GUIDE
Graphical User Interface Development Environment

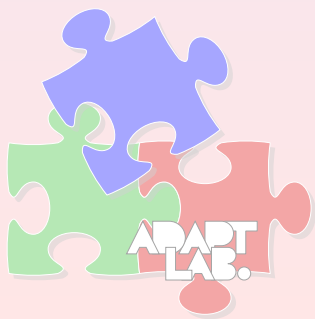
Andrea Longoni
Id. Number: 13896A

Università degli Studi di Milano
Computer Science Department
MSc in Computer Science

Advisor: Prof. Walter Cazzola
Co-Advisor: Dr. Luca Favalli
Co-Advisor: Mr. Matteo Brancaleoni

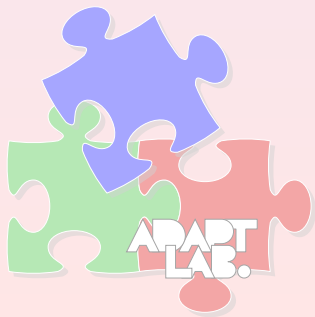
09/04/2025

LM-18 - Computer science
Academic Year 2023-2024



Problem Statement
Graphical User Interface Development

Over the years, Graphical User Interfaces (GUIs) have become a fundamental component of software development.

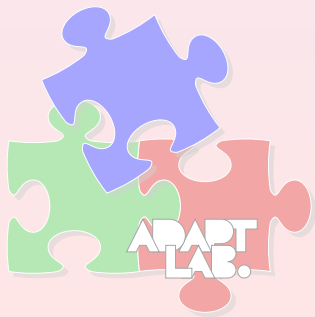


Problem Statement
Graphical User Interface Development

Over the years, Graphical User Interfaces (GUIs) have become a fundamental component of software development.

The increase in the number of UI technologies presents a major challenge:

- More technologies mean more languages to learn.
- UI migration requires a full code rewrite.
- Maintaining consistency across platforms adds time and effort.



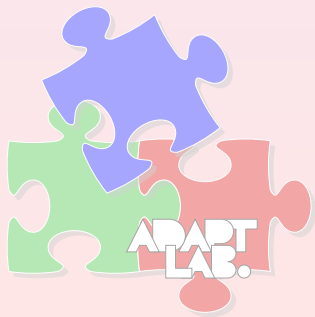
Problem Statement
Graphical User Interface Development

Over the years, Graphical User Interfaces (GUIs) have become a fundamental component of software development.

The increase in the number of UI technologies presents a major challenge:

- More technologies mean more languages to learn.
- UI migration requires a full code rewrite.
- Maintaining consistency across platforms adds time and effort.

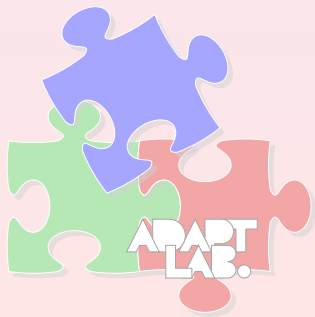
A more modular and framework-independent approach is needed to enable automatic UI generation across different technologies.



Domain-Specific Languages (DSLs)

A **Domain-Specific Language (DSL)** is a programming language tailored to a specific domain:

- **Higher-level abstraction** than General-Purpose Languages (GPLs)
- **Improves productivity** by focusing on domain concepts
- **Reduces errors** by limiting expressiveness to necessary operations
- **Examples**: SQL (databases), HTML (web pages), Gradle (build scripts)

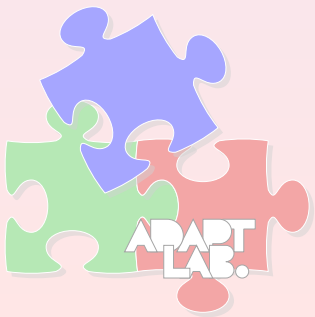


Domain-Specific Languages (DSLs)

A Domain-Specific Language (DSL) is a programming language tailored to a specific domain:

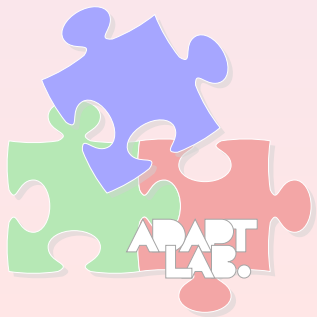
- Higher-level abstraction than General-Purpose Languages (GPLs)
- Improves productivity by focusing on domain concepts
- Reduces errors by limiting expressiveness to necessary operations
- Examples: SQL (databases), HTML (web pages), Gradle (build scripts)

GUIDE's DSL was implemented using Neverlang, a language workbench designed to facilitate modular language development with composition and reuse of language features.



Software Product Lines (SPLs)

A Software Product Line (SPL) is an approach to software development that enables the creation of multiple related software products from a shared set of core assets.

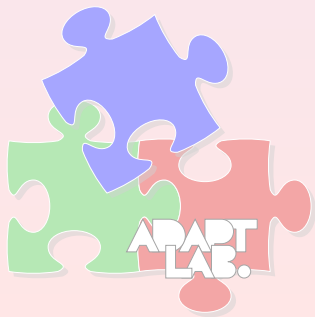


Software Product Lines (SPLs)

A Software Product Line (SPL) is an approach to software development that enables the creation of multiple related software products from a shared set of core assets.

SPLs rely on:

- Core Assets: Shared components and libraries
- Variability Management: Mechanisms to enable/disable features
- Automation: Tools to generate product variants efficiently



Software Product Lines (SPLs)

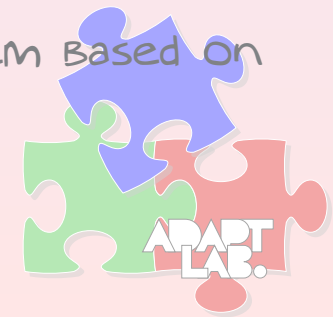
A Software Product Line (SPL) is an approach to software development that enables the creation of multiple related software products from a shared set of core assets.

SPLs rely on:

- Core Assets: Shared components and libraries
- Variability Management: Mechanisms to enable/disable features
- Automation: Tools to generate product variants efficiently

Key Benefits:

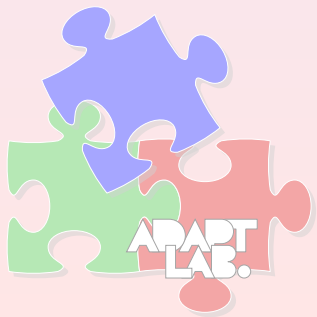
- Reuse: Common components across different product variants
- Scalability: New features can be added without major redesigns
- Customization: Generate tailored versions of a system based on feature selection



Architecture
Introduction

GUIDE (Graphical User Interface Development Environment) is a system that combines:

- Domain-Specific Language (DSL) for UI definition
- Software Product Line (SPL) for modularization
- Code generation framework targeting multiple languages



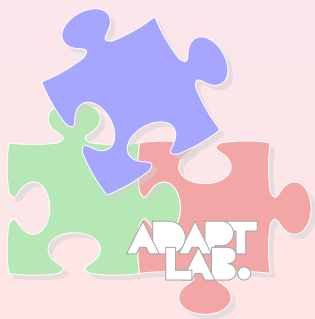
Architecture
Introduction

GUIDE (Graphical User Interface Development Environment) is a system that combines:

- Domain-Specific Language (DSL) for UI definition
- Software Product Line (SPL) for modularization
- Code generation framework targeting multiple languages

Key Features:

