Tel-Aviv University, summer 2013

SUMMAPHOTO

Summing up your day!

Software documentation

Handed by: Omri Koshorek & Yonatan Wilkof

Under the guidance of Prof. Lior Wolf & Noga Levy

Introduction:

SUMMAPHOTO ® *is a photo app that creates hot-outta-the-oven new collages consisting of your life's most precious moments. SummaPhoto divides your day into its primary events, and then makes a beautiful collage out of your photos! It's like an automatic diary that you can view, share and enjoy forever.*
Simply choose your preferred mode of operation, your preferred style, and you're set!

We thought about this idea, together with our supervisor Prof. Lior Wolf. We figure that because the Photo Gallery in our phones is usually not well organized and is a long strip of photos - it's difficult to find the hot pictures of the day that really sum up your day. And of course, it's always nice to remember where you take your photos…. This is where we thought SummaPhoto will come in handy.

Specifications:

The application was developed using JRE6, as required on Android.

The application was tested on the following:

Minimum API Level **Android 2.2–2.2.3 Froyo (API level 8)**

Maximum API Level: **Android 4.3 Jelly Bean (API level 18**

Permissions Required:

-Read/Write to Internal Storage

-Connection to the Internet

The app was tested on: Nexus 4, Nexus One

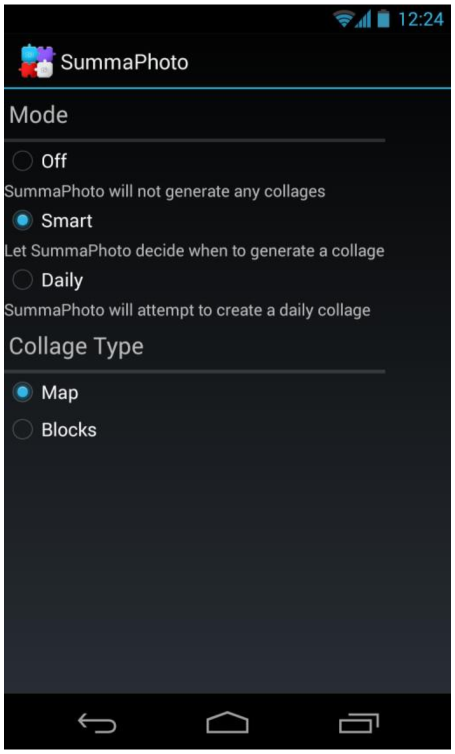
Installation:

1. Download the app file (summaphoto.apk) from the native gmail app on the Android phone. Simple installation instructions will follow. Installation is also possible by going to "Downloads" in the apps window.
2. If this message appears: "Installation Blocked: For security, your phone is set to block installation of apps obtained from unknown sources", then go to Settings>Security and check "unkown sources" to allow installing the app.

3. The app must have “Geo-tagging” ON in the phone. This option is enabled in a different manner in every device, so please check with your vendor how to do so. The app will prompt the user to enable this option if a certain amount of photos with no locations are identified.

User instructions

1. In the app’s main screen choose which mode you’d like to work in. *Off* means that the app will **not** collect your live photos and will therefore create no collage. If you’d like the app to collect photos and create collage, then choose Smart of Daily modes. After that, choose the collage type you desire – Map or Blocks.



12:24

SummaPhoto

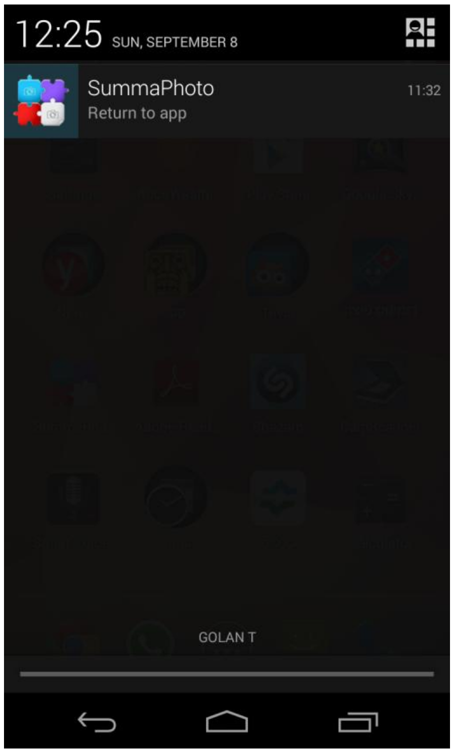
Mode

( ) Off
SummaPhoto will not generate any collages
[✓ Smart]
Let SummaPhoto decide when to generate a collage
( ) Daily
SummaPhoto will attempt to create a daily collage

