Test Plan and Results

Overall Test Plan

We will test several different aspects of the Large Pizza Order Generator. The fundamental components to be tested include the mobile app interface, the API, and the underlying algorithm for order generation. In addition to testing these individual components, we will also perform several tests to ensure that the components work well together, and are able to fail gracefully when connections are broken. Some tools we plan on using for testing include: Postman (for API testing), unit testing frameworks in the languages of our implementation, network request overwrites (for testing the frontend without needing backend support), and manual usage of the app on real and simulated devices.

Test Case Descriptions

CO 1.1 **Create Order Test 1**
CO 1.2 This test will validate the happy path for creating an order from the mobile app while connected to the backend.
CO 1.3 Navigate to the front page of the mobile app and press the “create order” button. No error message should be displayed, and the user should be navigated to an order admin page with the generated order code displayed and a list of members, with the ability to generate their order.
CO 1.4 Inputs: a single press of “Create Order”
CO 1.5 Outputs: an order code and navigation to the expected order admin page
CO 1.6 Normal
CO 1.7 Blackbox
CO 1.8 Functional
CO 1.9 Integration

CO 2.1 **Create Order Test 2**
CO 2.2 This test will validate the unhappy path for creating an order from the mobile app.
CO 2.3 Ensure that you are in some sort of error state with respect to the connection between the frontend and backend (easiest would be disconnected from the internet or the backend is not running at all). Navigate to the front page of the mobile app and press the “create order” button. An error should be displayed that explains what went wrong, an user should remain on home page.
CO 2.4 Inputs: a single press of “Create Order”
CO 2.5 Outputs: descriptive error message and lack of navigation to admin page
CO 2.6 Abnormal
CO 2.7 Blackbox

CO 2.8 Functional
CO 2.9 Integration

MAJO 1.1 **Mobile App Join Order Test 1**
MAJO 1.2 This test will validate the ability for a user to join a pre-existing order through the mobile app.
MAJO 1.3 Navigate to the front page of the mobile app and enter a known valid order code (i.e., one that has already been created, perhaps through **CO1** test, or one that has a specific mock response locally to always return a valid code), and tap “Join Order”. No error message should be displayed, and the user should be navigated to the page to select their preferences.
MAJO 1.4 Inputs: valid order code
MAJO 1.5 Outputs: no error messages, and navigation to the expected topping selection page
MAJO 1.6 Normal
MAJO 1.7 Blackbox
MAJO 1.8 Functional
MAJO 1.9 Unit

MAJO 2.1 **Mobile App Join Order Test 2**
MAJO 2.2 This test will validate that invalid orders cannot be joined through the mobile app.
MAJO 2.3 Navigate to the front page of the mobile app and enter a known invalid order code (one that is impossible or does not exist, potentially overwritten locally to always return an invalid response), and tap “Join Order”. An error message should be displayed, which prompts the user that their code was invalid and to ensure they are entering the right code and try again.
MAJO 2.4 Inputs: invalid order code
MAJO 2.5 Outputs: error message
MAJO 2.6 Abnormal
MAJO 2.7 Blackbox
MAJO 2.8 Functional
MAJO 2.9 Unit

TOP 1.1 **Topping Selector Test 1**
TOP 1.2 This test will validate that the topping selector works as expected given the constraints and/or preferences of the order.
TOP 1.3 Navigate to the topping selector (this will require a valid order code), and toggle topping (and/or crust) selections. Verify that the possible selections make sense with the given constraints (i.e., if the algorithm requires constraints on the amount of

selections any given user can have, they cannot make more selections than that, must choose at least 1 crust preference, etc.).
TOP 1.4 Inputs: valid order code, topping selections
TOP 1.5 Outputs: only allowed to make preference selections that make sense and fall within constraints of algorithm, otherwise there will be a prompt to make corrections.
TOP 1.6 Normal and Abnormal
TOP 1.7 Whitebox
TOP 1.8 Functional
TOP 1.9 Integration (algorithm's constraints will need to be passed from backend + admin selections)

ALGO 1.1 **Algorithm Test 1**
ALGO 1.2 This test will be used to ensure that the order generation algorithm can complete its calculations and outputs in a reasonable amount of time with expected volume of inputs.
ALGO 1.3 Using a unit testing framework in the language in which the algorithm is written, we will pass the preferences of an amount of users close to what we would consider the ceiling in a practical use-case (maybe 150-200 randomly generated preference lists). We will use the framework to measure the time it took to calculate, and compare against our expectations/requirements.
ALGO 1.4 Inputs: randomly generated preferences for a large number of users
ALGO 1.5 Outputs: amount of time (in seconds) that it took for the algorithm to complete the order generation
ALGO 1.6 Normal
ALGO 1.7 Whitebox
ALGO 1.8 Performance
ALGO 1.9 Unit