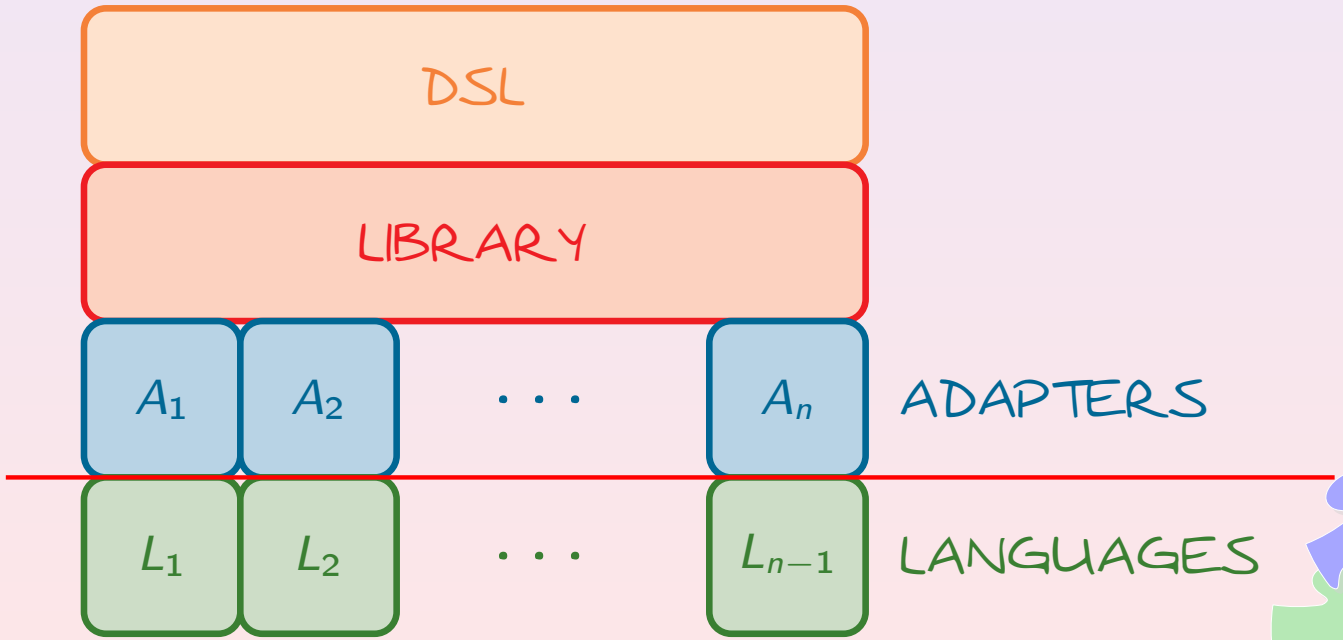
- Multi-language: Supports HTML, Elixir and Python
- Multi-framework: Supports Bootstrap, Phoenix LiveView, and Tkinter
- Multi-platform: Supports Web and Desktop applications
- Extensible: New components, layouts, and languages
- Automated: Reduces manual coding effort

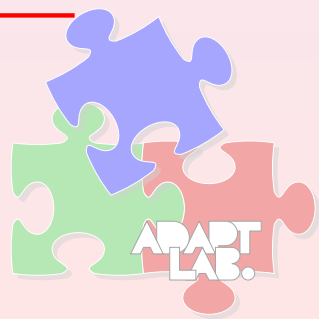


Architecture
Overview

The architecture of GUIDE is organized into three layers:

- DSL Layer: Defines the UI structure and behavior
- Library Layer: Provides core logic and reusable components
- Adapter Layer: Translates UI definitions into target languages





Architecture
Module Structure

DSL

Library

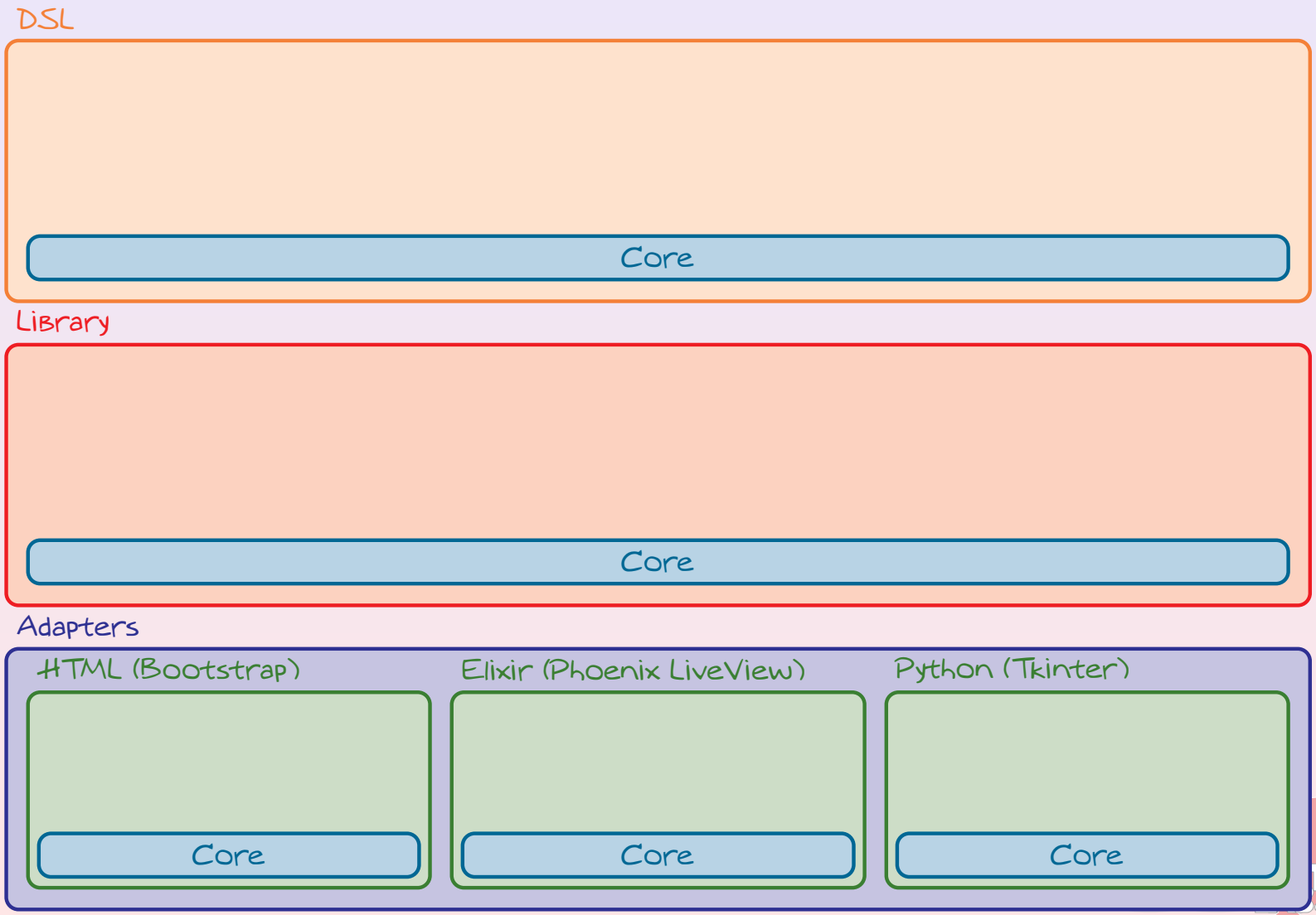
Adapters

HTML (Bootstrap)

