Avgust: Automating Usage-Based Test Generation from Videos of App Executions

Presented by : Anushka Jain, Abhishek Manayam, Rishitha Pokalkar

Basic Introduction

1. What is Software Testing?

2. Why we do Testing?

3. How many type of Testing are there?

What is Usage Based UI Testing?

Need for Automatic Testing Method

1. Manual testing requires significant human effort and time.

2. It's challenging to cover every possible scenario or use case manually due to limited knowledge.

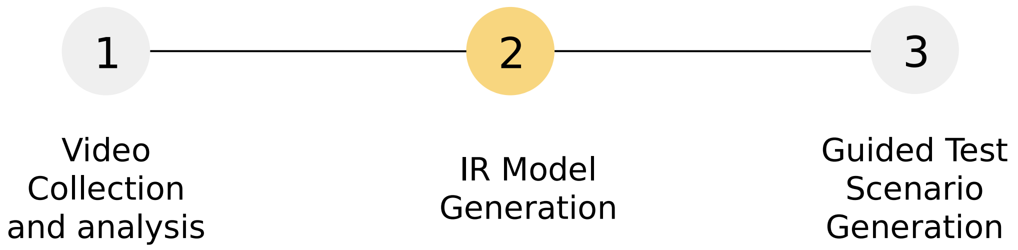
3. Manual testing is expensive.

What is this paper about?

● **Avgust** is an **semi** - automated tool that assists developers in generating usage-based tests for mobile applications.

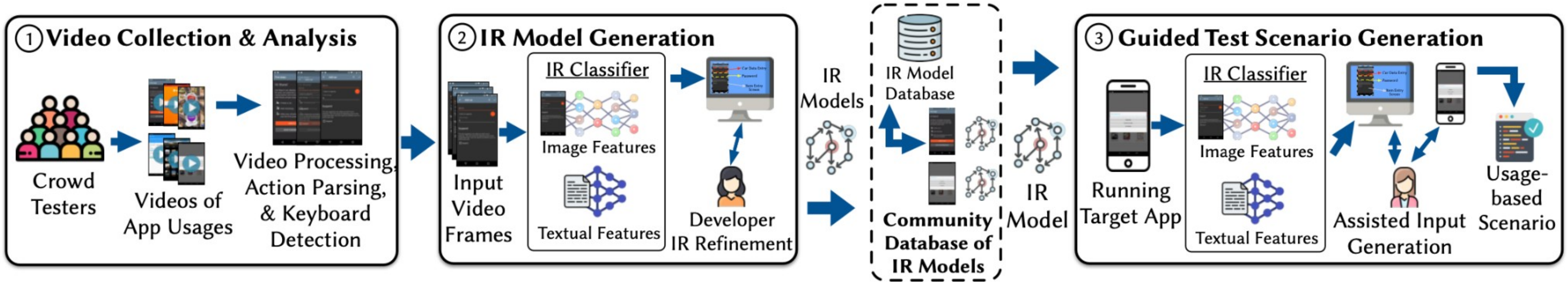
AVGUST

AVGUST generates usage-based tests for target mobile apps by analyzing user interaction videos of similar applications.


1
Video Collection and analysis
2
IR Model Generation
3
Guided Test Scenario Generation

IR - Intermediate Representation (next slides)

