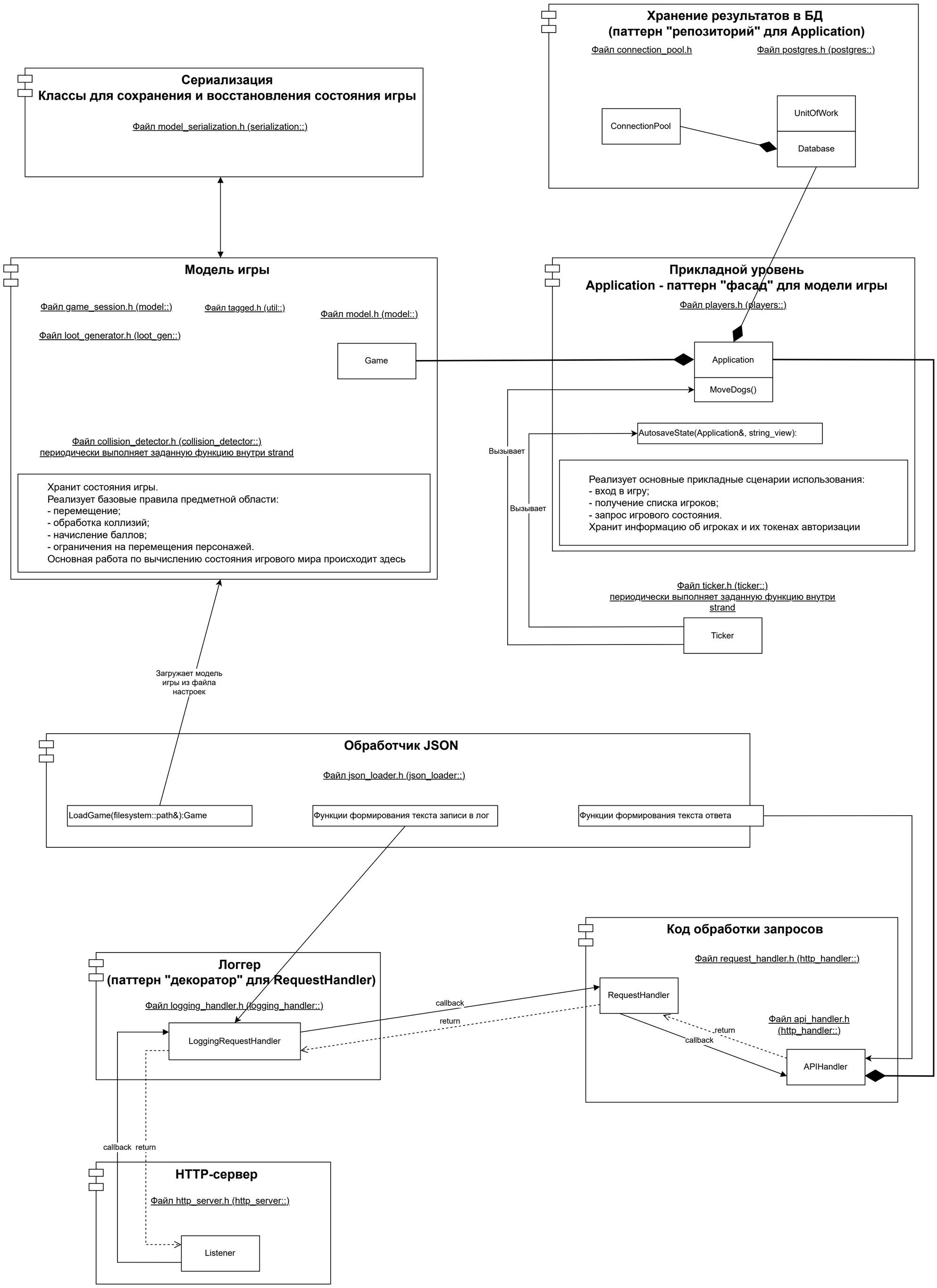

Сериализация
Классы для сохранения и восстановления состояния игры
Файл model_serialization.h (serialization::)
Хранение результатов в БД
(паттерн "репозиторий" для Application)
Файл connection_pool.h
Файл postgres.h (postgres::)
ConnectionPool
UnitOfWork
Database
Модель игры
Файл game_session.h (model::)
Файл tagged.h (util::)
Файл model.h (model::)
Файл loot_generator.h (loot_gen::)
Game
Файл collision_detector.h (collision_detector::)
периодически выполняет заданную функцию внутри strand
Хранит состояния игры.
Реализует базовые правила предметной области:
- перемещение;
- обработка коллизий;
- начисление баллов;
- ограничения на перемещения персонажей.
Основная работа по вычислению состояния игрового мира происходит здесь
Прикладной уровень
Application - паттерн "фасад" для модели игры
Файл players.h (players::)
Application
MoveDogs()
AutosaveState(Application&, string_view):
Вызывает
Вызывает
Реализует основные прикладные сценарии использования:
- вход в игру;
- получение списка игроков;
- запрос игрового состояния.
Хранит информацию об игроках и их токенах авторизации
Файл ticker.h (ticker::)
периодически выполняет заданную функцию внутри strand
Ticker
Загружает модель игры из файла настроек
Обработчик JSON
Файл json_loader.h (json_loader::)
LoadGame(filesystem::path&):Game
Функции формирования текста записи в лог
Функции формирования текста ответа
Код обработки запросов
Файл request_handler.h (http_handler::)
RequestHandler
callback
return
return
callback
Файл api_handler.h
(http_handler::)
APIHandler
Логгер
(паттерн "декоратор" для RequestHandler)
Файл logging_handler.h (logging_handler::)
LoggingRequestHandler
callback return
HTTP-сервер
Файл http_server.h (http_server::)
Listener

Модель игры

Сокращения:

[] - std::vector<>

{} - std::map<>

shptr<> - std::shared_ptr<>

- важное

- последнее добавленное

Файл model.h (model::)

Файл game_session.h (model::)

Game
- -maps_:[Map]
- -map_id_to_index_:unordered{}
- -loot_generator_: LootGenerator