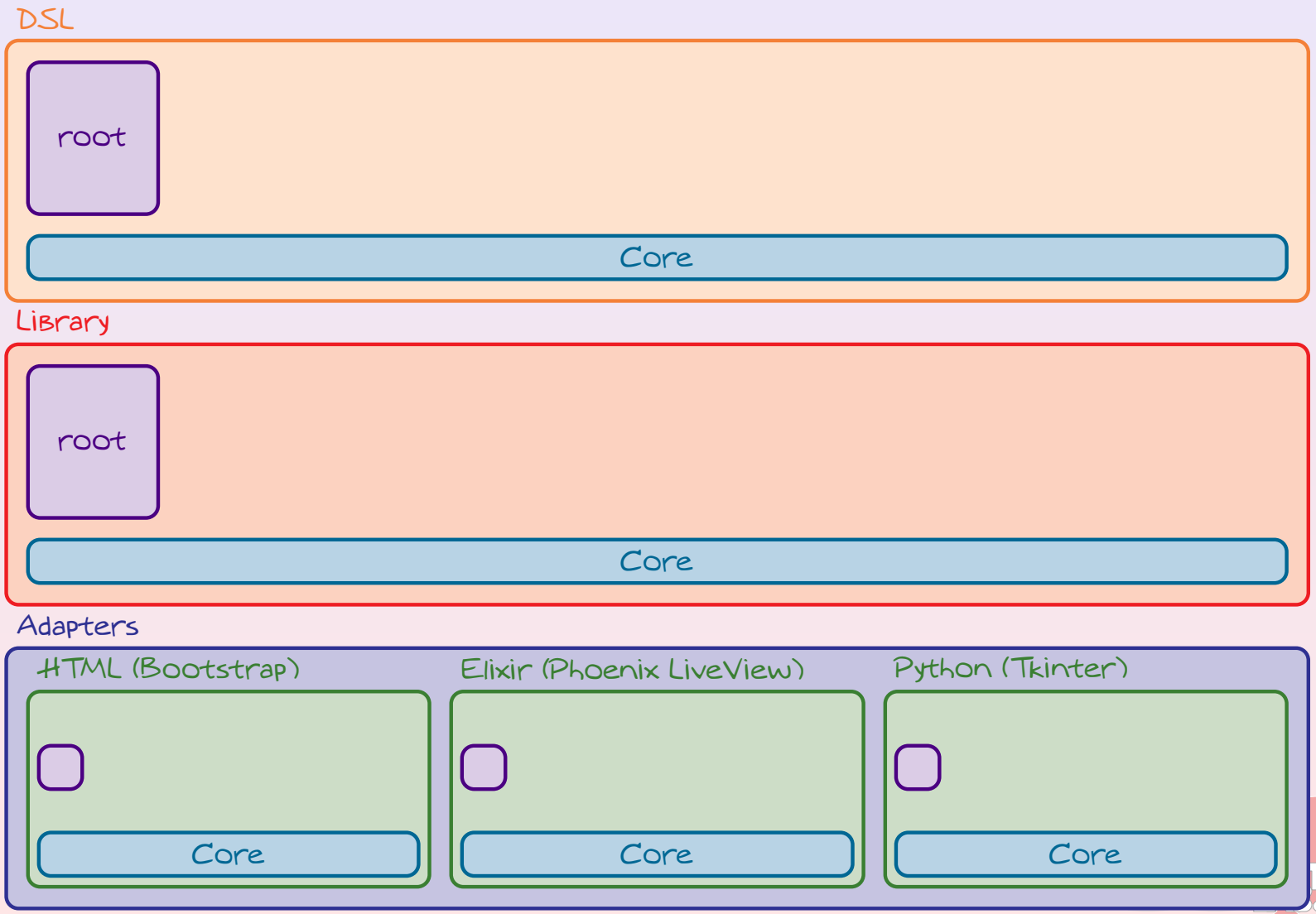
Elixir (Phoenix LiveView)

Python (Tkinter)

