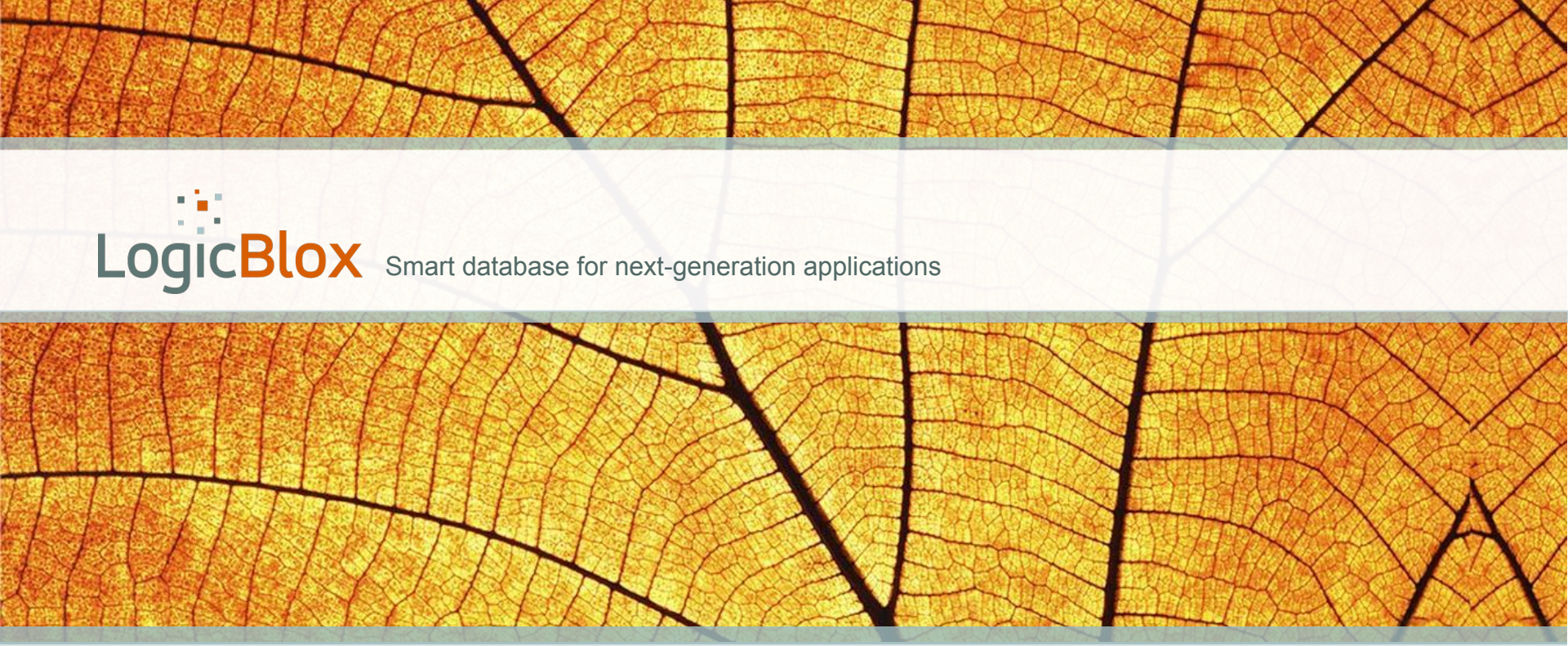
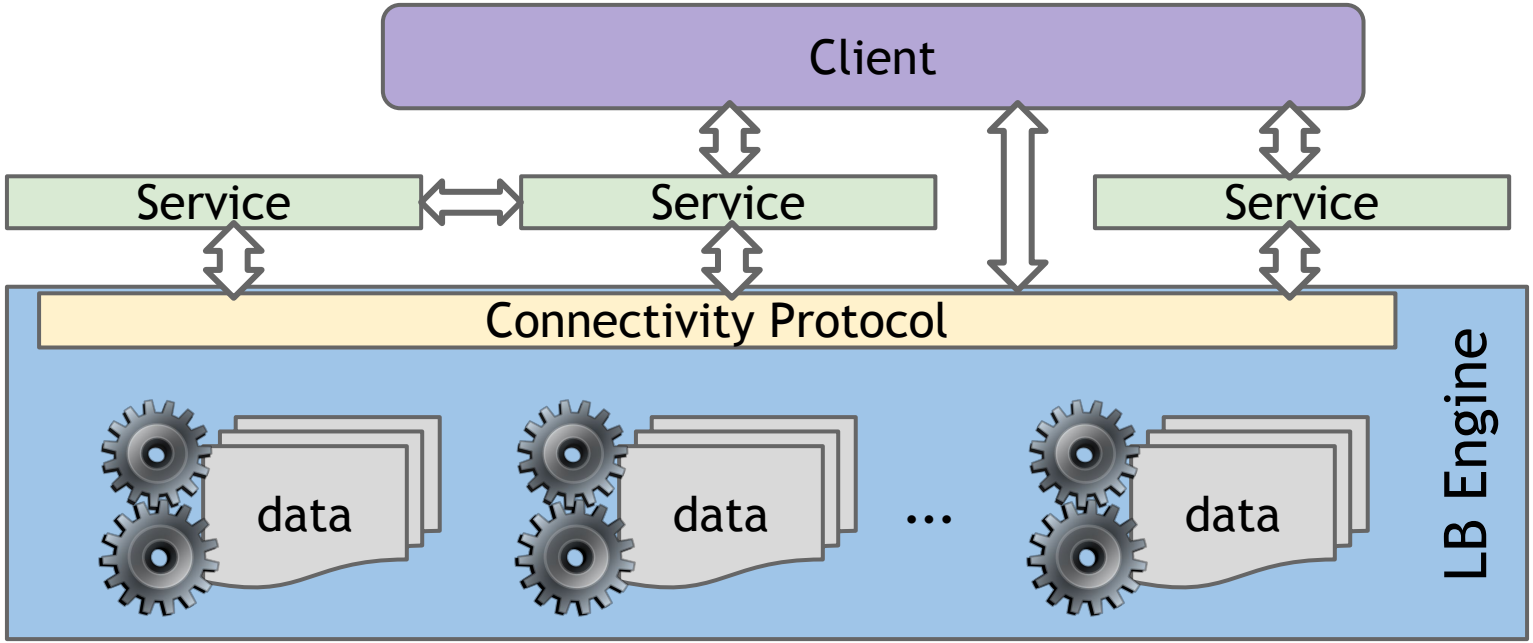