Workflow of AVGUST



Video Collection and analysis

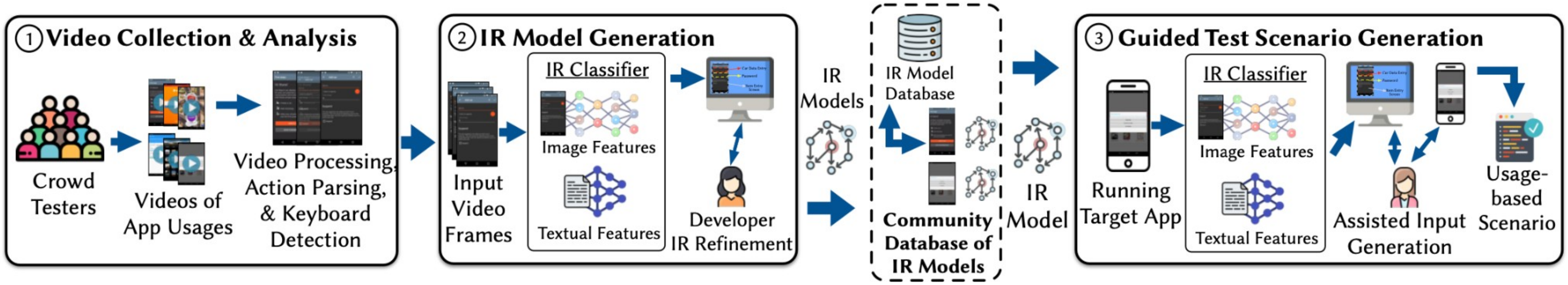
1. **Action Identification & Event Frame Extraction -** Avgust builds upon the analyses introduced by V2S to identify user actions and event frames.

   The outputs are
   (1) a sequence of event frames of a given video and their associated user actions (i.e., click, long tap, and swipe).
   (2) the coordinates of the touch indicator in each event frame

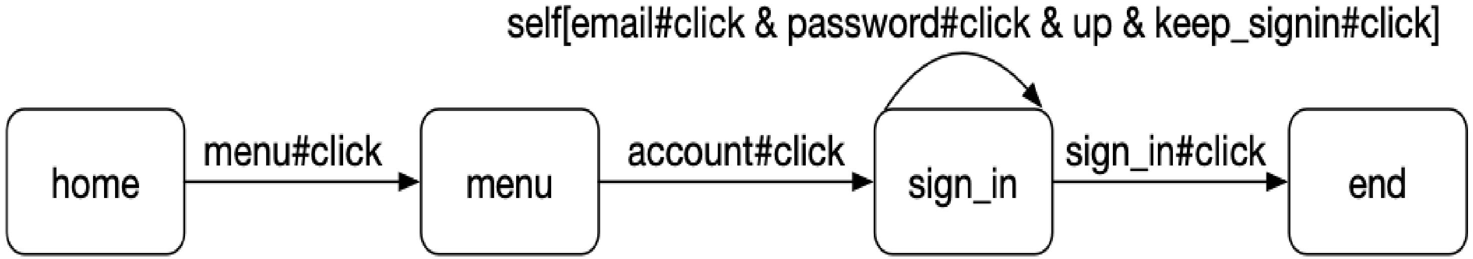
2. **Event Frame Filtering -** Avgust filters the extracted event frames by eliminating the frames that are associated with the typing action, since they may expose user's private information such as password.

   Clicking the password field
   Frames associated with typing each individual character in the password

Workflow of AVGUST



IR MODEL



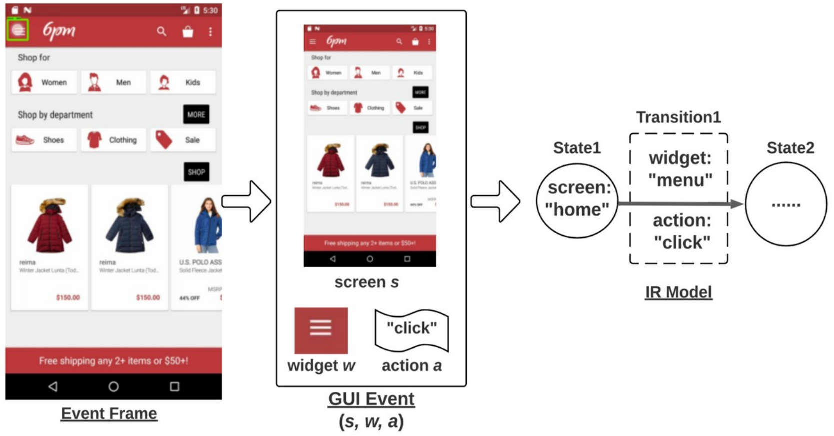
Avgust’s IR Model is defined as a finite state machine (FSM) that captures app usages.