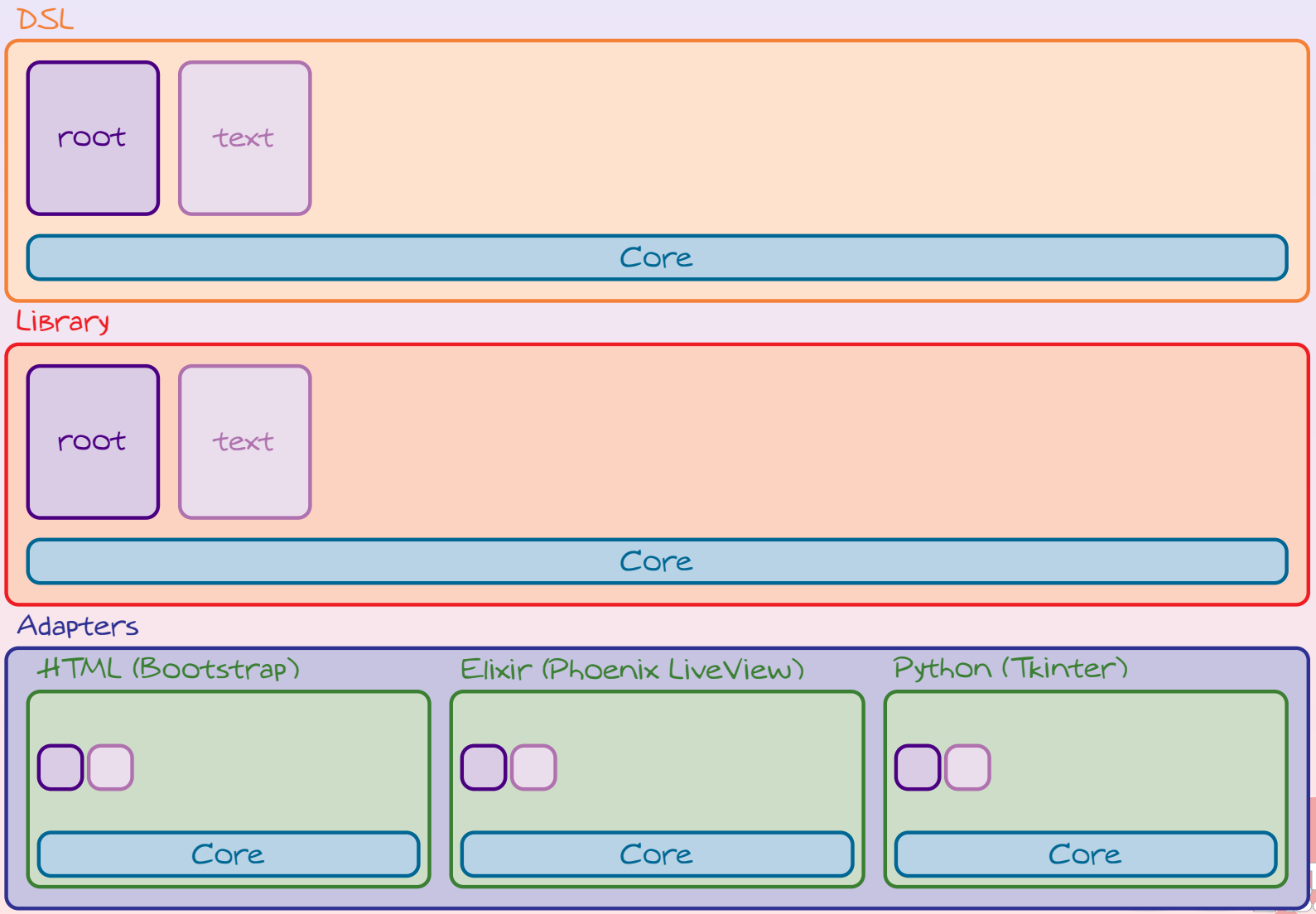
Architecture
Module Structure



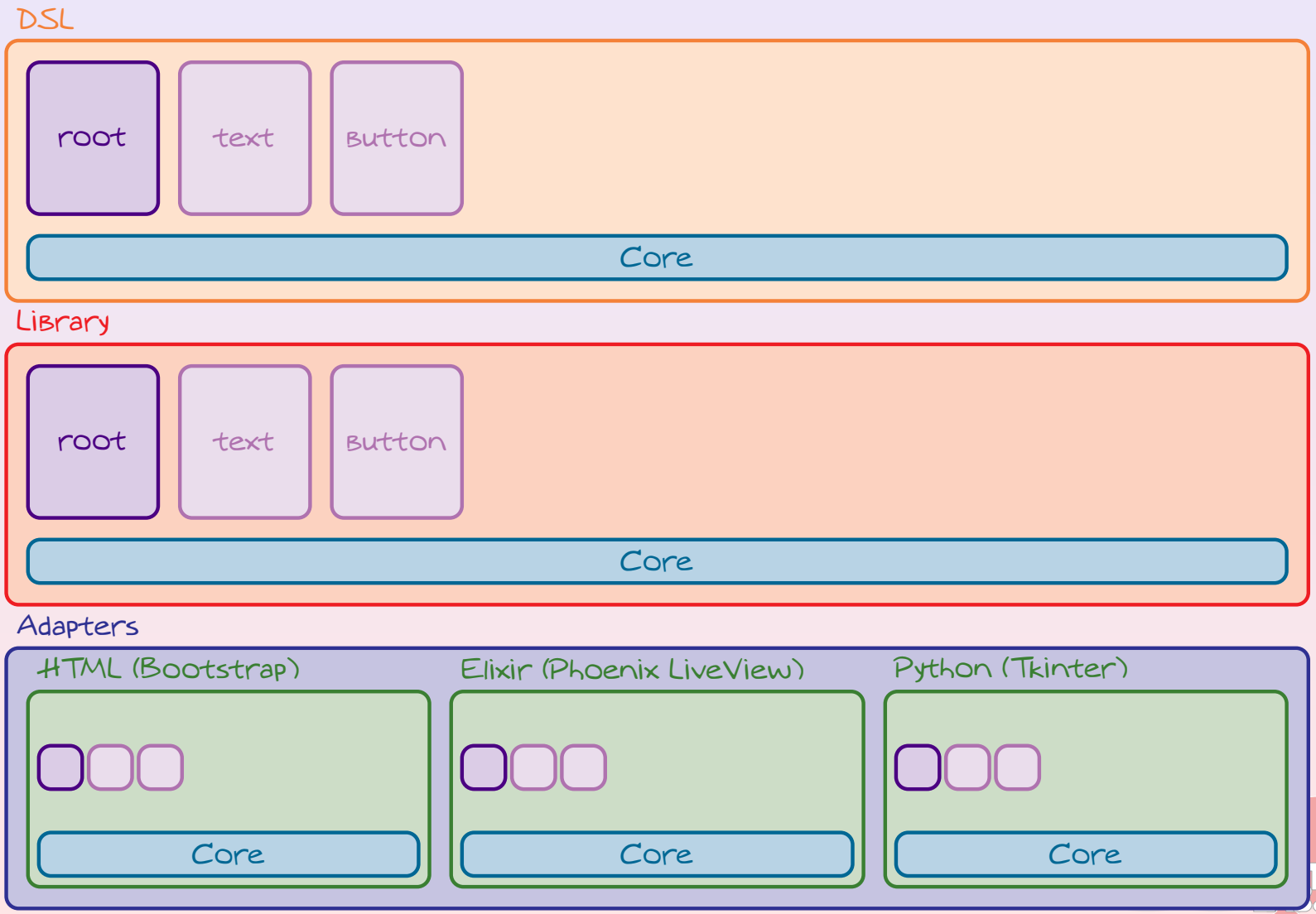
Architecture
Module Structure



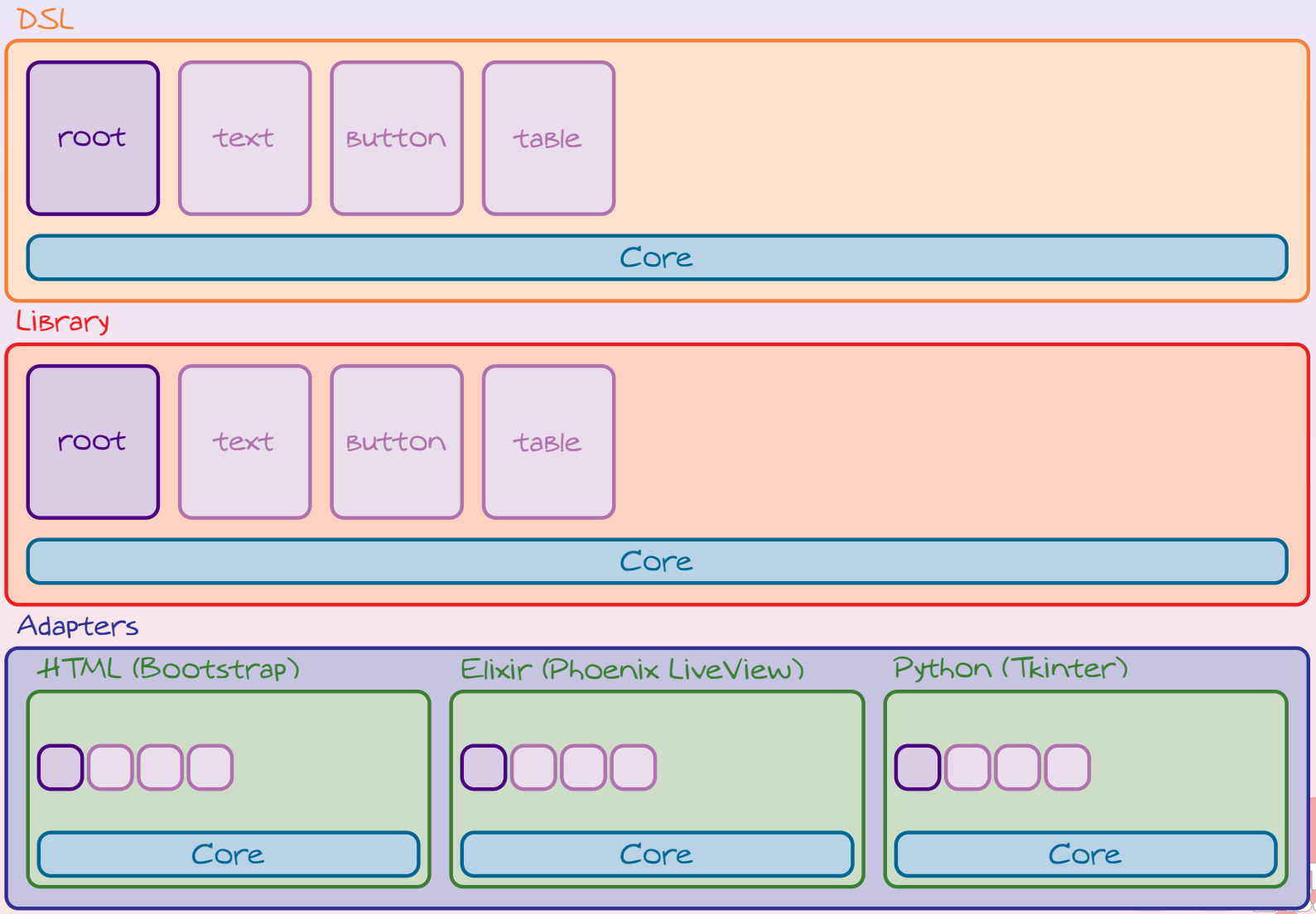
Architecture
Module Structure



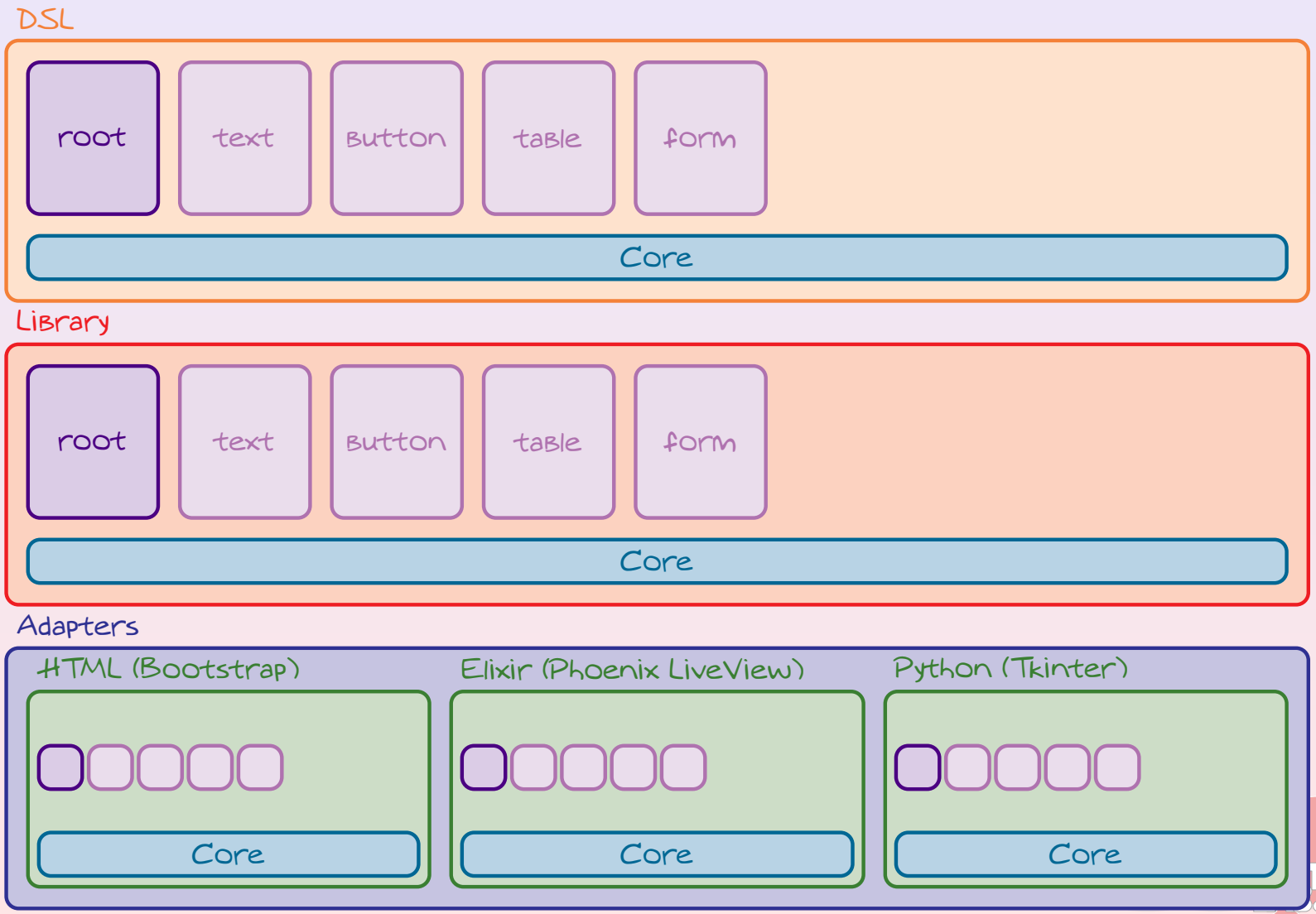
Architecture
Module Structure



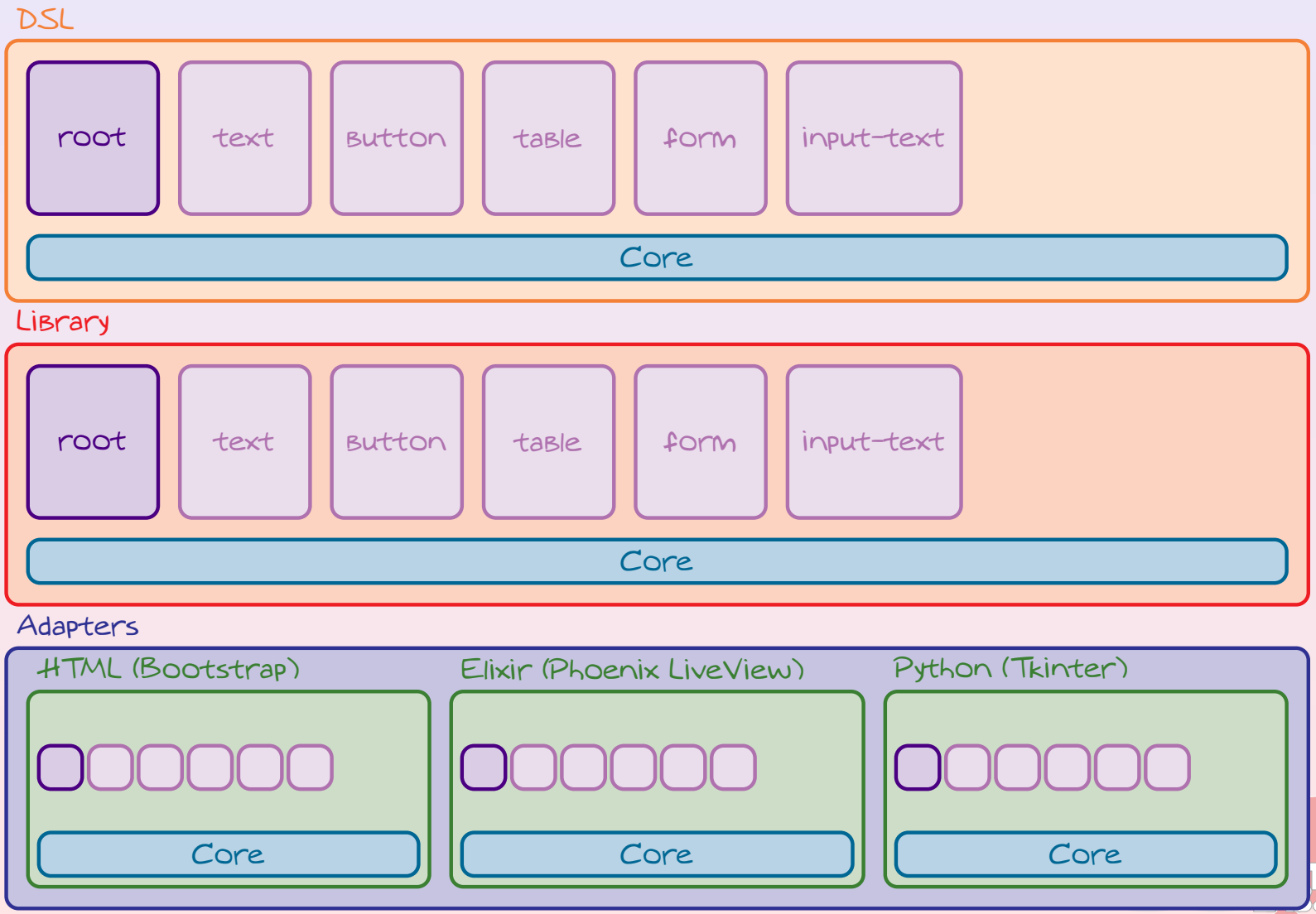
Architecture
Module Structure



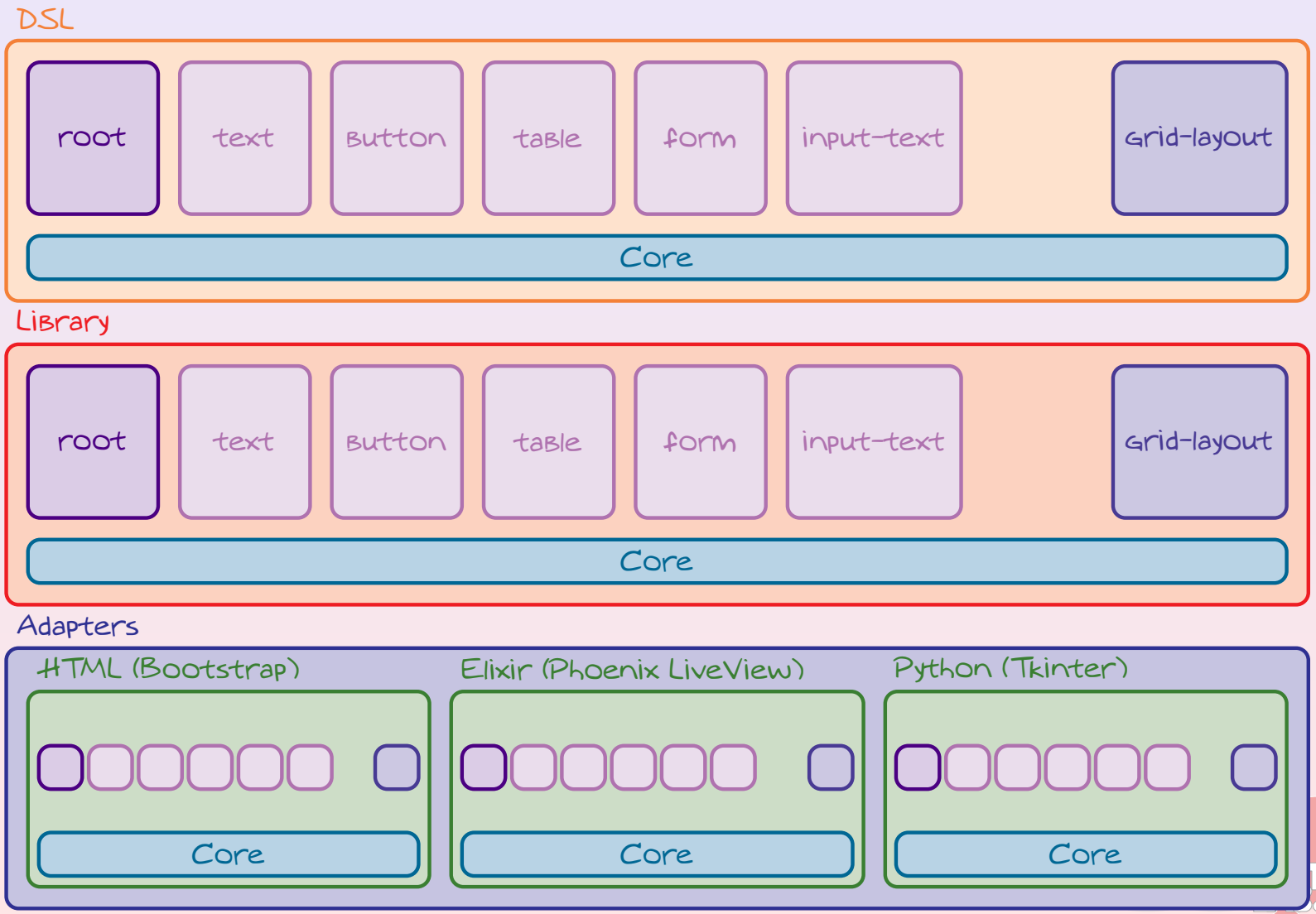
Architecture
Module Structure



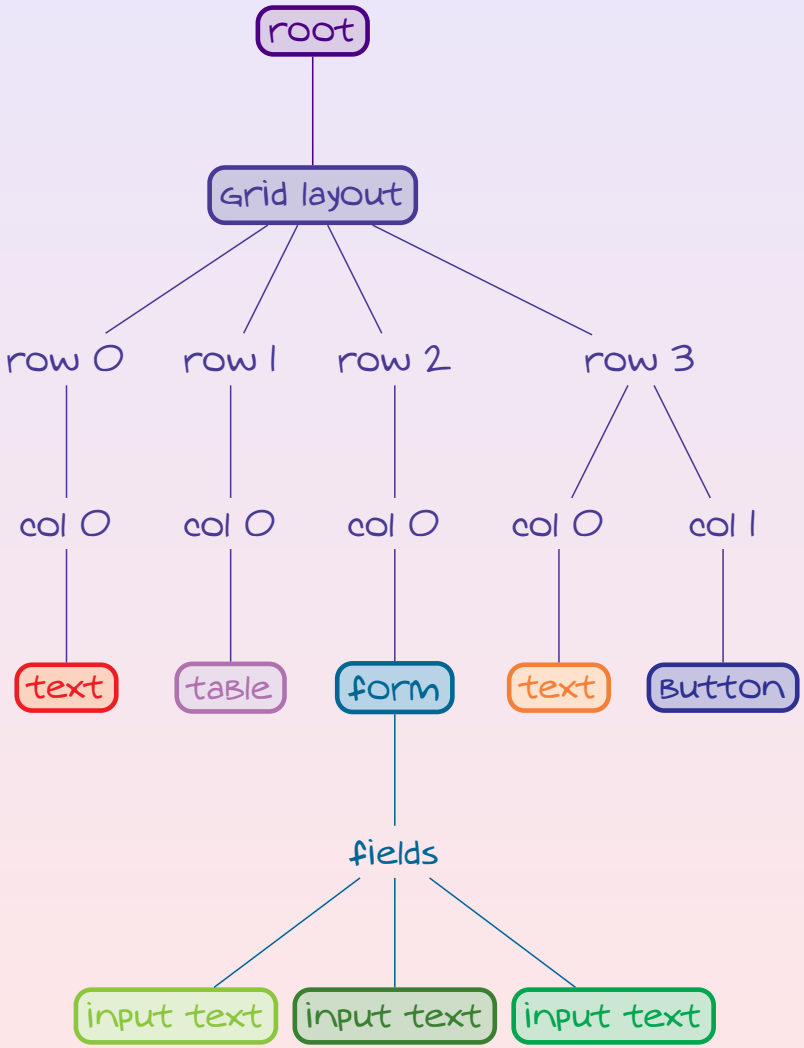
Architecture
Module Structure

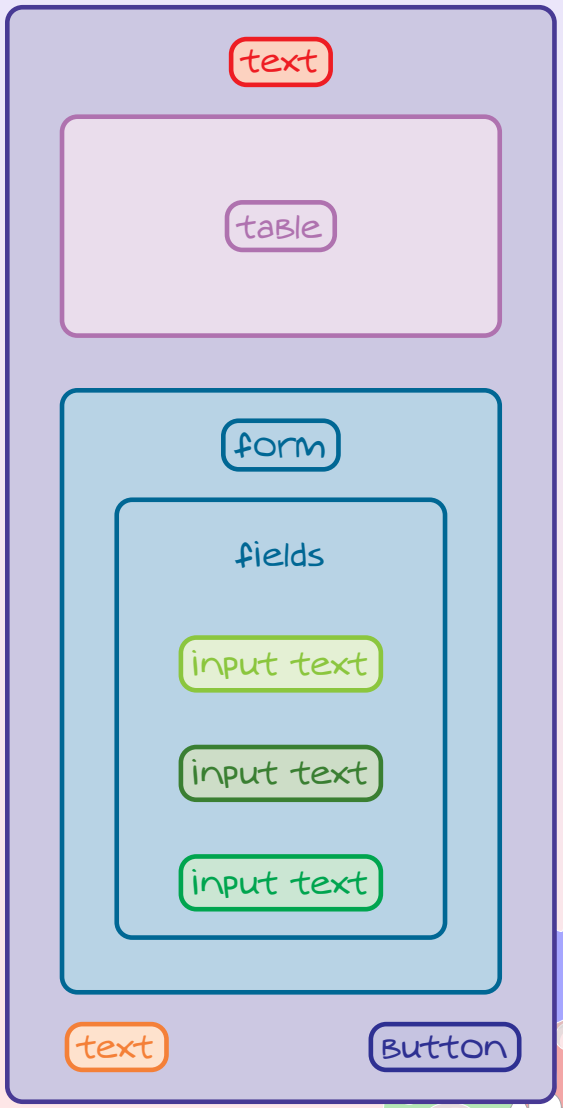


Architecture
Module Structure



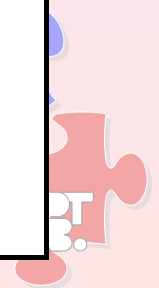
GUIDE in Action
Example UI Structure





GUIDE in Action