- -sessions_:[[shptr<GameSession>]]
- -dog_retirement_time_: double
- +AddMap(Map)
- +GetMaps(): [Map]&
- +FindMap(Map::Id): Map*
- +PlacePlayerOnMap(Map::Id&):GameSession*
- +SetLootGenerator(chrono::milliseconds, double):
- +GetLootGenerator(): LootGenerator
- +RestoreSessions([Sessions]):
- +SetDogRetirementTime(double):
- +GetDogRetirementTime(): double

GameSession
- -map_:Map*
- -dog_ids_:[size_t]
- -map_id_to_it_:unordered{}
- -lost_objects_: [shptr<LostObject>]
- +GameSession(Map*)
- +AddDog(shptr<Dog>):
- +GetDogIds():const [size_t]&
- +CountDogsInSession():const size_t
- +GetMap():Map*
- +GetLostObjects(): [shptr<LostObjects>]
- +AddLostObjectsOnSession(LootGenerator&, TimeInterval):
- +CountLostObjects(): [size_t]
- +RemoveObjectsFromLost([bool])
- +GetLastObjectId(): size_t
- +RestoreLostObjects(LootObjects, size_t):
- +DeleteDog(dog_id: size_t):

LostObject
- -type_: size_t
- -position_: Position
- -id_: size_t
- -width_: double
- +LostObject(size_t, Position, size_t, double)
- +GetType(): size_t
- +GetPosition(): Position&
- +GetId(): size_t
- +GetWidth(): double

Модель игры Генератор трофеев

Файл loot_generator.h (loot_gen::)

LootGenerator
- -base_interval_:chrono::milliseconds
- -probability_: double
- -time_without_loot_:chrono::milliseconds
- -random_generator_: function<double()>
- -DefaultGenerator(): double
- +LootGenerator(base_interval, probability, RandomGenerator)
- +Generate(time_delta, loot_count, looter_count): unsigned
- +SetLootPeriod(chrono::milliseconds):
- +GetLootPeriod(): chrono::milliseconds
- +SetLootProbability(double):
- +GetLootProbability(): double

Road
- -start_:Point
- -end_:Point
- +Road(HorizontalTag, Point, Coord):
- +Road(VerticalTag, Point, Coord):
- +IsHorizontal():bool
- +IsVertical():bool
- +GetStart():Point
- +GetEnd():Point

Building
- -bounds_:Rectangle
- +Building(Rectangle)
- +GetBounds():const Rectangle&

Office
- -id_:Id
- -position_:Point
- -offset_: Offset
- +GetId(): const Id&
- +GetPosition(): Point
- +GetOffset(): Offset
- +Office(Id, Point, Offset)