IR Model Generation

Transforming Event Frames to GUI Events -

Used existing UIED tool to detect textual and visual GUI elements and produces their bounding boxes.

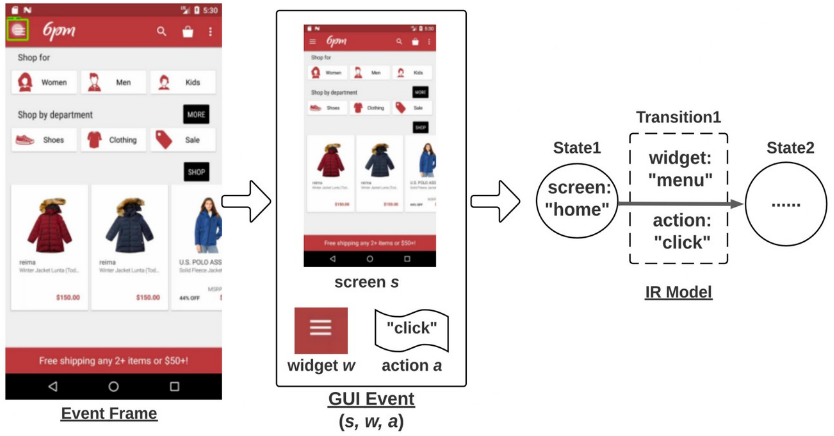
To identify the action a in the GUI event triple, Avgust leverages V2S's action identification procedure



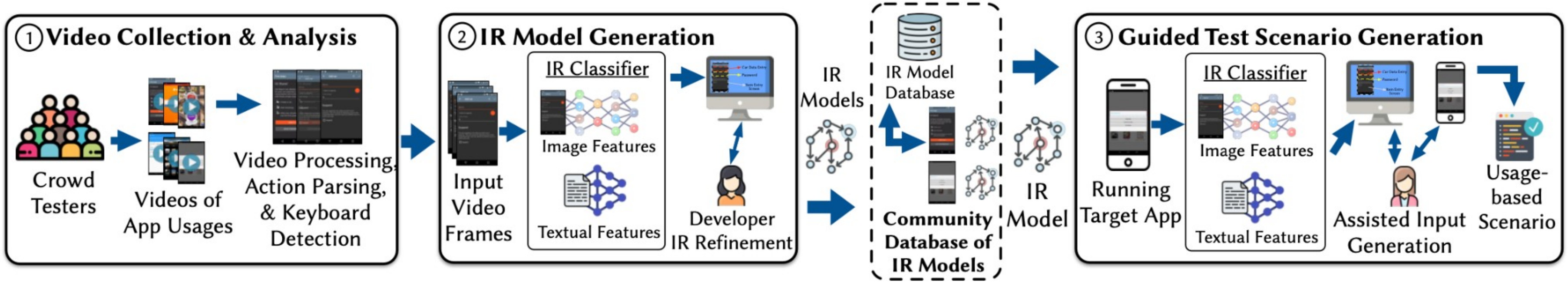
IR Model Generation

Transforming GUI Events to IR Models -

Avgust provides recommendations to assist developers in translating GUI events into their app-independent IR representations



Workflow of AVGUST



IR Classifier - App-specific screens and widgets are classified into their canonical counterparts (categories) that are shared across different apps.

Example canonical screens are “home screen”, “password assistant page”, and “shopping cart page”. Example canonical widgets are “account”, “help”, and “buy”.