LogicBlox Smart database for next-generation applications

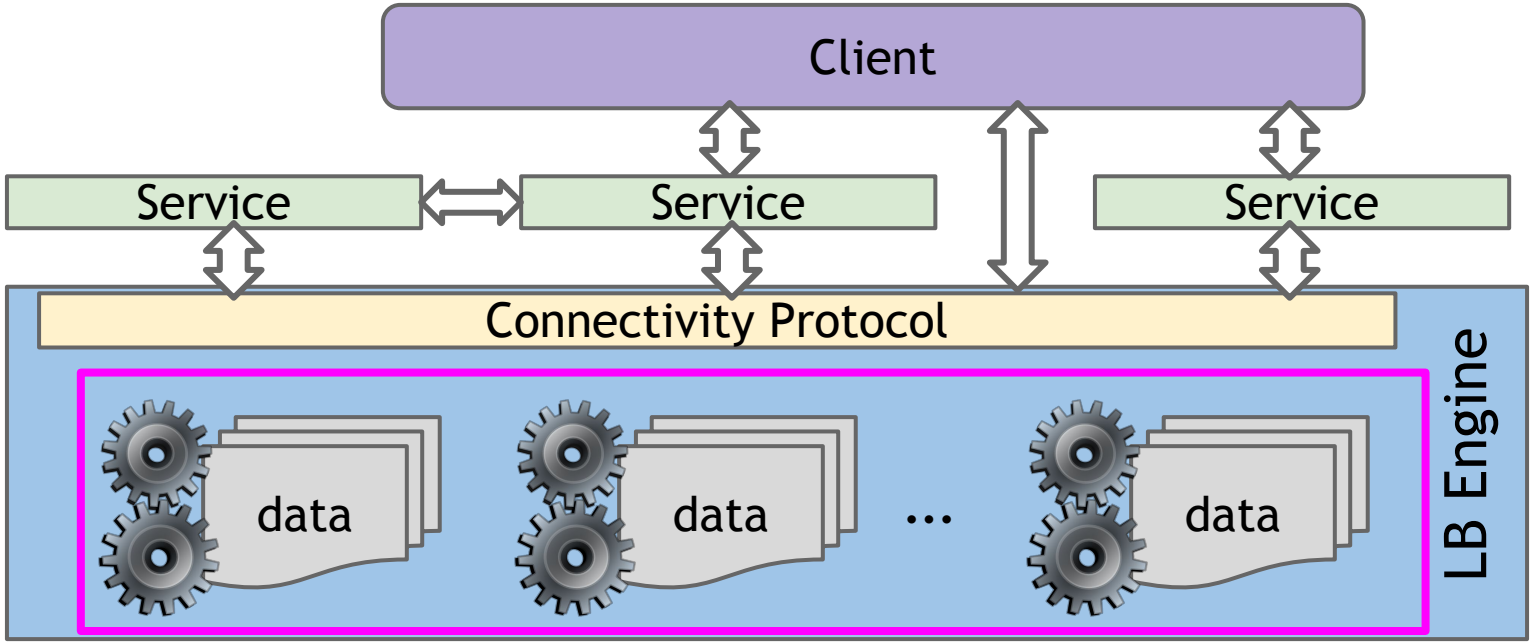
LESSON 4 : PROJECT ORGANIZATION, UNIT TESTING, AND CONSTRAINTS

LOGICBLOX TRAINING

TYPICAL APPLICATION ARCHITECTURE

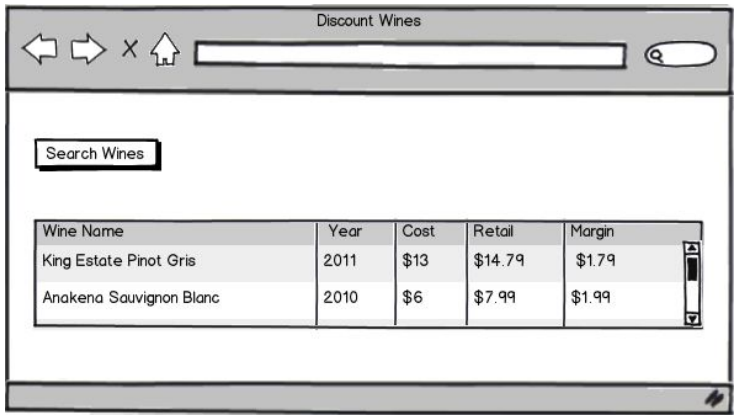


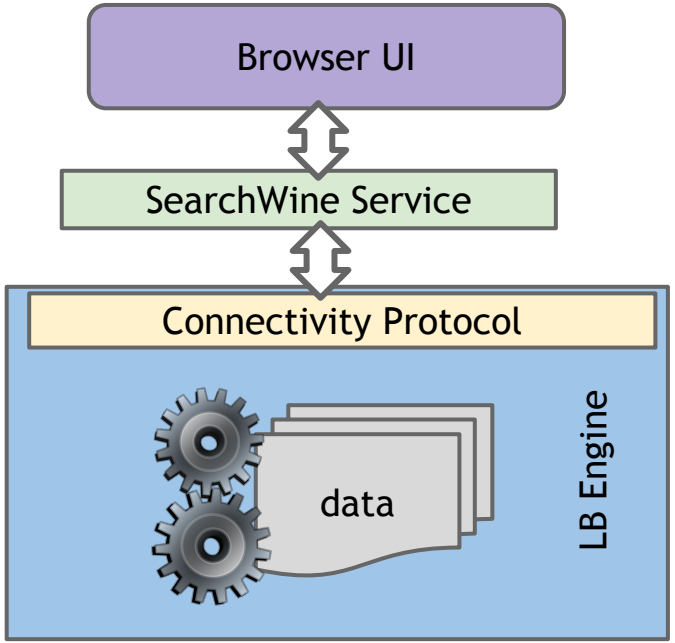
TYPICAL APPLICATION ARCHITECTURE



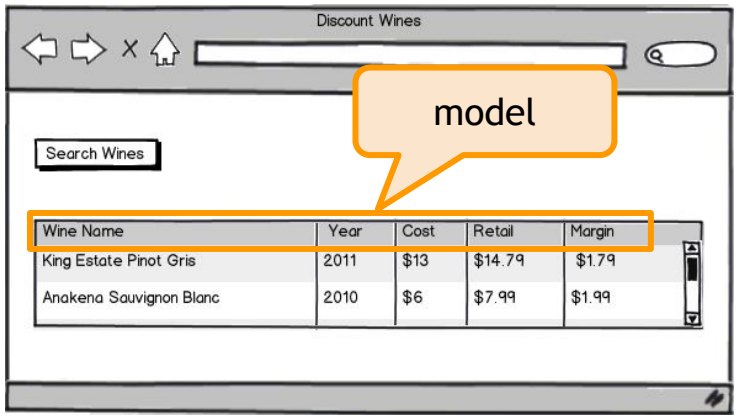
DISCOUNTWINES

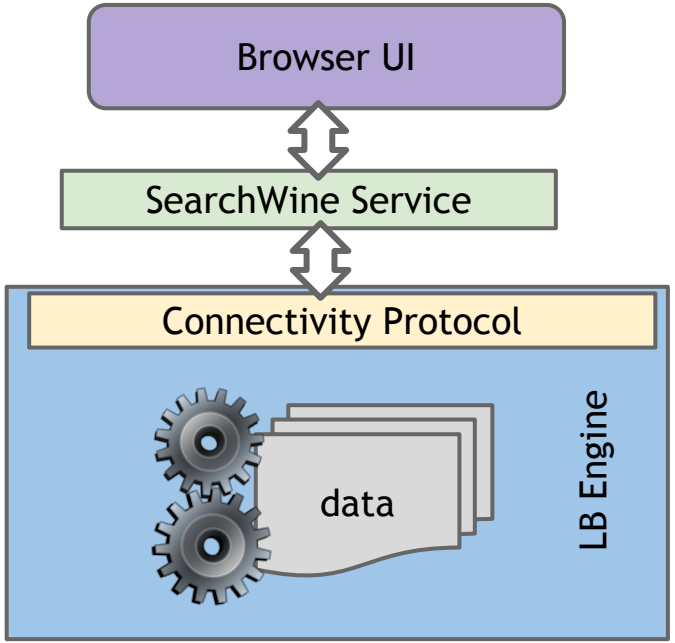
DISCOUNTWINES ARCHITECTURE