LootType
- -name_: string
- -file_: string
- -type_: string
- -rotation_: optional<int>
- -color_: optional<string>
- -scale_: double
- -scores_cost_: size_t
- +LootType(name, file, type, rot, color, scale, score)
- +GetName(): string_view
- +GetFile(): string_view
- +GetType(): string_view
- +GetRotation(): optional<int>
- +GetColor(): optional<string_view>
- +GetScale(): double
- +GetScores(): size_t

Map
- -id_:Id
- -name_: string
- -speed: double
- -speed: size_t
- -warehouse_id_to_index_: unordered{}
- -roads_: [Road]
- -buildings_: [Building]
- -offices_: [Office]
- -normal_roads: [Road]
- -loot_types_: [LootType]
- -GetRoadByPosition(Position&): [size_t]
- +Map(id:Id, name:string, speed:double, bag:size_t)
- +GetId(): const Id&
- +GetName(): const string&
- +GetBuildings(): const [Building]&
- +GetRoads(): const [Road]&
- +GetOffices(): const [Office]&
- +GetSessions(): [Sessions]&
- +GetSpeed(): double
- +GetLootTypesCount():size_t
- +GetLootTypes():[LootType]&
- +GetBagCapacity(): size_t
- +GetLootByPosition(size_t): LootType&
- +AddBuilding(const Building&)
- +AddRoad(const Road&)
- +AddOffice(const Office&)
- +GetRandomPositionOnRoads():Position
- +GetTestPositionOnRoads():Position
- +MoveDog(Dog*, time:double):DogState
- +AddLootType(const LootType&)

Dog
- -id_:Id
- -name_:string
- -state_:DogState
- -objects_: [PickedObject]
- -scores_: size_t
- -inactive_time_: double
- -total_time_: double
- +Dog(size_t, string, Position)
- +Dog(size_t, string, DogState, [PickedObject], size_t)
- +GetDogId():size_t
- +GetDogName():const string&
- +GetDogState():const DogState&
- +SetPosition(Position&):
- +SetVelocity(Velocity&):
- +SetDirection(Direction&):
- +SetState(DogState&):
- +AddPickedObject(PickedObject, size_t): bool
- +GetPickedObjects(): [PickedObjects]&
- +IsBagEmpty(): bool
- +FlushPickedObjects(): [PickedObjects]
- +AddScores(size_t):
- +GetScores(): size_t
- +IncInactive(double):
- +ResetInactive():
- +GetInactive(): double
- +IncTotalTime(double):
- +GetTotalTime(): double

PickedObject
- -id_: size_t
- -type_: size_t
- +PickedObject(size_t, size_t)
- +GetId(): size_t
- +GetType(): size_t

Воздействует на сущности игровой модели

Файл ticker.h (ticker::) периодически выполняет заданную функцию внутри strand

Использован в main()

библиотека boost

Ticker
- -strand_:Strand
- -period_:chrono::milliseconds
- -timer_:steady_timer
- -handler_:Handler
- -last_tick_:steady_clock::time_point
- +Ticker(Strand, milliseconds, Handler)
- +Start():
- -ScheduleTick():
- -OnTick(error_code)

struct Point: ...

struct Rectangle: ...

struct Size: ...

struct Offset: ...

detail::
- RoundPosition(pos:double):Coord
- FoundRoad([size_t], [size_t]):bool
- FoundBigger(double&, size_t&, double, size_t):bool
- DoubleIsZero(double):bool

Файл tagged.h (util::)

template <typename Value, typename Tag> class Tagged
- -value_:Value
- + ...