IR Model Generation

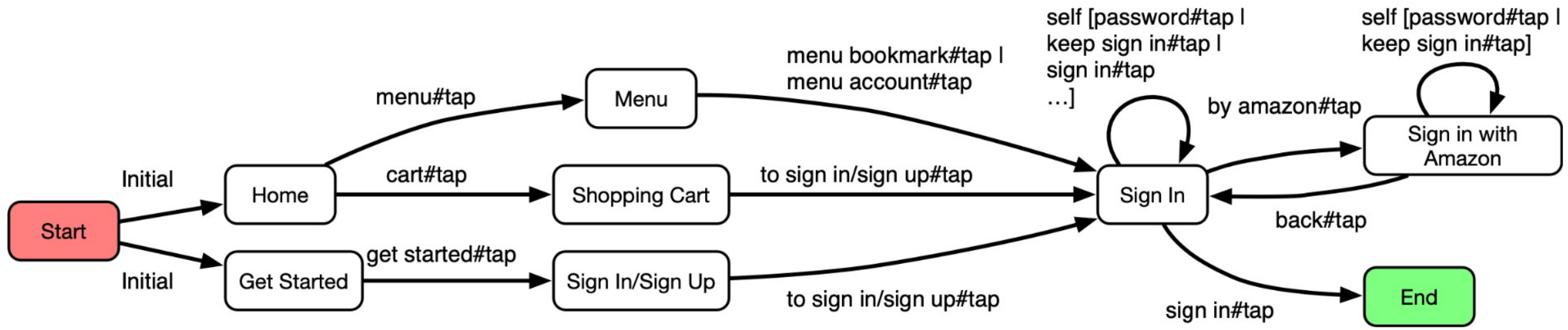


Figure 7: A simplified example of the merged IR Model learned from three sign-in videos of the 6pm and Etsy apps.

Guided Test Scenario Generation

Generates the test inputs based on the target app’s current state, until the end condition is met (i.e., the target feature is executed)

1. The process begins by launching the target app and running Avgust’s Screen Classifier to retrieve the most likely canonical categories of the target app’s starting screen.

2. Avgust then presents the developer with these top-k classification results, and the canonical category selected by the developer is used to match the target app’s current device screen to the canonical screen states in the IR Model

3. Avgust recommends the top-k app widgets for developers to interact with by using its Widget Classifier to map the canonical widgets in the IR Model to the widgets on the target app’s current screen.

Guided Test Scenario Generation

4. Avgust then checks whether the test should complete based on whether the widget chosen by the developer will lead to the end state in the IR Model. If not, the chosen widget is triggered, the target app’s next state becomes its current state, and this process repeats until the end state in the unified IR Model for a given usage is reached.

Research Questions

1. How effective is Avgust in generating successful usage-based tests?

2. How well does Avgust perform in classifying UI elements compared to the S2V model?

1

RQ1

1. For each of 18 different app usages, 3 target apps were chosen randomly for testing (with an exception for three usages where only 2 apps were selected due to data extraction issues from some commercial apps).

2. For each target app, Avgust used a merged IR Model created from the data of the other 17 apps to generate tests, showcasing Avgust's ability to generate tests for apps it hasn't directly learned from.

3. 35 out of the 51 generated tests (68.6%) successfully achieved the intended app usage, indicating that Avgust can accurately generate correct tests.

4. For the tests that didn't fully meet the usage criteria, their similarity to the closest human-generated test was assessed in terms of precision (how much of the generated test was relevant) and recall (how much of the human test was covered by the generated test).

1

RQ1

1. On average, 79% of the states and 47% of the transitions in the generated tests matched those in the closest human test.
2. Conversely, the generated tests captured 68% of the states and 37% of the transitions found in the human tests.
3. This suggests that while some generated tests didn't completely achieve the intended usage, they were still partially aligned with human testers' actions, potentially reducing the effort required for developers to write tests.