DSL Example


```
Module salaries {




}
```



GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

  gridLayout = GridLayout {
    titleText;
    salariesTable;
    addForm;
    deleteText, deleteButton;
  }

}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

  gridLayout = GridLayout {
    titleText;
    salariesTable;
    addForm;
    deleteText, deleteButton;
  }

  titleText = Text {
    textColor: #color("blue"),
    backgroundColor: #color("#FFFF00"),
    content: title
  }

  deleteText = Text {
    textColor: deletePrimaryColor,
    backgroundColor: deleteSecondaryColor,
    content: "Delete all salaries: "
  }

}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

  gridLayout = GridLayout {
    titleText;
    salariesTable;
    addForm;
    deleteText, deleteButton;
  }

  deleteButton = Button {
    id: deleteButton,
    textColor: deletePrimaryColor,
    backgroundColor: deleteSecondaryColor,
    content: "Delete",
    onClick: @deleteButtonOnClick
  }

}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

  gridLayout = GridLayout {
    titleText;
    salariesTable;
    addForm;
    deleteText, deleteButton;
  }

  salariesTable = Table {
    id: table,
    headers: ["Name", "Surname", "Salary"],
      rows: [
          "John", "Doe", "1000";
          "Jane", "Doe", "2000";
      ]
  }
}
```

GUIDE in Action
DSL Example


