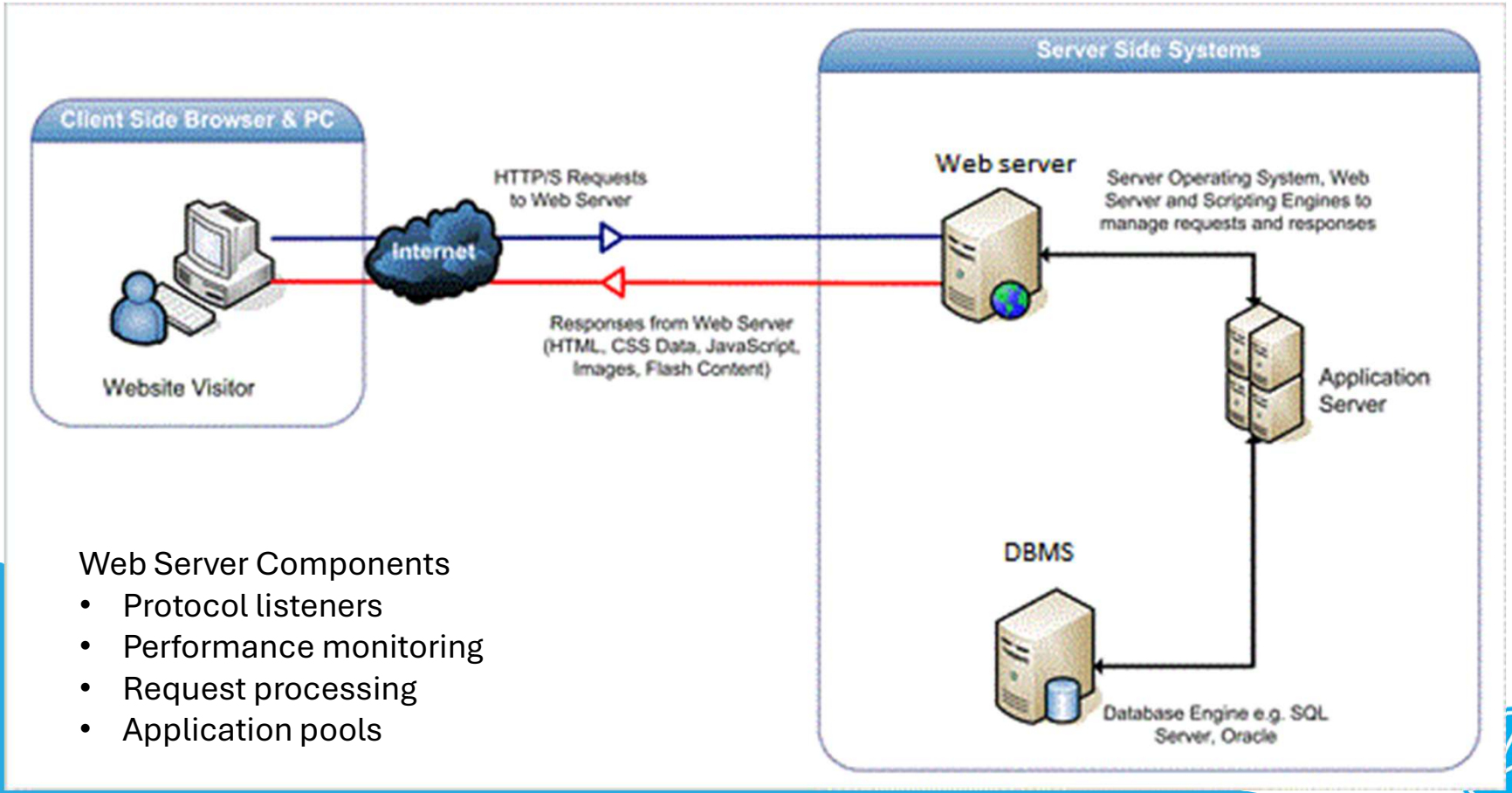
MODULE 2

Server Side API: Part 1

TECH ELEVATOR
A Stride Company

Anatomy of an HTTP request



Web Server Components

- Protocol listeners
- Performance monitoring
- Request processing
- Application pools

MVC (Model View Controller) Pattern

Model (Database)

- application state and business logic
- only part of the application that talks to the database
- Models could be classes