Collage Type

[✓ Map]
( ) Blocks

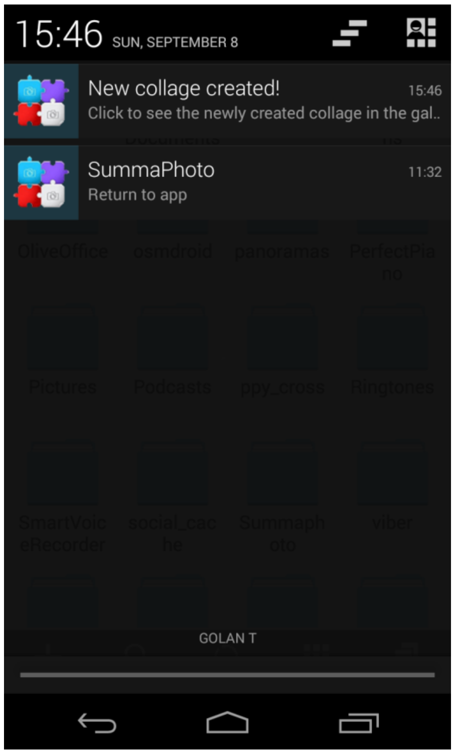
2. You may leave the app using the home button. If you’d selected either Smart or Daily mode, a notification reminding about the app will appear in the notification screen.


12:25 SUN, SEPTEMBER 8

SummaPhoto 11:32
Return to app

GOLAN T

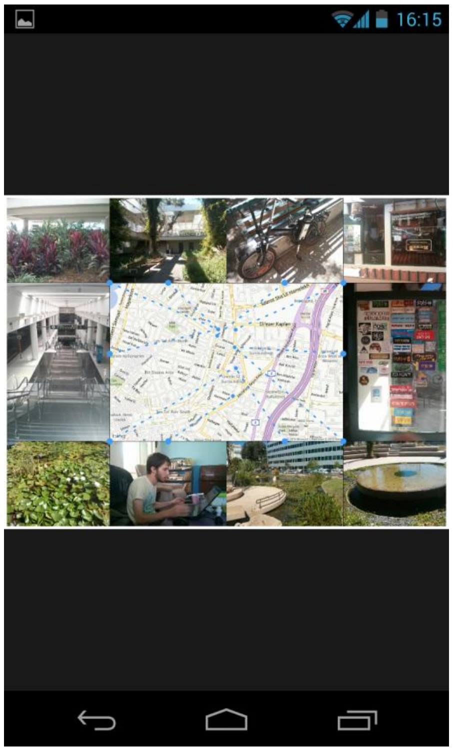
3. If the application created a new collage, a notification will appear and a ringtone will sound. By pressing the notification you will be forwarded to the gallery to view the collage where you can even share it with your friends!


15:46 SUN, SEPTEMBER 8

New collage created! 15:46
Click to see the newly created collage in the gal..

SummaPhoto 11:32
Return to app

GOLAN T



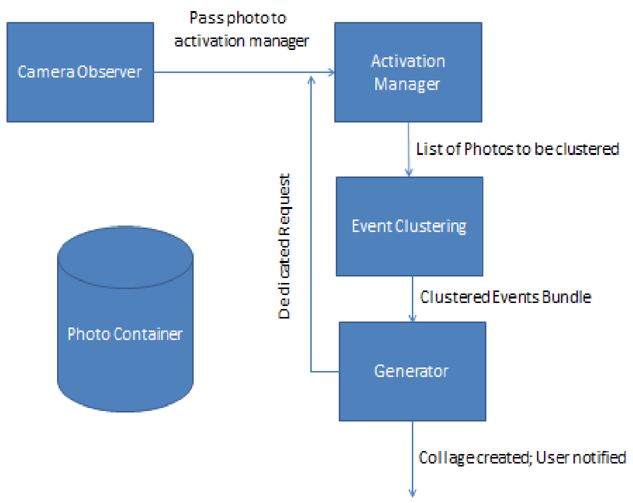
Flows

As long as Summaphoto is working (visible or non-visible) it listens online to all of the photos the user takes with his camera app.