template <typename TaggedValue> TaggedHasher
- +operator():size_t

struct Velocity
- ...
- +IsZero(): bool

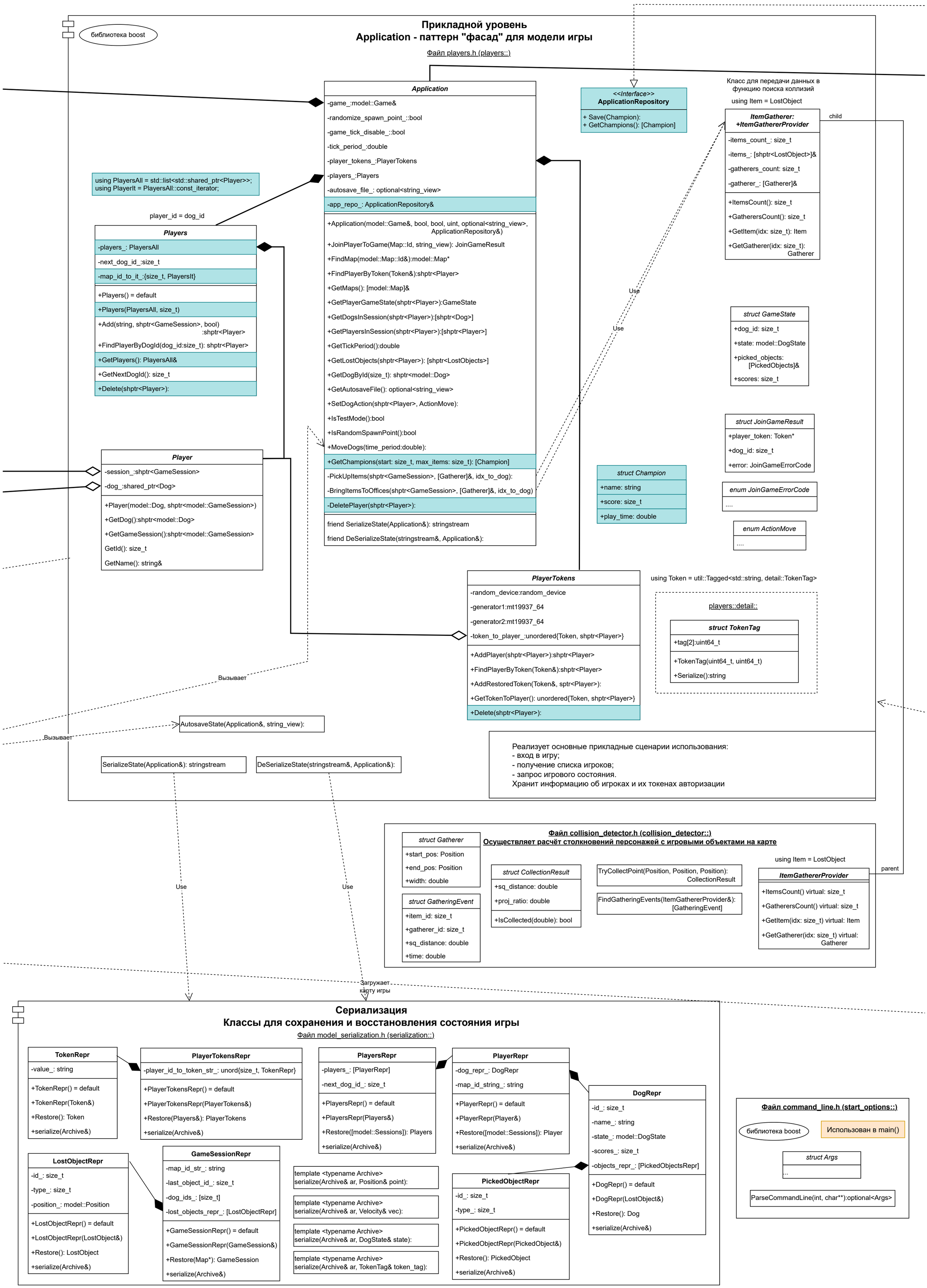
struct DogState
- +position:Position
- +velocity:Velocity
- +direction:Direction

struct Position: ...

enum Directon: ...

Хранит состояния игры.
Реализует базовые правила предметной области:
- перемещение;
- обработка коллизий;
- начисление баллов;
- ограничения на перемещения персонажей.