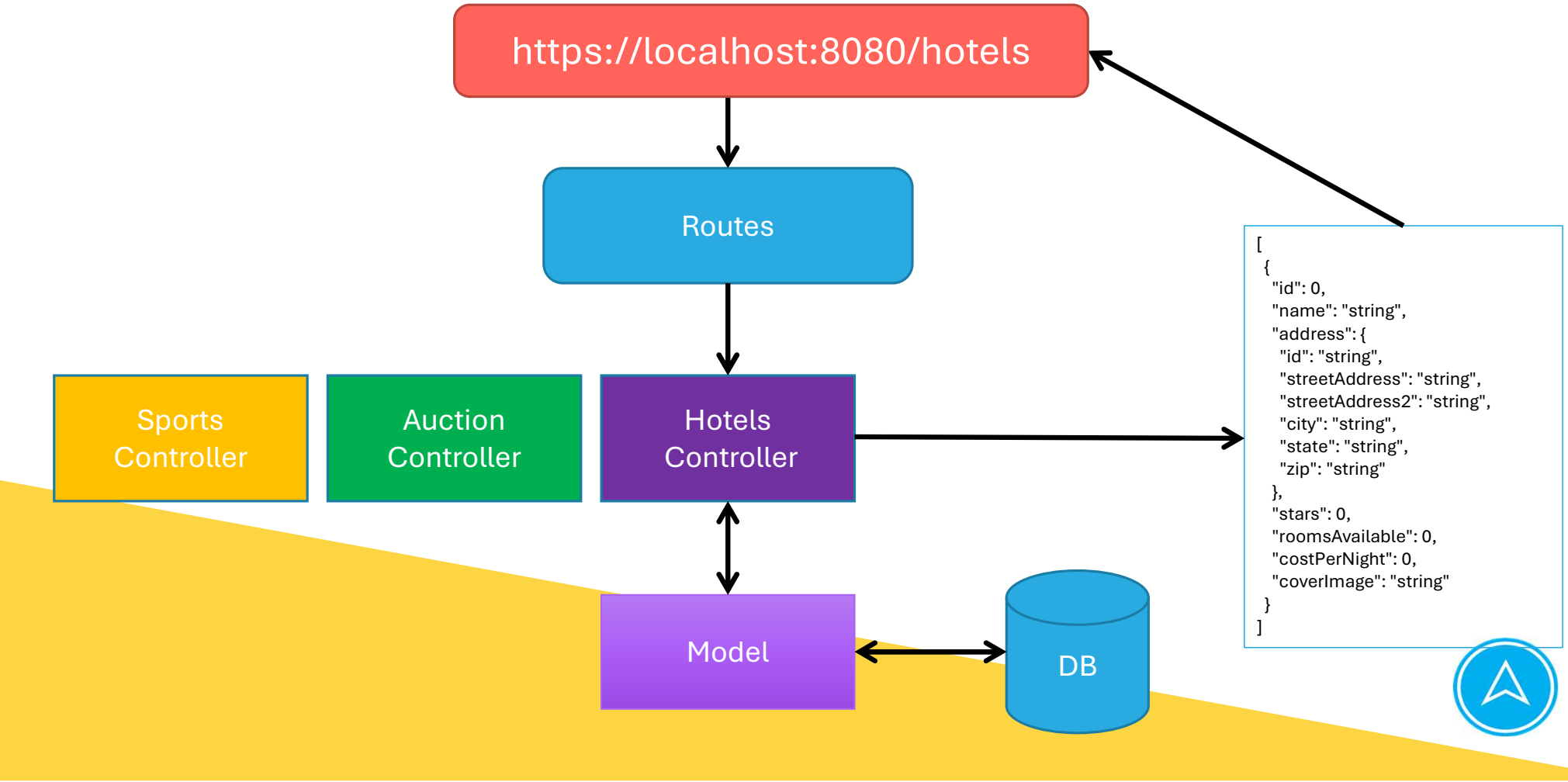
View (Front End)

- presents data to user
- accepts input from user
- Views might be a desktop display, a mobile display, a file output based on a model

Controller (API/Back End)

- Takes input from the view and passes it to the appropriate model objects
- Grabs all necessary building blocks and organizes them for the output
- Takes results from the model and passes it to the appropriate view