Summaphoto allows two modes:

- *Smart Mode* – The application divides into events and decides if a collage should be created or not.
- *Scheduled Mode* – A collage creation will be made (if possible) at a specific time chosen by the user, daily.

Smart Mode:

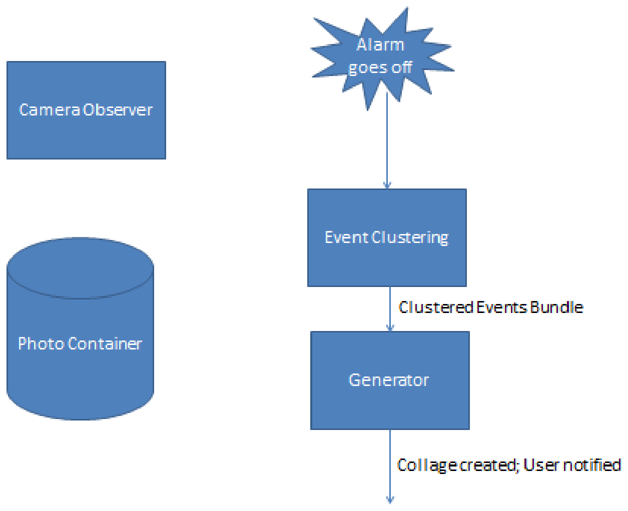


- Incoming photo from Camera Observer invokes Activation Manager (AM) if it's not already invoked, but is also accumulated in the photo container.
- In regular mode, AM "guesses" events by last recieved photo time. This is to avoid invoking the whole flow for every photo, and to save resources. Practically, the first photo is always an event, and any photos arrived in less than *NEW_CANDIDATE_THRESHOLD_DELTA* will be considered as the same event. Any photo that arrives in more than *NEW_CANDIDATE_THRESHOLD_DELTA* will be considered as a new event. After *CANDIDATE_EVENTS_FOR_COLLAGE,* the rest of the flow is invloked.
- Only if *MIN_TIME_BETWEEN_COLLAGES* has passes since last collage was created, then clustering algorithm will be invoked. This is to avoid running the clustering algorithm too many times and to accumulate more photos for the collage that spread over more time.
- Clustering algorithm returns all collected photos so far arranged in events. More about the algorithm, later.

- An attempt to generate a collage out of these events will be created if Clustering algorithm recognized more than *MIN_EVENTS* when analyzing the photos it was given.
- A template that fits the photos in the events will be chosen, populated and filled with photos to make the final collage.
- If such a template was not found, the "closest" one will be chosen, and the activation manager will be set to Dedicated Mode. This means that when a vertical/horizontal photo(s) required to fill the "close" template will be received in the AM, a collage will be created immediately again, by triggering the clustering algorithm and so forth.
- If all worked correctly (template filled with photos), a collage will be saved to the gallery and a notification will state that for the user.

Scheduled Mode:

- Incoming photos are accumulated.
- An alarm will go off at the time the user set, which triggers the clustering algorithm for all accumulated photos, and from there on it's the same as in the smart mode.
- One exception: There are no dedicated requests in this mode. If a collage cannot be created when the user requested, it will not be created at all, until the next day when another attempt will be made.



Algorithms in use

1. **Clustering algorithm – DBScan.**

One of the cores of the application is the clustering algorithm. The program has to divide the input photos into events, in order to pick the pictures which represent different activities of the user along the day. Later on the flow, from each event will be chosen some pictures. The division of the algorithm is based on 2 dimensions: The **time** and the **distance**. We assume that pictures which were taken in short time intervals (e.g. 5 seconds) represent the same event of the user's life. Also, pictures that were taken at the same area, are likely to represent the same event. By those two dimensions we managed to divide the different pictures into events. Of course, this division is statistical and can never fix in 100% to the real "events of life" of the user. In addition, some users can divide the same set of pictures into different sets of event, so our mission was tough.

The algorithm that we choose for the described purpose was the DBScan, which is a density-based clustering algorithm. The main advantage of this algorithm, which fits our app needs, is that the number of clusters is not a parameter of the algorithm. Also, the algorithm can conclude that some photos are noise, as happens in real life photos, and in such a way to filter them. The algorithm has 2 parameters: the *epsilon-distance* parameter, which describes how to conclude if 2 photos are "close" to each other, and *the minimum number of points in cluster*. In high level, the algorithm goes over all points in data base, and for each one of them conclude if it has enough epsilon-distanced-neighbors (in comparison to min-number-of-points). If so, it opens a new cluster with the point and all its neighbors, and for each neighbor does the same process as described, (without opening a new cluster).