```
Module salaries {

  root = Root {
    module: "salaries",
    title: title,
    height: 600,
    width: 800,
    callbacksFilename: "salaries_callbacks",
    layout: gridLayout
  }

  title = "Salaries"
  deletePrimaryColor = #color("red")
  deleteSecondaryColor = #color("#000000")

  gridLayout = GridLayout {
    titleText;
    salariesTable;
    addForm;
    deleteText, deleteButton;
  }

  addForm = Form {
    id: addForm,
    submitButtonText: "Add salary",
    onSubmit: @addFormOnSubmit,
    fields: [
      "Name": InputText {
        id: name,
        placeholder: "Name"
      },
      "Surname" : InputText {
        id: surname,
        placeholder: "Surname"
      },
      "Salary" : InputText {
        id: salary,
        placeholder: "Salary"
      }
    ]
  }

}
```

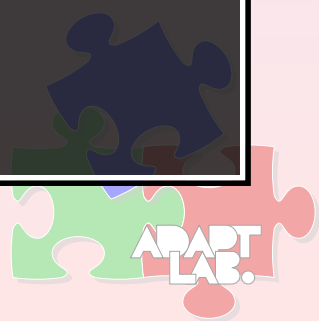
GUIDE in Action
Running the Example

GUIDE help:

```
> java -jar guide.jar --help
Usage: guide [-hV] -l=<language> -o=<output> -r=<rootName> [-s=<srcDir>]
Guide DSL compiler
  -h, --help              Show this help message and exit.
  -l, --language=<language>
                          Valid languages: python, elixir, html
  -o, --output=<output>   Output file
  -r, --root=<rootName>   Root name
  -s, --srcDir=<srcDir>   Source directory
  -V, --version           Print version information and exit.
```

Generate the code:

```
> java -jar guide.jar -l html -o salaries.html -r salaries.root
Successfully wrote to the file salaries.html

> java -jar guide.jar -l elixir -o salaries.ex -r salaries.root
Successfully wrote to the file salaries.ex

> java -jar guide.jar -l python -o salaries.py -r salaries.root
Successfully wrote to the file salaries.py
```



GUIDE in Action
HTML (Bootstrap) Generated Code


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Salaries</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
          ↪  rel="stylesheet">
    <!-- jQuery -->
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <!-- Callbacks functions -->
    <script src="salaries_callbacks.js"></script>
  </head>
  <body>
    <div id="HtmlGridLayout8" class="container">
  <div class="row">
  <div class="col">
<p id="HtmlText0" class="text-start" style="color:#0000FF; background-color:#FFFF00;">Salaries</p>
</div>
</div>
<div class="row">
  <div class="col">
<table id="table" class="table">
  <thead>
    <tr><th>Name</th><th>Surname</th><th>Salary</th></tr>
  </thead>
  <tbody>
```

GUIDE in Action
HTML (Bootstrap) Result

Salaries

Name	Surname	Salary
John	Doe	1000
Jane	Doe	2000

Name

Name

Surname

Surname

Salary

Salary

Add salary

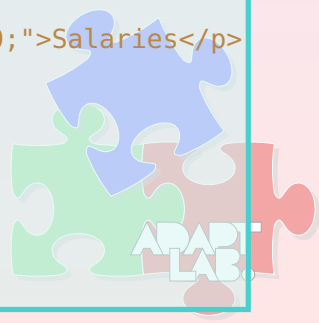
Delete all salaries:

Delete

GUIDE in Action

Elixir (Phoenix LiveView) Generated Code


```
defmodule Salaries do
  use Phoenix.LiveView
  alias Salaries_callbacks

  def mount(_params, _session, socket) do
    socket = assign(socket, :tableheaderOnLoad, ["Name", "Surname", "Salary"])
    socket = assign(socket, :tabletableOnLoad, [["John", "Doe", "1000"], ["Jane", "Doe", "2000"]])
    socket = assign(socket, :addFormOnSubmit, Salaries_callbacks.addFormOnSubmit())
    {:ok, socket}
  end

  def handle_event("addFormOnSubmit", value, socket) do
    Salaries_callbacks.addFormOnSubmit(value, socket)
  end

  def handle_event("deleteButtonOnClick", value, socket) do
    Salaries_callbacks.deleteButtonOnClick(value, socket)
  end

  def render(assigns) do
    ~H"""
<div id="ElixirGridLayout8" class="container mx-auto">
  <div class="grid grid-cols-1">
  <div class="col-span-1">
<p id="ElixirText0" class="text-left" style="color:#0000FF; background-color:#FFFF00;">Salaries</p>
</div>
</div>
<div class="grid grid-cols-1">
  <div class="col-span-1">
<table id="table" class="min-w-full divide-y divide-gray-200 table-auto">
  <thead class="bg-gray-50">
```