Основная работа по вычислению состояния игрового мира происходит здесь


Прикладной уровень
Application - паттерн "фасад" для модели игры
Файл players.h (players::)
библиотека boost
Application
-game_:model::Game&
-randomize_spawn_point_::bool
-game_tick_disable_::bool
-tick_period_:double
-player_tokens_:PlayerTokens
-players_:Players
-autosave_file_: optional<string_view>
-app_repo_: ApplicationRepository&
+Application(model::Game&, bool, bool, uint, optional<string_view>, ApplicationRepository&)
+JoinPlayerToGame(Map::Id, string_view): JoinGameResult
+FindMap(model::Map::Id&):model::Map*
+FindPlayerByToken(Token&):shptr<Player>
+GetMaps(): [model::Map]&
+GetPlayerGameState(shptr<Player>):GameState
+GetDogsInSession(shptr<Player>):[shptr<Dog>]
+GetPlayersInSession(shptr<Player>):[shptr<Player>]
+GetTickPeriod():double
+GetLostObjects(shptr<Player>): [shptr<LostObjects>]
+GetDogById(size_t): shptr<model::Dog>
+GetAutosaveFile(): optional<string_view>
+SetDogAction(shptr<Player>, ActionMove):
+IsTestMode():bool
+IsRandomSpawnPoint():bool
+MoveDogs(time_period:double):
+GetChampions(start: size_t, max_items: size_t): [Champion]
-PickUpItems(shptr<GameSession>, [Gatherer]&, idx_to_dog):
-BringItemsToOffices(shptr<GameSession>, [Gatherer]&, idx_to_dog)
-DeletePlayer(shptr<Player>):
friend SerializeState(Application&): stringstream
friend DeSerializeState(stringstream&, Application&):
<<Interface>>
ApplicationRepository
+ Save(Champion):
+ GetChampions(): [Champion]
Класс для передачи данных в функцию поиска коллизий
using Item = LostObject
ItemGatherer:
+ItemGathererProvider
-items_count_: size_t
-items_: [shptr<LostObject>]&
-gatherers_count: size_t
-gatherer_: [Gatherer]&
+ItemsCount(): size_t
+GatherersCount(): size_t
+GetItem(idx: size_t): Item
+GetGatherer(idx: size_t): Gatherer
child
Use
Use
using PlayersAll = std::list<std::shared_ptr<Player>>;
using PlayerIt = PlayersAll::const_iterator;
player_id = dog_id
Players
-players_: PlayersAll
-next_dog_id_:size_t
-map_id_to_it_:{size_t, PlayersIt}
+Players() = default
+Players(PlayersAll, size_t)
+Add(string, shptr<GameSession>, bool) :shptr<Player>
+FindPlayerByDogId(dog_id:size_t): shptr<Player>
+GetPlayers(): PlayersAll&
+GetNextDogId(): size_t
+Delete(shptr<Player>):
struct GameState
+dog_id: size_t
+state: model::DogState
+picked_objects: [PickedObjects]&
+scores: size_t
struct JoinGameResult
+player_token: Token*
+dog_id: size_t
+error: JoinGameErrorCode
enum JoinGameErrorCode
....
struct Champion
+name: string
+score: size_t
+play_time: double
enum ActionMove
....
Player
-session_:shptr<GameSession>
-dog_:shared_ptr<Dog>
+Player(model::Dog, shptr<model::GameSession>)
+GetDog():shptr<model::Dog>
+GetGameSession():shptr<model::GameSession>
GetId(): size_t
GetName(): string&
PlayerTokens
-random_device:random_device
-generator1:mt19937_64
-generator2:mt19937_64
-token_to_player_:unordered{Token, shptr<Player>}
+AddPlayer(shptr<Player>):shptr<Player>
+FindPlayerByToken(Token&):shptr<Player>
+AddRestoredToken(Token&, sptr<Player>):
+GetTokenToPlayer(): unordered{Token, shptr<Player>}
+Delete(shptr<Player>):
using Token = util::Tagged<std::string, detail::TokenTag>
players::detail::
struct TokenTag
+tag[2]:uint64_t
+TokenTag(uint64_t, uint64_t)
+Serialize():string
Вызывает
Вызывает
AutosaveState(Application&, string_view):
SerializeState(Application&): stringstream
DeSerializeState(stringstream&, Application&):
Реализует основные прикладные сценарии использования:
- вход в игру;
- получение списка игроков;
- запрос игрового состояния.
Хранит информацию об игроках и их токенах авторизации
Файл collision_detector.h (collision_detector::)
Осуществляет расчёт столкновений персонажей с игровыми объектами на карте
struct Gatherer
+start_pos: Position
+end_pos: Position
+width: double
struct GatheringEvent
+item_id: size_t
+gatherer_id: size_t
+sq_distance: double
+time: double
struct CollectionResult
+sq_distance: double
+proj_ratio: double
+IsCollected(double): bool
TryCollectPoint(Position, Position, Position): CollectionResult
FindGatheringEvents(ItemGathererProvider&): [GatheringEvent]
using Item = LostObject
ItemGathererProvider
+ItemsCount() virtual: size_t
+GatherersCount() virtual: size_t
+GetItem(idx: size_t) virtual: Item
+GetGatherer(idx: size_t) virtual: Gatherer
parent
Use
Use
Загружает карту игры
Сериализация
Классы для сохранения и восстановления состояния игры
Файл model_serialization.h (serialization::)
TokenRepr
-value_: string
+TokenRepr() = default
+TokenRepr(Token&)
+Restore(): Token
+serialize(Archive&)
PlayerTokensRepr
-player_id_to_token_str_: unord{size_t, TokenRepr}
+PlayerTokensRepr() = default
+PlayerTokensRepr(PlayerTokens&)
+Restore(Players&): PlayerTokens
+serialize(Archive&)
PlayersRepr
-players_: [PlayerRepr]
-next_dog_id_: size_t
+PlayersRepr() = default
+PlayersRepr(Players&)
+Restore([model::Sessions]): Players
+serialize(Archive&)
PlayerRepr
-dog_repr_: DogRepr
-map_id_string_: string
+PlayerRepr() = default
+PlayerRepr(Player&)
+Restore([model::Sessions]): Player
+serialize(Archive&)
DogRepr
-id_: size_t
-name_: string
-state_: model::DogState
-scores_: size_t
-objects_repr_: [PickedObjectsRepr]
+DogRepr() = default
+DogRepr(LostObject&)
+Restore(): Dog
+serialize(Archive&)
LostObjectRepr
-id_: size_t
-type_: size_t
-position_: model::Position
+LostObjectRepr() = default
+LostObjectRepr(LostObject&)
+Restore(): LostObject
+serialize(Archive&)
GameSessionRepr
-map_id_str_: string
-last_object_id_: size_t
-dog_ids_: [size_t]
-lost_objects_repr_: [LostObjectRepr]
+GameSessionRepr() = default
+GameSessionRepr(GameSession&)
+Restore(Map*): GameSession
+serialize(Archive&)
template <typename Archive> serialize(Archive& ar, Position& point):
template <typename Archive> serialize(Archive& ar, Velocity& vec):
template <typename Archive> serialize(Archive& ar, DogState& state):
template <typename Archive> serialize(Archive& ar, TokenTag& token_tag):
PickedObjectRepr
-id_: size_t
-type_: size_t
+PickedObjectRepr() = default
+PickedObjectRepr(PickedObject&)
+Restore(): PickedObject
+serialize(Archive&)
Файл command_line.h (start_options::)
библиотека boost
Использован в main()
struct Args
...
ParseCommandLine(int, char**):optional<Args>