Our implementation has the complexity of O(n^2). The main parameters of our implementation, as described below are: ***MaxSecondsIntervals*** and ***MaxMetersInterval*** for the "epsilon distanced" definition, and ***minNumberOfPointsInCluster***. Another scenario that our algorithm deals with is set of noisy pictures: Noisy pictures are those whose cluster's size is lower than the ***minNumberOfPointsInCluster***. For such a case we added the ***minNumberOfPointsInClusterForNoisyPictures***, which is used in case that a set of points is considered very noisy (the ratio of noisy pictures of the set is more than ***MaxRatioOfNoise***). In such a case, we re-cluster the whole set with the new parameter. We assume, that some users tend to take more dense pictures, and some not.

2. **Algorithm for Minimum Lines crossing in Map Collage:**

   Each map collage that our app creates consists of the map and 8-12 slots of the picked pictures from the different events of the user. From each slot we draw a line to the pushpin of the place on the map where the picture that populates the slot was taken. This algorithm is used to populate the different slots in such a way that reduces the number of line intersections for esthetic output.

   The algorithm is recursively and is includes the following steps:

   - Go over the slots and pushpins and find a slot-PushPin tuple, that a line between them will divide the plane in such a way that the number of push pins above the line (or under the line) equals the number the slots above the line (or equal the line). Such a tuple must exist!
   - Divide the other pushpins and slots according to the line (slots above line, slots under line, pushpin above line…), and continue recursively.

   Since our collage consists of vertical and horizontal photos, we run this algorithm separately for vertical slots and pushpins, and horizontal slots and pushpins. In order to reduce the number of line intersections, as part of the vertical run we try to populate extra photos (e.g. if there are 4 vertical slots we will try to populate 8 pictures), and during the algorithm we choose the photos that minimize the number of line intersections.

3. **Algorithm for choosing "closest" template:**

   When a user takes certain photos and Smart mode decides a collage should be made, a correct template of collage should be chosen. By correct we mean that the bundle of photos have enough horizontal/vertical photos as the template requires. For both Map and Block collage type, there is more than one template and each one varies by number of horizontal / vertical photos that comprise it. If there is no correct template, we must choose the closest one, so when the needed photos are received by the app, the collage will be sent to the user instantly. The closest one is the one in which there is roughly the same amount of horizontal and vertical photos missing, and has the least total number of photos missing.

Template score is calculated for each template with respect to the given bundle of events and the photos in them.

- For each template:
    - Calculate difference between the number of needed horizontal photos to the number of supplied horizontal photos in the bundle. Do the same for vertical photos.
    - Subtract the two diffs from on another – *diff score*. The lower it is, the more "even" are the needed numbers of horizontal and vertical photos. (e.g. 1 horizontal needed and 1 vertical needed, diff = 0)
    - Calculate the total number of photos for template- this is the *remaining score*. The lower it is, the lower is the number of photos missing to fill the template. (e.g. 1 horizontal needed and 1 vertical needed, remaining = 2)
    - Calculate a *total closeness* based on *diff score* and *remaining score,* where *remaining score* gets more weight.
    - Check if *total closeness score* == 0, is so then the template it is a perfect match for the bundle of photos. (0 is as close as it gets)
- In the end, If there is more than one perfect match, choose one template randomly. If there are none, choose the "closest template" - the template with minimum *total closeness score.*

Code Structure:

1. ***com.summaphoto -***

   This package holds the only Activity and Service used by the app.

   - SettingsActivity – The main and only activity (GUI) in the application.
   - PhotoListenerService – The main service in the application. It runs as long as the app is open (not necessarily visible) and consists of a CameraObserver object that listens to the DCIM\Camera folder for new (or deleted) photos.
   - ScheduledModeService – A repated Alarm that goes off at a time which the user chooses, daily.
   - SmartModeFlow – A set of function that support the Smart mode of the app.

2. ***Bing**:*

   This package includes the objects which are used to get the map data from Bing services for the map collage-

   - StaticMap – object that represents the map that is retrieved from Bing, include all its meta data.
   - BingServices- includes the methods that are used to get the map from Bing over the net.
   - PushPin- A pin on the map that we receive from Bing.