GUIDE in Action
Elixir (Phoenix LiveView) Result

Salaries

Name	Surname	Salary
John	Doe	1000
Jane	Doe	2000

Name

Name

Surname

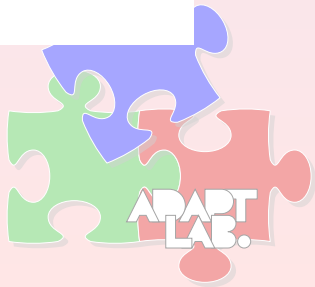
Surname

Salary

Salary

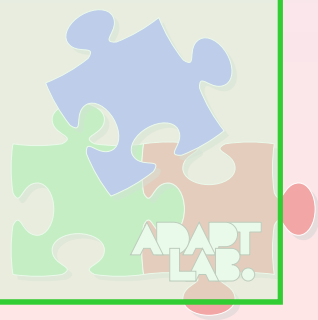
Add salary

Delete all salaries: Delete

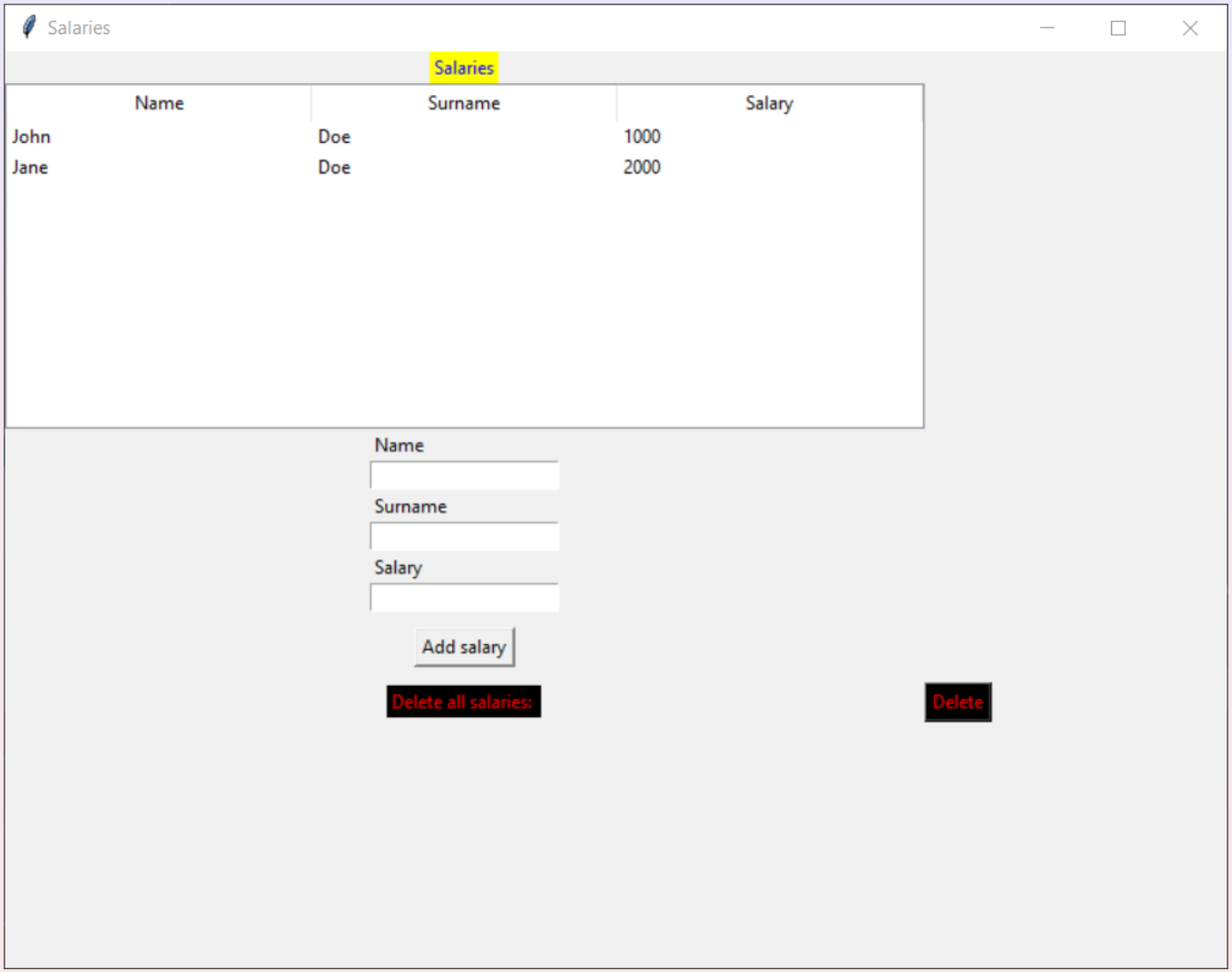


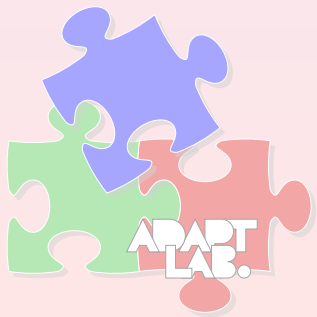
GUIDE in Action
Python (Tkinter) Generated Code


```
import tkinter as tk
from tkinter import ttk
import salaries_callbacks

class salaries:
  def __init__(self, PythonRoot9):
    self.PythonRoot9 = PythonRoot9
    self.PythonRoot9.title("Salaries")
    self.PythonRoot9.geometry("800x600")

    self.PythonText0 = tk.Label(master=self.PythonRoot9, text="Salaries", fg="#0000FF", bg="#FFFF00")
    self.PythonText0.grid(row=0, column=0)

    self.table = ttk.Treeview(master=self.PythonRoot9, columns=(0, 1, 2), show="headings")
    self.table.heading("0", text="Name")
    self.table.heading("1", text="Surname")
    self.table.heading("2", text="Salary")
    self.table.insert("", tk.END, values=["John", "Doe", "1000"])
    self.table.insert("", tk.END, values=["Jane", "Doe", "2000"])
    self.table.grid(row=1, column=0)

    self.addForm = tk.Frame(master=self.PythonRoot9)

    self.nameLabel = tk.Label(master=self.addForm, text="Name")
    self.nameLabel.pack(anchor="w")
    self.name = tk.Entry(master=self.addForm, state=tk.NORMAL)
    self.name.insert(0, "")
    self.name.pack(fill="x")

    self.surnameLabel = tk.Label(master=self.addForm, text="Surname")
    self.surnameLabel.pack(anchor="w")
    self.surname = tk.Entry(master=self.addForm, state=tk.NORMAL)
```



GUIDE in Action
Python (Tkinter) Result





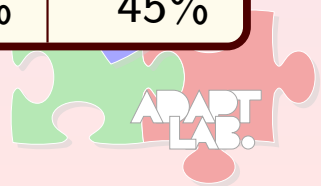
Experiments
Code Generation

Compare the amount of code written manually vs. generated by GUIDE

The GUIDE program in the example consists of:

- 63 lines of code (LOC)
- 1353 characters (Char)
- 60 minutes of development time

Language	LOC Gen.	Expansion Factor (LOC)	Char Gen.	Expansion Factor (Char)	Time Saved
HTML	62	−2%	2067	+53%	51%
Elixir	81	+29%	3232	+139%	63%
Python	55	−13%	2517	+86%	45%



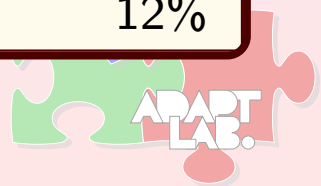
Experiments
Feature Extension

Evaluate the ease of adding new features using the code generation tool

Automatucally generated code for the new feature includes:

- Gradle module structure with dependencies and Build scripts
- Java classes declaration
- Test files declaration
- Factory interfaces definition
- Neverlang modules declaration

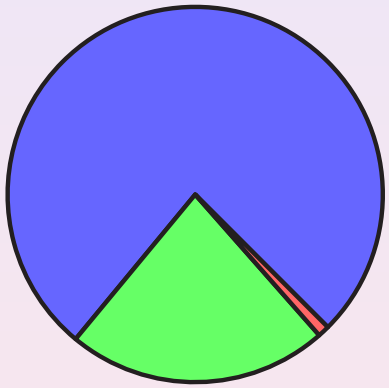
Feature Type	Feature LOC	Gen. LOC	Effort Reduction
New Language Adapter	$\sim$ 458	108	24%
New Component	$\sim$ 594	68	12%



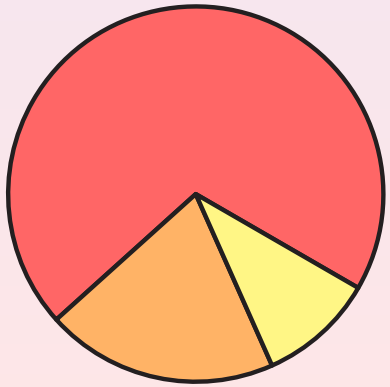
Experiments
Feature Selection

Measure the impact of feature selection on storage and distribuition efficiency

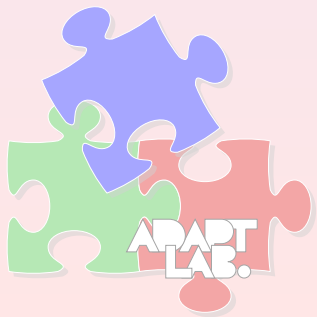
Size of the generated JAR file with all features enabled:



- Neverlang (77%)
- Other Dependencies (23%)
- GUIDE (1%)



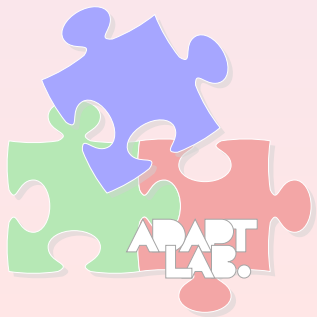
- GUIDE DSL (70%)
- Adapters (20%)
- GUIDE Library (10%)



Conclusion

Interesting results:

- GUIDE **simplifies GUI development** by combining DSLs and SPLs.
- Significant **reduction in manual code writing**.
- **Multi-language support** enables flexibility but requires adapters for new languages.
- Feature selection enables **delivering only paid features** to clients in enterprise contexts.



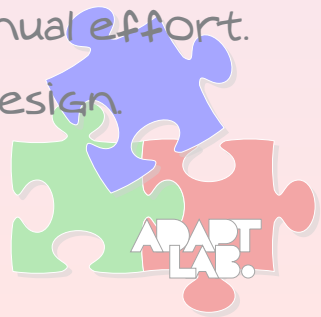
Conclusion

Interesting results:

- GUIDE **simplifies GUI development** by combining DSLs and SPLs.
- Significant **reduction in manual code writing**.
- **Multi-language support** enables flexibility but requires adapters for new languages.
- Feature selection enables **delivering only paid features** to clients in enterprise contexts.

Future Work:

- Extend GUIDE to support **additional GUI frameworks** and languages.
- Improve dependency management for **better storage efficiency**.
- Automate **callback file generation** to further reduce manual effort.
- Develop a **visual UI editor** for drag-and-drop interface design.



Thanks for your attention!