Код обработки запросов

библиотека boost

Файл request_handler.h (http_handler::)

Файл api_handler.h (http_handler::)

APIHandler

-app_:Application&

+APIHandler(Application&)

+ReturnAPIResponse(StringRequest&&, uri:string) :StringResponse

template <typename Fn>
-ExecuteAuthorized(auth_header:string_view, Fn&&) :StringResponse

-HandleJoining(body:string_view, uint, bool):StringResponse

-HandlePlayersList(shptr<Player>, uint, bool, head_only:bool) :StringResponse

-HandleGameState(shptr<Player>, uint, bool, head_only:bool) :StringResponse

-HandleGameState(shptr<Player>, uint, bool, head_only:bool) :StringResponse

-HandleAction(shptr<Player>, body:string_view, uint, bool) :StringResponse

-HandleTick(body:string_view, uint, bool):StringResponse

-HandleChampions(StringRequest&&): StringResponse

RequestHandler

-api_handler_:unique_ptr<APIHandler>

-root_path_:filesystem::path

-api_strand_:Strand

+RequestHandler(players::Application&, filesystem::path, Strand api_strand)

-ReturnFile(filesystem::path, uint, bool): variant<EmptyResponse, StringResponse, FileResponse>

template <typename Body, typename Allocator, typename Send, typename LogFunc>
+operator()(request&&, Send&&, LogFunc&&):

AssureContentTypeIsJSON(string_view, unsigned, bool): optional<StringResponse>

AssureMethodIsPOST(http::verb, unsigned, bool): optional<StringResponse>

AssureMethodIsGetHead(http::verb, unsigned, bool): optional<StringResponse>

LoadGETParams(string_view): pair<size_t, size_t>

struct FileExt

...

GetContentType(string):string_view

DecodeURI(string_view):string

IsSafePath(filesystem::path&, filesystem::path):int

TryToExtractToken(string_view):optional<Token>

MakeStringResponse(...):StringResponse

struct ContentType

...