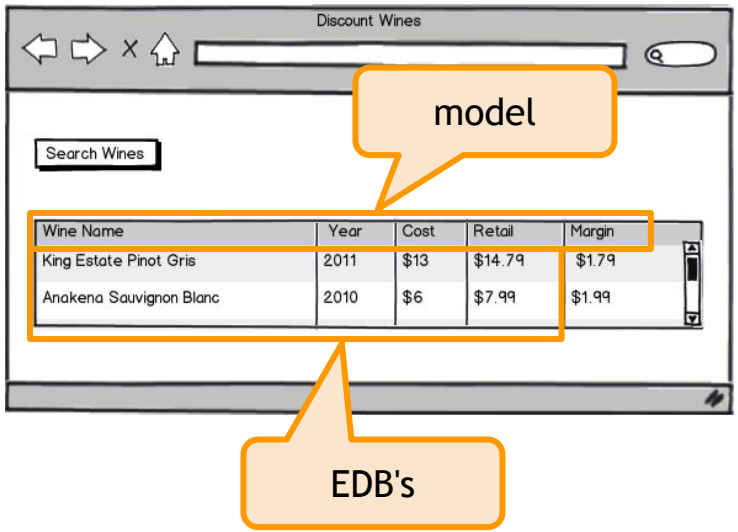


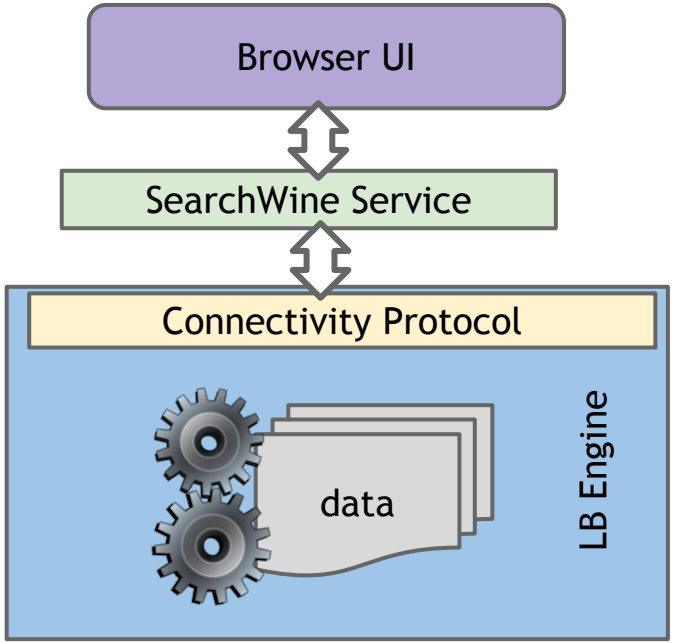
DISCOUNTWINES ARCHITECTURE



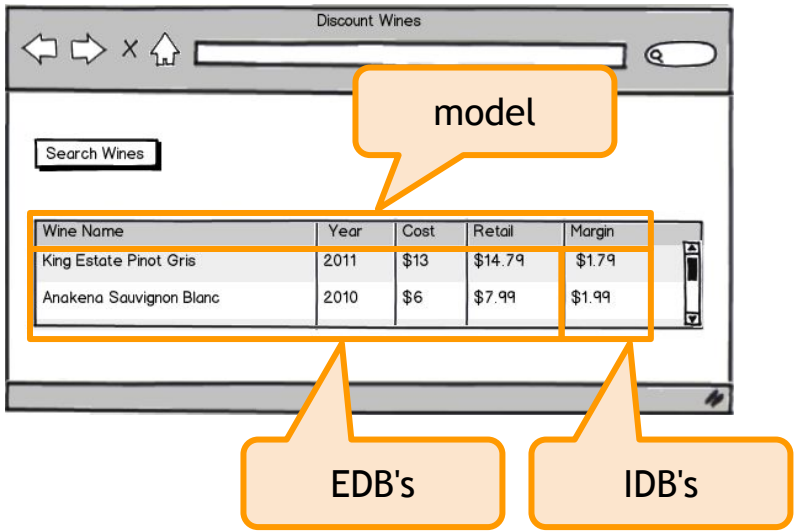


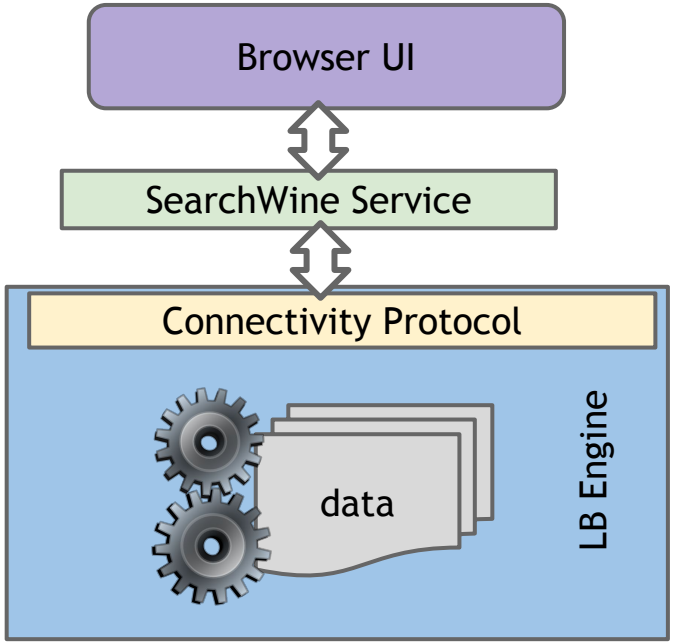
DISCOUNTWINES ARCHITECTURE



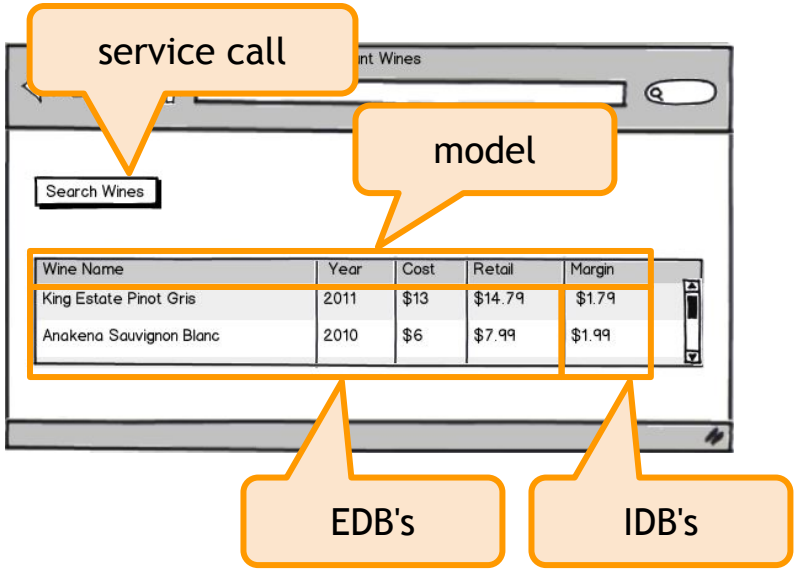


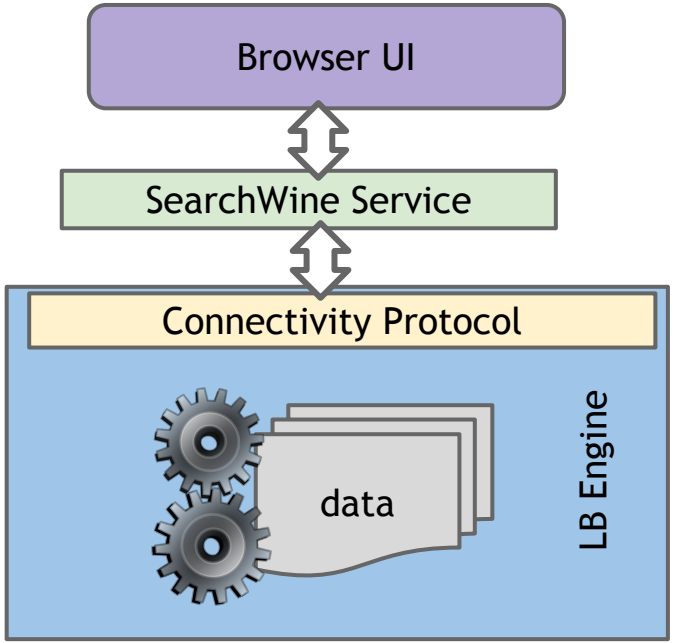
DISCOUNTWINES ARCHITECTURE





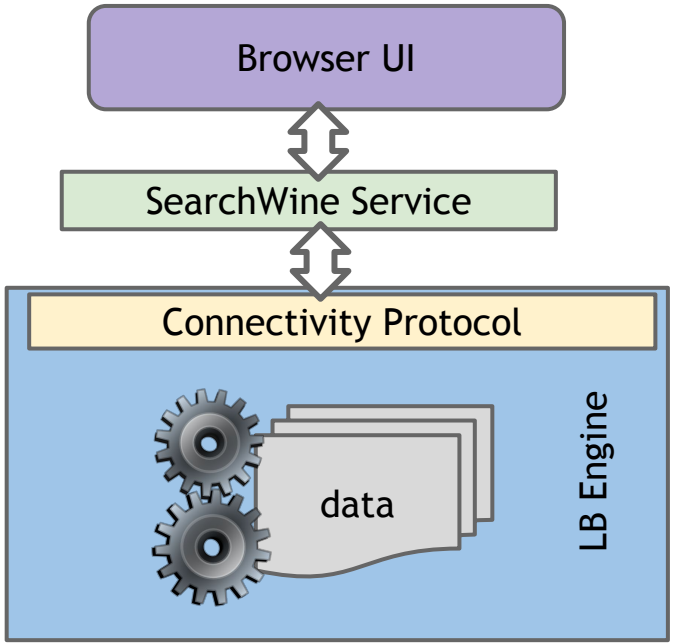
DISCOUNTWINES ARCHITECTURE





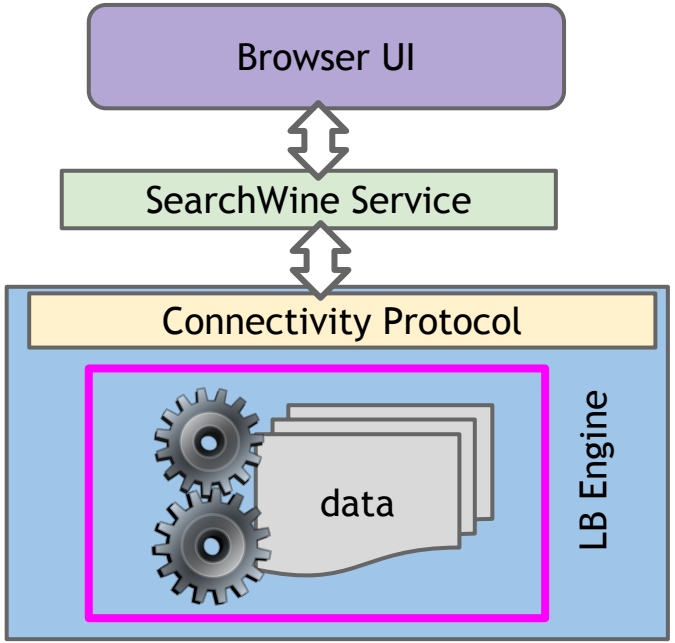