3. ***Common**:*

   This package consists of the objects that are used by more than one other package in the app.

   - ActualEvent – A sole event that can be recognized by DBScan algorithm
   - ActualEventBundle – The output of a the clustering algorithm. It's a bunch of ActualEvents ties together for the collage to choose photos from.
   - Photo – representation of a photo file, including the EXIF metadata that comes with it, such as: Width, height, Date Taken, orientation.
   - PhotoContainer – A singleton that contains a list of fresh photos intercepted by the listener service, and a list of photos that are ready to be reviewed by the clustering algorithm.

4. **ActivationManager**:

   - ActivationManager- the implementation of the algorithm for AM.
   - DedicatedRequest – the object that changes the AM from Regular mode to Dedicated mode. Sent to AM by the Generator.

5. ***Partitioning**:*

   The package consists of different object that are used to divide the photos events:

   - Cluster- describe a cluster of photos according to the algorithm.
   - DBScan- includes the method of the implementation of the algorithm.
   - PhotoObjectForClustering- an extended photo for the clustering algorithm.
   -

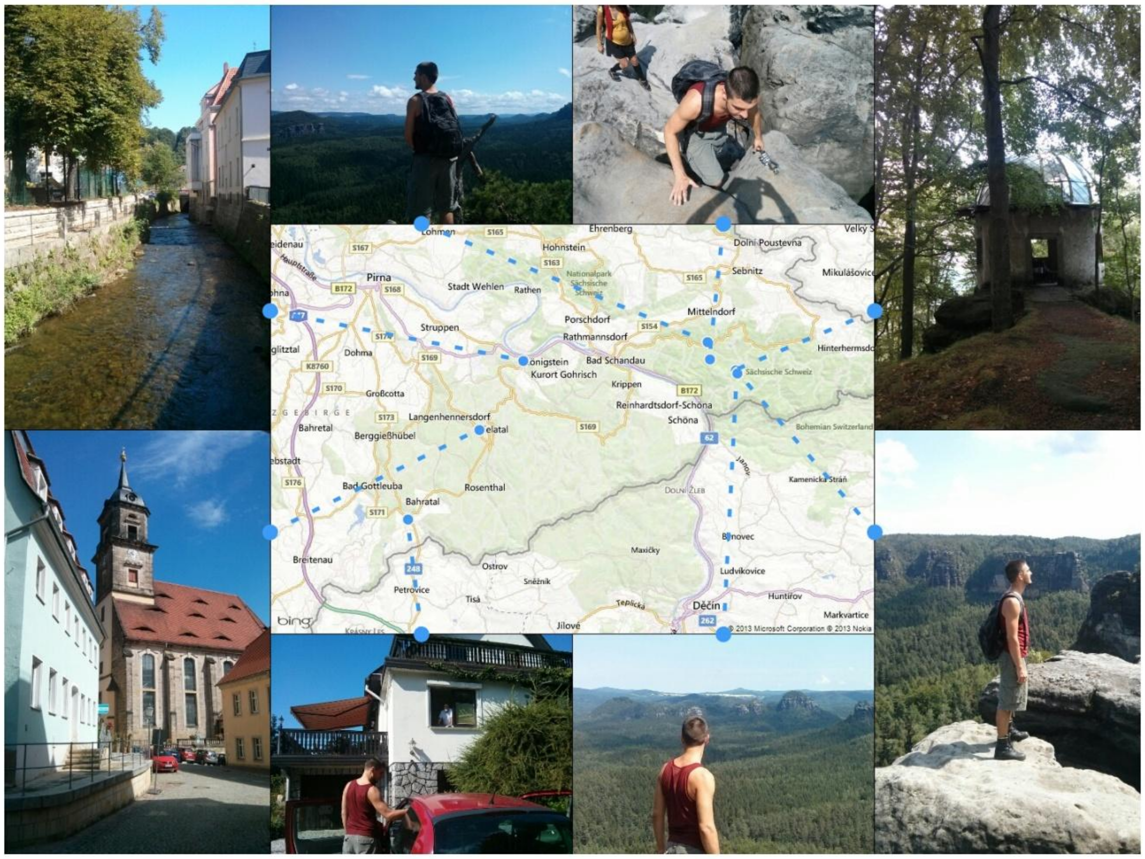
6. **Generator**:

   - AbstarctBuilder- abstract class which consists of mutual functions for the derived classes MapCollageBuilder & BlockCollageBuilder.