HTTP-сервер

библиотека boost

Файл http_server.h (http_server::)

parent

child

SessionBase

-request_:HttpRequest

-buffer_:flat_buffer

#stream_:tcp_stream

#SessionBase(socket&&)

-Read()

-OnRead(error_code, size_t)

#template <typename Body, typename Fields> Write(response<Body, Fields>&&)

-OnWrite(bool, error_code, size_t)

-Close()

-HandleRequest(HttpRequest&&): virtual

-GetSharedThis(): shared_ptr<> virtual

template <typename RequestHandler> class Session

-request__handler_:RequestHandler

+template <typename Handler> Session(socket&&, Handler&&)

-HandleRequest(HttpRequest&&): override

-GetSharedThis(): shared_ptr<> override

/*
Внутри Session управление по очереди получают методы: Read → OnRead → HandleRequest → Write → OnWrite, а затем снова Read.
Здесь нет рекурсии, так как методы OnRead и OnWrite вызываются асинхронно — их вызывает не сама программа, а операционная система и Boost.Asio.
*/

Use

template <typename RequestHandler> class Listener

-ioc_:io_context&

-acceptor_:acceptor

-request__handler_:RequestHandler

-AsyncRunSession(socket&&):

-DoAccept():

-OnAccept(error_code, socket):

template<typename Handler>
+Listener(io_context&, endpoint&, Handler&&)

+Run():

Use

/* здесь осуществляется запуск сервера*/
template <typename RequestHandler>
ServeHttp(io_context&, endpoint&, RequestHandler&&)

call

callback Handler

Реализует обработку Http-запросов

Формирует текст ответов в формате JSON

Логгер
(паттерн "декоратор" для RequestHandler)

Файл logging_handler.h (logging_handler::)

библиотека boost

template <class MainRequestHandler> class LoggingRequestHandler

-decorated_:shared_ptr<MainRequestHandler>

-LogRequest(StringRequest&, string): static

-LogResponse(string, int, int, string_view): static

+LoggingRequestHandler(shared_ptr<MainRequestHandler>)

+template <typename Body, typename Allocator, typename Send> void operator()(request<>&&, Send&&, endpoint&)

LogStartServer(endpoint&):

LogStopServer(int, string):

LogNetworkError(int, string_view, string_view):

GetTimeStampString():string

LogFormatter(record_view&, formatting_ostream&):

Вызывает сценарии использования

Формирует текст записи в лог

Обработчик JSON

Файл json_loader.h (json_loader::)

LoadGame(filesystem::path&):Game

template <typename BoostJSONType, typename ValueType>
ReadOptionalValue(BoostJSONType&, string_view, ValueType) : ValueType

LoadAndAddRoads(...):

LoadAndAddBuildings(...):

LoadAndAddOffices(...):

LoadAndAddLootTypes(...):

LoadAndSetLootSettings(...):

GetLogServerStart(timestamp:string, srv_addr:string, port:int): string

GetLogServerStop(timestamp:string, return_code:int, exception:string): string

GetLogRequest(timestamp:string, client_addr:string, uri:string, method:string): string

GetLogResponse(timestamp:string, client_addr:string, calc_time:int, response_code:int, content_type:string): string

GetLogError(timestamp:string, error_code:int, error:string, where:string): string