APPLICATION SOURCE CODE

```
$ ls discountwines

  logiql/
    core.project
    core/
    services/

  data/
```



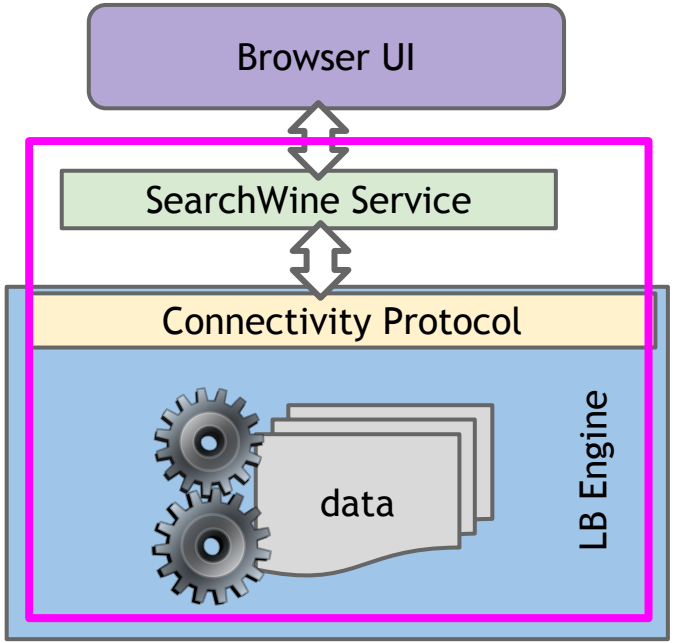
THIS LESSON

- code organization
  - project
  - modules
- unit testing the logic
  - lb unit
  - constraints (assertions)



NEXT LESSON

- services
  - protocol
  - logic
  - configuration
- testing services



BUILDING DISCOUNTWINES

```
$ cd discountwines

$ lb config

$ make
```

creates a workspace `discountwines`

