a JSON LoginDetails object

We fake an implementation of login for this example that always succeeds as "jane doe".

This LoginForm is deserialized from the client's JSON payload

LoginDetails is serialized to JSON when it is returned from this method.

```java
package api;

import business.account.LoginForm;
import business.account.LoginDetails;
import javax.ws.rs.*;

@ApplicationPath("/")
@Path("/account")
public class AccountResource {

    @POST
    @Path("login")
    @Consumes(javax.ws.rs.core.MediaType.APPLICATION_JSON)
    @Produces(javax.ws.rs.core.MediaType.APPLICATION_JSON)
    public LoginDetails login(LoginForm loginForm) {
        try {
            LoginDetails loginDetails = new LoginDetails();
            loginDetails.setEmail("jane@doe.com");
            loginDetails.setFirstName("jane");
            loginDetails.setLastName("doe");
            return loginDetails;
        } catch (ApiException e) {
            throw e;
        } catch (Exception e) {
            throw new ApiException("login failed", e);
        }
    }
}
```

- Let us see how the deserializing and serializing works.

Deserialize (JSON to Java)

JSON fields match @JsonProperty names

```
{
  "email": "jane@doe.com",
  "passwordHash": "f8ab56b3058d01189c1421304b4a8fb85b40057f8972bbc5f6ed0a9b6466986d"
}
```

@JsonCreator indicates how to start constructing the object.

```
package business.account;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class LoginForm {

    private String email;
    private String passwordFingerprint;

    @JsonCreator
    public LoginForm(@JsonProperty("email") String email,
                     @JsonProperty("passwordHash") String passwordHash) {
        this.email = email;
        this.passwordFingerprint = passwordHash;
    }

    public String getEmail() { return email; }

    public String getPasswordFingerprint() { return passwordFingerprint; }

}
```

- Annotations guide the process and allow the Java field names (passwordFingerprint) to differ from the JSON field names (passwordHash)

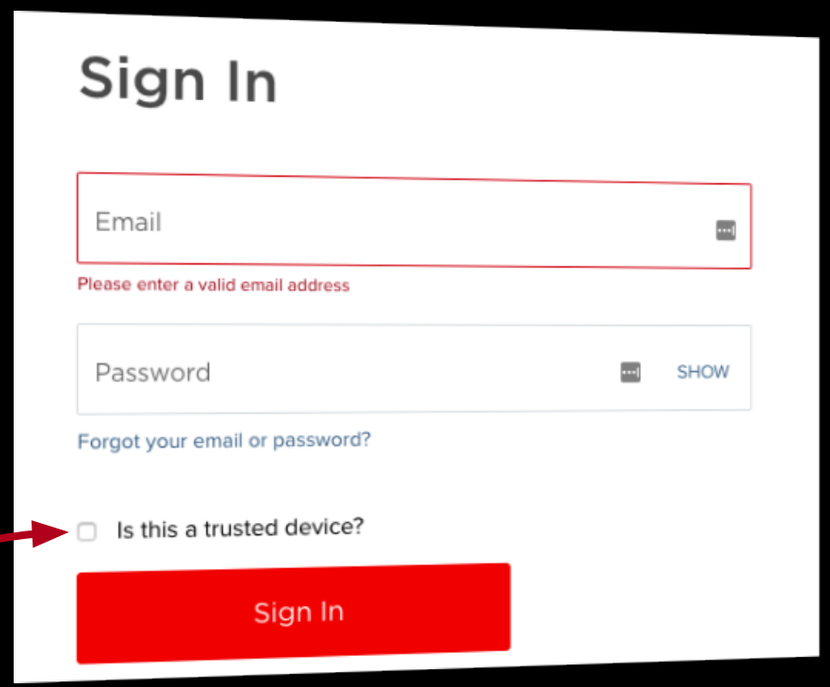
Serialize (Java to JSON)

```java
package business.account;

public class LoginDetails {

    private String firstName;
    private String lastName;
    private String email;

    public String getFirstName() { return firstName; }

    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return lastName; }

    public void setLastName(String lastName) { this.lastName = lastName; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }
}
```

```json
{
    "firstName": "jane",
    "lastName": "doe",
    "email": "jane@doe.com"
}
```

- The getters are used to derive field names for JSON. @JsonProperty can be used to rename the JSON properties if desired.