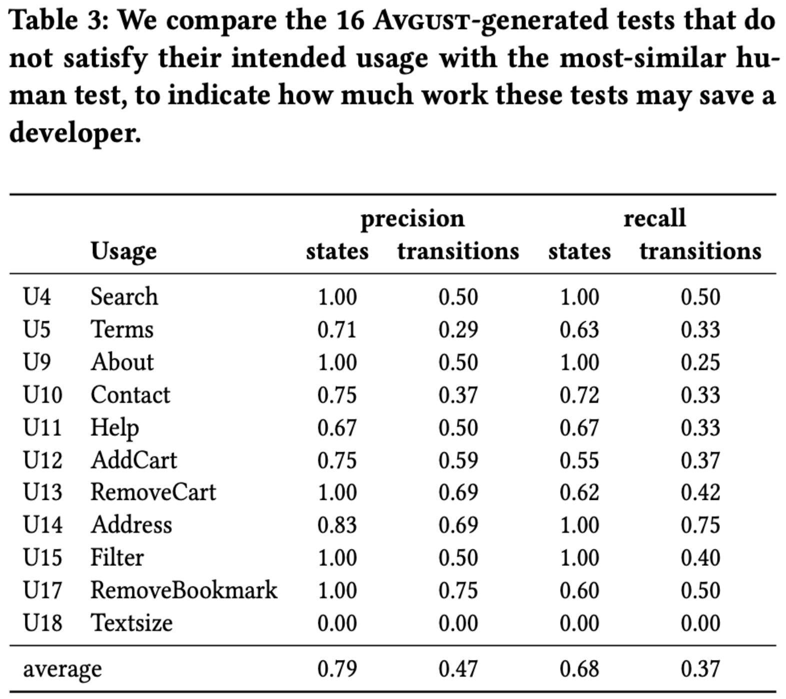


Table 3: We compare the 16 AVGUST-generated tests that do not satisfy their intended usage with the most-similar human test, to indicate how much work these tests may save a developer.

		precision		recall	
	Usage	**states**	**transitions**	**states**	**transitions**
U4	Search	1.00	0.50	1.00	0.50
U5	Terms	0.71	0.29	0.63	0.33
U9	About	1.00	0.50	1.00	0.25
U10	Contact	0.75	0.37	0.72	0.33
U11	Help	0.67	0.50	0.67	0.33
U12	AddCart	0.75	0.59	0.55	0.37
U13	RemoveCart	1.00	0.69	0.62	0.42
U14	Address	0.83	0.69	1.00	0.75
U15	Filter	1.00	0.50	1.00	0.40
U17	RemoveBookmark	1.00	0.75	0.60	0.50
U18	Textsize	0.00	0.00	0.00	0.00
average		0.79	0.47	0.68	0.37

1

RQ1

Author’s found that 69% of Avgust’s generated tests successfully accomplish the desired usage, saving the developer from having to manually write the test from scratch. For the remaining 31% of the tests, they found that those tests capture significant portions of the behavior in the most- similar test a human would write, again, potentially saving human effort.

2

RQ2

The evaluation is divided into two parts: Assessing Avgust as a standalone tool and then evaluating its performance in the context of test generation.

Using a labeled dataset, Avgust's classification accuracy for both screens and widgets is assessed through leave-one-out cross-validation. This involves training Avgust's model on data from all apps except one and testing on the excluded app.

Screen Classification: Three variants of Avgust's screen classifier are evaluated:
The standard Avgust classifier.
An autoencoder (AE) variant with K-Nearest Neighbors (KNN).
An AE variant with a Multilayer Perceptron (MLP).

Widget Classification: Similarly, Avgust's widget classifier is compared to S2V's, focusing on the accuracy of identifying widgets' roles based on visual features and other attributes.

2

RQ2

1. Screen Classification: Avgust's composit classifier (combining visual and textual features) outperforms all other variants and S2V, indicating the importance of using both types of features for accurate classification.

2. Widget Classification: Avgust also surpasses S2V in widget classification, highlighting the value of considering multiple widget features beyond just textual information.