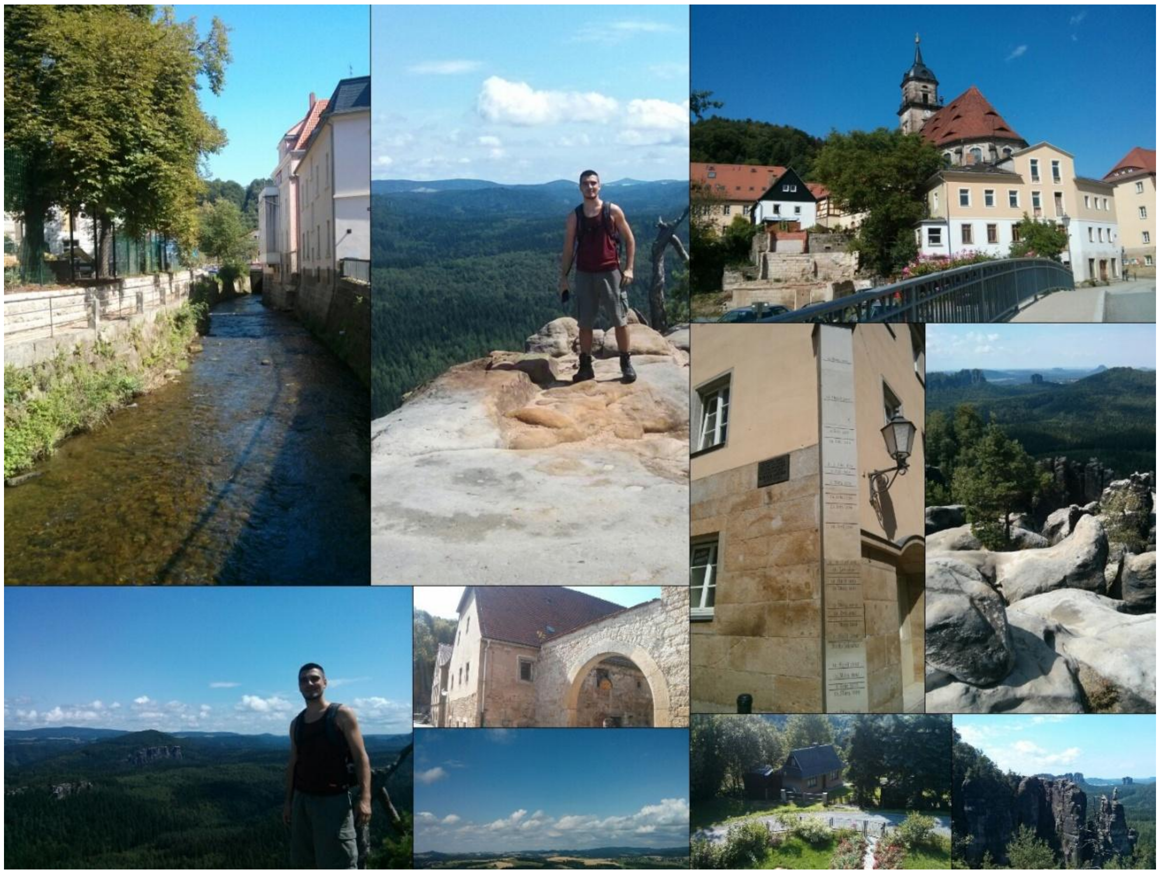
- MapCollagebuilder – functionatily for choosing a map template, and filling it correctly (minimum lines intersecting)
- BlockCollageBuilder- functionality for choosing a block template and filling it
- AbstractTemplate – abstract class which consists of mutual functions for derived MapTemplate and BlockTemplate
- MapTemplate – defines an empty map template to fill (we have 2 hard-coded templates)
- BlockTemplate- defines an empty block template to fill (we have 3 hard-coded templates)

Output examples:

Map Collage



Blocks Collage



finally{

The project before you represents hours of hard work, many efforts (and wandering aimlessly through the city while taking photographs), but also a lot of challenging moments and funky collages that led us to the final product.

We truly hope that you find the application interesting at least as we do.

Sincerely,

SUMMAPHOTO team

Omri Koshorek & Yonatan Wilkof

}