Adding a Login Form Element



We introduce a new login form element “trustedDevice” to capture the sense of whether this is a personal or public device.

- What complications does this extra field cause?

Deserializing the Extended Login Form

JSON fields match @JsonProperty names

```
{
  "email": "jane@doe.com",
  "passwordHash": "f8ab56b3058d01189c1421394b4a8fb85b40057f8972bbc5f6ed0a9b6466986d",
  "trustedDevice": false
}
```

?

```
package business.account;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class LoginForm {

    private String email;
    private String passwordFingerprint;

    @JsonCreator
    public LoginForm(@JsonProperty("email") String email,
                     @JsonProperty("passwordHash") String passwordHash) {
        this.email = email;
        this.passwordFingerprint = passwordHash;
    }

    public String getEmail() { return email; }

    public String getPasswordFingerprint() { return passwordFingerprint; }
}
```

- What happens with the extra trustedDevice JSON field?

Handling Extra Fields Gracefully

- What happens with the extra trustedDevice JSON field?

- What's the fix?

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class LoginForm {
```

- By ignoring extra properties using the @JsonIgnoreProperties annotation we can still build a LoginForm with the default of "false" for trustedDevice.

Further Topics (Optional)

JAX-RS: Java API for RESTful Web Services

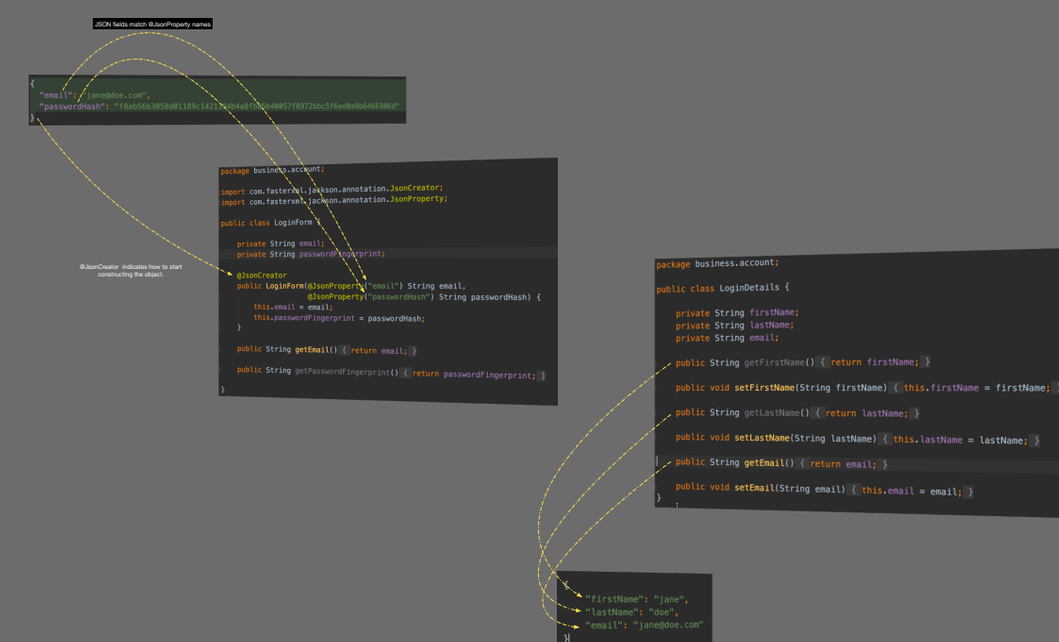
JAX-RS specifies all the annotations' behavior

```
package api;

import business.account.LoginForm;
import business.account.LoginDetails;

import javax.ws.rs.*;

@ApplicationPath("/")
@Path("/account")
public class AccountResource {

    @POST
    @Path("login")
    @Consumes(javax.ws.rs.core.MediaType.APPLICATION_JSON)
    @Produces(javax.ws.rs.core.MediaType.APPLICATION_JSON)
    public LoginDetails login(LoginForm loginForm) {...}

}
```

...including the login endpoint

https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services
https://jax-rs.github.io/apidocs/2.0/

The Jackson Library for JSON-Java Conversion



https://www.baeldung.com/jackson-annotations