Web Service Parts



The MVC Application Lifecycle

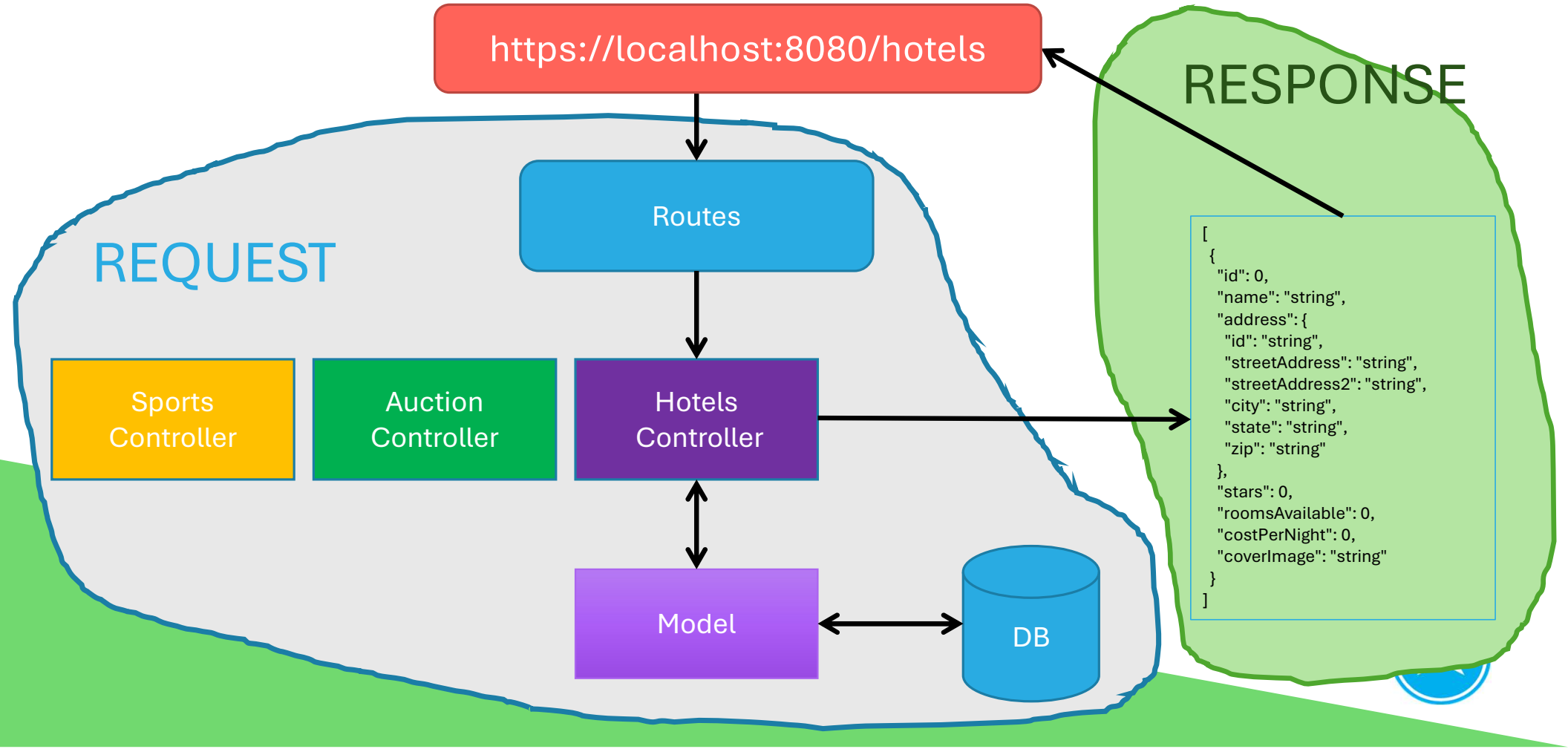
Request

- Creation of Request begins with a URL
- When the request arrives the route maps to a controller and a specific action
- The request arrives with additional parameters that are accessible in code

Response

- The action is executed to provide a response
- Response JSPs, Files, JSON, etc.