installs application logic

loads test data

RUNNING THE UI CLIENT

```
$ cd discountwines

$ lb config

$ make
```

creates a workspace `discountwines`

installs application logic

loads test data

```
$ make start-nginx
```

runs UI client to service on http://localhost:8086

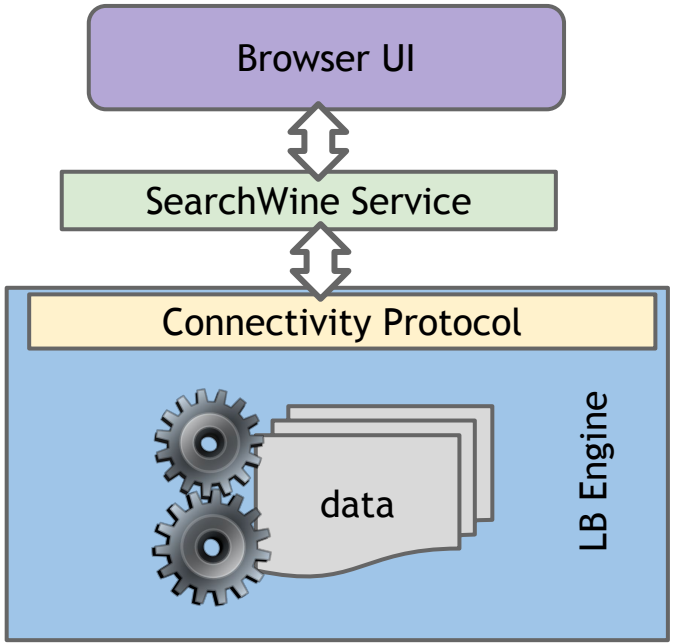
MODEL, LOGIC, AND DATA

```
$ ls discountwines
```

```
logiql/
  core.project
  core/

  services/
  protocols/

data/
```