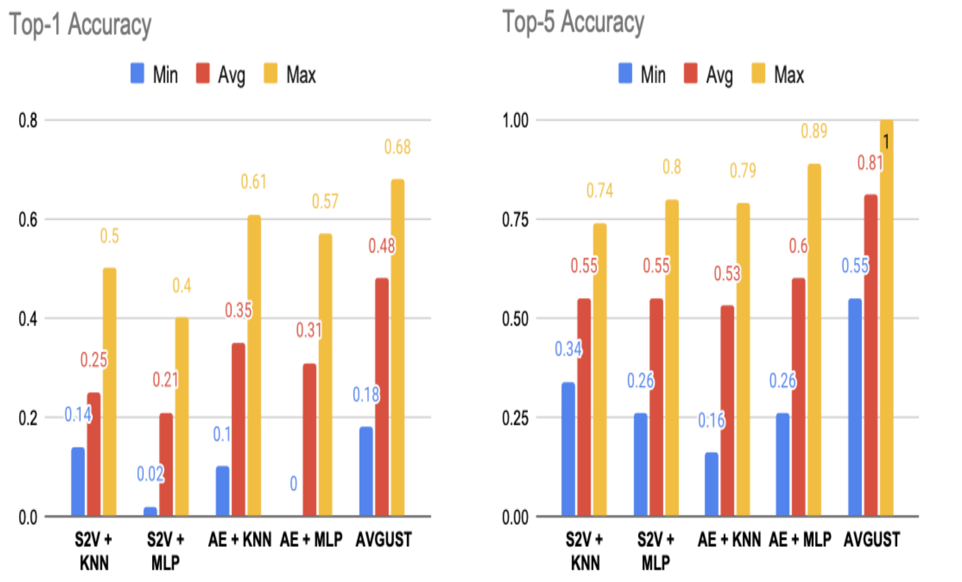


Figure 8: AVGUST's vision-only screen classification outperforms SV2 [47] and the other AVGUST classifier variants in both top-1 and top-5 accuracy.

2

RQ2

Avgust's classification accuracy is further evaluated within the test generation context. This involves tracking Avgust's top recommendations at each step of test creation and assessing their accuracy.

When generating tests, Avgust's recommendations for widgets accurately matched the intended actions suggested by its IR Model 77.4% of the time, demonstrating its effectiveness in guiding test creation.

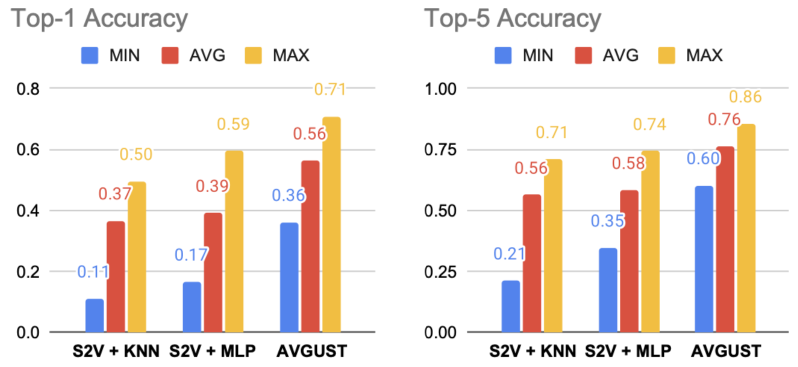


Figure 9: Avgust's vision-only widget classification consistently outperforms S2V [47].

2

RQ2

Avgust’s classifiers consistently outperform the state-of-the-art S2V tool. Author’s found that using textual and visual features together improves accuracy, but adding run- time features decreases accuracy, perhaps because these features are too different from the ones used to train Avgust’s vision-only models.

Novelty of Avgust

Existing automated UI testing often focuses on code coverage or crash detection.

Avgust is novel in its ability to generate usage-based tests, replicating specific user interactions.

Assumptions and Limitations

1. Avgust depends on the quality of the input videos. Blurry videos might lead to inaccurate identification of screens, widgets, and user actions.

2. Avgust can only recognize a limited set of user actions like taps, swipes, and long taps. It might not be able to handle complex gestures or interactions with specific UI elements.

3. Although it reduces manual effort, Avgust still requires developer involvement in labeling screens, widgets, and refining classifications during test generation.

Discussion Points

1. Many modern applications feature Complex UI elements that change based on user inputs or other factors. Extending Avgust to handle these complex elements would be essential for comprehensive testing.

2. Exploring ways to enhance the developer-in-the-loop approach.

3. How can avgust assist developers in creating templates for applications?

THANK YOU