Web Service Parts



The wizard behind the curtain

GET https://localhost:44322/api/hotels

```java
@RestController
public class HotelController {

    private HotelDao hotelDao;
    private ReservationDao reservationDao;

    public HotelController() {
        this.hotelDao = new MemoryHotelDao();
        this.reservationDao = new MemoryReservationDao(hotelDao);
    }

    /**
     * Return All Hotels
     *
     * @return a list of all hotels in the system
     */
    @RequestMapping(path = "/api/hotels", method = RequestMethod.GET)
    public List<Hotel> list() { return hotelDao.list(); }

    /**
     * Return hotel information
     *
     * @param id the id of the hotel
     * @return all info for a given hotel
     */
    @RequestMapping(path = "/api/hotels/{id}", method = RequestMethod.GET)
    public Hotel get(@PathVariable int id) { return hotelDao.get(id); }
```

The wizard behind the curtain

GET https://localhost:44322/api/hotels/1

```java
@RestController
public class HotelController {

    private HotelDao hotelDao;
    private ReservationDao reservationDao;

    public HotelController() {
        this.hotelDao = new MemoryHotelDao();
        this.reservationDao = new MemoryReservationDao(hotelDao);
    }

    /**
     * Return All Hotels
     *
     * @return a list of all hotels in the system
     */
    @RequestMapping(path = "/api/hotels", method = RequestMethod.GET)
    public List<Hotel> list() { return hotelDao.list(); }

    /**
     * Return hotel information
     *
     * @param id the id of the hotel
     * @return all info for a given hotel
     */
    @RequestMapping(path = "/api/hotels/{id}", method = RequestMethod.GET)
    public Hotel get(@PathVariable int id) { return hotelDao.get(id); }
```

Annotations

@RestController	Marks a class as a REST controller
@RequestMapping	Creates the route for a method by defining the url and request method
@PathVariable	Identifies a parameter as a variable in the url path.
@RequestBody	Identifies the object type and variable name for deserialization.
@RequestParam	Identifies the querystring parameters
@ResponseStatus	Sets the HTTPResponse

HttpGet

- In APIs, it is common to provide at least two HttpGet actions:
  - An action that retrieves all resources.
  - An action that retrieves a single resource given an identifier.

```
@RequestMapping(path = "/hotels", method = RequestMethod.GET)
public List<Hotel> list() {
    return hotelDao.list();
}
```

```
@RequestMapping(path = "/hotels/{id}", method = RequestMethod.GET)
public Hotel get(@PathVariable int id) {
    return hotelDao.get(id);
}
```

HttpPost

- For adding new information to the system.
    - If you are updating, use HttpPut

```
@RequestMapping( path = "/reservations", method = RequestMethod.POST)
 public Reservation addReservation(@RequestBody Reservation reservation)
 {
     return reservationDao.create(reservation, reservation.getHotelID());
 }
```

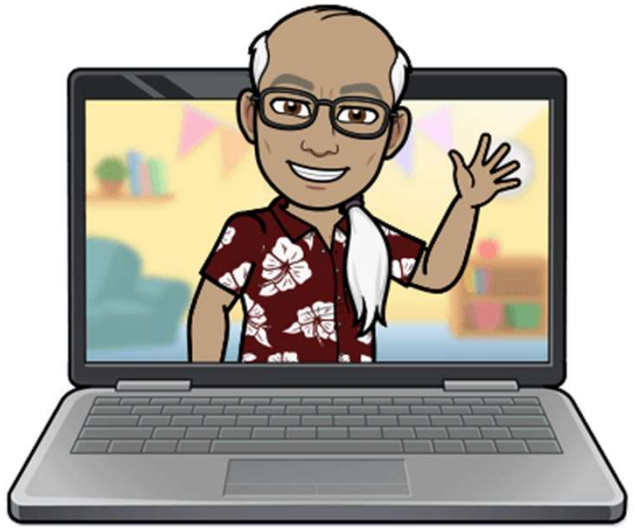
HttpPut

```
@RequestMapping(path = "/reservations/{id}", method = RequestMethod.PUT)
    public Reservation update(@Valid @RequestBody Reservation reservation, @PathVariable int id)
                        throws ReservationNotFoundException {
        return reservationDao.update(reservation, id);
    }
```

HttpDelete

```
@ResponseStatus(HttpStatus.NO_CONTENT)
@RequestMapping(path = "/reservations/{id}", method = RequestMethod.DELETE)
public void delete(@PathVariable int id) throws ReservationNotFoundException {
   reservationDao.delete(id);
}
```

LET'S CODE!



WHAT QUESTIONS DO YOU HAVE?

Reading for tonight:

Server Side APIs Part 2