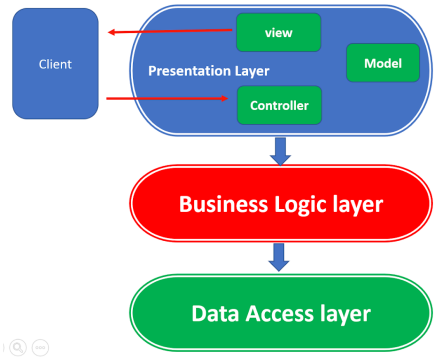
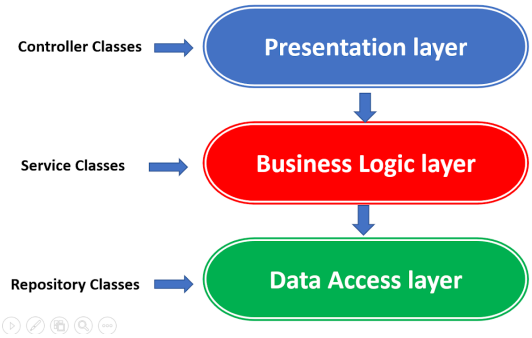
Three-Layer Architecture:

Three Tier (Three Layer) Architecture in Spring MVC Web Application
blog post, video





Presentation layer

- Eclipse/Java: **controller** class(es), JSP files w/ JSTL
- Spring MVC
- *MVC* components reside here
- *Model* contains the data that will be displayed to the user
    - data is made available by way of the Business Logic Layer, which subsequently connects to the Persistence Layer, then to the database
- *View* is a visual representation of the data (JSP, HTML/Bootstrap, CSS)
- *Controller* handles user requests, interacts with the Model, and represents the connection between Presentation and Business Logic layers

Business Logic layer

- Eclipse/Java: **service** class(es)
- acts as the interface between Presentation and Persistence layers
- functional process logic: calculations, sorting, conditionals

Persistence/Data Access layer

- Eclipse/Java: **DAO/repository** class(es), Spring-ORM, Hibernate Core, Hibernate Validator, c3p0
- sole purpose is persisting data to the database using CRUD functionality

GitHub Repos vs Work Silos vs Architecture:

How does the three-layer approach, plus the database, translate to dividing work into silos that can be combined into one Eclipse dynamic web project later? It makes sense to divide repos between Eclipse packages, but also consider whether they are front-end, middle, or back-end.

Table linked to “work silos & repos” sheet

web	layer	function	package / path	classes / files	repo/branch	team
front	presentation	controller	gogo.controller	ControllerSOAP.java	front-controller	CK, DA
		view	src/main/webapp//WEB-INF/	views/viewnames.jsp	front-view	AK, DA, AS, CK
				res/css/sheets.css, bootstrap		
				Dispatcher-servlet.xml		
				web.xml		
mid	business	service	gogo.service	UserService.java [i]	mid-service	GW, CM
				UserServiceImpl.java		
				MoneyInfoService.java [i]		
				MoneyInfoServiceImpl.java		
				FormOptionsService.java		
back	persistence	data access	gogo.dao	MoneyInfoDao.java [i]	back-dao	GW, CM, HM, DC
				MoneyInfoDaoImpl.java		
				UserDao.java [i]		
				UserDaoImpl.java		
		db abstraction	gogo.entity	MoneyInfo.java	back-entity	HM, DC
				Transaction.java		
				User.java		
		database	MySQL	gogo.sql		

Jing’s thoughts:

- Jing and Clayton talked Mon Mar 2
- use one repo and branches rather than multiple repos
- establish DAO first: create a list of requirements
- when DAO v1 is established, team disperses to other teams, then reconvenes as necessary
- other teams make requests for DAO changes/updates throughout lifecycle
- IMPORTANT: establish GitHub workflow rules for order of operations (i.e. ALWAYS pull first, when to push?, when to merge to main?, etc) to avoid overwriting important work

Other Notes:

- use 3-layer architecture
- everybody using Eclipse JEE with JRE 11 or 16 (need everyone on one version)
- no need for REST controller .. therefore no need for Jackson-databind
- CPS 278 grads: AK, DC, GW, CM, CK | current: AS, HM | never: DA
- CK “floats” across repos

old version .. ignore

eb	function	package / path	classes / files	repo	team
ont	controller	gogo.controller	ControllerSOAP.java	front-controller	
	view	src/main/webapp/	/WEB-INF/views/viewnames.jsp	front-view	AK DA
			/WEB-INF/res/css/sheets.css		
			/WEB-INF/Dispatcher-servlet.xml		
			/WEB-INF/web.xml		
d	business /	gogo.service	UserService.java [i]	mid-service	
	service		UserServiceImpl.java		
			MoneyInfoService.java [i]		
			MoneyInfoServiceImpl.java		
ck	persistence /	gogo.dao	MoneyInfoDao.java [i]	back-dao	
	dao		MoneyInfoDaoImpl.java		
			UserDao.java [i]		
			UserDaoImpl.java		
	database	gogo.entity	MoneyInfo.java	back-db	
			Transaction.java		
			User.java		
	database	MySQL	gogo.sql		