GetListOfMaps(Game&): string

GetMap(Map::Id&, Game&): string

GetPlayerAddedAnswer(token:string, player_id:size_t): string

GetSessionPlayers(GameSession::Dogs&): string

LoadJSONJoinGame(requestbody:string_view): JoinGame

LoadActionMove(requestbody:string_view): optional<string>

LoadTimeDelta(requestbody:string_view): optional<double>

GetRoadsArray(...): json::array

GetBuildingsArray(...): json::array

GetOfficesArray(...): json::array

GetLootTypesArray(...): json::array

struct JoinGame

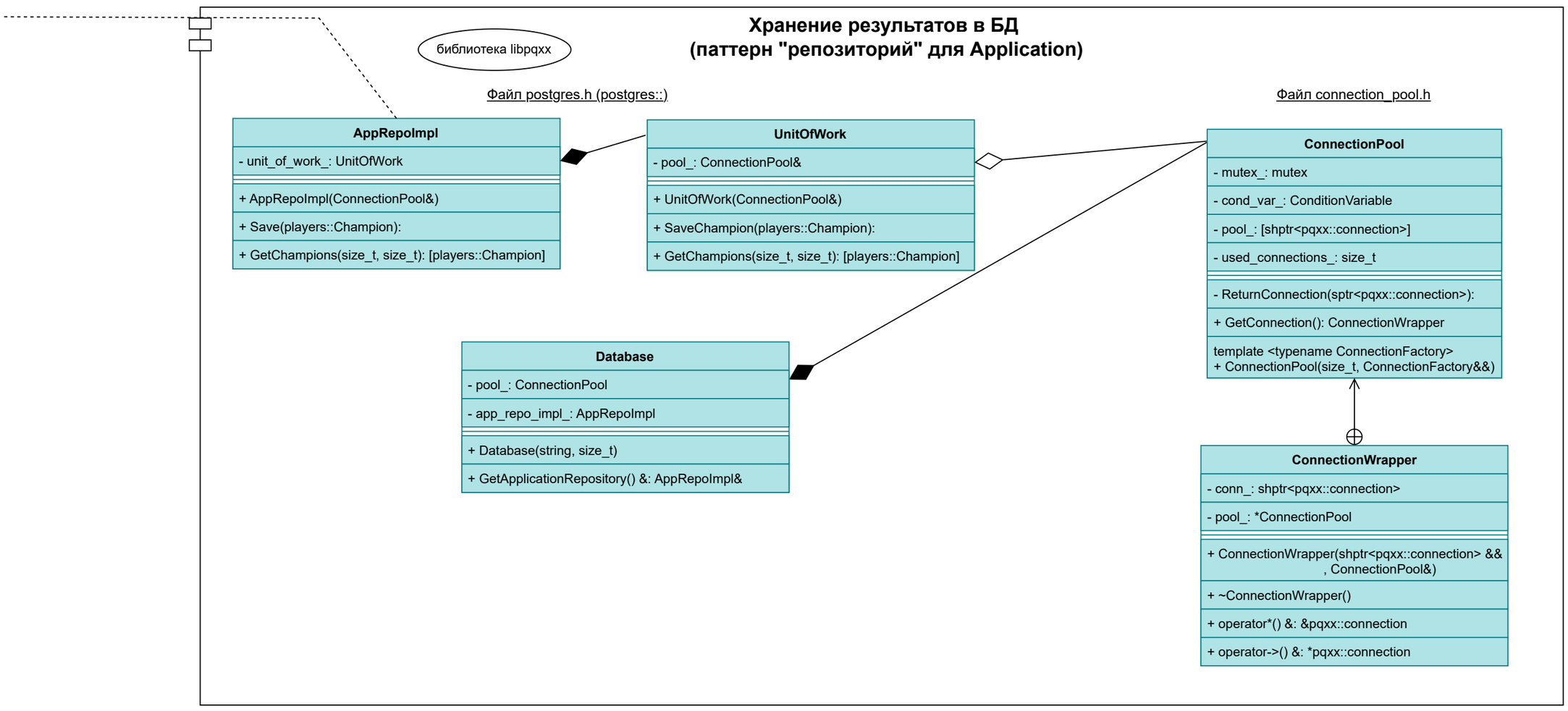
...

MakeErrorString(err_code:string&&, err_text:string&&): string

MakeChampionsAnswer([players::Champion]): string

MakeGameStateAnswer([players::GameState]): string

DogDirectionToString(direction:model::Direction): string


Хранение результатов в БД
(паттерн "репозиторий" для Application)
библиотека libpqxx
Файл postgres.h (postgres::)
Файл connection_pool.h
AppRepoImpl
- unit_of_work_: UnitOfWork
+ AppRepoImpl(ConnectionPool&)
+ Save(players::Champion):
+ GetChampions(size_t, size_t): [players::Champion]
UnitOfWork
- pool_: ConnectionPool&
+ UnitOfWork(ConnectionPool&)
+ SaveChampion(players::Champion):
+ GetChampions(size_t, size_t): [players::Champion]
ConnectionPool
- mutex_: mutex
- cond_var_: ConditionVariable
- pool_: [shptr<pqxx::connection>]
- used_connections_: size_t
- ReturnConnection(sptr<pqxx::connection>):
+ GetConnection(): ConnectionWrapper
template <typename ConnectionFactory>
+ ConnectionPool(size_t, ConnectionFactory&&)
Database
- pool_: ConnectionPool
- app_repo_impl_: AppRepoImpl
+ Database(string, size_t)
+ GetApplicationRepository() &: AppRepoImpl&
ConnectionWrapper
- conn_: shptr<pqxx::connection>
- pool_: *ConnectionPool
+ ConnectionWrapper(shptr<pqxx::connection> &&
, ConnectionPool&)
+ ~ConnectionWrapper()
+ operator*() &: &pqxx::connection
+ operator->() &: *pqxx::connection