ALGO 2.1 **Algorithm Test 2**
ALGO 2.2 This test will be used to ensure that the orders generated by the order generation algorithm make sense and satisfy the conditions for a "good" order.
ALGO 2.3 Using a unit testing framework in the language in which the algorithm is written, we will pass the preferences of a reasonably large, but not huge, number of users (maybe 20) into our order generation algorithm. The output order should satisfy the following conditions (and this can be asserted in the test framework): there are no less than N/2 pizzas (where N is the number of users), and for each user, there exists at least 1 pizza with no toppings they dislike, and at least 1 topping that they do like.
ALGO 2.4 Inputs: topping preferences of a reasonable number of users
ALGO 2.5 Outputs: generated order, boolean describing whether it meets the requirements

ALGO 2.6 Normal
ALGO 2.7 Whitebox
ALGO 2.8 Functional
ALGO 2.9 Unit

GEN 1.1 **Generate Order Test 1**
GEN 1.2 This test essentially functions as a full-system sanity check, ensuring that an admin can generate an order with the preferences of their group.
GEN 1.3 Using several devices and/or simulators running the mobile app, create an order using one device as the admin, and join that order on the remaining clients, filling out and submitting their preferences (users should appear on the admin's member list as they join). Once finished, press "Generate Order" on the admin page, and the result should be an order that satisfies everyone's preferences.
GEN 1.4 Inputs: a created order (order code to join), preferences of several users, and pressing the "generate order" button
GEN 1.5 Outputs: a list of users who have joined and a satisfactory order
GEN 1.6 Normal
GEN 1.7 Blackbox
GEN 1.8 Functional
GEN 1.9 Integration

API 1.1 **Backend Service Availability**
API 1.2 This test is used to make sure all defined endpoints are available and working correctly
API 1.3 Can be tested by hand via a web request application like Postman
API 1.4 Inputs: A request to each URL with appropriate parameters
API 1.5 Outputs: Correct request responses
API 1.6 Normal
API 1.7 Whitebox
API 1.8 Functional
API 1.9 Unit

API 2.1 **Backend Service Performance**
API 2.2 This test is used to ensure that our backend can handle a large volume of requests
API 2.3 Depending on where we deploy our service, this service can be tested by using that cloud platform's tools see
API 2.4 Inputs: Inputs to stress testing tools specifying quantity of load to test
API 2.5 Outputs: Performance metrics based on tests
API 2.6 Abnormal

API 2.7 Whitebox
API 2.8 Performance
API 2.9 Integration

Test Case Matrix

Test Id	**Normal/ Abnormal**	**Blackbox/ Whitebox**	**Functional/ Performance**	**Unit/Integration**
CO1	Normal	Blackbox	Functional	Integration
CO2	Abnormal	Blackbox	Functional	Integration
MAJO1	Normal	Blackbox	Functional	Unit
MAJO2	Abnormal	Blackbox	Functional	Unit
TOP1	Both	Whitebox	Functional	Unit
ALGO1	Normal	Whitebox	Performance	Unit
ALGO2	Normal	Whitebox	Functional	Unit
GEN1	Normal	Blackbox	Functional	Integration
API1	Normal	Whitebox	Functional	Unit
API2	Abnormal	Whitebox	Performance	Integration

Test Case Matrix (With Results)

Test Id	**Normal/ Abnormal**	**Blackbox / Whitebox**	**Functional/ Performance**	**Unit/ Integration**	**Result and Notes**
CO1	Normal	Blackbox	Functional	Integration	**PASS-** Calls to Create Order endpointsucceed from app, and resulting order can be looked up with /order endpoint
CO2	Abnormal	Blackbox	Functional	Integration	**PASS-** Without a connection to the active backend, an error message is shown when an attempt is made to create an order
MAJO1	Normal	Blackbox	Functional	Unit	**PASS-** Order created using postman and a user can join via mobile app using the associated code
MAJO2	Abnormal	Blackbox	Functional	Unit	**PASS-** Using a random invalid code to join an order results in an error message
TOP1	Both	Whitebox	Functional	Unit	**PASS-** Once joined into an order, users are shown only toppings from the database, and are able to select them as expected
ALGO1	Normal	Whitebox	Performance	Unit	**FAIL-** HTTP requests to generate orders time out

					when a group is of 40 users or larger
ALGO2	Normal	Whitebox	Functional	Unit	**PASS-** All attempted test inputs on the algorithm generate N/2 or more pizzas, with every user being satisfied
GEN1	Normal	Blackbox	Functional	Integration	**PASS-** Using about 10 devices, we were able to create, join, and generate an order that makes sense
API1	Normal	Whitebox	Functional	Unit	**PASS-** Running a Postman collection that hits all endpoints was successful
API2	Abnormal	Whitebox	Performance	Integration	**PASS-** Running our Postman collections several times in parallel was handled gracefully