Model & Logic

Data



ANATOMY OF A LOGICBLOX PROJECT

- project manifest

logiql/core.project

```
core, projectname

core, module
services, module
```

ANATOMY OF A LOGICBLOX PROJECT

- project manifest

logiql/core.project

```
core, projectname


core, module

services, module
```

ANATOMY OF A LOGICBLOX PROJECT

- project manifest

logiql/core.project

```
core, projectname

core, module
services, module
```

ANATOMY OF A LOGICBLOX PROJECT

- project manifest

logiql/core.project

```
core, projectname

core, module

services, module
```

- module
  - a collection of inter-dependent .logic files
  - directory structure -> namespace

```
core/
  wine.logic
  attributes.logic
```

ANATOMY OF A LOGICBLOX PROJECT

- project manifest

logiql/core.project

```
core, projectname

core, module

services, module
```

- module
  - a collection of inter-dependent .logic files
  - directory structure -> namespace

```
core/
    wine.logic
    attributes.logic
```

core:wine*

ANATOMY OF A LOGICBLOX BLOCK

- project manifest

logiql/core.project

```
core, projectname

core, module

services, module
```

- module
  - a collection of inter-dependent .logic files
  - directory structure -> namespace

```
core/
    wine.logic
    attributes.logic
```

core:wine

core:attributes

ANATOMY OF A MODULE BLOCK

logiql/core/wine.logic

```
block(`wine) {
  alias(`attributes:year, `year),

  export(`{
    wine(wine), wine_id(wine : id) -> int(id).
    ...
  }),

  clauses(`{
    margin(wine, margin)
      <- retail(wine, retail),
         cost(wine, cost),
          margin = retail - cost.

  })
} <-- .
```

ANATOMY OF A MODULE BLOCK

logiql/core/wine.logic

```
block(`wine) {
  alias(`attributes:year, `year),

  export(`{
    wine(wine), wine_id(wine : id) -> int(id).
  }),

  clauses(`{
    margin(wine, margin)
      <- retail(wine, retail),
         cost(wine, cost),
          margin = retail - cost.

  })
} <-- .
```

core:wine:wine

CLOSER LOOK : CORE/WINE.LOGIC

```
block (`wine) {
  alias(`attributes:year, `year),
  export(`{
    wine(wine), wine_id(wine : id) -> int(id).
    hasDescription(wine, name) -> wine(wine), string(name).
    ofYear(wine,year) -> wine(wine), year(year).
    cost(wine,price) -> wine(wine), float(price).
    retail(wine,price) -> wine(wine), float(price).
    margin(wine,margin) -> wine(wine), float(margin).
  }),
  clauses(`{
    margin(wine,margin)
      <- retail(wine,retail),
         cost(wine,cost),
     margin = retail - cost.
  })
} <-- .
```

CORE/ATTRIBUTES.LOGIC

logiql/core/attributes.logic

```
block(`attributes) {

  export(`{

    year(x), year_id(x : id) -> int(id).

  })

} <-- .
```

POP QUIZ

logiql/core/attributes.logic

```
block(`attributes) {

  export(`{

    year(x), year_id(x : id) -> int(id).

  })

} <-- .
```

- what is the fully qualified name of year?
- what about year_id?

POP QUIZ ANSWER

logiql/core/attributes.logic

```
block(`attributes) {

  export(`{

    year(x), year_id(x : id) -> int(id).

  })

} <-- .
```

- what is the fully qualified name of year?
  - core:attributes:year
- what about year_id?
  - core:attributes:year_id

GET ANSWERS WITH LB COMMANDS

```
$ lb list discountwines
```

```
boolean
boolean:and
boolean:bitand
boolean:bitnot
boolean:bitor
boolean:bitxor
boolean:cond
boolean:eq_2
boolean:eq_3
boolean:ge_2
boolean:ge_3
boolean:gt_2
boolean:gt_3
boolean:le_2
boolean:le_3
boolean:lt_2
boolean:lt_3
boolean:ne_2
boolean:ne_3
boolean:not
boolean:or
boolean:string:convert
core:attributes:year
...
```

GET ANSWERS WITH LB COMMANDS

```
$ lb list discountwines | grep year

 core:attributes:year
 core:attributes:year:cond
 core:attributes:year:eq_2
 core:attributes:year:ne_2
 core:attributes:year:string:convert
 core:attributes:year_id
 datetime:year
 datetime:yearTZ
 protocols:searchWines:Wine_year
```

GET ANSWERS WITH LB COMMANDS

```
$ lb list discountwines | grep year
 core:attributes:year
 core:attributes:year:cond
 core:attributes:year:eq_2
 core:attributes:year:ne_2
 core:attributes:year:string:convert
 ....

$ lb predinfo discountwines core:attributes:year

 info {
   name: "core:attributes:year"
   qualified_name: "core:attributes:year"
   arity: 1
   key_arity: 1
   value_arity: 0
   key_argument: "core:attributes:year"
   is_entity: true
 ...
 }
```

COMPILING A PROJECT

```
$ lb compile project datalog/core.project

$ lb compile -h
```

COMPILING A PROJECT

```
$ lb compile project datalog/core.project

$ lb compile -h
```

- for this application

```
$ make [clean]
```

LOADING DATA

- check config.py:

```
def load_test_data(ws):
  return [
     'echo "Adding sample wine data"',
     'lb exec ' + ws + ' --file data/wine.logic'
  ]
```

- check data/wines.logic
  - EDB rules
  - clearly not scalable to 100,000's of wines
  - how to load data in real applications: lesson 7

IN-CLASS EXERCISE : QUERIES

- provide the lb query command that queries all wines and their descriptions

- provide the lb query command that queries all wines of year < 2010

IN-CLASS EXERCISE : QUERIES

- provide the lb query command that queries all wines and their descriptions

```
$ lb query discountwines '
    _(wine, desc)
      <- core:wine:hasDescription(wine, desc).'
```

- provide the lb query command that queries all wines of year < 2010

IN-CLASS EXERCISE : QUERIES

- provide the lb query command that queries all wines and their descriptions

```
$ lb query discountwines '
    _(wine, desc)
      <- core:wine:hasDescription(wine, desc).'
```

- provide the lb query command that queries all wines of year < 2010

```
$ lb query discountwines '
    _(wine,desc)
      <- core:wine:ofYear(wine, year),
         core:attributes:year_id(year, id),
          id < 2010 ,core:wine:hasDescription(wine, desc) .'
```

UNIT TESTING LOGIC : LB UNIT

- suites of tests, where each suite contains
  - setup script
    - setUp.lb
  - teardown script
    - tearDown.lb
  - test scripts
    - test1.lb, test2.lb, etc.
  - setup and teardown script run for each test script

USING LB UNIT

- run a suite

```
$ lb unit --suite-dir tests/lb_unit [ --progress ]
```

- run a single test

```
$ lb unit --test tests/lb_unit/suiteBasic/test1.lb
```

CLOSER LOOK : SETUP.LB & TEARDOWN.LB

tests/lb_unit/suiteBasic/setUp.lb

```
create --unique

transaction

installProject --dir ${DISCOUNTWINES_HOME}/build/sepcomp/core

exec --file ${DISCOUNTWINES_HOME}/data/wine.logic

commit
```

CLOSER LOOK : SETUP.LB & TEARDOWN.LB

tests/lb_unit/suiteBasic/setUp.lb

```
create --unique

transaction

installProject --dir ${DISCOUNTWINES_HOME}/build/sepcomp/core

exec --file ${DISCOUNTWINES_HOME}/data/wine.logic

commit
```

tests/lb_unit/suiteBasic/tearDown.lb

```
close --destroy
```

WHAT'S IN A TEST

tests/lb_unit/suiteBasic/testMargin.lb

```
transaction

// test: retail = margin + cost
exec <doc>

core:wine:margin(wine, margin),
core:wine:cost(wine, cost)
  -> core:wine:retail(wine, margin + cost).

</doc>

commit
```

WHAT'S IN A TEST

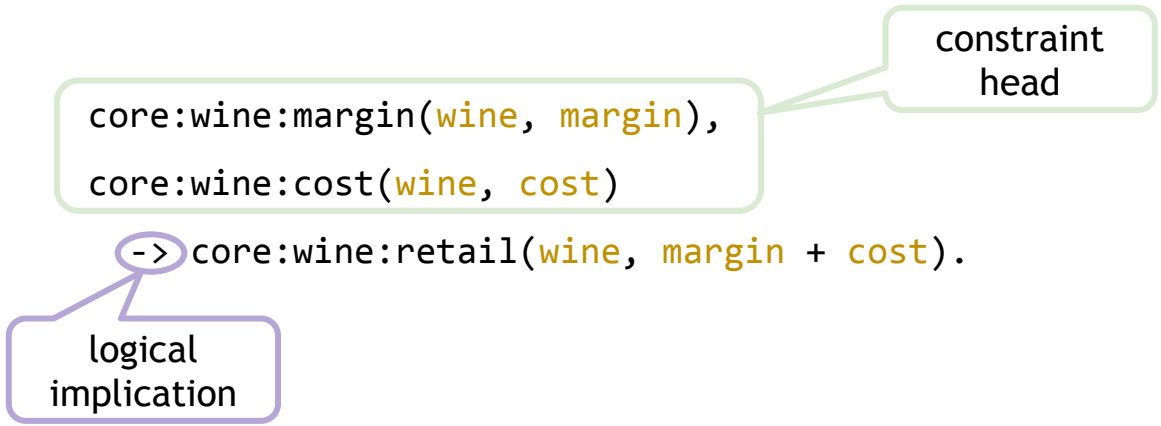
tests/lb_unit/suiteBasic/testMargin.lb

```
transaction

// test: retail = margin + cost
exec <doc>

core:wine:margin(wine, margin),
core:wine:cost(wine, cost)
  -> core:wine:retail(wine, margin + cost).

</doc>

commit
```

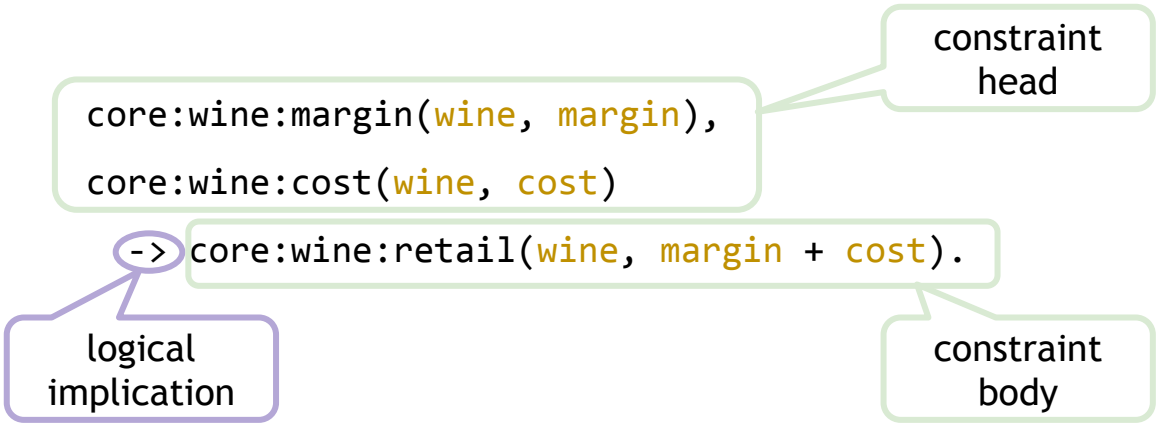
CONSTRAINTS == ASSERTION

```
core:wine:margin(wine, margin),
core:wine:cost(wine, cost)
  -> core:wine:retail(wine, margin + cost).
```

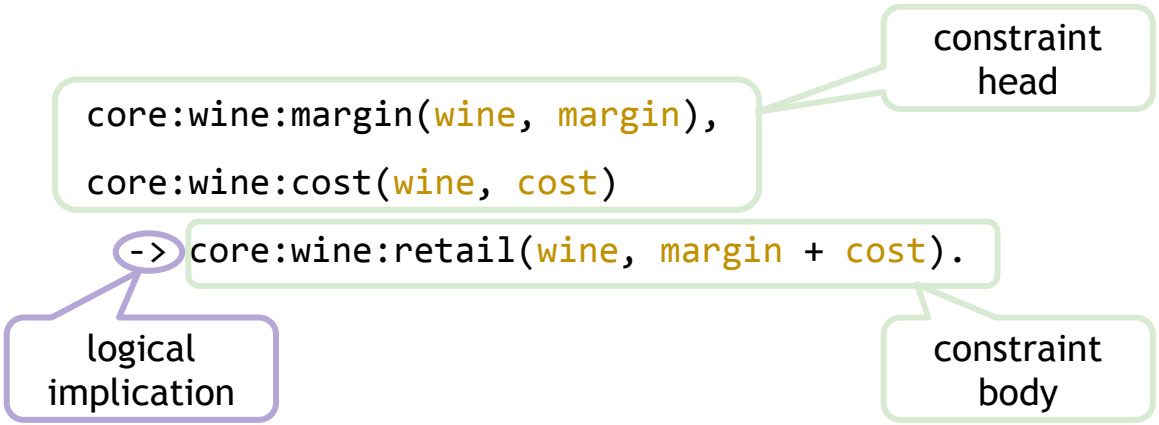
CONSTRAINTS == ASSERTION



CONSTRAINTS == ASSERTION



CONSTRAINTS == ASSERTION



If constraint head is true, then constraint body must be true
Otherwise, it's a constraint violation

CHECK CONSTRAINT

```
core:wine:margin(wine, margin),
core:wine:cost(wine, cost)
  -> core:wine:retail(wine, margin + cost).
```

source

```
!(core:wine:margin(wine, margin),
  core:wine:cost(wine, cost),
  !core:wine:retail(wine, margin + cost)).
```

runtime

CHECK CONSTRAINT

```
core:wine:margin(wine, margin),
core:wine:cost(wine, cost)
    -> core:wine:retail(wine, margin + cost).
```

source

```
!(core:wine:margin(wine, margin),
   core:wine:cost(wine, cost),
   !core:wine:retail(wine, margin + cost)).
```

runtime

UNDERSTANDING CONSTRAINT VIOLATION MESSAGE

```
1) tests: tests/lb_unit/suiteBasic/testMargin
   (E) testMargin line 13:
...
__block0 (line 1, column 1): Integrity constraint violation.
  Occurred in logic clause:

!(core:wine:margin(wine, margin),
  core:wine:cost(wine, cost),
  float64:add[margin, cost] = retail,
  !core:wine:retail(wine, retail)).

----------------------------
Constraint with values that caused the violation:

!(core:wine:margin(0, 5.49),
  core:wine:cost(0, 4.5),
  float64:add[5.49, 4.5] = 9.99,
  !core:wine:retail(0, 9.99)).
```

UNDERSTANDING CONSTRAINT VIOLATION MESSAGE

```
1) tests: tests/lb_unit/suiteBasic/testMargin
   (E) testMargin line 13:
...
__block0 (line 1, column 1): Integrity constraint violation.
  Occurred in logic clause:

!(core:wine:margin(wine, margin),
  core:wine:cost(wine, cost),
  float64:add[margin, cost] = retail,
  !core:wine:retail(wine, retail)).

----------------------------
Constraint with values that caused the violation:

!(core:wine:margin(0, 5.49),
  core:wine:cost(0, 4.5),
  float64:add[5.49, 4.5] = 9.99,
  !core:wine:retail(0, 9.99)).
```

UNDERSTANDING CONSTRAINT VIOLATION MESSAGE

```
1) tests: tests/lb_unit/suiteBasic/testMargin
   (E) testMargin line 13:
...
__block0 (line 1, column 1): Integrity constraint violation.
  Occurred in logic clause:

!(core:wine:margin(wine, margin),
  core:wine:cost(wine, cost),
  float64:add[margin, cost] = retail,
  !core:wine:retail(wine, retail)).

----------------------------
Constraint with values that caused the violation:

!(core:wine:margin(0, 5.49),
  core:wine:cost(0, 4.5),
  float64:add[5.49, 4.5] = 9.99,
  !core:wine:retail(0, 9.99)).
```

RULE VS. CONSTRAINT

- constraint : the implication must hold
  - otherwise, it's an exception

```
core:wine:margin(wine,margin),
core:wine:cost(wine,cost)
  -> core:wine:retail(wine, margin + cost).
```

- rule : MAKE the implication hold
  - by deriving values into the head

```
core:wine:retail(wine, margin + cost)
  <- core:wine:margin(wine,margin),
     core:wine:cost(wine,cost).
```

RUNTIME VS. COMPILE-TIME

- constraints checked @ runtime

```
core:wine:margin(wine,margin),
core:wine:cost(wine,cost)
  -> core:wine:retail(wine, margin + cost).
```

- constraints checked @ compile-time
  - compiler guarantees that they will never ever be violated, for any possible data, i.e., predicate type declarations

```
margin(wine,margin)
  -> wine(wine), float(margin).
```

FUNCTIONAL DEPENDENCY

- a common constraint

```
core:wine:ofYear(wine,year1),
core:wine:ofYear(wine,year2)
  -> year1 = year2.
```

FUNCTIONAL DEPENDENCY

- a common constraint

```
core:wine:ofYear(wine,year1),
core:wine:ofYear(wine,year2)
  -> year1 = year2.
```

a function

FUNCTIONAL DEPENDENCY

- a common constraint

```
core:wine:ofYear(wine,year1),
core:wine:ofYear(wine,year2)
  -> year1 = year2.
```

a function

- SQL analogy

```
CREATE TABLE core:wine:ofYear (
  wine int NOT NULL,
  year int NOT NULL
  PRIMARY KEY (wine)
)
```

FUNCTIONAL DEPENDENCY

- a common constraint

```
core:wine:ofYear(wine,year1),
core:wine:ofYear(wine,year2)
  -> year1 = year2.
```

a function

- common enough we created a short-hand

```
core:wine:ofYear[wine] = year
  -> core:wine:wine(wine),
     core:attributes:year(year).
```

SUMMARY OF CONCEPTS

- code organization
  - LogicBlox project specification and compilation
  - modules: how, why, fully qualified names
- unit testing with “lb unit”
  - structure of test suites & test scripts
- constraints == assertions
  - how to read constraint violation messages
  - runtime vs. compile-time constraints
  - functional dependency constraints

LESSON 4 LAB EXERCISES

THANK YOU.